ECE656 Course Project

Ricardo Alejandro Manzano, ramanzano@uwaterloo.ca

Jingyu Niu, jingyu.niu@uwaterloo.ca

Zhilun Chang, zhilun.chang@uwaterloo.ca

ABSTRACT:

Yelp database is a useful collection of entities related through a well structured and organized relations about businesses, users and reviews, which provides useful information to owners, users, students, data analysts to find interesting patterns to predict or analyze data in different fields. In the present work, this database has been used to evaluate different hypothesis about prediction using techniques and methods in pre-processing , data cleaning, data analysis and security to database access to insurance a trustworthy results of the hypothesis.

Keywords: YELP, Data cleaning & indexing, Data analysis

1. Introduction

Yelp website has become one of the most popular sites choose by users to find suggestions and leave reviews about different services in different kind of businesses. Due flexibility of the options that the website offers to users such as reviews, stars, tips, likes, cools and other useful information, the website has reached a millions of reviews and thousands of users. All the information of this website feeds the database of Yelp. Thus, this database has favorable information in order to do data mining and prediction of data.

The present work has been divided in some phases which search to provide confident results about data analysis. The first phase is to determine an entitiy-relational model, this phase will identify how the different actors in the database as business, users, reviews, tips, friends, hours and others are related among them through reasoned relations. Of this phase depends the next phases. Furthermore, a correct of identification of the E-R model ensures for example other important properties to avoid redundancy in the database which can cause different problems in the moment to process data such as inconsistency of results or delays in processing queries.

The second phase is pre-processing the data. This phase involves to verify self-consistency and sanity of the data which is stored in the database YELP. Self-consistency is related specifically with same attributes contain in different relations. It means that if an attribute is a primary key in one relation, the instances of the same attribute in other relations should be consistent. It is impossible that an attribute which is a foreign key has instances which are not in the relation which has the primary key. While sanity is related with the validity of some facts that in reality cannot be occurred, especially validation about dates which are contained in the database. To illustrate, the sanity phase takes in account validation of data which cannot be in the future, or data which can exist before the creation of the YELP database. Furthermore, the sanity phase implicates some important validations such as a user cannot have a review before that his account was created, and other validations which will expose in the section. Finally, it is important to denote that the sanity goes beyond the checking of the facts that we exposed above. The present project exposes the data should be chosen in order to have consistent results. To do viable the processing of all the information exposed above and considering the amount of data stored in the database in the project was created indexes in order to improve the time of processing of each query.

The third phase considered in this project is data-analysis which involves some stages. The first stage is the suitable selection of data depending that the fact which will be analyzed. It is important to highlight, that the efficient selection of data depends of the indexes which will be created analyzing the performance of queries. The second stage considered the trend or coeffective factors based on the diagrams we plotted, and final stage is analyzing the data in order to infer a result.

Finally, the fourth phase is related with the security and application development in the database of YELP. This phase considered the creation of different kind of users with distinct permissions to handle data of database. As a plus is exposed a simple API developed in Phyton which allows to give more security to the database depending of the kind of user.

1.1 E-R diagram

The E-R diagram we created for the YELP database is shown as following.

