

Rochester Institute of Technology

Introduction to Big Data - CSCI 620

Phase 2 : Team Unagi - Group No. 7

Ishan Guliani
Rochester Institute of Technology
Rochester, New York
ig5859@rit.edu

Srushti Vijay Kotak
Rochester Institute of Technology
Rochester, New York
sk7735@rit.edu

Nikunj Rajesh Kotecha
Rochester Institute of Technology
Rochester, New York
nrk1787@rit.edu

Soniya Rode
Rochester Institute of Technology
Rochester, New York
sr5055@rit.edu

Aishwarya Sudhir Sontakke
Rochester Institute of Technology
Rochester, New York
as4897@rit.edu

1. PROJECT OUTLINE

. In this project, we explore the dom' of big data. In this component, we look at the data management aspect. The dataset used in this component is *Yelp Dataset* provided by *Yelp*. Here, we look at the description of the dataset and understand the data. Using the Relational DBMS, we design the database structure, create and load in a MySQL database. We define the integrity constr's and implement indexing by detecting functional dependencies within each table and check whether normalization is necessary. Finally, a query system will be developed that will allow users to access the data conveniently. At the end, the system will have a user-friendly interface that will allow multiple users to concurrently access data and execute queries for not only single table but also multiple and nested tables.

2. DATASET

. The dataset used in this project is compiled by *Yelp* and is available for public use at <https://www.yelp.com/dataset>. It cont' a subset of their business, reviews and user data that is made available public for educational purposes. The data is available in *JSON* format and have six different files - business, users, reviews, tips, checkin and photos. The *business.json* cont' important data about location, attributes and list of business categories. The *review.json* cont' the full review text data that includes the *user_id* and *business_id* to indicate which user wrote which review about which business. The *user.json* includes all the user's friend information and the mapping of all the metadata associated with the user. The *checkin.json* cont' the number of check ins made to the particular business. The *tips.json* cont' text which are shorter than reviews and tend to convey quick suggestions to businesses. The *photo.json* cont' photos with caption and classification information. Considering such significant and detailed amount of information we found here, it makes a fit choice for us to implement data management components.

3. ER DIAGRAM

. The ER diagram shown below illustrates the logical structure of our database.

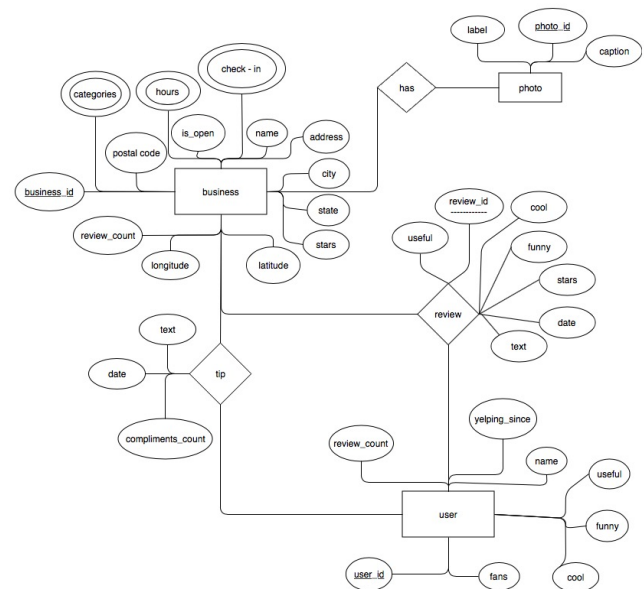


Fig 1. Entity Relationship model illustrates the modified logical structure. We have made modifications so as to minimize distinct entities and relations for simpler understanding. For instance -

- **Normalization:** Instead of making *Checkin* as a separate table, we normalize the relationship by making it a multi valued attribute *count_of_checkins* in the *business* table.

Relational schema:

- *business* (*business_id*, postal code, is_open, name, address, city, stars, latitude, longitude, review_count)
- *user* (*user_id*, yelping_since, name, review_count, useful, funny, cool, fans, average_stars)

- review (business_id, user_id, review_id, useful, cool, funny, stars, date, text) FK1: business_id, FK2: user_id
- tip (business_id, user_id, text, date, compliments_count) FK1: business_id, FK2: user_id
- categories (business_id, categories) FK1: business_id
- hours (business_id, hours_Monday, hours_Tuesday, hours_Wednesday, hours_Thursday, hours_Friday) FK1: business_id
- photo (business_id, caption, label, photo_id) FK1: business_id
- checkin_count (business_id, number_of_checkins) FK1: business_id

4. DESCRIPTION OF MYSQL SCRIPTS

Following are the MySQL scripts for creation, modification and querying.

- create.sql : Contains creation scripts of the database and the tables. The database is named Yelp and the tables created are business, user, Reviews, tip, categories, hours, checkin_count and photo which includes separate tables created for multi-valued attributes.
- Insertion of data in tables is done using python script files.
- modify.sql : It contains SQL statements for functional dependency detection, normalization and specifying integrity constraints. There is also a view dup_business which was created to remove anomalies in the data. Normalization resulted into creation of table business1 addressing the functional dependency postal_code -> state.
- query.sql : Queries involving single-table queries, multiple-table queries, nested queries, and aggregation have been implemented in MySQL.

