DATABASE SYSTEMS

ECE-656

Project Report

UNIVERSITY OF WATERLOO

Tanaya Tanaya

ID- 20740187

Navrisham Bhullar

ID- 20739654

Mansukh Singh Seerha

ID- 20690558

INTRODUCTION

For this project we are using ten different tables namely business, review, user, checkin, tip, hours, user, attribute, friend and elite_years.

In order to move ahead with the data analysis of the Yelp dataset, it is important for the data to be cleaned and indexed properly. The data cleaning and indexing is done on MYSQL. While the data analysis has been done using Python.

PART I

DATA CLEANING

This section involves preparing the data in order to get the dataset ready for the data analysis part. This tasks involves more time consuming than the actual data analysis. In reality, data cleaning might involve the removal of the typographical errors, it also depends on the type of data that is used in the dataset that is being cleansed. However, it is also common to mistake data cleansing with data validation, i.e. validation actually deals with the data at the entry level whereas cleansing can be done even after the data has been entered. Even though we have many tools available in the market to carry out this process, but for this project report, we will be doing this manually using the knowledge that we have gained from the ECE-656 course.

Following are the points that were taken into consideration while doing the cleaning part:

- Cross checking
  This depends on the type of data being used and its context. For the current scenario, we are dealing with different restaurants for which the reviews and ratings were stored in the form of the dataset. So, in order to take this data set into consideration for the data analysis, we need to make sure that the review table contains tangible data that can be used to provide analysis.

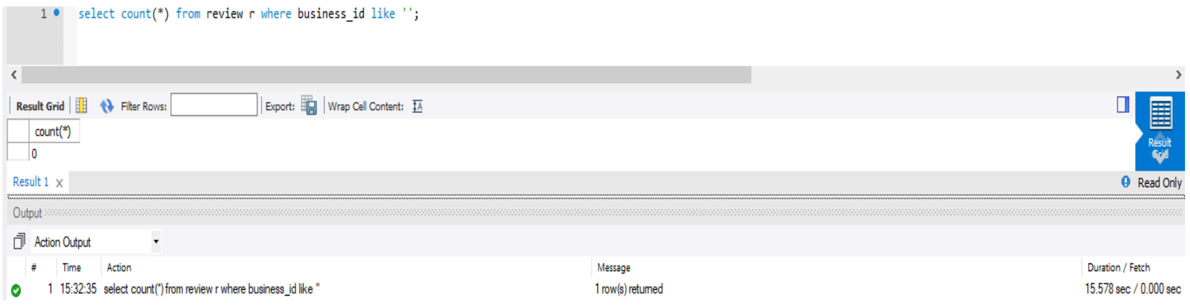
- Checking for NULL values
  This involves check for null values and removing the corresponding tuples. The null values are already cleaned in the dataset.

1.1 Consistency and Sanity Checking

The following sanity checks were done as a part of the data cleansing process, keeping in mind the above mentioned points.

1. No Empty business_id in Review table

For the review table, the business_id should not be an empty string. It is expected that any review with missing business information is invalid, meaning that the review should not be submitted in Yelp. So here the data is already cleaned as count is 0, therefore no review information is required to be removed before indexing and analysis of data.



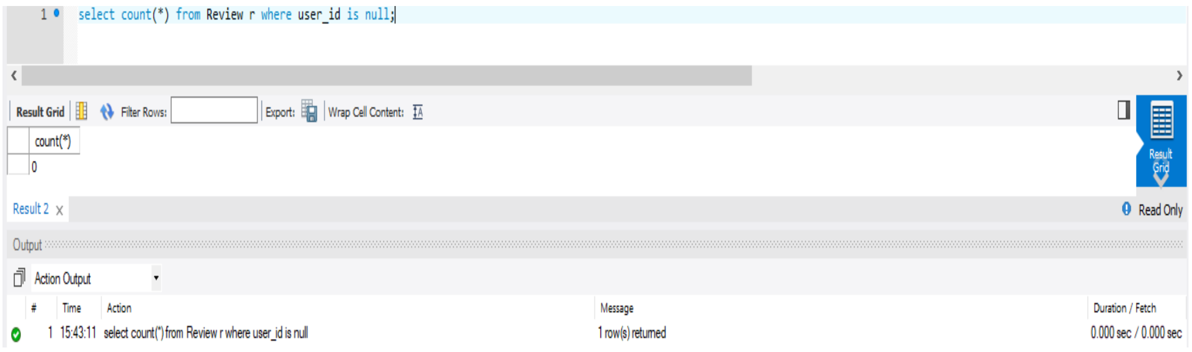
```
select count(*) from review r where business_id like '';
```

```
Result Grid | Filter Rows: | Export: | Wrap Cell Content:
count(*)
0
Result 1                                                         Read Only
Output
Action Output
#  Time      Action                                                  Message             Duration / Fetch
1  15:32:35  select count(*) from review r where business_id like ''  1 row(s) returned   15.578 sec / 0.000 sec
```

Snippet 1: Finding if business_id is an empty string in the review table.

2. No Null user id in Review table

This is similar to the first sanity check, the user_id should not be null. It is also expected that any review with missing user information is invalid, meaning that Yelp must not have those review. As there are no such cases, therefore data is cleaned here.



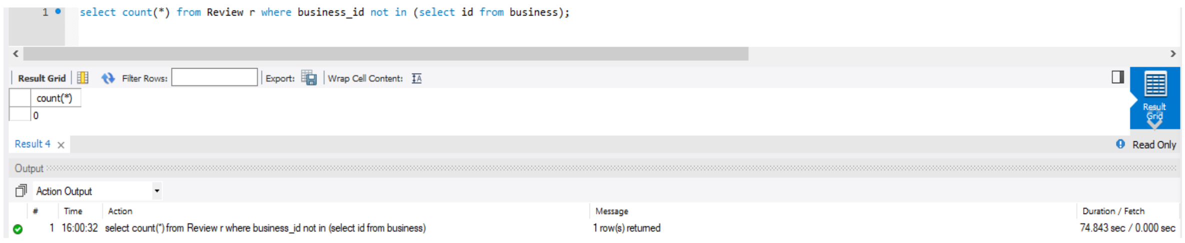
```
select count(*) from Review r where user_id is null;
```

```
Result Grid | Filter Rows: | Export: | Wrap Cell Content:
count(*)
0
Result 2                                                         Read Only
Output
Action Output
#  Time      Action                                              Message             Duration / Fetch
1  15:43:11  select count(*) from Review r where user_id is null  1 row(s) returned   0.000 sec / 0.000 sec
```

Snippet 2: Finding if user_id is null in the review table.

3. No case where business_id in Review table does not exist in Business table

It is observed that, no review information contains business_id which does not belong to the Business table. Hence, data is cleaned.



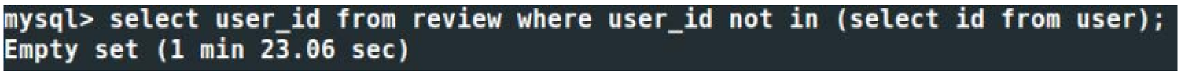
```
select count(*) from Review r where business_id not in (select id from business);
```

```
Result Grid | Filter Rows: | Export: | Wrap Cell Content:
count(*)
0
Result 4                                                         Read Only
Output
Action Output
#  Time      Action                                                                          Message             Duration / Fetch
1  16:00:32  select count(*) from Review r where business_id not in (select id from business)  1 row(s) returned   74.843 sec / 0.000 sec
```

Snippet 3: Finding if there is any business_id in review table which does not exist in business table.

4. User id in Review table does not exist in User table

In this sanity check, we are making sure that the user_id from the review table only contains the corresponding user_id that are found in the user table. We have found no review information which contains user_id that does not exist in the User table.



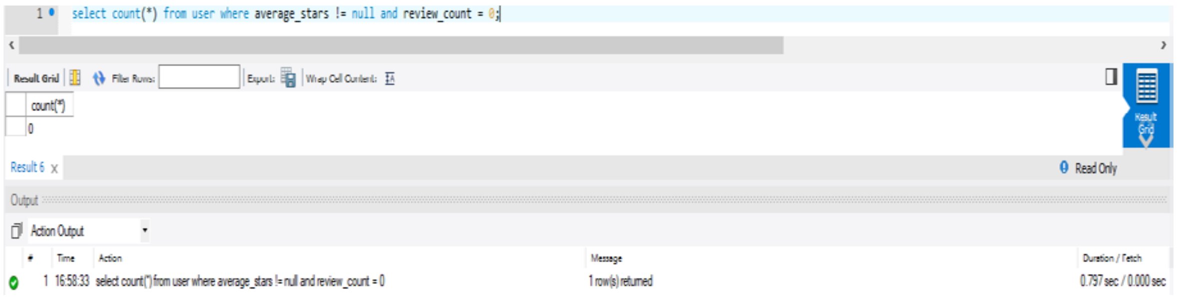
```
mysql> select user_id from review where user_id not in (select id from user);
Empty set (1 min 23.06 sec)
```

Snippet 4: Finding if there is any user_id from review table which does not exist in user table.

Therefore, data was found to be already cleaned.