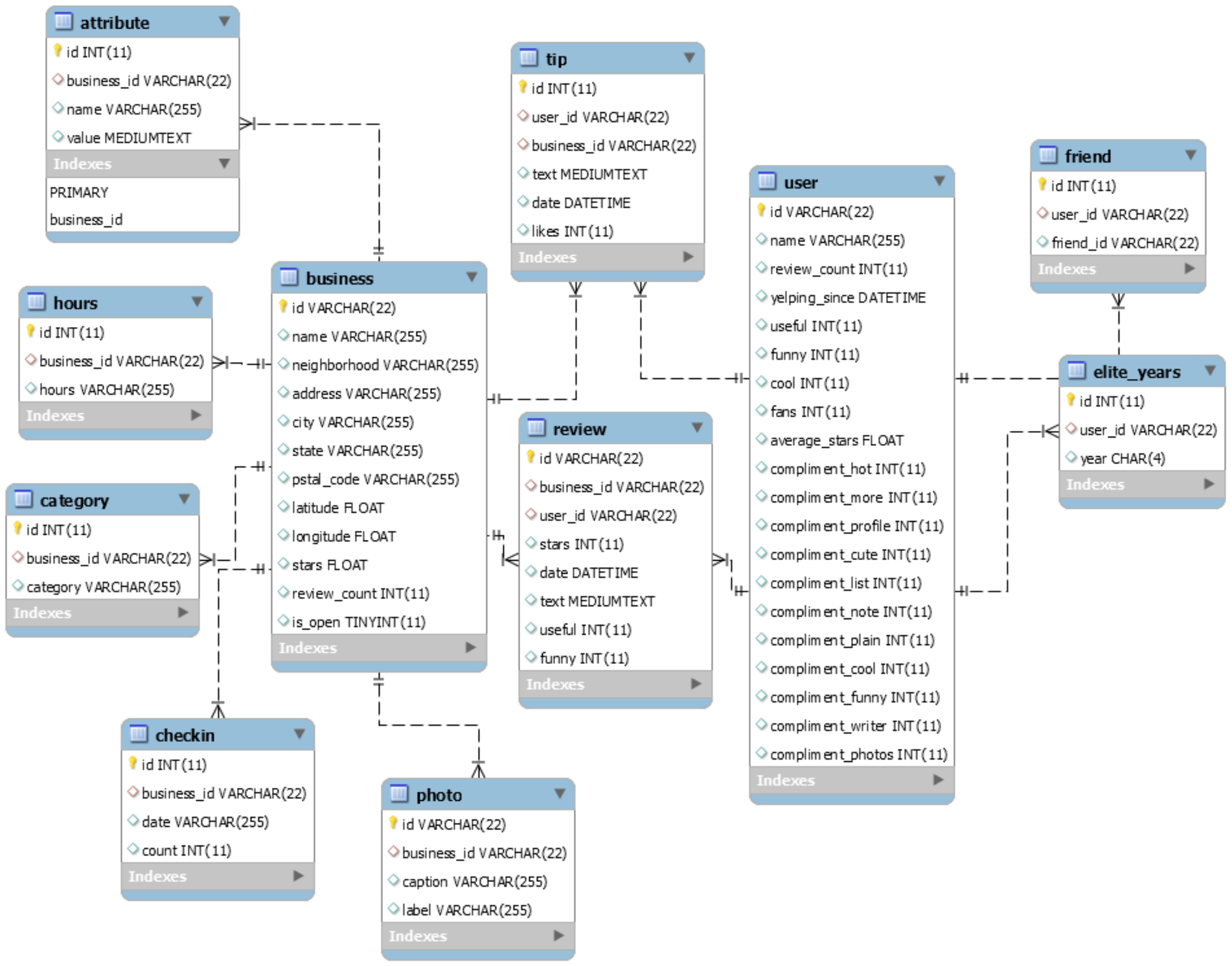


Figure 1.1 E-R diagram

2. Data cleaning and self-consistency – question part 1

To execute data cleaning and check self-consistency in the database in the present project has been stablished a plan which has been divided in two parts.
The first part checked self-consistency of some attributes in the database, while the second part checked sanity in the database taking in account some facts according with the information provided by YELP database and valid assumptions proposed in the present work.

2.1 Part 1

It is important to highlight that the database has only primary keys in the beginning. Therefore, it is important to erase all the information that can be inconsistent with the database, and after that create foreign keys.
To check self-consistency have been defined primary keys and foreign keys in each relation depending of correlated information among all the relations. The goal of this part is to determine if the attributes which are foreign keys in some relations can have entities which are not contained in the relation which has the primary key. These entities should be erased of the database because it is inconsistent information.

2.1.1 Check self-consistency in all the relations which have information about Business.

The database has the relation business where id(related with business_id in other relations) is the primary key. Some other relations have the attribute business_id as a foreign key as we exposed in the table below. To check self-consistency in the present work has been analyzed if the relations, where business_id is foreign Key, have values of business_id which are not contained in the relation business which has the primary key. As we can see in the results exposed in the table below, there are not inconsistency of this type due to the fact that all the results are EMPTY SET. Therefore, it is not necessary to erase data to guarantee self-consistency in information related to business.

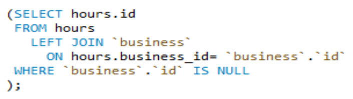
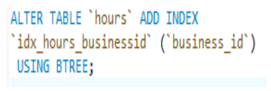
Relation	Pk	Foreign Keys	Query to check self-consistency	Execution time without index	Index used to improve query	Result and time of execution with index
Business	id		 (SELECT hours.id FROM hours LEFT JOIN `business` ON hours.business_id= `business`.`id` WHERE `business`.`id` IS NULL); In order to calculate self-consistency of business_id in all the tables that contain this attribute as a foreign key. We use a query like the query that we showed above. We only changed the relation hours by the relation that we want to check self-consistency	Time out	 ALTER TABLE `hours` ADD INDEX `idx_hours_businessid` (`business_id`) USING BTREE; In order to improve the time of query we created an index for each table. The index is like the exposed above only we changed the relation	
Hours	id	business_id				Empty set (11.24 sec)
Category	id	business_id				Empty set (9.65 sec)
Attribute	id	business_id				Empty set (17.49 sec)
Checkin	id	business_id				Empty set (56.59 sec)
Tip	id	business_id, user_id				Empty set (15.19 sec)
Photo	id	business_id				Empty set (4.13 sec)
Review	id	business_id, user_id				Empty set (1 min 18.77 sec)

Figure 2.1

2.1.2 Check self-consistency in all the relations which have information about User.

The database has the relation user which has the attribute id(related with user_id in other relations) as a primary key. Other relations have the attribute user_id as a foreign key as we can see in the table below. To check self-consistency is necessary to evaluate if there are values of user_id as a foreign key which do not exist in the attribute id(primary key) of the relation user. This values should be erased. As we can see in the table below the results are Empty Set, it means that there are not inconsistent information in the attribute user_id in other relations.

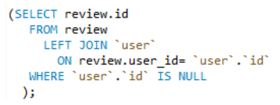
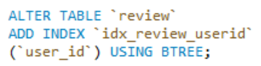
Relation	Pk	Foreign Keys	Query to check self-consistency	Execution time without index	Index used to improve query	Result and time of execution with index
User	id		 (SELECT review.id FROM review LEFT JOIN `user` ON review.user_id= `user`.`id` WHERE `user`.`id` IS NULL); In order to calculate self-consistency of user_id in all the tables that contain this attribute as a foreign key. We use a query like the query that we showed above. We only changed the relation review by the relation that we want to check self-consistency	Time out	 ALTER TABLE `review` ADD INDEX `idx_review_userid` (`user_id`) USING BTREE; In order to improve the time of query we created an index for each relation. The index is like the exposed above only we changed the relation	
Friend	id	user_id				Empty set (9 min 2.40 sec)
Elite_years	id	user_id				Empty set (13.01 sec)
Review	id	business_id, user_id				Empty set (1 min 7.98 sec)
Tip	id	business_id, user_id				Empty set (23.76 sec)

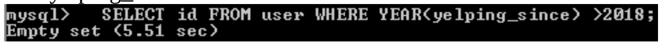
Figure 2.2

2.2 Part 2

The second part comprises sanity of some data that can be inconsistent especially when we validate data related to dates and other valid assumptions.

2.2.1 Check sanity in the table User according to date of creation of an account in YELP and future dates

• It is impossible that a user can create an account in the future in YELP. Therefore, it is necessary to erase all the entities in the table User which have in the attribute yelping_since a date in the future.