5. LIST OF DELIVERABLES

- A MySQL script file containing all the SQL command involved in database creation, population and query.
- A query system to be able to perform following tasks:
 - Group Business in a particular area based on their star rating.
 - List Businesses such as Restaurants, shopping, salons, bars, etc open at a given time on a given day.
 - Find top 10 users based on the review count.
 - Find top 15 Businesses with maximum number of check-ins
 - Find top 50 most reviewed businesses for a given category such as Restaurants, Shopping, Fashion, Salons, Bars, etc.
 - Get count of businesses of a particular category such as Squash, Art, Rental, Pub, Racing, etc per city.
 - List categories having highest number of reviews
 - Find all users who have not been satisfied by service of a given business such as IHOP, Starbucks, Sushiya, KFC, etc.
 - Get top 10 users based on popularity.(number of fans)
 - Listing names of restaurants that have menu photos
 - List number of tips grouped according to the date posted
 - A TRIGGER to increment the review count for a business whenever a new review about it is added by a user.

- Web Interface for accessing data and executing queries
- Task sheet which stands as the document that will establish our team's progress towards the project goals.

6. STEPS IN DATA MANAGEMENT

6.1 Data Preprocessing

The data management process involves the acquisition, validation, storage and processing of information relevant to a business or entity. Data goes through a series of steps during preprocessing such as removing duplicate/incorrect data and data redundancy, dropping unwanted attributes, handling null values and multi-valued attributes, finding functional dependencies and normalizing the tables accordingly. We have carried out the following steps for preprocessing:

- Converting the json data to csv files using the json.loads() function.
- Loading the csv files into the pandas dataframe by using pandas.read_csv.
- Parsing the pandas dataframe for loading data into Google Cloud database server.
- To establish a connection with the cloud database, using connect_to_db(host, user, password, database) function, the following parameters are used:

```
host = '35.193.29.72'
user = 'unagi'
password = 'unagi@123'
database = 'Yelp'
db = connect_to_db(host, user, password,
database)
```

- Before inserting the data, some unwanted attribute columns are dropped using dataframe.drop[attributes], and null values are dropped using dataframe.dropna() function.
- Following code is used to load the values into the database.

```
def create_MySQL_table(db, tableName, dataframe,
verbose=False):
    cursor = db.cursor()
    #insert the values into the table
    colStr = "" # generate a string of column
names
separated by ", "
    for colName in dataframe.columns:
        colStr += colName + ","
    colStr = colStr.strip(",")
    for row in dataframe.iterrows():
        sql = "INSERT INTO " + tableName +
        " (" + colStr + ") VALUES ("
        formatStr = ""; valueStr = ""
        sql += ", ".join(str(row[1].values)
        .strip("[]").split(' ')) + ")"
        print(sql)
        cursor.execute(sql)
        db.commit()
```

The first for loop is used to get the column names for the table by generating a string of column names separated by

", ". The second for loop iterates through all the dataframe rows and inserts each row into the table.

- The multi-valued attributes are given as comma separated values in the same column. Since multi-valued attributes are to be stored in a separate table, we do dataframe manipulations such as split by ','.

For example consider business table having business_id:"1SWheh84yJXfytoVILXOAQ" and categories: "Golf, Active Life, Fun, Sports". Now we create table categories, having attributes business_id and categories. For the above example the categories table will have 4 rows for business_id:"1SWheh84yJXfytoVILXOAQ".

```
cat = cat[['business_id', 'categories']]
categories = cat['categories'].str.
split(',').apply(pd.Series, 1).stack()
categories.index = categories.index.
droplevel(-1)
categories.name = 'categories'
del cat['categories']
categories = cat.join(categories)
```

In the above code, cat dataframe stores business_id and categories from the business dataframe. To separate the multi-valued categories, we use the str.split(',') function and stack the split categories from columns to rows by apply(pd.Series, 1).stack() and store the results in categories dataframe by join() function.

6.1.1 Normalization.

- Removing duplicate business ids/incorrect data

```
SELECT count(*), name, address, city,
state, postal_code
FROM business
GROUP BY name, address, postal_code, city,
state
HAVING count(*)>1;
```

Firing the above query, we discovered that there was an anomaly in the dataset with different business_ids for same name and addresses. After investigating the data in this regard, we noticed the entire row was identical except for the review_count. The row with the maximum review_counts is selected as the most recent entry for that name and address. This was achieved using the following queries:

```
CREATE view dup_business as
SELECT MIN(review_count) as review_count,
name, address, city, state, postal_code
FROM business
GROUP BY name, address, postal_code,
city, state
HAVING count(*)>1;
```

Here, we created a view dup_business that contains rows with duplicate business_ids having minimum review_counts so as to delete these rows from the business table.

```
DELETE from business
WHERE business_id
IN (SELECT business_id
```

```
FROM (select * from business ) as b,
dup_business
WHERE b.name=dup_business.name
AND b.address= dup_business.address
AND b.review_count =
dup_business.review_count
AND b.postal_code =
dup_business.postal_code
AND b.city = dup_business.city
AND b.state = dup_business.state);
```

The above query was used for deletion by mapping the business_ids from the view dup_business to the business table.