5. Average stars is not null when review count is 0 in User table

In User table, if the total number of review written by a user is 0, then the average stars rated by the user should be null by default. The default value should not be zero since zero could mean the user had the worst experience during visit. It is found that there is no such case where users have review count of 0, but they have value for the average stars (which is supposed to be 0 as well). Therefore, for this case data is cleaned as well.



```
1  select count(*) from user where average_stars != null and review_count = 0;
```

```
count(*)
0
```

Result 6 | Read Only

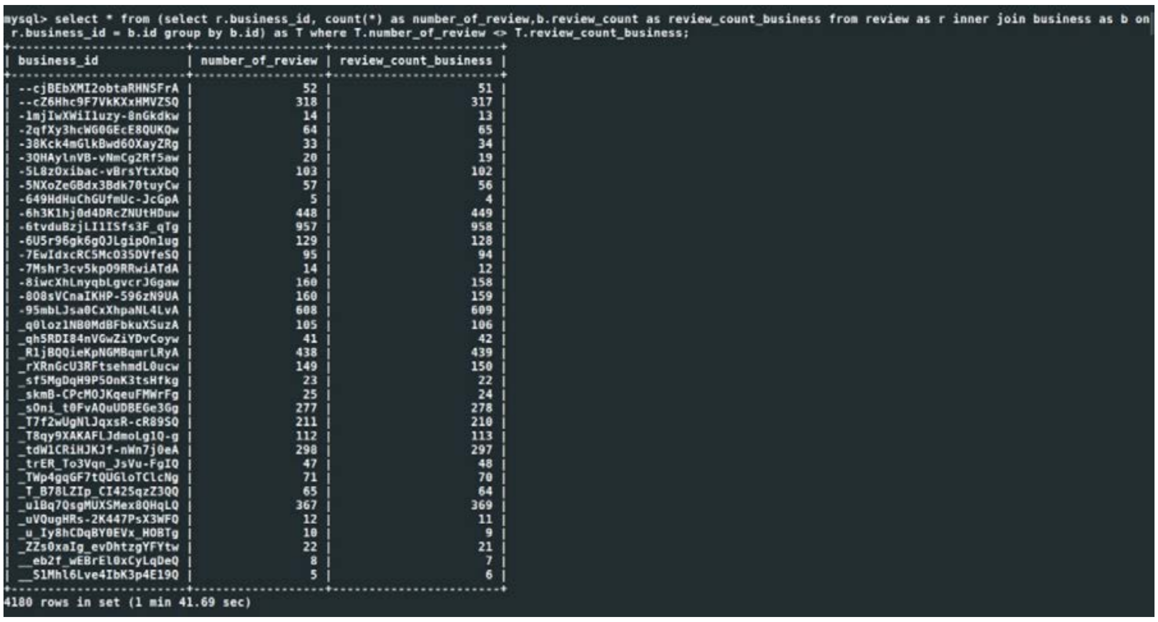
Action Output

```
#   Time      Action                                                                   Message            Duration / Fetch
1   16:58:33  select count(*) from user where average_stars != null and review_count = 0   1 row(s) returned   0.797 sec / 0.000 sec
```

Snippet 5: Finding if average_stars and review_count is undefined in the user table.

6. For all business, the number of review in Review table should be same as review count in Business table.

Users can share their experience after or during visit, and that information is added into the review table. The total number of reviews for a business in Review table should be the same as the review_count value of the business in Business table. It is found that 4180 businesses return not matching value. This can result that stars and review_count value of Business table are incorrect; therefore, the missing review information should be updated.



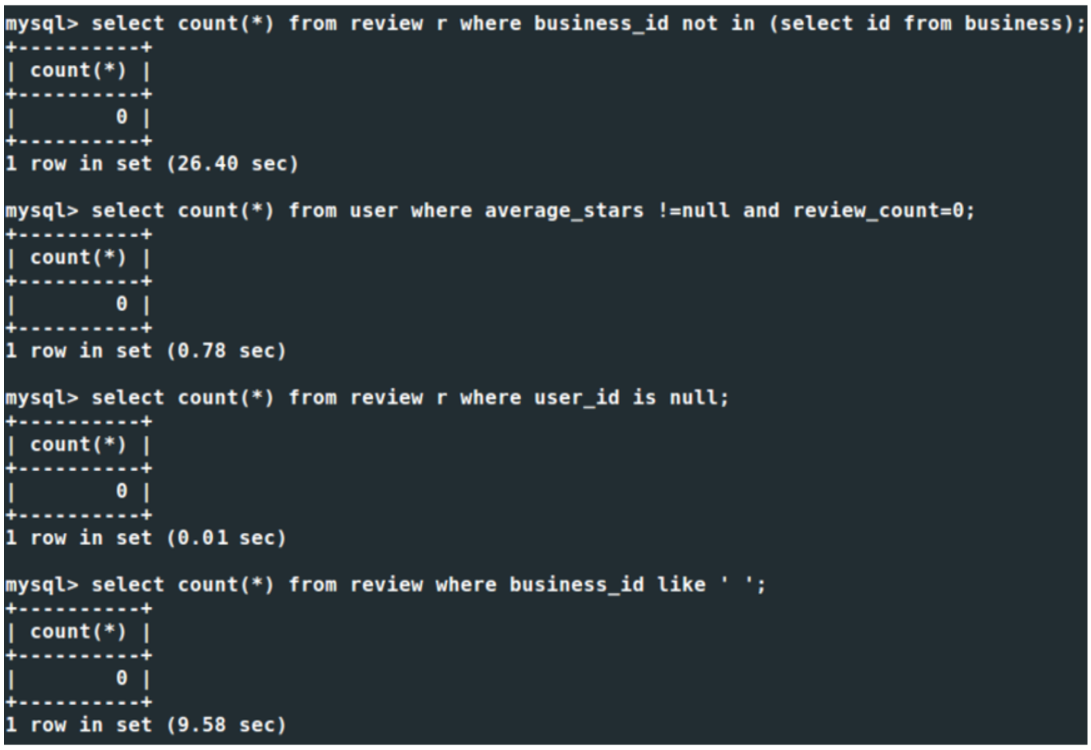
```
mysql> select * from (select r.business_id, count(*) as number_of_review,b.review_count as review_count_business from review as r inner join business as b on
 r.business_id = b.id group by b.id) as T where T.number_of_review <> T.review_count_business;
+------------------------+------------------+-----------------------+
| business_id            | number_of_review | review_count_business |
+------------------------+------------------+-----------------------+
| --cjBEbXMI2obtaRHNSFrA |               52 |                    51 |
| --cZ6Hhc9F7VkKXxHMVZSQ |              318 |                   317 |
| -1mjIwXWiIluzy-8nGkdkw |               14 |                    13 |
| -2qfXy3hcWG0GEcE8QUKQw |               64 |                    65 |
| -38Kck4mGlkBwd6OXayZRg |               33 |                    34 |
| -3QHAylnVB-vNmCg2Rf5aw |               20 |                    19 |
| -5L8zOxibac-vBrsYtxXbQ |              103 |                   102 |
| -5NXoZeGBdx3Bdk70tuyCw |               57 |                    56 |
| -649HdHuChGUfmUc-JcGpA |                5 |                     4 |
| -6h3K1hj0d4DRcZNUtHDuw |              448 |                   449 |
| -6tvduBzjLI1ISfs3F_qTg |              957 |                   958 |
| -6U5r96gk6gQJLgipOnlug |              129 |                   128 |
| -7EwIdxcRC5Mc035DVfeSQ |               95 |                    94 |
| -7Mshr3cv5kpO9RRwiATdA |               14 |                    12 |
| -8iwcXhLnyqbLgvcrJGgaw |              160 |                   158 |
| -8O8sVCnaIKHP-596zN9UA |              160 |                   159 |
| -95mbLJsa0CxXhpaNL4LvA |              608 |                   609 |
| _q0loz1NB0MdBFbkuXSuzA |              105 |                   106 |
| _qh5RDI84nVGwZiYDvCoyw |               41 |                    42 |
| _R1jBQQieKpNGMBqmrLRyA |              438 |                   439 |
| _rXRnGcU3RFtsehmdL0ucw |              149 |                   150 |
| _sf5MgDqH9P5OnK3tsHfkg |               23 |                    22 |
| _skmB-CPcMOJKqeuFMWrFg |               25 |                    24 |
| _sOni_t0FvAQuUDBEGe3Gg |              277 |                   278 |
| _T7f2wUgNlJqxsR-cR89SQ |              211 |                   210 |
| _T8qy9XAKAFLJdmoLg1Q-g |              112 |                   113 |
| _tdW1CRiHJKJf-nWn7j0eA |              298 |                   297 |
| _trER_To3Vqn_JsVu-FgIQ |               47 |                    48 |
| _TWp4gqGF7tQUGloTClcNg |               71 |                    70 |
| _T_B78LZIp_CI425qzZ3QQ |               65 |                    64 |
| _u1Bq7QsgMUXSMex8QHqLQ |              367 |                   369 |
| _uVQugHRs-2K447PsX3WFQ |               12 |                    11 |
| _u_Iy8hCDqBY0EVx_HOBTg |               10 |                     9 |
| _ZZs0xaIg_evDhtzgYFYtw |               22 |                    21 |
| __eb2f_wEBrEl0xCyLqDeQ |                8 |                     7 |
| __S1Mhl6Lve4IbK3p4E19Q |                5 |                     6 |
+------------------------+------------------+-----------------------+
4180 rows in set (1 min 41.69 sec)
```

Snippet 6: The preview of the output (not all tuples are shown) of the input query.