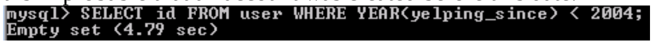


```
mysql>   SELECT id FROM user WHERE YEAR(yelping_since) >2018;
Empty set (5.51 sec)
```

Figure 2.3

As we can see in the figure above there are not users created in the future because the result of the query is Empty set. Thus, it is not necessary to erase nothing in the database

• It is impossible that a user can create an account before the date of creation of YELP. According to the web page YELP was founded in 2004. For this reason it is improbable that an account was created before this date.



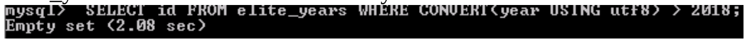
```
mysql> SELECT id FROM user WHERE YEAR(yelping_since) < 2004;
Empty set (4.79 sec)
```

Figure 2.4

As we can see in the figure above there are not users created before the creation of the platform YELP because the result of the query is Empty set. Thus, it is not necessary to erase nothing in the database

2.2.2 Check sanity in the table ELITE_YEARS according to date of creation of an account in YELP, future dates, and other consideratiions

• It is impossible that a user can become an elite member of the YELP in a date in the future. Therefore, it is necessary to erase all the entities in the table Elite_years which have in the attribute year a date in the future.



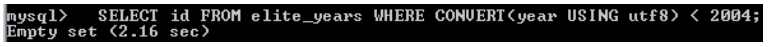
```
mysql>  SELECT id FROM elite_years WHERE CONVERT(year USING utf8) > 2018;
Empty set (2.08 sec)
```

Figure 2.5

As we can see in the figure above there are not users in the group elite members with a date in the future because the result of the query is Empty set. Thus, it is not necessary to erase nothing in the database.

• It is impossible that a user can become an elite member of the YELP in a date before YELP database was created 2004. Thus, it is necessary to erase the entities which have in the attribute year a year less than 2004.

Results:



```
mysql>   SELECT id FROM elite_years WHERE CONVERT(year USING utf8) < 2004;
Empty set (2.16 sec)
```

Figure 2.6

As we can see in the figure above there are not users in the group elite members with a date less than the year where Yelp was created because the result of the query is Empty set. Thus, it is not necessary to erase nothing in the database.

• Finally, it is improbable that a user, who created an account in YELP in 2008, becomes an elite member before 2008. In this query we are analyzing if the there is a user that it is in the table elite before that the account in yelp had been created for the user. This information is inconsistent; therefore, we have to erase it.

Without indexes is impossible to obtain a result. Thus, in this query has been created the following indexes.

```
ALTER TABLE `user` ADD INDEX `idx_eliteyears_useryear` (`yelping_since`) USING BTREE;
ALTER TABLE `elite_years` ADD INDEX `idx_useryear_eliteyears` (`year`) USING BTREE;
```

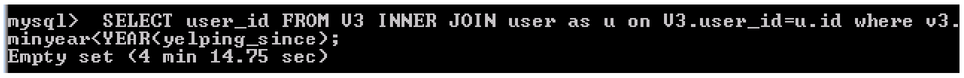
Figure 2.7

After that a view has been created in order to find the minimum year per user in order to compare this with the year when the user created an account

```
create view  v3 as select user_id, min(CONVERT(year USING utf8)) AS minyear from elite_years group by user_id order by user_id;
```

Fugure 2.8

Results:



```
mysql>  SELECT user_id FROM V3 INNER JOIN user as u on V3.user_id=u.id where v3.
minyear<YEAR(yelping_since);
Empty set (4 min 14.75 sec)
```

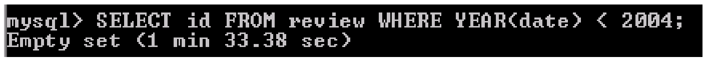
Figure 2.9

As we can see in the figure above there are not users who are in elite group before they had created an account in Yelp.

2.2.3 Check sanity in the table Reviews according to date of creation of an account in YELP, future dates, and other considerations

• It is impossible that a review can have a date less than the date of creation of YELP database 2004. Thus, it is necessary to erase all the entities that has this information.

Results:



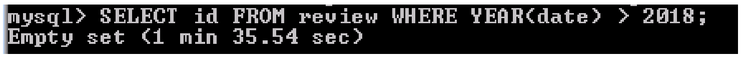
```
mysql> SELECT id FROM review WHERE YEAR(date) < 2004;
Empty set (1 min 33.38 sec)
```

Figure 2.10

As we can see in the figure above there is not a review with year less than 2004(creation of the database Yelp).

• It is impossible that there is a review of the future. For this reason, it is necessary to erase the reviews with a date greater than 2018

Results:



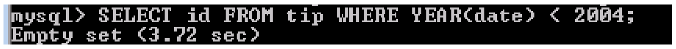
```
mysql> SELECT id FROM review WHERE YEAR(date) > 2018;
Empty set (1 min 35.54 sec)
```

Figure 2.11

As we can see in the figure above there is not a review with date in the future. Therefore, there is not inconsistent data in this relation.

2.2.4 Check sanity in the table Tip according to date of creation of an account in YELP, future dates, and other considerations

• It is impossible that a user may have given a tip in a date less than the date of creation of YELP database 2004. Thus, it is necessary to erase all the entities that has this information.

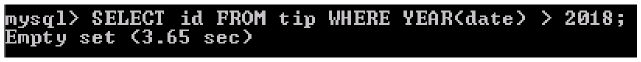


```
mysql> SELECT id FROM tip WHERE YEAR(date) < 2004;
Empty set (3.72 sec)
```

Figure 2.12

As we can see in the figure above there are not tips with date less than the year of creation of Yelp. For this reason, there is not inconsistency in the relation

• It is impossible that a user may have given a tip of the future. For this reason, it is necessary to erase the reviews with year greater than 2019



```
mysql> SELECT id FROM tip WHERE YEAR(date) > 2018;
Empty set (3.65 sec)
```

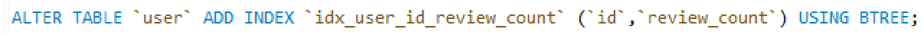
Figure 2.13

As we can see in the figure above there are not tips with date in the future. For this reason, there is not inconsistency in the relation

2.2.5 Check sanity number of reviews written by a user recorded in the User table cannot be smaller than the number of reviews in the relation review

Yelp web page provides useful information about this inconsistency. The relation user has the attribute review_count which has the total number of reviews done by a user while the number of reviews in the relation review are the reviews which were <u>recommended</u>. In addition, the reviews in the relation review only includes businesses that have had at least 3 reviews older than 14 days. Thus, it is impossible the attribute review_count in the relation user can be less than the count computed in the relation review.