- Functional Dependency Detection

Following queries were used:

Checking dependency for postal_code->city

```
SELECT count(city), postal_code
FROM business
GROUP BY postal_code;
```

Above query counts the number of cities for every postal_code. The result of this query had multiple number of cities for one postal_code. This led to the conclusion that there is no dependency from postal_code to the city.

Checking dependency for postal_code->state

```
SELECT count(state), postal_code
FROM business
GROUP BY postal_code;
```

Checking dependency for latitude, longitude->state

```
SELECT count(postal_code), latitude,
longitude
FROM business
GROUP BY latitude, longitude;
```

The combination of latitude and longitude determines only a single state and so does the postal_code alone. So, we select minimum of the candidate keys, i.e. the postal_code for the new table.

- Normalizing business table for dependency postal_code -> state by creating a table business1(postal_code, state) using the following query:

```
INSERT INTO business1
SELECT postal_code, state
FROM business
```

7. DEPLOYMENT

7.1 System Design

7.1.1 The Application Logic.

The manuscript that runs behind the home page of the web application uses some specific data structures and algorithms to minimise round trip time when executing queries. On startup, the script uses an instance of **Flask** app to connect to the remote database using the following parameters-

```
app = Flask(__name__)
app.config['MYSQL_USER'] = 'unagi'
```

```
app.config['MYSQL_PASSWORD'] = 'unagi@123'
app.config['MYSQL_DB'] = 'Yelp'
app.config['MYSQL_HOST'] = '35.193.29.72'
mysql = MySQL(app)
```

Two hashmaps take care of mapping each query id to its corresponding data. The hashmap **headerList** maps each *queryId* with its list of headers(that is the header row of each table). The hashmap **queryList** on the other hand maps each *queryId* to its corresponding query string.

The **queryId** in this case is a manually generated serial number of the queries in the order in which they appear on the home page. A sample of each hashmap is shown below -

```
headerNames = {
    1: ['Name', 'Stars', 'Address', 'Postal code'],
    2: ['Name'],
    3: ['Name', 'Review count'],
    4: ['Name', 'Address', 'City', 'Checkin count']
    ...
}

queryList = {
    1: '''SELECT distinct name,stars,address,postal_code
        from business
        where city="Toronto"
        and stars=4
        LIMIT 50;''',
    2: '''SELECT distinct name
        from business,hours,categories
        where business.business_id=hours.business_id
        and business.business_id=categories.business_id
        and hours.Friday="7:0-1:0" and categories like "%
        Restaurants%";'''
    ...
}
```

For queries accepting inputs from the user, we inject the selected value within the query in real time using standard **string find and replace algorithm**.

Once the query is fired and the results are obt'd, the webpage routes to **/query** path within the host to display the results of the query. We make use of the **jinja** framework to pass the id of the user-selected query to the **/query** webpage. This helps us map the list of headers with the results based on the id of the query selected by the user.

7.1.2 Google Cloud Platform.

. One database server and one application server host the code for the entire application. The two servers exist independent of each other and interface through RESTful APIs that are invoked when the user selects a query to execute from the dashboard. In selecting a cloud provider for deployment, we took into consideration the following factors

- **Cost effectiveness** - A service that ensures maximum up time while keeping cost as low as 1 USD per day

- **Ease of access** - The server should be easily *ssh*-able and should also provide a GUI for easy access
- **Security** - Emphasis was laid on *Dual Key Authentication* during logging in to the server
- **Flexibility** - Ability to add and remove *third party services* on-demand as and when needed

7.1.3 Database Server: Cloud SQL.

. Hosted on IP Address 35.193.29.72, a second generation master databases server houses the following characteristics-

- Hosted in *us-central1-b* zone
- Machine type : db-n1-standard-1
- 1 virtual CPU clocked @ 4Ghz
- Memory : 3.75 GB
- SSD storage : 10 GB
- MySQL database version : 5.7
- Auto-storage scaling : enabled
- Auto-backups : enabled
- Globally scalable : disabled

7.1.4 Application Server: App Engine.

. Hosted on URL <http://104.196.144.102/>, our application server is wsgi gateway enabled and runs *Flask* framework to process client requests. Our m'logic is written in *Python* programming language and can be compiled using the Python2.7 interpreter.-

. The web interface of the application has two significant landing pages. The *HOME* page is served by an *html* file that is served when the *root* directory of the server received a *GET request*. The *RESULT* page is served by another *html* file that is served when the *URL path* matches *\query* and *request type* is *POST*

. The key parameters of the virtual machine are listed below-

- Hosted in *us-east1-b* zone
- External IP address 104.196.144.102
- CPU platform : Intel Haswell
- Machine type : n1-standard-1
- 1 virtual CPU clocked @ 3.5 Ghz
- Memory : 3.75 GB
- SSD storage : 10 GB
- MySQL database version : 5.7

7.1.5 Accessing the Cloud SQL server from remote client.

. One can access the database server by adding their public IP address to the list of *white list IP addressed* in the configuration file on Google Cloud. Following this, the server can be accessed by simple *ssh* command as -

```
mysql -host=35.193.29.72 -user=admin -password.
```

7.1.6 Accessing the web interface from a remote client.

. The application server running on *Flask* framework is available for public access on the following address -

<http://104.196.144.102>.