Similarly, it is difficult to update additional 4180 reviews into each of 4180 business; therefore, we will rely on the information in Review table for these 4180 business.

1.2 Data Indexing

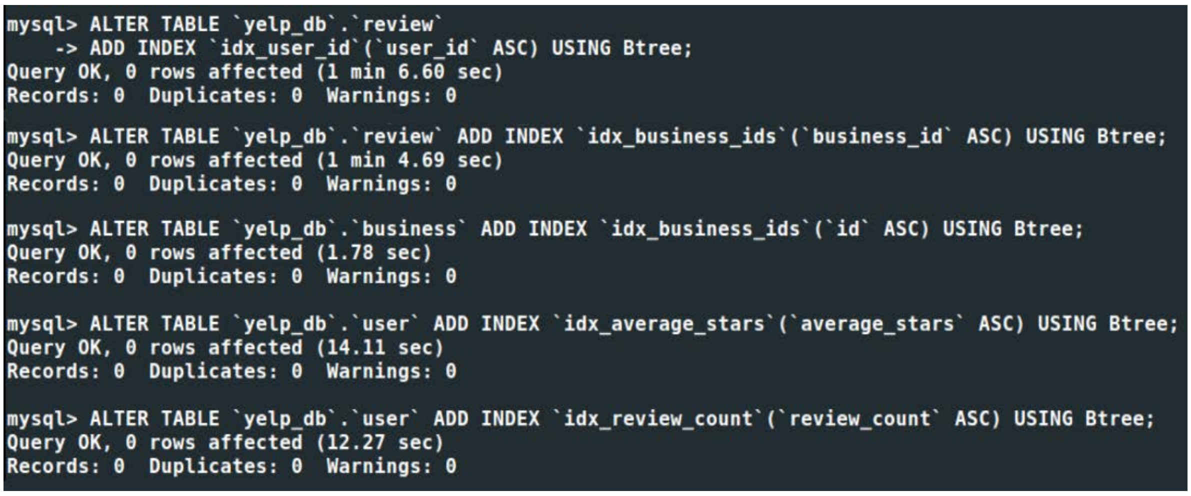
Data indexing is the process of adding index (plural indexes or indices) to the tables, which speeds up in looking-up of records by specific search keys. An index can be created for one or more columns in a table according to the need. Adding index to the table helps in executing the result faster, quick data retrieval and for sorting the data. However, it also decreases the performance of insert, update and delete statements.



```
mysql> select count(*) from review r where business_id not in (select id from business);
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (26.40 sec)

mysql> select count(*) from user where average_stars !=null and review_count=0;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.78 sec)

mysql> select count(*) from review r where user_id is null;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from review where business_id like ' ';
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (9.58 sec)
```

Snippet 7: The execution time of the queries before indexing.

For the first 4 queries used for sanity check, the execution speed is shown in the previous snippet. The following snippet gives the necessary commands used for indexing.

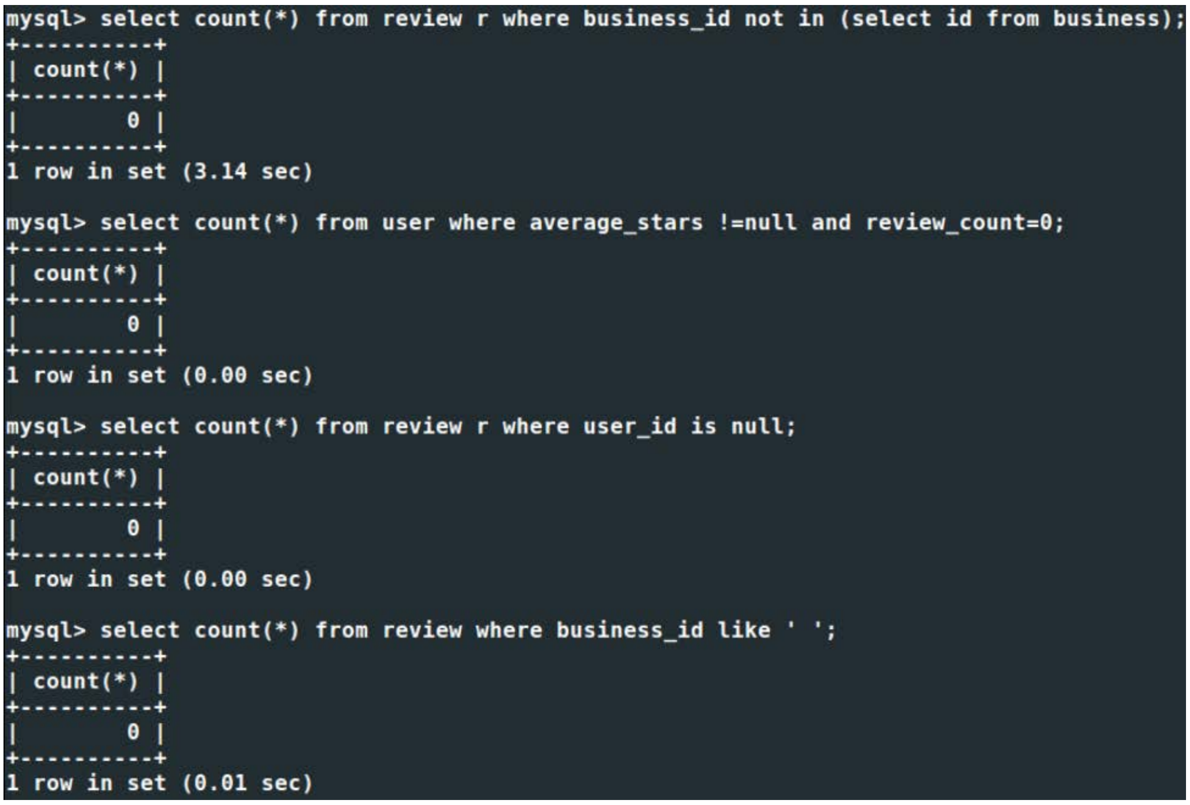


```
mysql> ALTER TABLE `yelp_db`.`review`
    -> ADD INDEX `idx_user_id`(`user_id` ASC) USING Btree;
Query OK, 0 rows affected (1 min 6.60 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE `yelp_db`.`review` ADD INDEX `idx_business_ids`(`business_id` ASC) USING Btree;
Query OK, 0 rows affected (1 min 4.69 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE `yelp_db`.`business` ADD INDEX `idx_business_ids`(`id` ASC) USING Btree;
Query OK, 0 rows affected (1.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE `yelp_db`.`user` ADD INDEX `idx_average_stars`(`average_stars` ASC) USING Btree;
Query OK, 0 rows affected (14.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE `yelp_db`.`user` ADD INDEX `idx_review_count`(`review_count` ASC) USING Btree;
Query OK, 0 rows affected (12.27 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Snippet 8: Indexing commands and their executing time.

It is advantageous to use index, because it improves the speed of operations in a table. For example, the index makes faster to find the rows matching a WHERE clause and more efficient to order the records. Even though it takes more time to execute INSERT and

UPDATE statement, the SELECT statements become much faster on those tables, meaning that adding indexes will be beneficial for the same 4 queries and their executing time (after adding indexes) is shown below:


```
mysql> select count(*) from review r where business_id not in (select id from business);
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (3.14 sec)