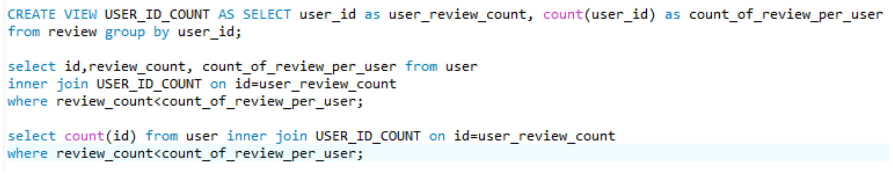
To find this inconsistency has been created the next indexes.



```
ALTER TABLE `user` ADD INDEX `idx_user_id_review_count` (`id`,`review_count`) USING BTREE;
```

Figure 2.14

The next queries help to find this inconsistency

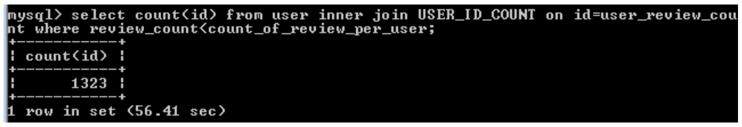


```
CREATE VIEW USER_ID_COUNT AS SELECT user_id as user_review_count, count(user_id) as count_of_review_per_user
from review group by user_id;

select id,review_count, count_of_review_per_user from user
inner join USER_ID_COUNT on id=user_review_count
where review_count<count_of_review_per_user;

select count(id) from user inner join USER_ID_COUNT on id=user_review_count
where review_count<count_of_review_per_user;
```

Figure 2.15



```
mysql> select count(id) from user inner join USER_ID_COUNT on id=user_review_cou
nt where review_count<count_of_review_per_user;
+-----------+
| count(id) |
+-----------+
|      1323 |
+-----------+
1 row in set (56.41 sec)
```

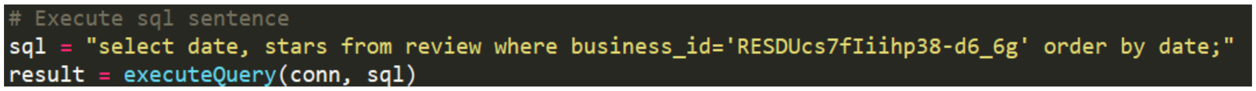
Figure 2.16

As we can see in the figure above there are 1323 users who presented this inconsistency. Thus, it is necessary to erase this users of the relation User and it is necessary to erase the reviews of these users in the relation review. It cannot affect the database because 1323 users represents 0.01% of the users in the database (total number of users in the database 1326101).

3. Data analysis – question part 1

3.1 Based on the data, determine if a business is declining or improve

To analyse the performance of a business, we simply choose a particular business example and get its ratings based on the time sequence. Firstly, we collected the useful data from SQL query and read them through Python just as following:



```
# Execute sql sentence
sql = "select date, stars from review where business_id='RESDUcs7fIiihp38-d6_6g' order by date;"
result = executeQuery(conn, sql)
```

Figure 3.1 read SQL query

Secondly, we calculate the average ratings of the business from the first day to the other days by recording the average value before now and the total number of records until now. The formula is set up as:

$$average\_ratings_{until\_now} = \frac{average\_ratings_{before\_now} * number\_records_{before\_now} + SUM(ratings_{now})}{number\_records_{before\_now} + number\_records_{today}}$$

Then, we also need to see the number of ratings per date during the time sequence. So, we created a query to count that.

Results:

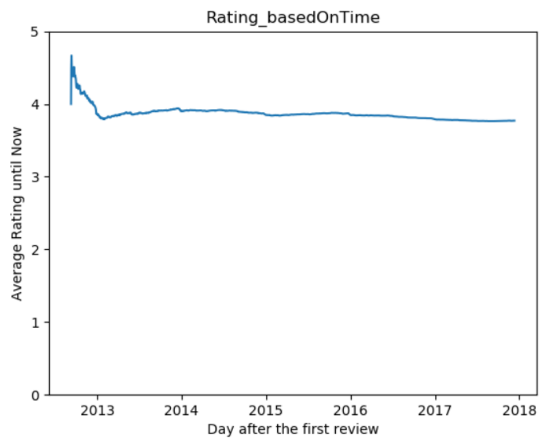


Figure 3.2 Rating_basedOnTime

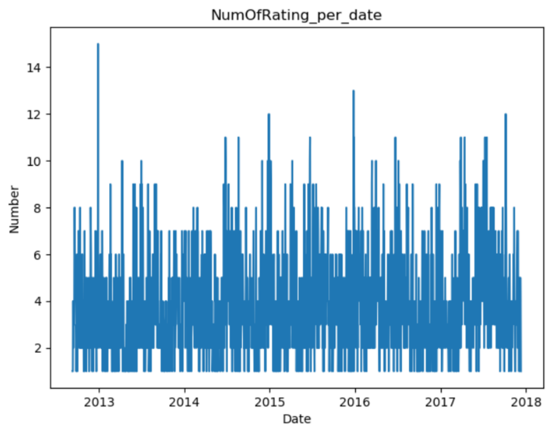
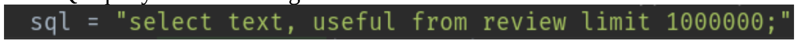


Figure 3.3 NumOfRating_per_date

As can be seen from figure 3.2, the curve fluctuates significantly before 2013(initial time when the business is open), then it remains steady during the period from 2013 to 2018 and remains a relatively high ratings(3.9 stars) during that period. According to that, we can see that after it becomes steady in 2013, the business remains a really good service quality. There is no improvement or decline. When consider the fluctuation part(before 2013), we need to compare it with our figure 3.3. Since there is no enough ratings at the begining, a particular rating will give a huge impact on the average result, which results in fluctuation.

