**MongoDB and Cassandra Queries**

For the purpose of this assignment, I have collected two datasets; one is taken from the Twitter API and the other is taken from the Openweathermap API. From the Twitter API, all the tweets having the hashtag #winter is collected and stored in a .csv file. From the Openweathermap API, a 16-day forecast data is collected from 10 different cities and stored in a .csv file.

Data from the twitter API:

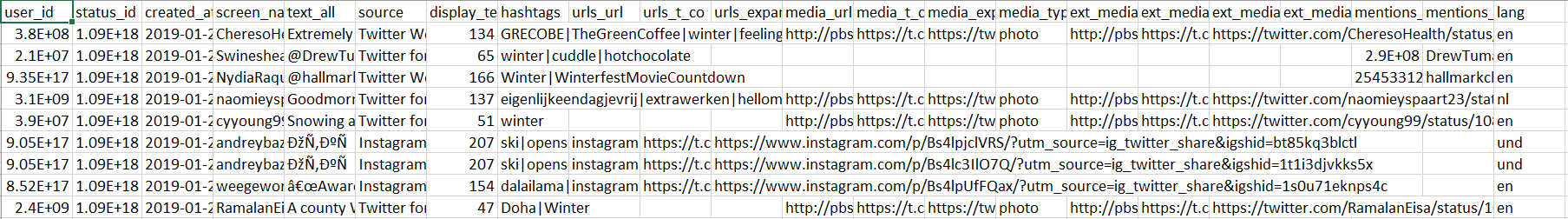
I have collected the tweets having the hashtag winter using the twitter API. For that, it is required to get a key from the Twitter API and using that key we can collect live tweet data for the desired duration and store it in a .csv format. I have done this using R language and R studio. The code for getting the tweet data for the hashtag winter is as follows:

The code for collecting the tweet data by using the twitter API is as follows:



In the above code, the required packages are twitter, streamR, rTweet and ROAuth. We first authenticate by using the consumer key and access token provided by the twitter API. This authentication is for rTweet. Next, we provide authentication for StreamR in the form of consumer keys and access tokens. Then, we pull 1000 historical tweets that contain the #winter and which is in English language and store it in a dataframe. Next, we pull live streaming data for a duration of 3 hours and store this data in another dataframe. Now, we merge both the dataframes using the rbind() function and then convert these into .csv file by using the data.table library in R.

The collected data in a .csv file looks like this:



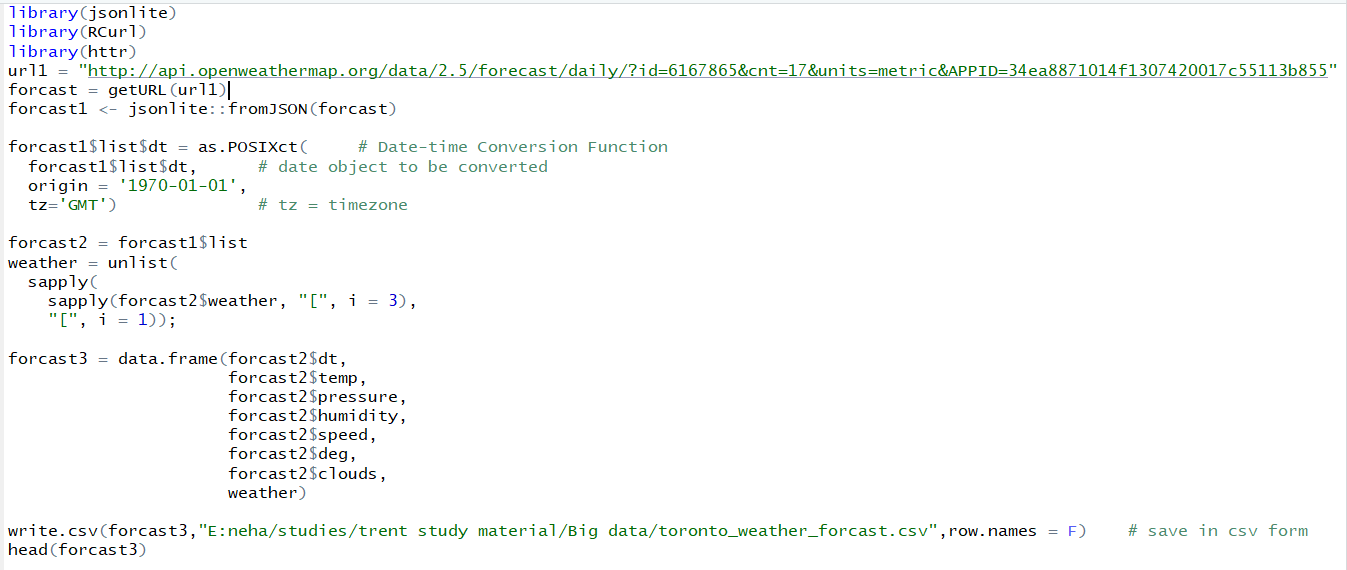
It contains columns like user\_id, status\_id, the actual tweet, the source of the tweet, the hashtags associated with it, media and url information and so on.