mysql> select count(*) from user where average_stars !=null and review_count=0;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from review r where user_id is null;
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from review where business_id like ' ';
+----------+
| count(*) |
+----------+
|        0 |
+----------+
1 row in set (0.01 sec)
```

Snippet 9: The execution time of the queries after indexing.

By re-executing the 4 queries above in tables with indexes, it is found that the execution speed was much faster as observed from the table below.

	Execution speed (before) (in secs)	Execution speed (after) (in secs)	Time taken to add index (in secs)
Query 1	26.46	3.14	1.78
Query 2	0.78	0.00	14.11 + 12.27
Query 3	0.01	0.00	66.60
Query 4	9.58	0.01	64.69

.

DATA ANALYSIS

3.1 ER Diagram of Yelp Dataset

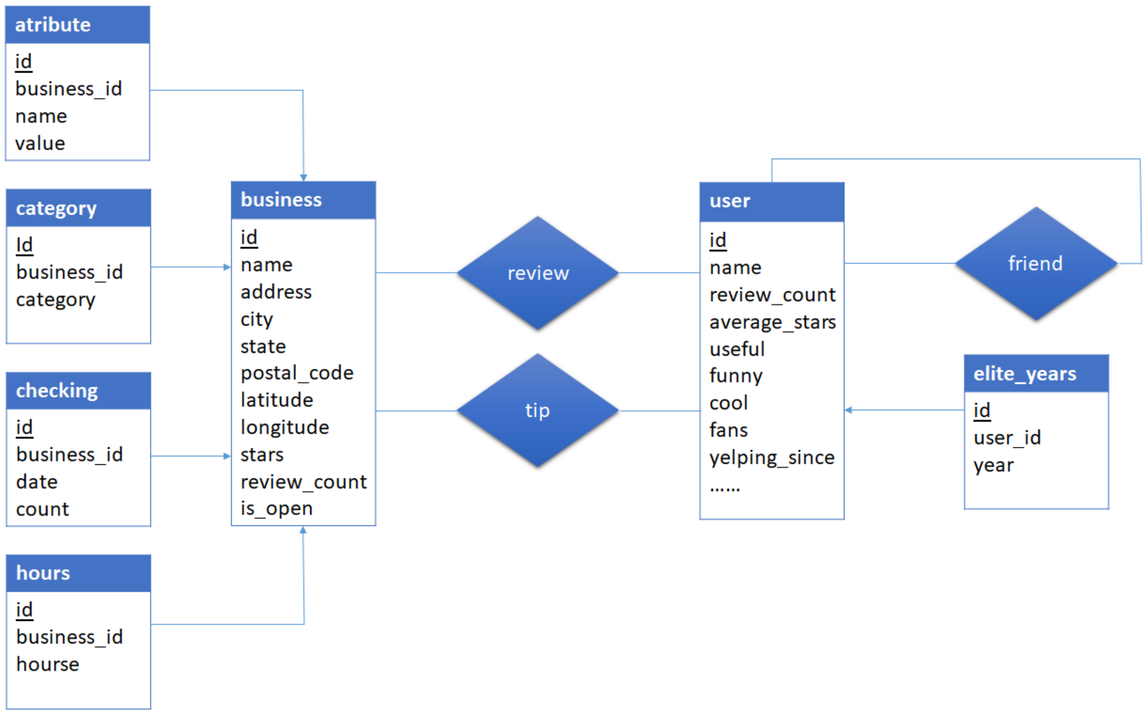
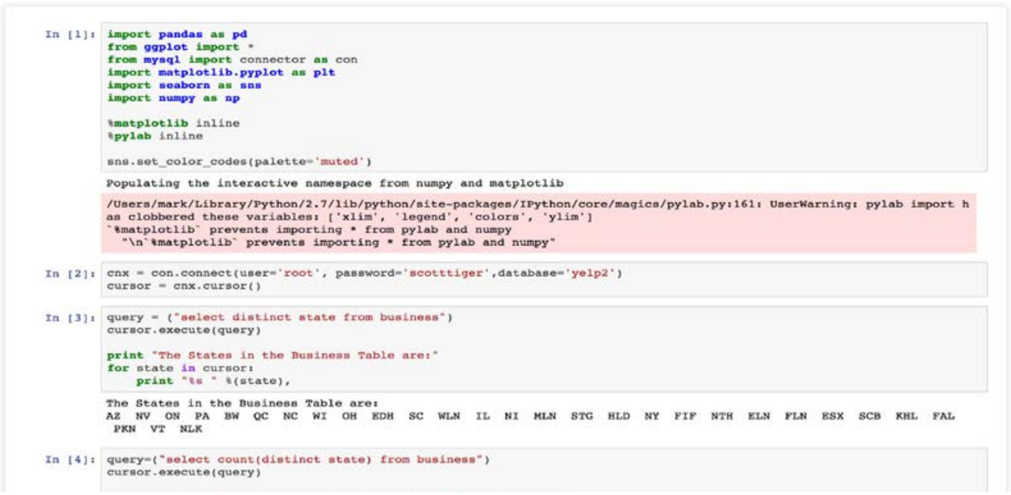


Figure 1: Entity-Relationship diagram of yelp dataset.

3.2 Tools for Data Analysis

We have performed our data analysis using Python and associated libraries that are available for data analysis. The python scripts are interfaced with the mysql database using the Python Driver for MySQL (mysql.connector) which is available on PyPI and documentation for this can be found at the MySQL website here. The database was queried from within the script and the results of the queries were stored to dataframes for analysis.



```
In [1]: import pandas as pd
        from ggplot import *
        from mysql import connector as con
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np

        %matplotlib inline
        %pylab inline

        sns.set_color_codes(palette='muted')

        Populating the interactive namespace from numpy and matplotlib

        /Users/mark/Library/Python/2.7/lib/python/site-packages/IPython/core/magics/pylab.py:161: UserWarning: pylab import h
        as clobbered these variables: ['xlim', 'legend', 'colors', 'ylim']
        `%matplotlib` prevents importing * from pylab and numpy
          "\n`%matplotlib` prevents importing * from pylab and numpy"

In [2]: cnx = con.connect(user='root', password='scotttiger',database='yelp2')
        cursor = cnx.cursor()

In [3]: query = ("select distinct state from business")
        cursor.execute(query)

        print "The States in the Business Table are:"
        for state in cursor:
            print "%s " %(state),

        The States in the Business Table are:
        AZ  NV  ON  PA  BW  QC  NC  WI  OH  EDH  SC  WLN  IL  NI  MLN  STG  HLD  NY  FIF  NTH  ELN  FLN  ESX  SCB  KHL  FAL
         PKN  VT  NLK

