**SQL – Things I Love & Hate**

This paper discusses what I encountered working with the Yelp dataset. This was the first time I had to work with Big Data so it was quite the challenge, but also proved very insightful.

**Capture**

Instead of working with the JSON files, I initially decided to work with the csv files. These were files that were converted to csv from JSON. I knew better than to expect these files to be clean and well-formed so I inspected them before I tried to get them into MySQL. Some of the files I couldn’t open with my go-to text editor, NotePad++. So I decided to try the CSVKit command-line tool kit. Since I never used it before, I was wasting a lot of valuable time just trying to learn to use it. With work and other responsibilities in mind, I decided to abandon using CSVkit and try a GUI csv file reader. I used CSVed which was able to open most of the csv files. The first file I opened and inspected was tip.csv. As suspected it contained dirty data. There were some rows of data that contained entries that had many commas. Other entries were missing data from columns. I believe that since a comma was used as a delimiter during the conversion process, entries that contained commas caused incorrect rows to be entered. Finally, I decided to use a Java program that parsed the data and used pipe symbols as delimiters. The resulting files were triple pipe separated files. I created a SQL schema based on the JSON object data of the dataset that is provided when you download the Yelp dataset. And finally imported the files into the database. It took over an hour to load the data into the dataset!

**ACCESS**

I love the aggregate functions like COUNT, MAX, AVG. The query below finds the business\_id, name and number of reviews for all business with greater than 1000 reviews. It took over a minute to get a result from the query, but this was actually fast compared to other queries I tried on the dataset.

**Query #1**

**SELECT** business**.**business\_id**,** name**,** **COUNT(**review**.**stars**)** **as** num\_reviews

**FROM** business **JOIN** review **ON** business**.**business\_id **=** review**.**business\_id

**GROUP** **BY** business\_id **Having** num\_reviews **>** 1000 **ORDER** **BY** num\_reviews**;**

+------------------------+---------------------------------------------+-------------+

| business\_id | name | num\_reviews |

+------------------------+---------------------------------------------+-------------+

| sNVGdeOPeitJ3OWUQBINzQ | BabyStacks Cafe | 1002 |

| IMLrj2klosTFvPRLv56cng | Honey Salt | 1005 |

| 4k3RlMAMd46DZ\_JyZU0lMg | Ramen Sora | 1008 |

| IsoLzudHC50oJLiEWpwV-w | Chelsea's Kitchen | 1012 |

| gTlDDzDEHyDQ6iwjNhpI6A | Mount Everest India's Cuisine | 1016 |

| 0NmTwqYEQiKErDv4a55obg | Scarpetta | 1028 |

| OARQDsxyoGnnX2FfSl9HjA | Le Reve - The Dream | 1031 |

| q-zZgXKAQFLEgMp9ZNllgQ | Umiya Sushi | 1033 |

| NCFwm2-TDb-oBQ2medmYDg | Fountains of Bellagio | 1035 |

| 3N9U549Zse8UP-MwKZAjAQ | Liberty Market | 1036 |

| PVTfzxu7of57zo1jZwEzkg | FEZ | 1039 |

| EnCIojgP5KTr1leaysFE3A | Viva Las Arepas | 1045 |

| UUGoM4q4i8rK2CBRS0xDAw | Jean Philippe Patisserie | 1046 |

...