Data from Openweathermap API:

Like the twitter API, to collect weather data from openweathermap, we need a key for authentication. After getting the key, we can get a 16-day forecast for any city that we need. An example of using this API is:

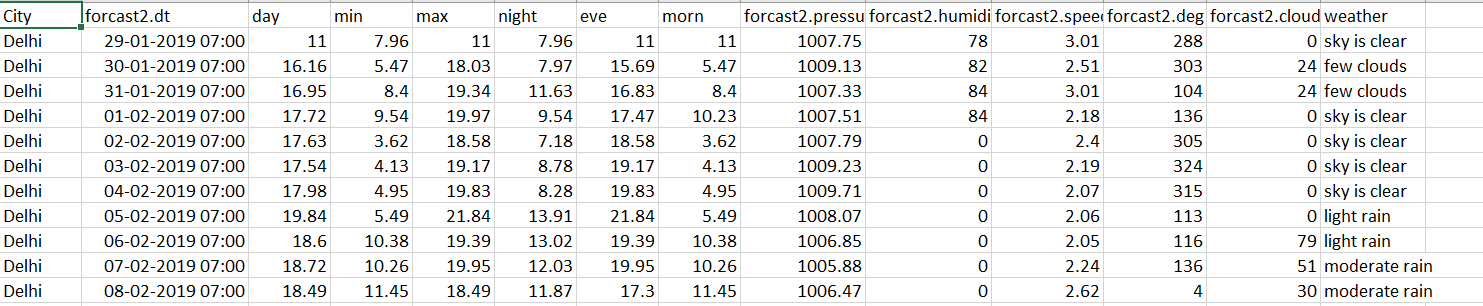
"http://api.openweathermap.org/data/2.5/forecast/daily/?id=6167865&cnt=17&units=metric&APPID=34ea8871014f1307420017c55113b855

In the above call, we get a 16-day forecast for Toronto which has the city id 6167865. We also specify the units as metrics to get the temperatures in degree Celsius. I then used this API call to get weather data about 10 different cities and then merged them in a single .csv file. I have used R to do this. The code for getting the data for Toronto and storing it in a file is as follows:



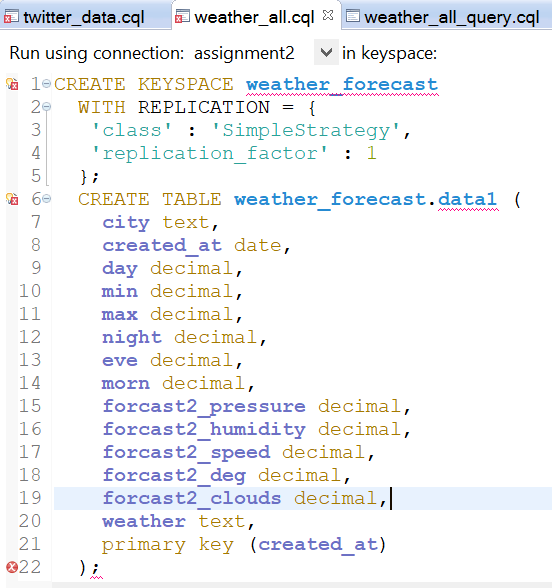
I have used the libraries jsonlite, RCurl and httr for this purpose. We then convert the date and time in proper format and store it appropriately. We then get the required columns for which we need the weather data and then write this data in a .csv file. We do this for different cities by just changing the city-id.