In [4]: query=("select count(distinct state) from business")
        cursor.execute(query)
```

Snippet 10: Using Jupyter notebook for analysis.

The code has been written and executed using Jupyter notebooks as seen from the snippet on page 6. Libraries that we have used for our project are listed below:

1. Pandas
2. Numpy
3. MySQL.Connector
4. Matplotlin
5. Seaborn
6. ggplot
7. Scikit-learn
8. Pillow (Python Imaging Library)
9. Wordcloud
10. Palettable

3.3 The Business table

The Business tables has several attributes of interest. These include the names of the States and Cities, the number of businesses in each state and city and the average star rating of the businesses in a state and city.

We begin by querying the database for the distinct states in the business table. We find that there are a total number of 68 states represented. We then turn our attention to the states with the most number of businesses and find that Ontario ranks very high (with 30208 businesses) and thus is a good candidate for analyzing trends.

First, we summarize the average star ratings per state of the ten states with the most number of businesses (it should be noted here that states include both US states and Canadian provinces). We find that, even though Ontario has the third highest number of businesses, it has the lowest ranking with both Arizona and Nevada performing significantly better in terms of star rating. Figure 2 shows the relationship between the states and their average star ratings.

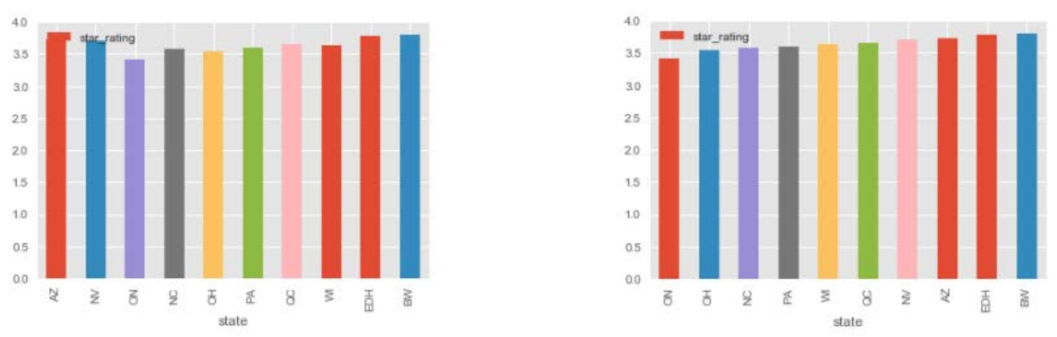


Figure 2: shows more clearly how Ontario businesses compare to other state's businesses on average(left picture), and average stars per state (right picture).

Next, we found the maximum, minimum, average and median review counts for businesses in Ontario. They are as follows:

ANALYSIS ON BUSINESSES IN ONTARIO	
The Maximum Review Count	1494.0000
The Minimum Review Count	3.0000
The Average Review Count	20.9965
The Median Review Count	8.0000

We also looked at the businesses ordered by highest star rating and highest number of reviews first and found three businesses with more than a hundred star ratings as well as a full 5 star rating.

We then sorted this result by highest number of reviews and plotted a bar chart of the highest reviewed businesses in Ontario against their average rating. This can be seen in figure 3. The business 'Pai Northern Thai Kitchen' is a top performer having a 4.5 star rating as well as the highest review count of 1494. 'Pai Northern Thai Kitchen' will reveal some interesting trends later in the Review table.

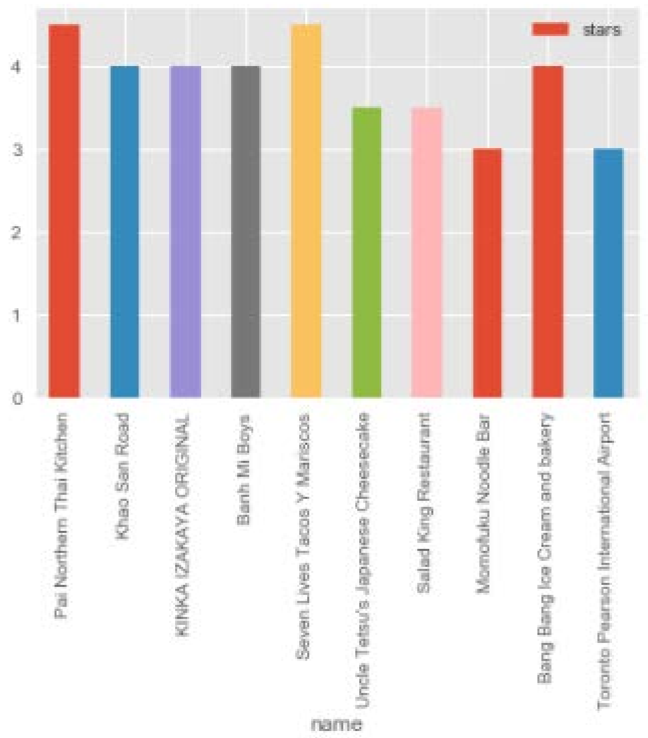
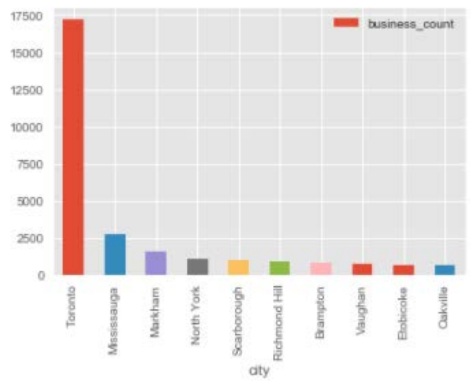


Figure 3: Most Rated Businesses.

Lastly, we take a look at the representation of the number of businesses per city in the province of Ontario. We found that Toronto is over-represented in this dataset. This can easily be attributed to the fact that Toronto, being the biggest metropolis, is naturally likely to have the most number of businesses. However the magnitude of how far ahead Toronto is in terms of number of businesses represented in the dataset is truly grasped by looking at the plot in figure 4 (left). Figure 4 (right) below shows the average star ratings of the businesses in each city while the legend shows the exact number of businesses.