3.2 Based on the data, determine the relation between review length and useful.

To indentify the relation between review length and useful we have to select data from review. Typically we selected 1000000 data from review table. The SQL query is as following:



```
sql = "select text, useful from review limit 1000000;"
```

Figure 3.4 read SQL query

To solve the question, we use text(review) length as x-axsis, and the number of user who regard the text is average useful number per specific length of text(count(useful number of a text_length L)/count(number of text with text_length L)) as y-axsis. By doing this, we can conclude diagrams showing the trend with the increasing the text_length. The trend could tell us the relation between the the two parameters. Besides, we use pearson correlation testing to test the correlation between the two paramters. From the figure we used as a test, we see the data that have a text_length longger than 200 is really small and is not respectively. So we select data with text_length between 0 and 200. And the basic diagram shows as following:

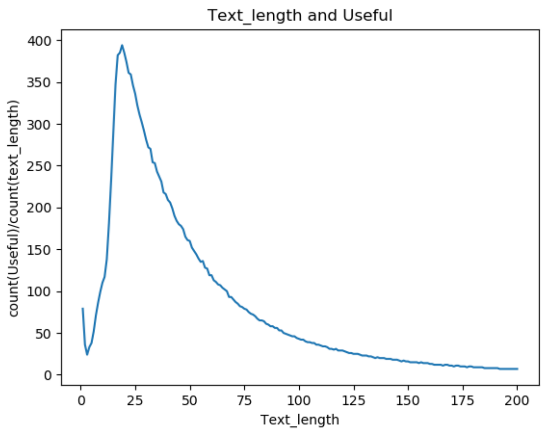
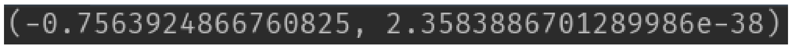


Figure 3.5 Text_length-Useful figure 1

The pearson correlation calculated is shown below:



```
(-0.7563924866760825, 2.3583886701289986e-38)
```

Figure 3.6 Correlation of figure 1

This figure illustrates the trend with the increasing of text_length. And we can see there is a maximum average of useful number and after that we can see a decreasing. The correlation is -0.76 which means there is a medium negative correlation between the two parameters.

To view the diagram better we divided the data into two parts. The first one with text_length from 0 to 20. And the figure of this part and correlation testing result are as following:

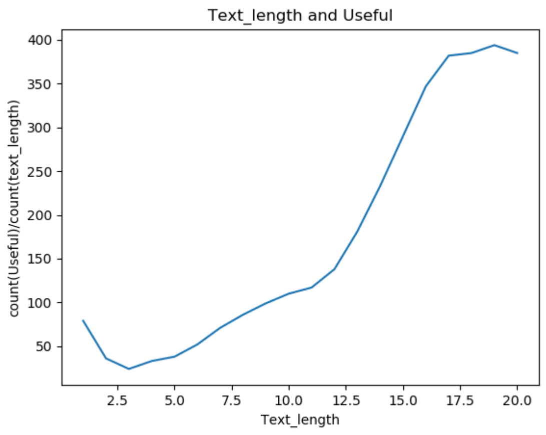
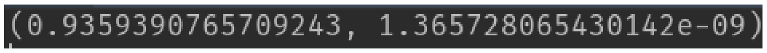


Figure 3.7 Text_length-Useful figure 2



```
(0.9359390765709243, 1.365728065430142e-09)
```

Figure 3.8 Correlation of figure 2

The figure shows that the average number of useful is increasing before the text_length reaching about 18. Though there is a small decreasing before text_length at 2.5. From the above analysis we know that after the maximum average of useful number the trend starts to decrease. A text with 18 length is most useful according to the data. The correlation of the two parameters in this figure is about 0.93 which is of high positive correlation.

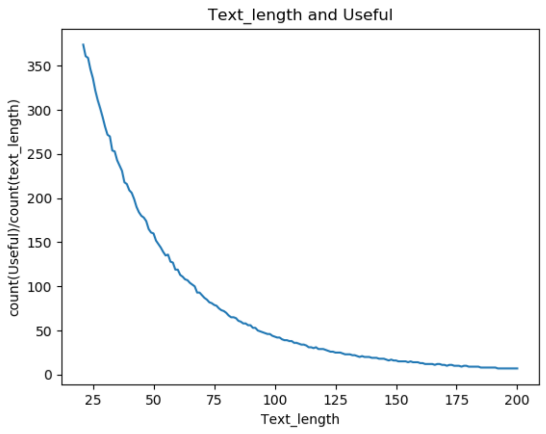
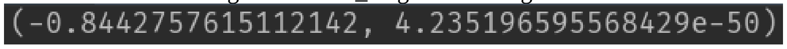


Figure 3.9 Text_length-Useful figure 3



```
(-0.8442757615112142, 4.235196595568429e-50)
```

Figure 3.10 Correlation of figure 3

The figure shows that with the increasing of text_length longger than 20, the average of useful number is decreasing. The pearson number of this figure is -0.84 whcih means the correlation between the paramters is of high nagetive correlation. The data and analysis result shows that people tend to mark a review with a short length but not very short with a more positive evaluation. We guess this situation is because a review with a very short length contains little information and it is not convinced. People would like to select a short or medium length review to have a look instead of a long one.