| Wxxvi3LZbHNIDwJ-ZimtnA | The Venetian Las Vegas | 2780 |

| MpmFFw0GE\_2iRFPdsRpJbA | XS Nightclub | 2828 |

| g8OnV26ywJlZpezdBnOWUQ | Hash House A Go Go | 2835 |

| FaHADZARwnY4yvlvpnsfGA | McCarran International Airport | 2865 |

| El4FC8jcawUVgw\_0EIcbaQ | MGM Grand Hotel | 3016 |

| AV6weBrZFFBfRGCbcRGO4g | Luxor Hotel and Casino Las Vegas | 3166 |

| iCQpiavjjPzJ5\_3gPD5Ebg | Secret Pizza | 3258 |

| rcaPajgKOJC2vo\_l3xa42A | Bouchon at the Venezia Tower | 3292 |

| SMPbvZLSMMb7KU76YNYMGg | ARIA Resort & Casino | 3385 |

| 5LNZ67Yw9RD6nf4\_UhXOjw | The Cosmopolitan of Las Vegas | 3400 |

| KskYqH1Bi7Z\_61pH6Om8pg | Lotus of Siam | 3458 |

| ujHiaprwCQ5ewziu0Vi9rw | The Buffet at Bellagio | 3482 |

| 2weQS-RnoOBhb1KsHKyoSQ | The Buffet | 3676 |

| f4x1YBxkLrZg652xt2KR5g | Hash House A Go Go | 3881 |

| eoHdUeQDNgQ6WYEnP2aiRw | Serendipity 3 | 3911 |

| hihud--QRriCYZw1zZvW4g | Gangnam Asian BBQ Dining | 4120 |

| DkYS3arLOhA8si5uUEmHOw | Earl of Sandwich | 4655 |

| cYwJA2A6I12KNkm2rtXd5g | Gordon Ramsay BurGR | 5116 |

| K7lWdNUhCbcnEvI0NhGewg | Wicked Spoon | 5216 |

| RESDUcs7fIiihp38-d6\_6g | Bacchanal Buffet | 5715 |

| 4JNXUYY8wbaaDmk3BPzlWw | Mon Ami Gabi | 6414 |

+------------------------+---------------------------------------------+-------------+

177 rows in set (1 min 8.56 sec)

These aggregation functions make it easy to get results of queries that involve mathematical operations. However, if the dataset is very large, SQL is slow as shown in the next query (Query #2 next page). I did not like how long this took. Computers are supposed to count very fast and this query took over 15 minutes. The user dataset is very, very large. The long running time of this query is likely directly proportional to the dataset size.

**Query #2**

**SELECT** **COUNT(DISTINCT** name**)** **FROM** **user;**

+----------------------+

| COUNT(DISTINCT name) |

+----------------------+

| 88722 |

+----------------------+

1 row in set (15 min 29.80 sec)

In the first query I had to join two tables just so I could get the name of the business. Joining tables also causes queries to take longer. In Query #3 I joined the user and review tables and the result took much longer. I believe that one reason for the additional time is that I joined the review table with the user table which is much larger than the business table of the first query. The result was the empty set. Over an hour just to get the empty set!

**Query #3**

**SELECT** review**.**user\_id**,** name

**FROM** **user** **NATURAL** **JOIN** review **WHERE** review**.**text **LIKE** '%great%'**;**

Empty set (1 hour 29 min 24.24 sec)

I dislike how SQL is slow when it works on large datasets. When SQL has to join multiple tables with millions of records it is ridiculously slow. I have better things to do than wait over an hour for a query to finish.

One thing I really like about SQL is how a query can include multiple subqueries. You can break a problem into multiple sub-problems and use the results of each sub-problem to solve the greater problem. The following query is a simple example that demonstrates this. It finds the users that have an average\_star value that is greater than the average of all the users average\_star values (names contained in the query result are not shown since there are over half million names in the result set).

**Query #4**

**SELECT** name **FROM** **user** **WHERE** average\_stars **>** **(SELECT** **avg(**average\_stars**)** **FROM** **user);**

...

+----------------------------------+

583834 rows in set (21 min 45.07 sec)

**Optimize**

I decided to try and optimize the first query of this paper by using a materialized view. I created a SQL script that creates a table that will hold the captured data. This script also runs the query that populates the materialized view. The materialized view will contain businesses that have over 1000 reviews. The file business\_1k\_reviews.sql does the work.