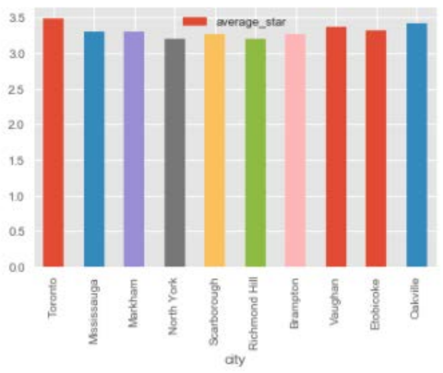


Figure 4: Cities with The Most Businesses in Ontario (left picture), and their average star rating (right picture).

The scatter plot (below) shows the scatter of number of business vs. average stars, with Toronto clearly standing out from rest of the cities.

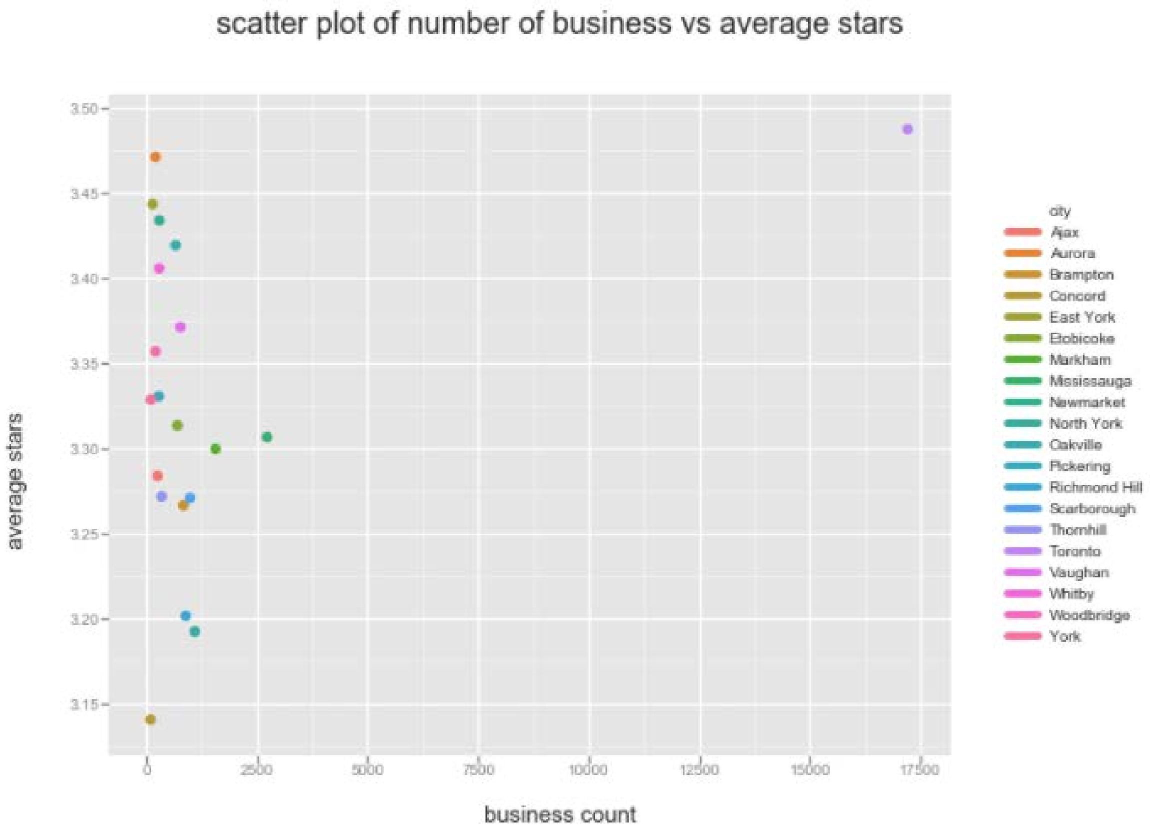


Figure 5: Number of Businesses Vs Star Rating.

The last graph, figure 6 (below) shows a scatter of the review counts against the star ratings for businesses in Toronto.

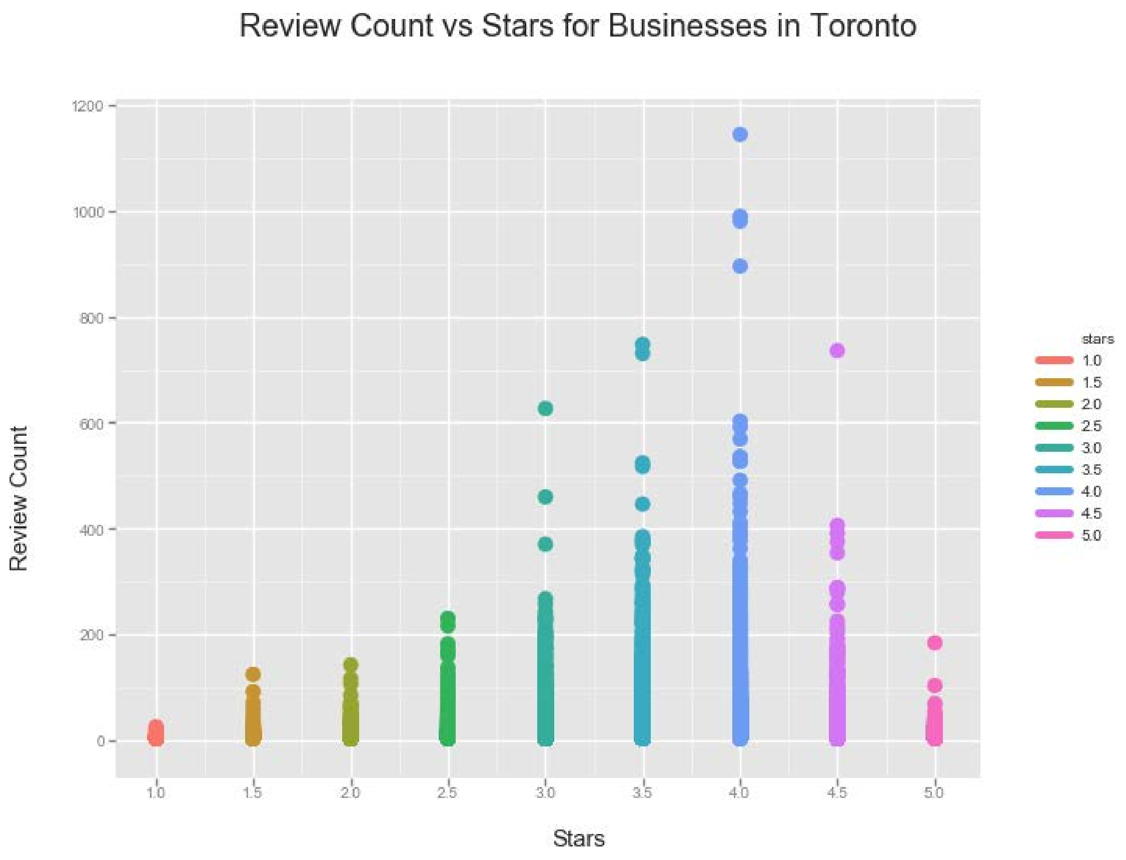


Figure 6: Number of Reviews Vs Star Rating.

3.4 The Review table

We begin our analysis in the Review table by creating a view for the business table that shows us all the businesses in Ontario. We then use this to query the top 50 businesses's from the city of Toronto in terms of highest number of reviews. We display the results and then extract the top 10 for further querying.

We iteratively query the database for the name, average(stars) and date (in terms of the year only) for each business_id that we have extracted all grouped by business_id, name and year. We now have a dataframe containing the average ratings for each of the top 10 businesses group year-wise and ready to be plotted. A graph of ratings versus time (in years) is plotted for the data and can be seen in figure 7.

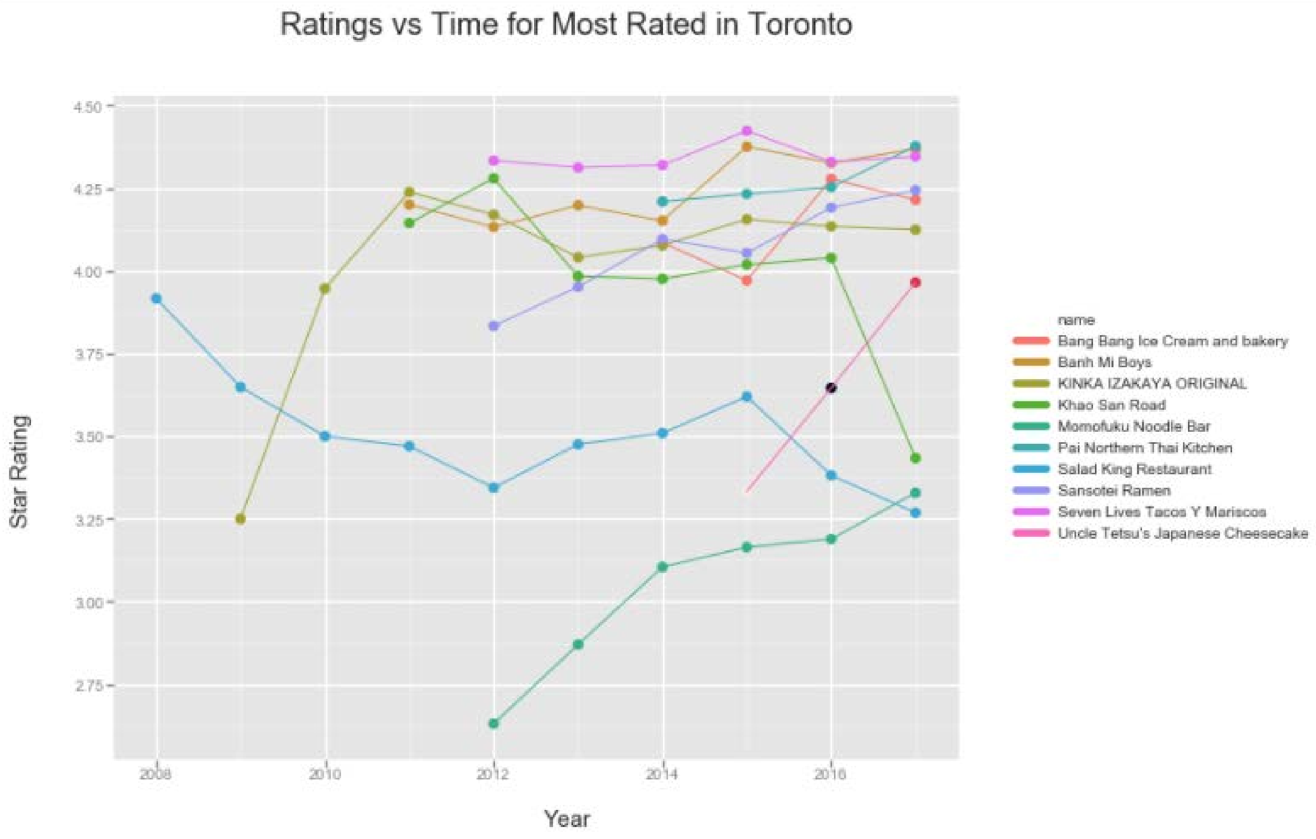


Figure 7: Rating Vs Time for Most Rated in Toronto.

From this plot two trends immediately stand out. First, there is ‘Pai Northern Thai Kitchen’ which we encountered in the business table. This business shows an impressive upward trend over the year 2016-18. Then there is ‘Salad King Restaurant’ which has been declining slowly and steadily over the course to 2015-18. In an attempt to see if these trends are reflected in the reviews of the businesses, we print word clouds for both.

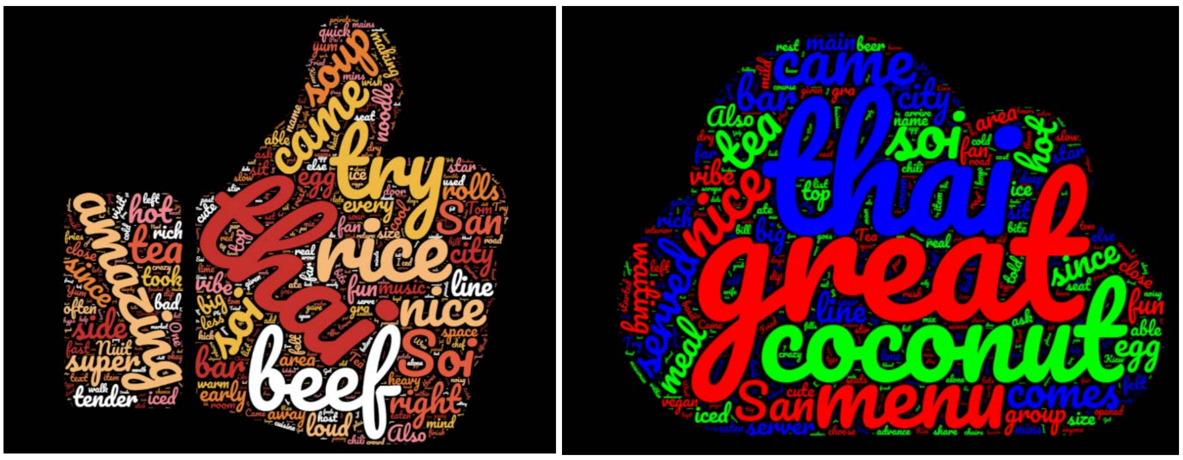


Figure 8: Word clouds for "Pai Northern Thai Kitchen" before 2016 (left) and after (right).

For 'Pai Northern Thai Kitchen', we have two word clouds: one for review before the steep upward trend of 2016- 17 and one for after the rating improved. These can be seen in figures in the previous page (figure 8). For 'Salad King Restaurant', we have two word clouds: one for the years just after the business began (around 2010) and one for after the steady decline in star rating. These can be seen in figures 9 and 10.

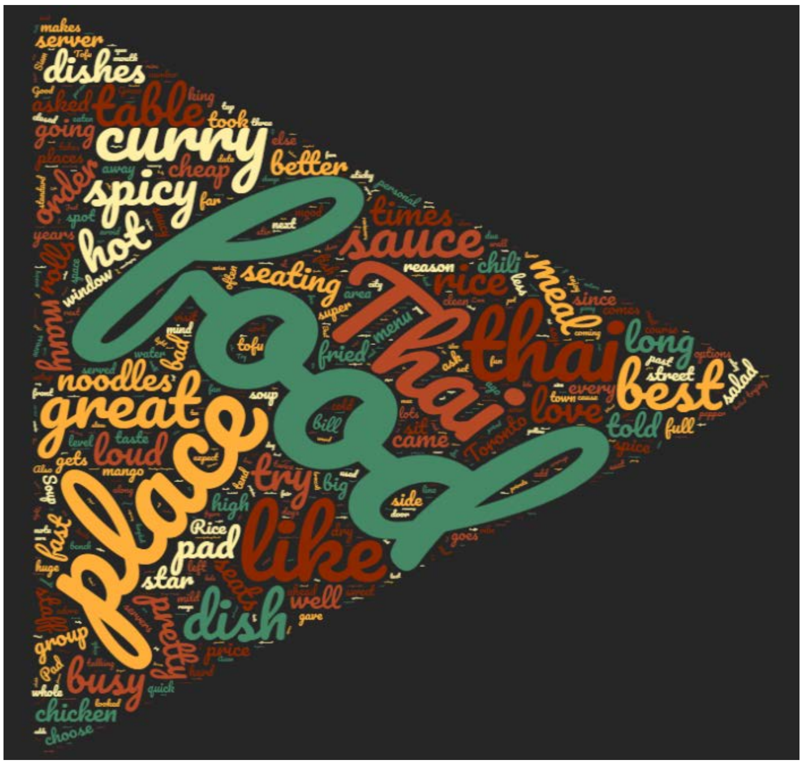


Figure 9: Word Cloud for "Salad King" in 2008-2010.

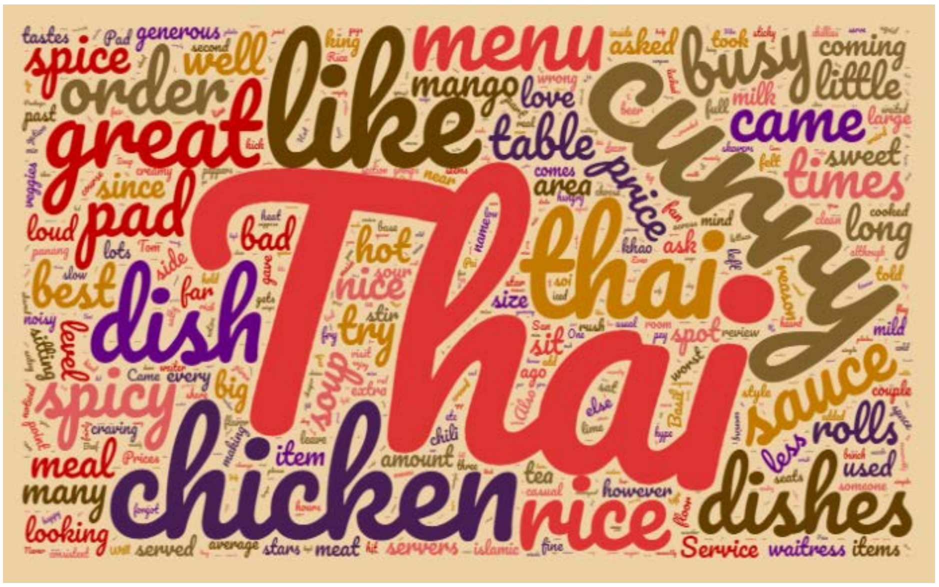


Figure 10: World Cloud for "Salad King" in 2017-2018.

It can be seen that for “Salad King” Restaurant, there are no significantly large negative words present in the word cloud for the period after 2017 which can be observed with the ratings of the business drop steadily. This can be attributed to the lack of proper review text representation for that period or simply that the reviewers do not feel the need to write strongly worded negative reviews. One word does stand out and that is ‘times’. This leads the analyst to investigate further if there might be an association of the word times with quantifiers such as ‘a lot’ or ‘long’ indicating that customers are dissatisfied with the service being too slow.