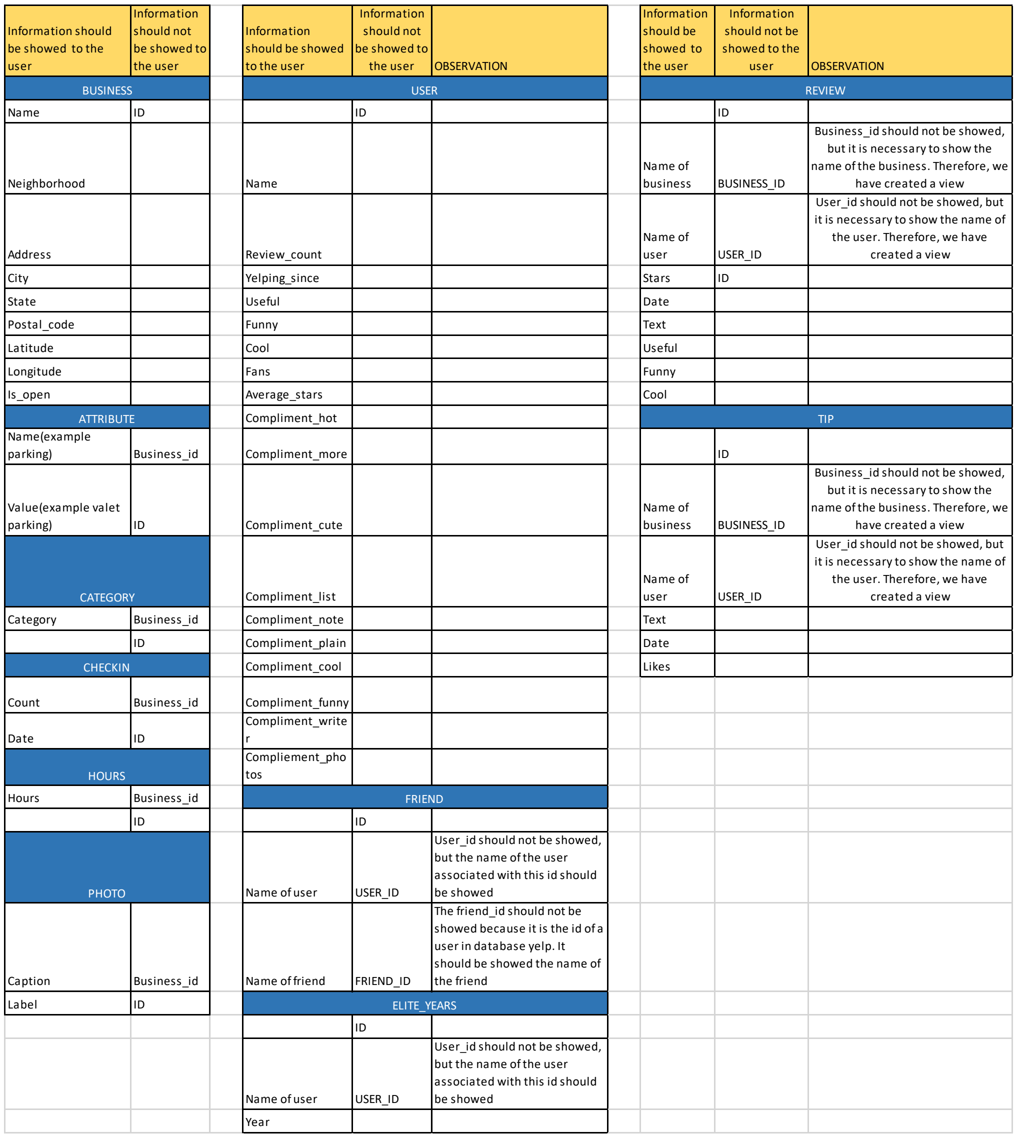
4. User access – question part 2

In this part of the project it is important to analyze what kind of actions a user can execute in the database depending of what kind of user is.

In this work has been defined 5 kind of users. The casual user, a registered user in yelp, a data analyst user, a developer user, and a root user.

4.1 Casual User

The casual user is a person who enters to the website YELP in order to search some information about a restaurant, bar or another business. This user does not have an account in YELP; thus, this kind of user cannot submit a review. This kind of user can see only informative data about the businesses, user and reviews. This kind of user for example cannot see the entire relation business; therefore, he cannot see the business_id, user_id or important related information. For this reason, in the next table has been defined what kind of information the casual user can see of the database



Information should be showed to the user	Information should not be showed to the user
BUSINESS	
Name	ID
Neighborhood	
Address	
City	
State	
Postal_code	
Latitude	
Longitude	
Is_open	
ATTRIBUTE	
Name(example parking)	Business_id
Value(example valet parking)	ID
CATEGORY	
Category	Business_id
	ID
CHECKIN	
Count	Business_id
Date	ID
HOURS	
Hours	Business_id
	ID
PHOTO	
Caption	Business_id
Label	ID

Information should showed to the user	Information should not be showed to the user	OBSERVATION
USER		
	ID	
Name		
Review_count		
Yelping_since		
Useful		
Funny		
Cool		
Fans		
Average_stars		
Compliment_hot		
Compliment_more		
Compliment_cute		
Compliment_list		
Compliment_note		
Compliment_plain		
Compliment_cool		
Compliment_funny		
Compliment_writer		
Compliement_photos		
FRIEND		
	ID	
Name of user	USER_ID	User_id should not be showed, but the name of the user associated with this id should be showed
Name of friend	FRIEND_ID	The friend_id should not be showed because it is the id of a user in database yelp. It should be showed the name of the friend
ELITE_YEARS		
	ID	
Name of user	USER_ID	User_id should not be showed, but the name of the user associated with this id should be showed
Year		

Information should be showed to the user	Information should not be showed to the user	OBSERVATION
REVIEW		
	ID	
Name of business	BUSINESS_ID	Business_id should not be showed, but it is necessary to show the name of the business. Therefore, we have created a view
Name of user	USER_ID	User_id should not be showed, but it is necessary to show the name of the user. Therefore, we have created a view
Stars	ID	
Date		
Text		
Useful		
Funny		
Cool		
TIP		
	ID	
Name of business	BUSINESS_ID	Business_id should not be showed, but it is necessary to show the name of the business. Therefore, we have created a view
Name of user	USER_ID	User_id should not be showed, but it is necessary to show the name of the user. Therefore, we have created a view
Text		
Date		
Likes		

Figure 4.1

In order to accomplish the rules that we defined in the above table in the present project has been created the following views which will be authorized to see to the casual user.