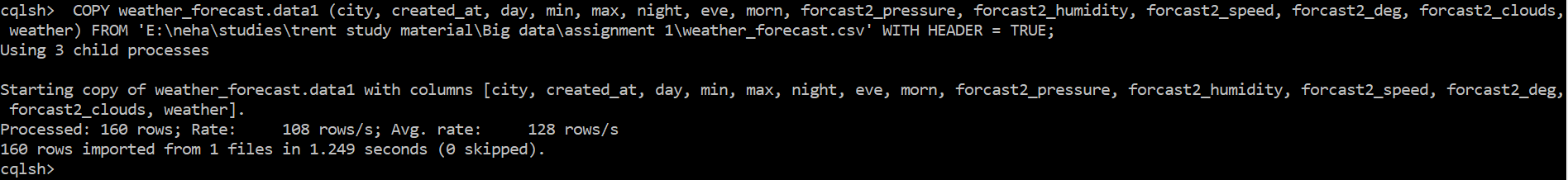
The final csv file looks somewhat like this:



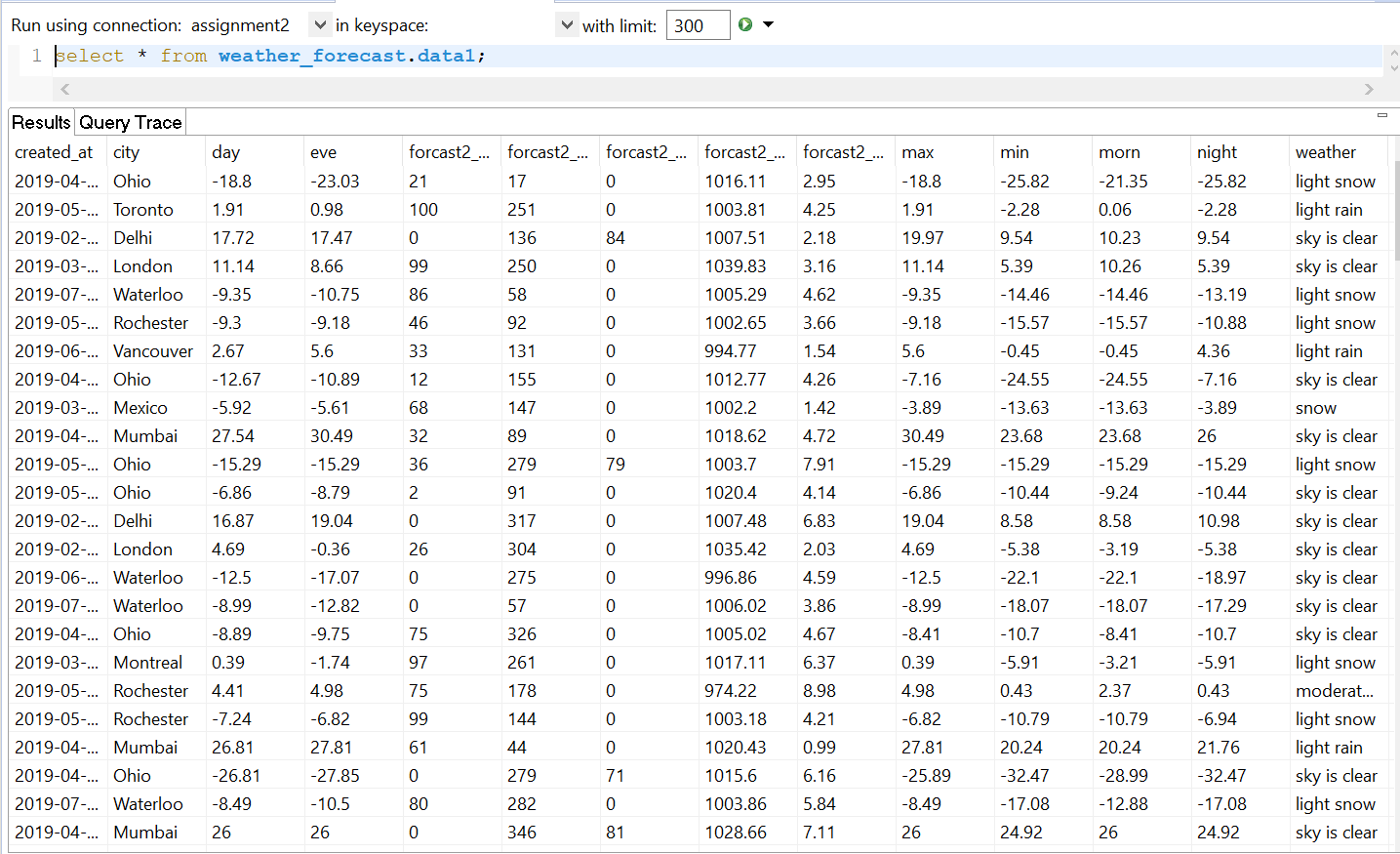
Loading the data in Cassandra and running queries on it:

First, I loaded the weather\_forecast data into Cassandra by using the COPY FROM command in the cql shell. To do that, I first created a keyspace named weather\_forecast and made a table called data1 with all the corresponding column names. Then I imported the data from the .csv file into the Cassandra table. The code is as follows:

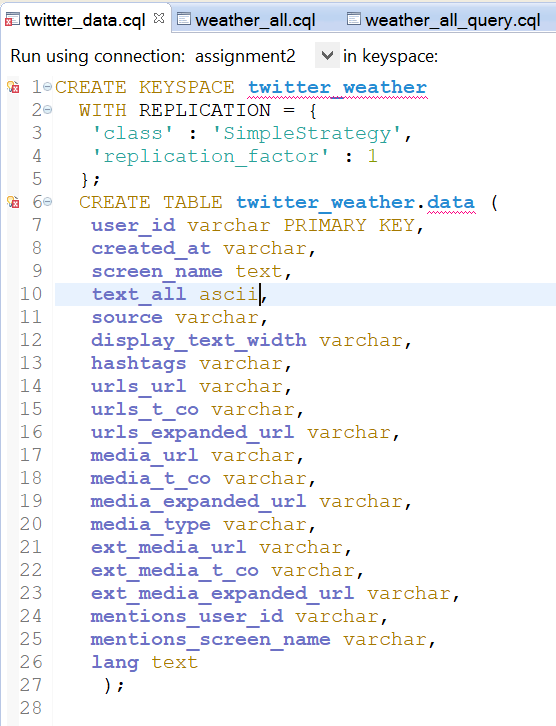


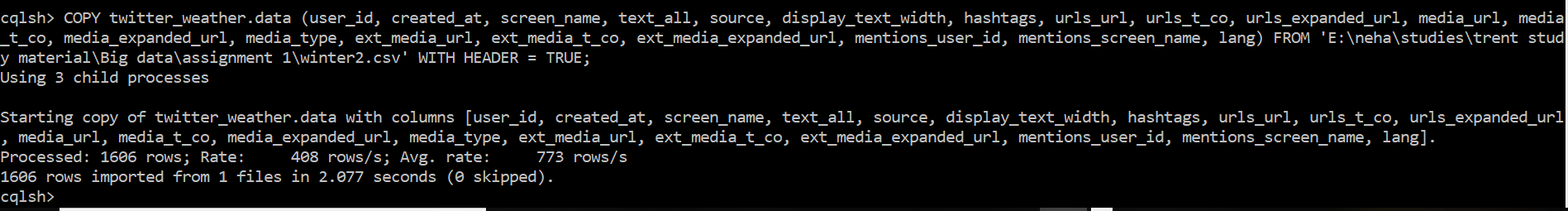


The table in Cassandra is as follows:

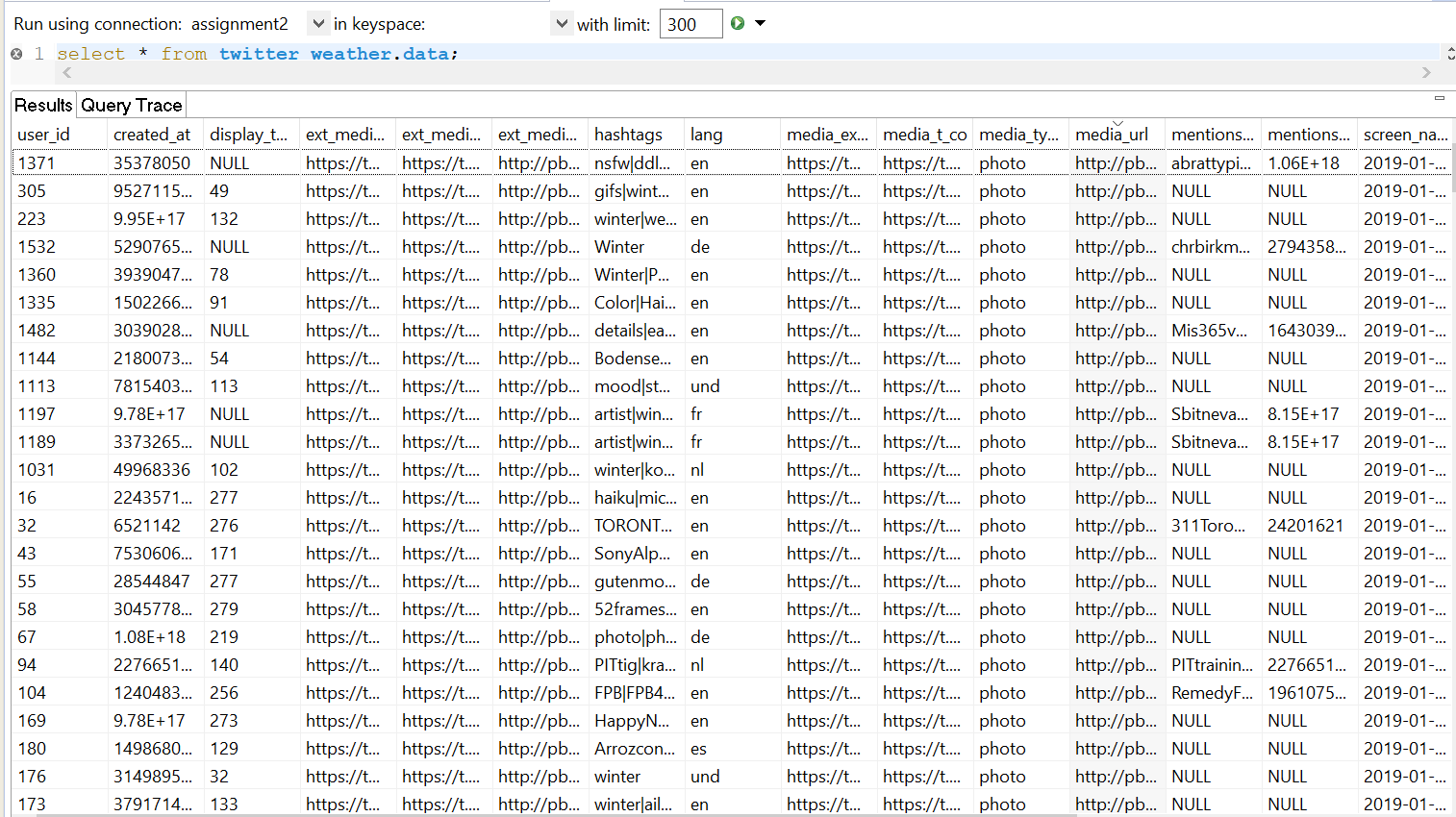


The second dataset (the twitter dataset for weather) is also loaded as follows:





The final table in Cassandra is as follows:



Query 1: From the weather forecast dataset, I find the cities that have weather forecast that contains the word “heavy”.