3.5 The User table and a Simple Recommendation System

We begin our analysis of the user table by looking for the top 10 users ranked by number of reviews (shown in figure 11). However, it is found that the top 10 users of the User table ranked by review count do not match the top 10 users ranked by review count but queried from the Review table. For our analysis we have used the top user from the Review table to allow for further work to be done using the user’s text data.

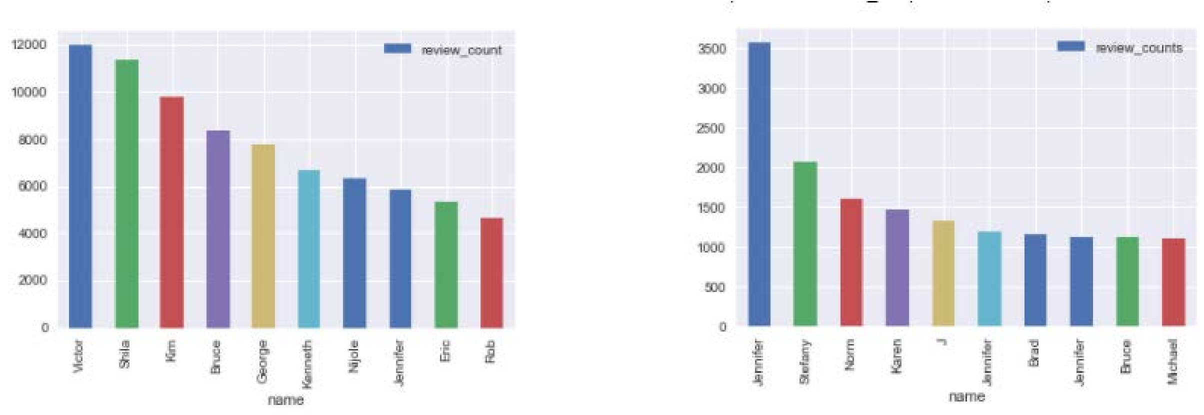


Figure 11: Top user from the user table (left picture) and review table (right picture).

The top user by review count from the review table is user_id: CxDOIDnH8gp9KXzpBHJYXw and has following attributes:

- name: Jennifer
- review_count: 3569

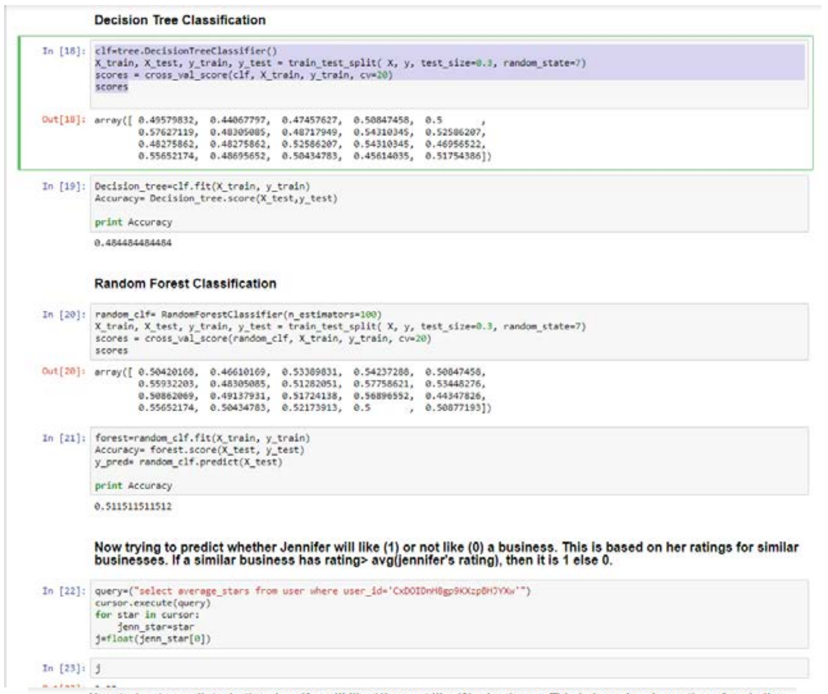
We have extracted the list of business_id’s for which Jennifer has written reviews. Then, we have queried the database for the categories of each of those businesses. Each review for a business was treated as a sample and the categories of those businesses were treated as features. The ratings Jennifer gave to each business was treated as class labels. These were then used to fit a Decision Tree Classifier and a Random Forest Classifier.

The 3569 samples were randomly divided into training and testing sets of sizes 70% and 30% which were randomly shuffled. The Decision Tree Classifier produced an accuracy of 48.4% and the Random Forest Classifier accurately predicted 51.1% of the testing set.

These results seem dismal, but our aim is not to accurately predict what score Jennifer or a user might give a particular business, rather it is to determine whether Jennifer will rate the business ‘well’ or not. That is to say, whether the user will ‘like’ or ‘not like’ the business. To this end, we converted the multi-class classification problem into a binary class problem.

Determining whether a user will ‘like’ or ‘not like’ is a very subjective problem. We have modelled it as a binary-class classification problem. If Jennifer has rated a place greater than her average rating (or median rating) she ‘likes’ the place and if not then she does ‘not like’ the place. Thus any rating equal to or above her average (or median) rating is a 1 and any rating below is a 0.

Using these class labels, we were able to achieve an accuracy of 66% with the Decision Tree Classifier. Though this accuracy may seem low, it basically represents that the Decision Tree was correctly able to determine 2 out of 3 places that Jennifer ‘likes’. This means that for every 3 places presented to Jennifer using this Decision Tree it is probable that she will ‘like’ 2 of them.



Decision Tree Classification

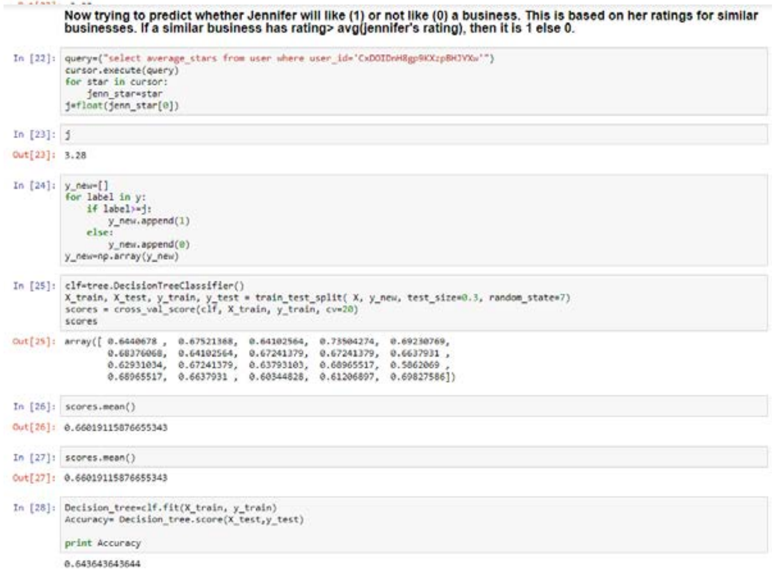
```
In [18]: clf=tree.DecisionTreeClassifier()
         X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=7)
         scores = cross_val_score(clf, X_train, y_train, cv=20)
         scores