7.2 Demonstration of system

We developed a website, which, can be accessed by multiple users concurrently and perform different actions such as checking a business star rating in particular area, getting a list of businesses which are open within the given time frame, to name a few. We even added a functionality where the users are able to give reviews to a business. Our system internally handles all of the required queries and inserts, delete or updates accordingly. The image shown below is the website that the user sees where a description is provided to obtain specific output.

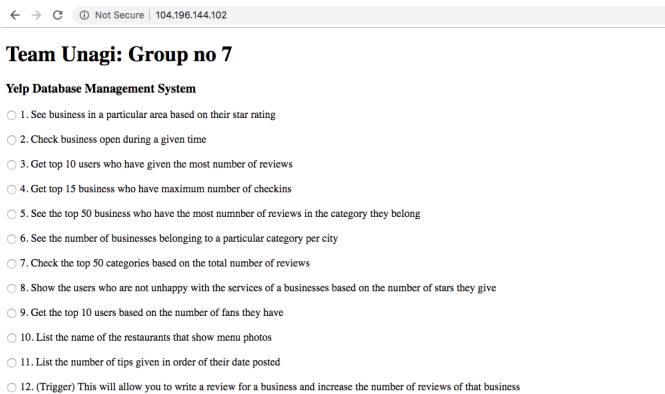


Fig 12. The website where any user can visit and get the results for performing tasks on the database
In sub sections below different types of queries are mentioned and their output will be shown.

7.2.1 Single table query.

See business in a particular area based on their star rating:

Query:
SELECT distinct name,stars,address,postal_code
FROM business
WHERE city="Toronto"
AND stars=4
LIMIT 50;



Fig 13.1 Here, the user is given the option to select from the list of different cities and the query will be executed accordingly.

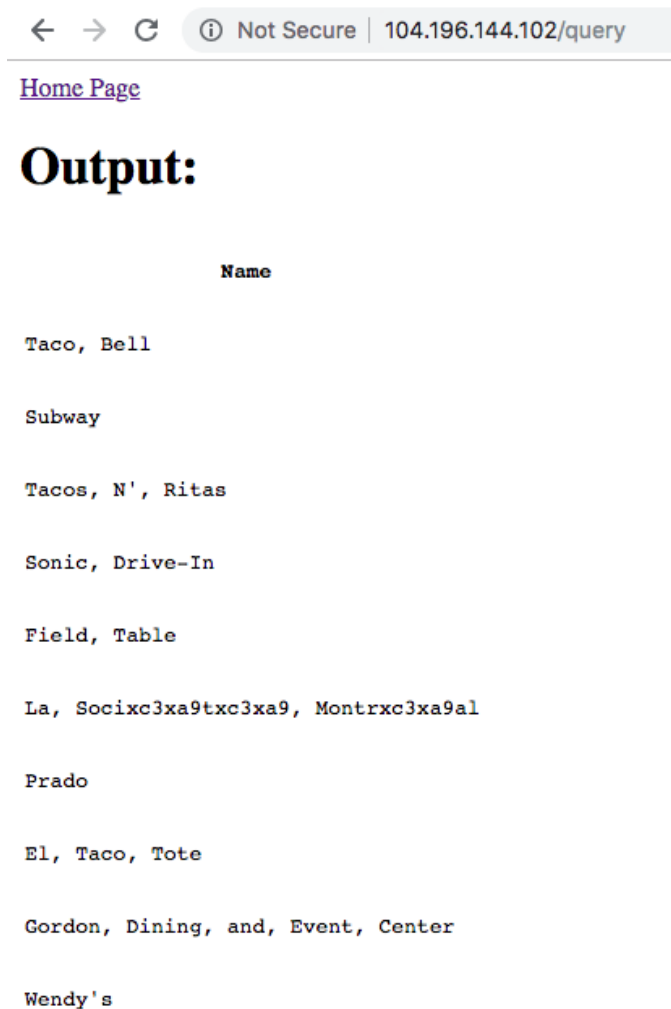
Name	Stars	Address	Postal code
GoodLife, Vapes	4.0	2095, Yonge, Street	M4B, 2A5
Ana's, Nail, Boutique, &, Spa	4.0	3349, Bloor, Street, W	M8X, 1E9
UVU, Nails, &, Spa	4.0	3503, Lake, Shore, Boulevard, W	M9W, 1N5
Iq, Food	4.0	613, King, Street, W	M5V, 1M5
Caffe, Furbo	4.0	55, Mill, Street,, Building, 59,, Unit, 104	M5A
East, West, Putons	4.0	126, Cartwright, Avenue	M6A, 1V2
Sho, Izakaya	4.0	1406, Queen, St, W	M6K, 1L9
CN, Tower	4.0	301, Front, Street, W	M5V, 2T6
Hamaru, Sushi	4.0	5469, Yonge, Street	M2N, 5S1
Booyah	4.0	16, Vaughan, Road, A	M6G, 2N1
GoodLife, Fitness	4.0	267, Richmond, Street, W	M5V, 3M6