**CREATE** **TABLE** Greater\_than\_1k**(**

business\_id **VARCHAR(**128**)** **NOT** **NULL,**

name **VARCHAR(**128**)** **NOT** **NULL,**

num\_reviews **INT** **NOT** **NULL,**

**UNIQUE** **INDEX** business\_mv **(**business\_id**)**

**);**

**INSERT** **INTO** Greater\_than\_1k

**SELECT** business**.**business\_id**,** name**,** **COUNT(**review**.**stars**)** **as** num\_reviews

**FROM** business **JOIN** review **ON** business**.**business\_id **=** review**.**business\_id

**GROUP** **BY** business\_id **Having** num\_reviews **>** 1000 **ORDER** **BY** num\_reviews**;**

mysql> source business\_1k\_reviews.sql;

Query OK, 0 rows affected (0.36 sec)

Query OK, 177 rows affected (1 min 12.08 sec)

Records: 177 Duplicates: 0 Warnings: 0

As you can see, to populate the materialized view took very close to the time to run Query #1 which is to be expected. However, the time to view the materialized view’s data was extremely fast – under a second!

**Query#5**

mysql> **SELECT** **\*** **FROM** Greater\_than\_1k**;**

+------------------------+---------------------------------------------+-------------+

| business\_id | name | num\_reviews |

+------------------------+---------------------------------------------+-------------+

| --9e1ONYQuAa-CB\_Rrw7Tw | Delmonico Steakhouse | 1311 |

| -ed0Yc9on37RoIoG2ZgxBA | Le Thai | 1117 |

| 0FUtlsQrJI7LhqDPxLumEw | Joe's Farm Grill | 1410 |

| 0NmTwqYEQiKErDv4a55obg | Scarpetta | 1028 |

| 0W4lkclzZThpx3V65bVgig | Schwartz's | 1603 |

| 2iTsRqUsPGRH1li1WVRvKQ | Carson Kitchen | 1225 |

| 2weQS-RnoOBhb1KsHKyoSQ | The Buffet | 3676 |

...

| YPavuOh2XsnRbLfl0DH2lQ | Lo-Lo's Chicken & Waffles | 1276 |

| yQab5dxZzgBLTEHCw9V7\_w | Charlotte Douglas International Airport | 1385 |

| z6-reuC5BYf\_Rth9gMBfgQ | St. Francis Restaurant | 1195 |

| zdE82PiD6wquvjYLyhOJNA | KoMex Fusion | 1128 |

| ZjSzUWHtnpCfjsa7CksSOg | The Signature at MGM Grand | 1356 |

| ZkGDCVKSdf8m76cnnalL-A | Le Village Buffet | 1864 |

| \_w5hBpkjHs5\_Hv3pLeHtIw | Julian Serrano | 1429 |

+------------------------+---------------------------------------------+-------------+

177 rows in set (0.00 sec)

The above query shows that using materialized views can greatly speed up data retrieval of queries that normally run slow. The down-side is that you must update materialized views periodically.

Query #1 time without optimization (1 min 8.56 sec)

Query #1 with optimization (0.00 sec)

**Data Quality**

My schema had some constraints in place, like NOT NULL, but it could stand to be more robust. The tables that contain a primary key of another table should use them as foreign keys. It’s important to add constraints to the schema to prevent bad data corruption and dangling pointers. Attribute based checks or tuple based checks can be used. I will add such checks to my schema if we are going to continue to work with this database. I will also consider adding triggers to help maintain the integrity of the data.

**Summary**

I learned that it is extremely important to inspect data before trying to import it into a database. This is especially true of big data. You can waist a lot of time trying to import bad data. Clean data first, then import it into database. Additionally, I learned how slow SQL is querying big datasets. Queries that involve joins are especially slow for obvious reasons. Materialized views can increase the speed of usually long queries. However, materialized views must be updated periodically. The frequency of the update depends on what the database is used for. The fact that you must maintain materialized views is a negative factor. It would be much nicer to use a different database type that can get similar results with regards to their speed.