VIEW BUSINESS: As we can see below this view does not show the id of business, and this view has all the related information to Business as we can see in Yelp website.

```
CREATE VIEW BUSINESSVIEWBUSINESS1 AS SELECT b.name as Nameofbusiness, b.neighborhood,
b.address, b.city, b.state, b.postal_code,latitude, b.longitude, b.stars, b.review_count, b.is_open,
ba.name as Nameofcategory,ba.value,
bc.category,
bch.count,bch.date,
bh.hours,
bp.caption, bp.label
FROM business as b inner join attribute as ba on b.id=ba.business_id
inner join category as bc on b.id=bc.business_id
inner join checkin as bch on b.id=bch.business_id
inner join hours as bh on  b.id=bh.business_id
inner join photo as bp on b.id=bp.business_id;
```

Figure 4.2

VIEW USER: As we can see below this view does not show the id of user, and this view has all the information about User as we can see in Yelp website. The auxuserfriend is used to find the names of the friends of the user, and added it to final view USERVIEW1. We will give access to this kind of users only to the view USERVIEW1.

```
CREATE VIEW AUXUSERFRIEND AS SELECT uf.user_id, u.name as Name_friend
from friend as uf inner join user as u on u.id=uf.friend_id;

CREATE VIEW USERVIEW1 AS SELECT u.name,u.review_count, u.yelping_since, u.useful, u.funny, u.cool,
u.fans, u.average_stars, u.compliment_hot, u.compliment_more,u.compliment_profile,u.compliment_cute,
u.compliment_list, u.compliment_note, u.compliment_plain,u.compliment_cool,u.compliment_funny,
u.compliment_writer,u.compliment_photos,
Auxuser.Name_friend,
uey.year
from user as u inner join AUXUSERFRIEND as Auxuser on u.id=Auxuser.user_id
inner join elite_years as uey on u.id=uey.user_id;
```

Figure 4.3

VIEW REVIEW: As we can see below this view does not show the id of user neither the id of business. This view only show the names of the user and the name of the business both attributes did unique to every register. In addition, both attributes are related with other attributes in the relation review.

```
CREATE VIEW USER_BUSINESS_REVIEW AS SELECT u.name as Username, b.name as BusinessName,
r.stars,r.date,r.text, r.useful, r.funny, r.cool
from review as r inner join user as u on u.id=r.user_id
inner join business as b on b.id=r.business_id;
```

Figure 4.4

VIEW TIP: As we can see below this view does not show the id of user neither the id of business. This view only show the names of the user and the name of the business both attributes did unique to every register. In addition, both attributes are related with other attributes in the relation Tip.

```
CREATE VIEW USER_BUSINESS_TIP AS SELECT u.name as Username, b.name as BusinessName,
t.text, t.date, t.likes
from tip as t inner join user as u on u.id=t.user_id
inner join business as b on b.id=t.business_id;
```

Figure 4.5

A casual user can only read the views described above. To give them this permission we will executed the next query. It is necessary highlight that this user does not have a password. In addition, the connection to mysql is local. In the next figure is showed how is created a user of this type and what are the permissions assigned to this user.

```
create user 'user1'@'localhost';
grant select on yelp_db.USER_BUSINESS_TIP to 'user1'@'localhost';
grant select on yelp_db.USER_BUSINESS_REVIEW to 'user1'@'localhost';
grant select on yelp_db.USERVIEW1 to 'user1'@'localhost';
grant select on yelp_db. BUSINESSVIEWBUSINESS1 to 'user1'@'localhost';
```

Figure 4.6

4.2 Registered users in YELP

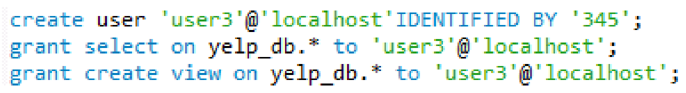
A user ,who has an account in Yelp, can log into the application, and he can execute the next actions:
The first action that a user can execute is read all the views that we described in part one.
Second, the user can only insert a new review, delete or alter previous reviews of his user (the user cannot submit or modify a review of another user), it means that the user can submit or modify a review only with the user which the user log in. This validation does the application. The relation review in the database yelp_db is the only relation where the user can insert, modify or delete his own reviews. In the next figure is showed the code in Mysql to assign the permissions to this kind of user

```
create user 'user2'@'localhost'IDENTIFIED BY '123';
grant select on yelp_db.USER_BUSINESS_TIP to 'user2'@'localhost';
grant select on yelp_db.USER_BUSINESS_REVIEW to 'user2'@'localhost';
grant select on yelp_db.USERVIEW1 to 'user2'@'localhost';
grant select on yelp_db. BUSINESSVIEWBUSINESS1 to 'user2'@'localhost';
grant insert,update ,delete on yelp_db.review to 'user2'@'localhost';
```

Figure 4.7

4.3 Data analyst users

This kind of user can read all the relations of the database with all the attributes. In addition, this user can create views.
In the next figure is showed all the permissions

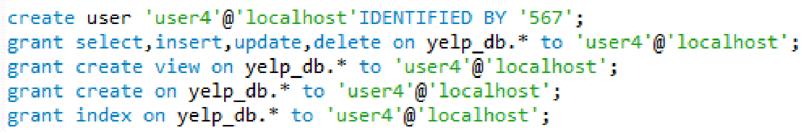


```
create user 'user3'@'localhost'IDENTIFIED BY '345';
grant select on yelp_db.* to 'user3'@'localhost';
grant create view on yelp_db.* to 'user3'@'localhost';
```