Out[18]: array([ 0.49579832,  0.44067797,  0.47457627,  0.50847458,  0.5       ,
                 0.57627119,  0.48305085,  0.48717949,  0.54310345,  0.52586207,
                 0.48275862,  0.48275862,  0.52586207,  0.54310345,  0.46956522,
                 0.55652174,  0.48695652,  0.50434783,  0.45614035,  0.51754386])

In [19]: Decision_tree=clf.fit(X_train, y_train)
         Accuracy= Decision_tree.score(X_test,y_test)

         print Accuracy

         0.484484484484
```

Random Forest Classification

```
In [20]: random_clf= RandomForestClassifier(n_estimators=100)
         X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=7)
         scores = cross_val_score(random_clf, X_train, y_train, cv=20)
         scores

Out[20]: array([ 0.50420168,  0.46610169,  0.53389831,  0.54237288,  0.50847458,
                 0.55932203,  0.48305085,  0.51282051,  0.57758621,  0.53448276,
                 0.50862069,  0.49137931,  0.51724138,  0.56896552,  0.44347826,
                 0.55652174,  0.50434783,  0.52173913,  0.5       ,  0.50877193])

In [21]: forest=random_clf.fit(X_train, y_train)
         Accuracy= forest.score(X_test, y_test)
         y_pred= random_clf.predict(X_test)

         print Accuracy

         0.511511511512
```

Now trying to predict whether Jennifer will like (1) or not like (0) a business. This is based on her ratings for similar businesses. If a similar business has rating> avg(jennifer's rating), then it is 1 else 0.

```
In [22]: query=("select average_stars from user where user_id='CxDOIDnH8gp9KXzpBHJYXw'")
         cursor.execute(query)
         for star in cursor:
             jenn_star=star
         j=float(jenn_star[0])

In [23]: j
```



Now trying to predict whether Jennifer will like (1) or not like (0) a business. This is based on her ratings for similar businesses. If a similar business has rating> avg(jennifer's rating), then it is 1 else 0.

```
In [22]: query=("select average_stars from user where user_id='CxDOIDnH8gp9KXzpBHJYXw'")
         cursor.execute(query)
         for star in cursor:
             jenn_star=star
         j=float(jenn_star[0])

In [23]: j
```

```
Out[23]: 3.28

In [24]: y_new=[]
         for label in y:
             if label>=j:
                 y_new.append(1)
             else:
                 y_new.append(0)
         y_new=np.array(y_new)

In [25]: clf=tree.DecisionTreeClassifier()
         X_train, X_test, y_train, y_test = train_test_split( X, y_new, test_size=0.3, random_state=7)
         scores = cross_val_score(clf, X_train, y_train, cv=20)
         scores

Out[25]: array([ 0.6440678 ,  0.67521368,  0.64102564,  0.73504274,  0.69230769,
                 0.68376068,  0.64102564,  0.67241379,  0.67241379,  0.6637931 ,
                 0.62931034,  0.67241379,  0.63793103,  0.68965517,  0.5862069 ,
                 0.68965517,  0.6637931 ,  0.60344828,  0.61206897,  0.69827586])

In [26]: scores.mean()

Out[26]: 0.66019115876655343

In [27]: scores.mean()

Out[27]: 0.66019115876655343

In [28]: Decision_tree=clf.fit(X_train, y_train)
         Accuracy= Decision_tree.score(X_test,y_test)

         print Accuracy

         0.643643643644
```

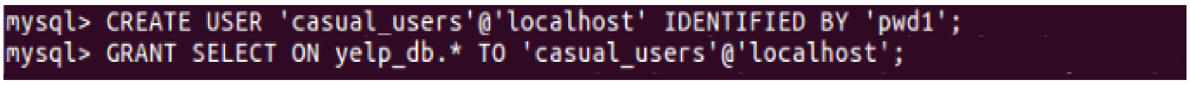
Snippet 11: Using decision classifier package on Jupyter notebook.

This approach is an extremely simple approach to a Recommender system. It has not considered the attributes table or the hours table, both of which are likely to impact the choices of users. The Decision Tree must also be fine-tuned to develop a more robust Recommender system.

PART II

For part 2 of the project, we were required to create an application which allows several types of users to access the Yelp Database. So, to allow such type of functionality, we would use the following queries, which would grant different permissions to each type of users.

1. Casual Users

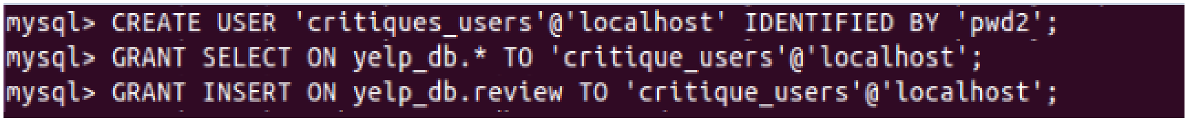


```
mysql> CREATE USER 'casual_users'@'localhost' IDENTIFIED BY 'pwd1';
mysql> GRANT SELECT ON yelp_db.* TO 'casual_users'@'localhost';
```

Snippet 12 Grant Permissions for Casual Users.

The above query ensures that a casual user could only browse but not submit any reviews.

2. Critiques

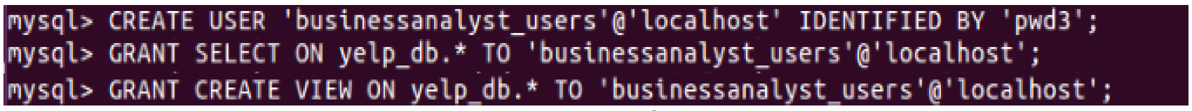


```
mysql> CREATE USER 'critiques_users'@'localhost' IDENTIFIED BY 'pwd2';
mysql> GRANT SELECT ON yelp_db.* TO 'critique_users'@'localhost';
mysql> GRANT INSERT ON yelp_db.review TO 'critique_users'@'localhost';
```

Snippet 13 Grant Permissions for Critiques.

The above query ensures that critiques would be read and write reviews.

3. Business Analysts

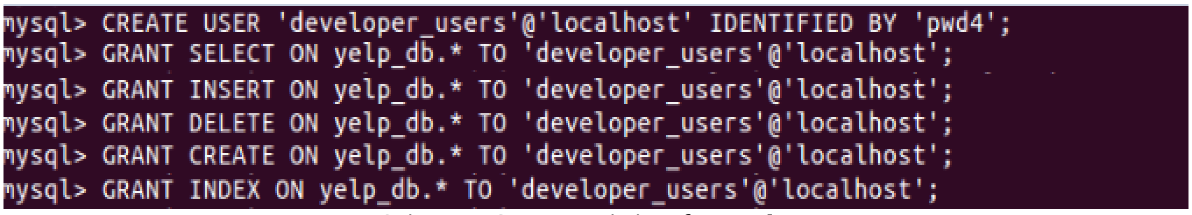


```
mysql> CREATE USER 'businessanalyst_users'@'localhost' IDENTIFIED BY 'pwd3';
mysql> GRANT SELECT ON yelp_db.* TO 'businessanalyst_users'@'localhost';
mysql> GRANT CREATE VIEW ON yelp_db.* TO 'businessanalyst_users'@'localhost';
```

Snippet 14 Grant Permissions for Business Analysts.

The above query ensures that a business analyst could select and create view operations.

4. Developers

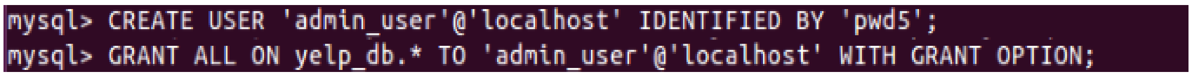


```
mysql> CREATE USER 'developer_users'@'localhost' IDENTIFIED BY 'pwd4';
mysql> GRANT SELECT ON yelp_db.* TO 'developer_users'@'localhost';
mysql> GRANT INSERT ON yelp_db.* TO 'developer_users'@'localhost';
mysql> GRANT DELETE ON yelp_db.* TO 'developer_users'@'localhost';
mysql> GRANT CREATE ON yelp_db.* TO 'developer_users'@'localhost';
mysql> GRANT INDEX ON yelp_db.* TO 'developer_users'@'localhost';
```

Snippet 15 Grant Permissions for Developers.

The above query ensures that developers could insert, update and delete data in the database.

5. Database Administrator



```
mysql> CREATE USER 'admin_user'@'localhost' IDENTIFIED BY 'pwd5';
mysql> GRANT ALL ON yelp_db.* TO 'admin_user'@'localhost' WITH GRANT OPTION;
```

Snippet 16 Grant Permissions for Database Admin.

The above query ensures that the admin has all the privileges for the database.