Fig 13.2 The output of the above query is shown in another web page. This is an example of single table query

7.2.2 Multiple table query.

Check business open during a given time

Query:
SELECT DISTINCT name
FROM business, hours, categories
WHERE business.business_id=hours.business_id
AND business.business_id=categories.business_id
AND hours_Friday="7:0-1:0" and categories like "%Restaurants%";



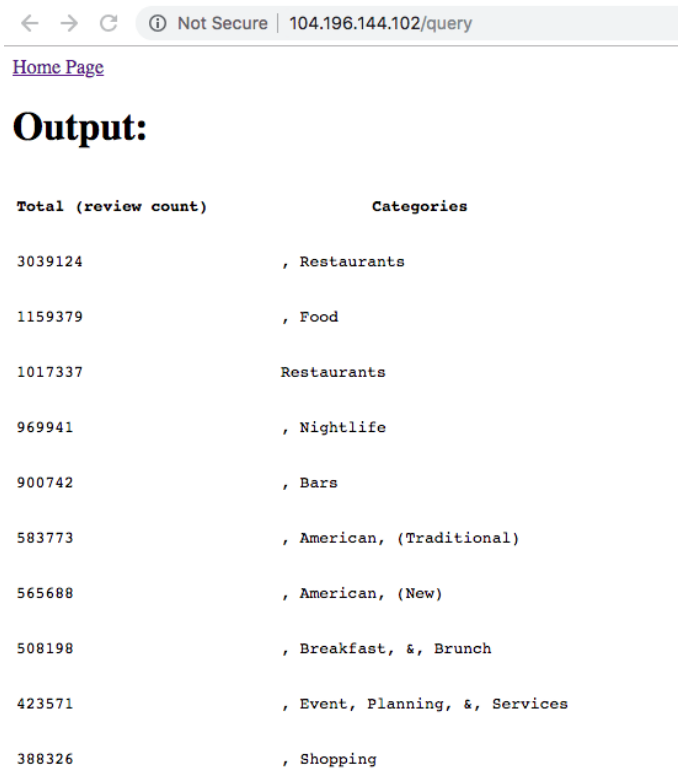
Name	review_count
Taco, Bell	3039124
Subway	1159379
Tacos, N', Ritas	1017337
Sonic, Drive-In	969941
Field, Table	900742
La, Socixc3xa9txc3xa9, Montrxc3xa9al	583773
Prado	565688
El, Taco, Tote	508198
Gordon, Dining, and, Event, Center	423571
Wendy's	388326

Fig 14. The output of this query is shown in another web page to the user.
This is a multiple table type of query.

7.2.3 Aggregation query.

Check the top 50 categories based on the total number of reviews.

Query:
 SELECT sum(review_count), categories.categories
 FROM business, categories
 WHERE business.business_id=categories.business_id
 GROUP BY categories.categories
 ORDER BY sum(review_count) DESC
 LIMIT 50;



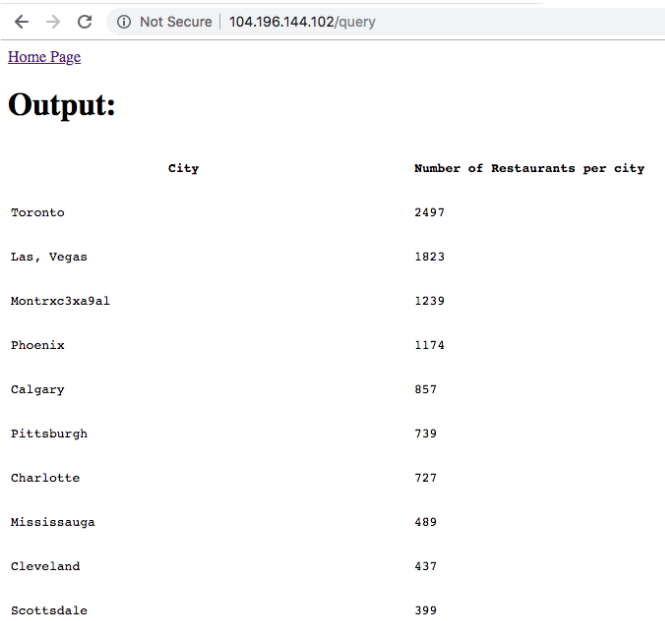
Total (review count)	Categories
3039124	, Restaurants
1159379	, Food
1017337	Restaurants
969941	, Nightlife
900742	, Bars
583773	, American, (Traditional)
565688	, American, (New)
508198	, Breakfast, &, Brunch
423571	, Event, Planning, &, Services
388326	, Shopping

Fig 15. The output of this query is shown in another web page to the user.
This is a aggregation type of query.

7.2.4 Inner join type query.

See the number of businesses belonging to a particular category per city.