Figure 4.8

4.4 Developer users

This kind of user can insert, update, or delete entities in all the relations. In addition, this user can create tables, indexes, and he can see all the information in all the relations. The permissions given to this user are showed in the next figure

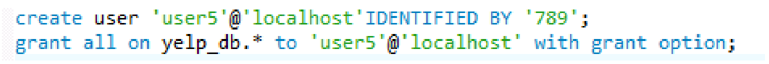


```
create user 'user4'@'localhost'IDENTIFIED BY '567';
grant select,insert,update,delete on yelp_db.* to 'user4'@'localhost';
grant create view on yelp_db.* to 'user4'@'localhost';
grant create on yelp_db.* to 'user4'@'localhost';
grant index on yelp_db.* to 'user4'@'localhost';
```

Figure 4.9

4.5 Root users

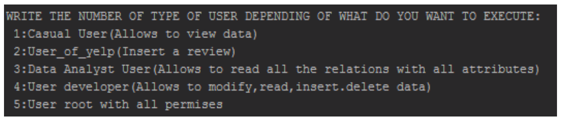
This kind of user has all the permissions of the database. It includes to create users and grants permission to other user.



```
create user 'user5'@'localhost'IDENTIFIED BY '789';
grant all on yelp_db.* to 'user5'@'localhost' with grant option;
```

Figure 4.10

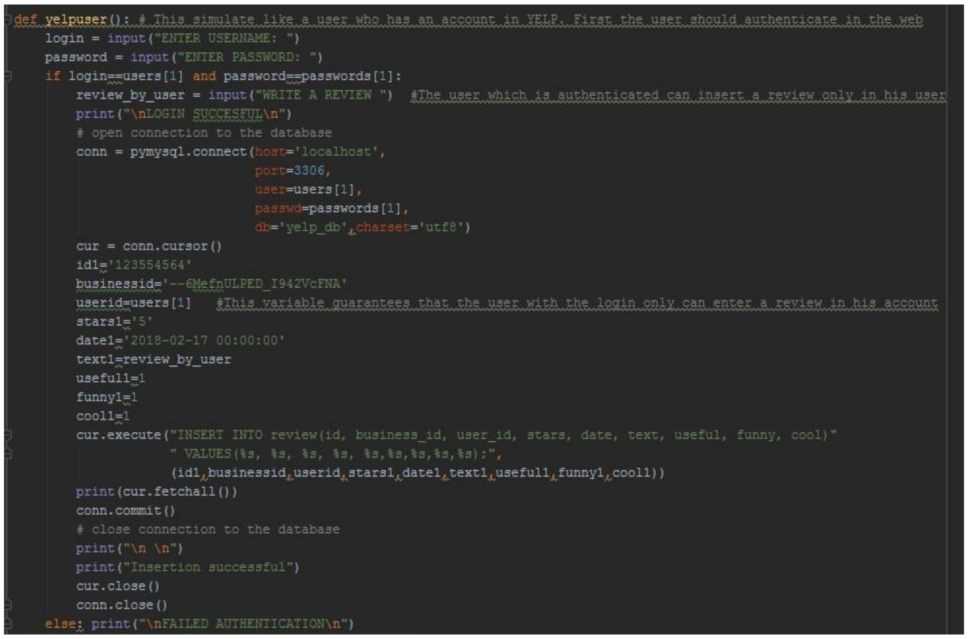
In order to complement this part of the project, a simple application has been developed in Phyton which verified the login and connects to the mysql database depending of the kind of user that we are.



```
WRITE THE NUMBER OF TYPE OF USER DEPENDING OF WHAT DO YOU WANT TO EXECUTE:
 1:Casual User(Allows to view data)
 2:User_of_yelp(Insert a review)
 3:Data Analyst User(Allows to read all the relations with all attributes)
 4:User developer(Allows to modify,read,insert.delete data)
 5:User root with all permises
```

Figure 4.11

This application for example validate that a registered user in YELP can only insert a review with his own user.



```
def yelpuser(): # This simulate like a user who has an account in YELP. First the user should authenticate in the web
    login = input("ENTER USERNAME: ")
    password = input("ENTER PASSWORD: ")
    if login==users[1] and password==passwords[1]:
        review_by_user = input("WRITE A REVIEW ")  #The user which is authenticated can insert a review only in his user
        print("\nLOGIN SUCCESFUL\n")
        # open connection to the database
        conn = pymysql.connect(host='localhost',
                               port=3306,
                               user=users[1],
                               passwd=passwords[1],
                               db='yelp_db',charset='utf8')
        cur = conn.cursor()
        id1='123554564'
        businessid='--6MefnULPED_I942VcFNA'
        userid=users[1]   #This variable guarantees that the user with the login only can enter a review in his account
        stars1='5'
        date1='2018-02-17 00:00:00'
        text1=review_by_user
        useful1=1
        funny1=1
        cool1=1
        cur.execute("INSERT INTO review(id, business_id, user_id, stars, date, text, useful, funny, cool)"
                    " VALUES(%s, %s, %s, %s, %s,%s,%s,%s,%s);",
                    (id1,businessid,userid,stars1,date1,text1,useful1,funny1,cool1))
        print(cur.fetchall())
        conn.commit()
        # close connection to the database
        print("\n \n")
        print("Insertion successful")
        cur.close()
        conn.close()
    else: print("\nFAILED AUTHENTICATION\n")
```

Figure 4.12

5. Conclusion

In this project, we verified self-consistency and sanity of the data which is stored in the database YELP, created a E-R diagram for the database. By doing data cleaning and indexing operations, we removed garbage data, fastened query, and prepared for the later part. And in the data analysis part, we analysed the behaviour of a particular business during the time period from the time before 2012 to 2017, which resulted in remaining steady in high ratings, determined the relation between review length and useful and found negative corralative property among them. Finally, we designed an access control rules that grant different authorities for 5 types of users.