Query:
 SELECT business.city, COUNT(*)
 AS "Number of Restaurants per City"
 FROM business
 INNER JOIN categories ON
 business.business_id= categories.business_id
 WHERE categories.categories = "Restaurants"
 GROUP BY business.city
 ORDER BY COUNT(*) DESC;



City	Number of Restaurants per city
Toronto	2497
Las, Vegas	1823
Montrxc3xa9al	1239
Phoenix	1174
Calgary	857
Pittsburgh	739
Charlotte	727
Mississauga	489
Cleveland	437
Scottsdale	399

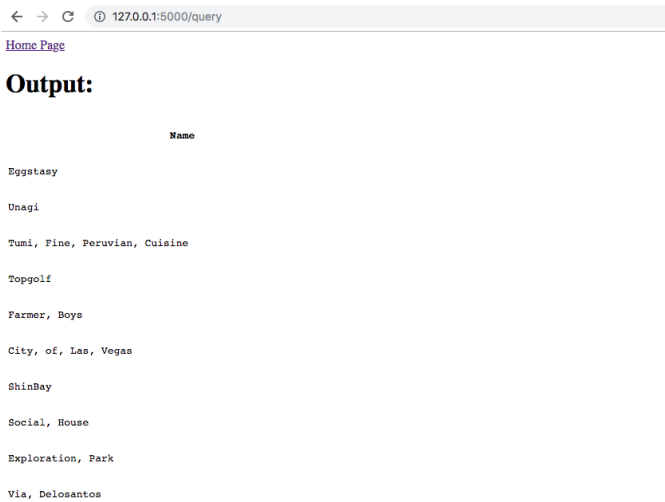
Fig 16. The output of this query is shown in another web page to the user.
This is a inner join type of query.

7.2.5 Nested table type query.

List top businesses with cool rated reviews

Query:

```
Select b.name
FROM business AS b
WHERE b.business_id in (SELECT r.business_id
FROM Reviews AS r
GROUP BY r.business_id HAVING SUM(r.cool) > 20
ORDER BY sum(r.cool) DESC);
```



Name
Eggstasy
Unagi
Tumi, Fine, Peruvian, Cuisine
Toppgolf
Farmer, Boys
City, of, Las, Vegas
ShinDay
Social, House
Exploration, Park
Via, Delosantos

Fig 17. The output of this query is shown in another web page to the user.
This is a nested table type of query.

7.2.6 Query that executes a trigger.

Here, user can give review to a restaurant. For this, an insert is executed. However, after this insert is executed, a trigger is executed which increases the review count of that business in the business table.

Query for insert:

```
INSERT INTO ReviewsVALUES(" 123",3,4,"abc1","2018-02-02, 10:03:06")
```

Query for Trigger:

```
delimiter //
CREATE TRIGGER ReviewCountIncrement AFTER
INSERT ON Reviews for each row
BEGIN UPDATE business
SET review_count = review_count + 1
WHERE business_id = NEW.business_id;
END;//
delimiter ;"
```

- ☐ 8. Show the users who are not unhappy with the services of a businesses based on the number of stars they give
- ☐ 9. Get the top 10 users based on the number of fans they have
- ☐ 10. List the name of the restaurants that show menu photos
- ☐ 11. List the number of tips given in order of their date posted
- ☒ 12. (Trigger) This will allow you to write a review for a business and increase the number of reviews of that business

Write a review for the new Starbucks opened in your area:

Love the coffee! Submit

Fig 18a. Here, the user can enter a review for business Starbucks.



Review_count
39

Fig 18b. The image shown above is the output of a previously executed query. It shows that the review count of the business 39.



Fig 18c. The image shown above is the output after the user writes the review for that business. It shows that the review count of the business incremented by 1 after the user wrote the review. Thus, the trigger was executed after the execution of insert query.

7.2.7 Other similar types of queries.

1. Get top 10 users who have given the most number of reviews

Query:
 SELECT name, MAX(review_count) as review_count
 FROM user
 GROUP BY name
 ORDER BY review_count DESC
 LIMIT 10;

← → ↻ ⓘ Not Secure | 104.196.144.102/query

[Home Page](#)

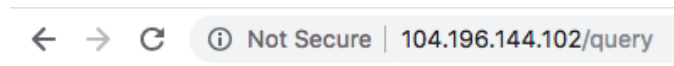
Output:

Name	Review count
Victor	13278
Shila	12390
Bruce	10022
George	7750
Nijole	7089
Kenneth	6738
Fox	6407
Jennifer	6314
Eric	5398
Rob	4888

Fig 19. The output of this query is shown in another web page to the user. This is a single table type of query.

2. Get the top 10 users based on the number of fans they have

Query:
 SELECT name, max(fans) AS no_of_fans
 FROM user
 GROUP BY name
 ORDER BY no_of_fans DESC
 LIMIT 10;



Output:

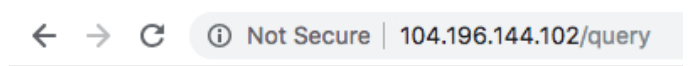
Name	No. of fans
Mike	9538
Katie	2964
Cherylynn	2434
Rugby	2383
Daniel	2132
Candice	2123
Peter	2086
Jessica	1964
Jeremy	1944
Brittany	1878

Fig 20. The output of this query is shown in another web page to the user.
This is a single table type of query.

3. List the number of tips given in order of their date posted

Query:

```
SELECT extract(year from date) as year, MONTHNAME(date)
as month, count(*) as Number_of_tips
FROM Tip
GROUP BY year, month
ORDER BY year DESC, COUNT(*) DESC;
```



Output:

Year	Month	No. of tips
2018	January	1387
2018	March	1130
2018	April	902
2018	February	565
2018	May	34
2017	March	2547
2017	June	1914
2017	July	1644
2017	April	1315
2017	September	1219

Fig 21. The output of this query is shown in another web page to the user.
This is a single table type of query.

4. Get top 15 business who have maximum number of checkins

Query:

```
SELECT b.name, b.address, b.city, MAX(number_of_checkins)
AS checkin_count
FROM business AS
b,checkin_countWHERE b.business_id = checkin_count.business_id
GROUP BY checkin_count.business_id
ORDER BY checkin_count DESC
LIMIT 15;
```

← → ↻ ⓘ Not Secure | 104.196.144.102/query

[Home Page](#)

Output:

Name	Address	City	Checkin count
McCarran, International, Airport	5757, Wayne, Newton, Blvd	Las, Vegas	143061
Phoenix, Sky, Harbor, International, Airport	3400, E, Sky, Harbor, Blvd	Phoenix	123126
The, Cosmopolitan, of, Las, Vegas	3708, Las, Vegas, Blvd, S	Las, Vegas	46384
Kung, Fu, Tea	5030, Spring, Mountain, Rd	Las, Vegas	38277
ARIA, Resort, &, Casino	3730, Las, Vegas, Blvd, S	Las, Vegas	34353
The, Venetian, Las, Vegas	3355, South, Las, Vegas, Boulevard	Las, Vegas	32343
Caesars, Palace, Las, Vegas, Hotel, &, Casino	3570, Las, Vegas, Blvd, S	Las, Vegas	30782
MGM, Grand, Hotel	3799, Las, Vegas, Blvd, S	Las, Vegas	30098
Bacchanal, Buffet	3570, S, Las, Vegas, Blvd	Las, Vegas	28872
Planet, Hollywood, Las, Vegas, Resort, &, Casino	3667, Las, Vegas, Blvd, S	Las, Vegas	25878
Earl, of, Sandwich	3667, Las, Vegas, Blvd, S	Las, Vegas	25835

Fig 22. The output of this query is shown in another web page to the user.
This is a multiple table and aggregation type of query.

5. See the top 50 business who have the most number of reviews in the category they belong

Query:

```
SELECT business.name, business.city, business.review_count,
business.stars
FROM business
INNER JOIN categories on
business.business_id= categories.business_id
WHERE categories.categories = "Shopping"
ORDER BY business.review_count DESC
LIMIT 50;
```

← → ↻ ⓘ Not Secure | 104.196.144.102/query

[Home Page](#)

Output:

Name	City	Review count	Stars
Bacchanal, Buffet	Las, Vegas	8339	4.0
Gordon, Ramsay, BurGR	Las, Vegas	5484	4.0
Yardbird, Southern, Table, &, Bar	Las, Vegas	3576	4.5
Paris, Las, Vegas, Hotel, &, Casino	Las, Vegas	2513	3.0
The, Oyster, Bar	Las, Vegas	2423	4.5
Nacho, Daddy	Las, Vegas	2400	4.5
The, Buffet, at, ARIA	Las, Vegas	2297	3.0
Studio, B, Buffet	Henderson	2241	4.0
TAO, Nightclub	Las, Vegas	2174	3.0
Pai, Northern, Thai, Kitchen	Toronto	2121	4.5

Fig 23. The output of this query is shown in another web page to the user.
This is a multiple table type of query.

6. Show the users who are unhappy with the services of a business based on the number of stars they give

Query:

```
SELECT user.name AS USERNAME, business.name
AS BUSINESS_NAME, Reviews.stars AS STARS_GIVEN
FROM business, user, Reviews
WHERE business.business_id = Reviews.business_id
AND Reviews.user_id =user.user_id
AND Reviews.stars < 2 and business.name="IHOP";
```

← → ↻ ⓘ Not Secure | 104.196.144.102/query

[Home Page](#)

Output:

Username	Business name	Stars given
Kristina	IHOP	1
Eugene	IHOP	1
Travis	IHOP	1
Nikki	IHOP	1
Laura	IHOP	1
JP	IHOP	1
Jaime	IHOP	1

Fig 24. The output of this query is shown in another web page to the user.
This is a multiple table type of query.

7. List the restaurant names that have menu photos to show

Query:

```
SELECT distinct business.name
FROM photo, business, categories
WHERE business.business_id = photo.business_id
AND business.business_id= categories.business_id
AND label = 'menu'
AND categories.categories LIKE '%Restaurants%'
LIMIT 50;
```



Fig 25. The output of this query is shown in another web page to the user.
This is a multiple table type of query.

9. CONTRIBUTIONS

- Query Formation :
-Done by Sonia, Aishwarya and Srushti
- Development, testing and deployment of query system including communication between the web interface and SQL server to execute different types of queries:
-Done by Nikunj and Ishan
- Report writing:
-Equally distributed amongst all team members
- Good communication between team member and transfer of knowledge in between all the phases were always maintained
- Find a good data set and brainstorm for data management:
-Done by all the team members and discussed in group meetings
- Designing the database and brainstorming on Entity Relational diagram:
-Done by all the team members and discussed in group meetings
- Creation and population of database:
-MySQL and Python scripts for this written by Srushti, Sonia and the documentation completed by Aishwarya, Ishan and Nikunj
- Detection of integrity constraints, functional dependencies, creating indexes, normalizing each table to BCNF :
-Done by Sonia, Aishwarya and Srushti

Rochester Institute of Technology

Introduction to Big Data - CSCI 620

Team Unagi - Group No. 7

Phase 2: Data Mining Description

Ishan Guliani
Rochester Institute of Technology
Rochester, New York
ig5859@rit.edu

Srushti Vijay Kotak
Rochester Institute of Technology
Rochester, New York
sk7735@rit.edu

Nikunj Rajesh Kotecha
Rochester Institute of Technology
Rochester, New York
nrk1787@rit.edu

Soniya Rode
Rochester Institute of Technology
Rochester, New York
sr5055@rit.edu

Aishwarya Sudhir Sontakke
Rochester Institute of Technology
Rochester, New York
as4897@rit.edu

1. MOTIVATION

. In this project, we explore the domain of big data and look at the data mining aspect of it. We aim to do sentiment analysis on topics extracted from review text in order to determine what aspects of businesses are important to customers. We want to understand whether the reviews given by the user are genuine and factually correct to help businesses out with their practices.

2. DESCRIPTION OF DATA

. The data set used in this project is compiled by Yelp and is available for public use at <https://www.yelp.com/dataset>. It contains a subset of their business, reviews and user data that is made available public for educational purposes. The data is available in JSON format and has six different files - business, users, reviews, tips, checkin and photos. The business.json contains important data about location, attributes and list of business categories. The review.json contains the full review text data that includes the user_id and business_id to indicate which user wrote which review about which business. The user.json includes all the user's friend information and the mapping of all the metadata associated with the user. The checkin.json contains the number of check ins made to the particular business. The tips.json contains text which are shorter than reviews and tend to convey quick suggestions to businesses. The photo.json contains photos with caption and classification information. Considering such significant and detailed amount of information we found here, it makes a fit choice for us to implement data management components.

3. PROPOSED DATA ANALYTICS TASKS

. A language understanding model which has ability to do sentiment analysis on the texts in a review to classify it as a good or bad review. Classifying these reviews will help to determine the business's attributes which lead to higher customer satisfaction. Being

able to see what reviews the users give to a business, we could provide feedback on the features and factors to a business to increase its popularity and customer experience. This predictive modelling will help to extract a business's strengths and weaknesses.

4. PLANNED ACTIVITIES

. Initially, we will have to clean the data and perform pre-processing steps on it as the reviews given by the user can be in many different forms. Next, we will do stemming and lemmatization steps to get the root of all the words to decrease the variance in all the text. After this, we will try two different approaches - Bag of words and Term Frequency Classifier (TF-IDF). Both of these approaches can be used to find the frequency of words, however TF-IDF approach is known to provide better output. For classification, we plan to consider other factors as features and implement logistic regression in order to classify a review accurately. For eg. 5 star reviews having the word 'rooftop' would indicate that rooftop is a good attribute for a business to have and would result in high customer satisfaction.

5. TIMELINE

Here is what we plan to achieve over the next few weeks-

- Defining domain constraints - *Week 0 - Week 1*
Gather resources, assumptions, constraints and other parameters which need to be mined from the data set. Then create a document defining the data mining goals including different dimensions to be taken into account.
- Assess gross and surface properties - *Week 1 - Week 3*
Build reporting and visualization components from the data set. This includes extracting nuances to convert the information to knowledge.
- Data preparation - *Week 4*
Select, clean, construct and format the data into the desired form. This includes implementing the commonly recurring patterns based on the data.

- Modeling - Week 5
Finalise modeling techniques and begin testing the set scenarios to validate the quality and validity of the model. Create all models as defined earlier.
- Evaluation - Week 5 - Week 6
Evaluate the created models and add features discovered during the development process. In this phase, new requirements may be raised due to the new patterns that have been discovered in the model results or from other factors.

6. WORKLOAD DISTRIBUTION

- Brainstorming on data analytics and to which data mining steps to perform
 - To be done by all team members
- Pre-processing and data cleaning steps which includes the stemming and lemmatization process
 - To be done by Nikunj and Srushti
- Visualize the data and find for important attributes and features
 - Initially to be explored by Ishan and Aishwarya and discussed with all team members
- Writing report for phase 3 and showing progress
 - To be distributed amongst all team members
- Exploring Bag of words, TD-IDF approach for classification
 - To be done by Sonia and Nikunj
- Applying the method and performing classification on the clean data
 - To be done by Srushti and Ishan
- Visualization and discussion of the analysis
 - To be done by all team members
- Final report and submission - To be done by all team members