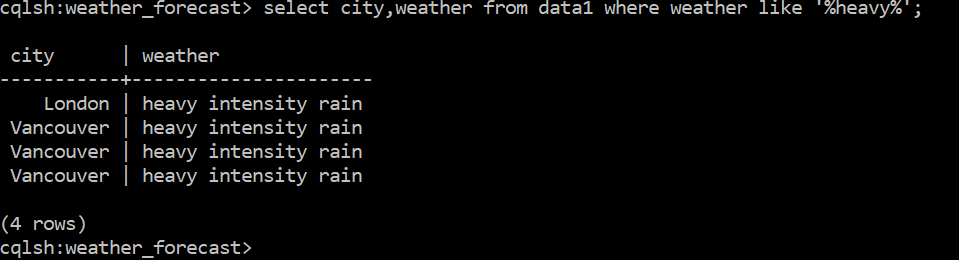
For this, I used the LIKE operator in CQL. But to use the LIKE operator, it is required to use SASI indexing for the columns on which we will be using the LIKE operator. The SASI index is generated as follows:

CREATE CUSTOM INDEX fn\_prefix ON weather\_forecast.data1 (weather)

USING 'org.apache.cassandra.index.sasi.SASIIndex';

Now, the query can be executed using the LIKE operator.

The query and result are as follows:

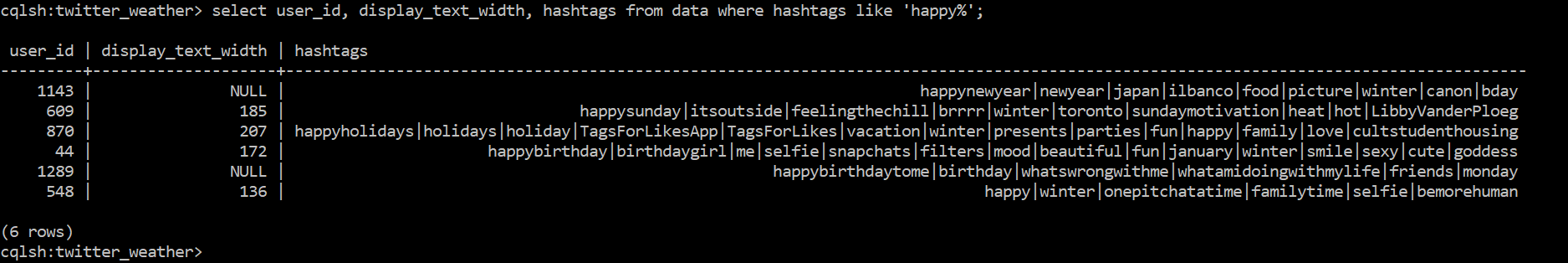


In the twitter weather dataset, I found the rows that start with the hashtag “happy”.

Here too, we have to have a SASI index.

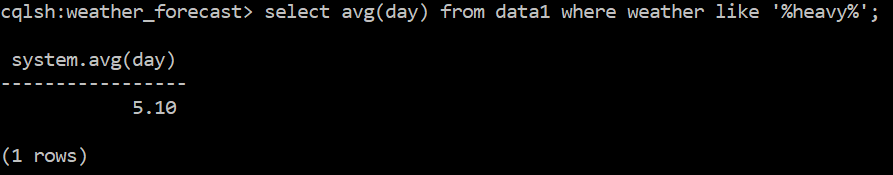
CREATE CUSTOM INDEX fn\_p ON data (hashtags)  
USING 'org.apache.cassandra.index.sasi.SASIIndex' WITH OPTIONS = {  
'mode': 'CONTAINS',  
'analyzer\_class': 'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',  
'case\_sensitive': 'false'  
};

The output and the result are as follows:



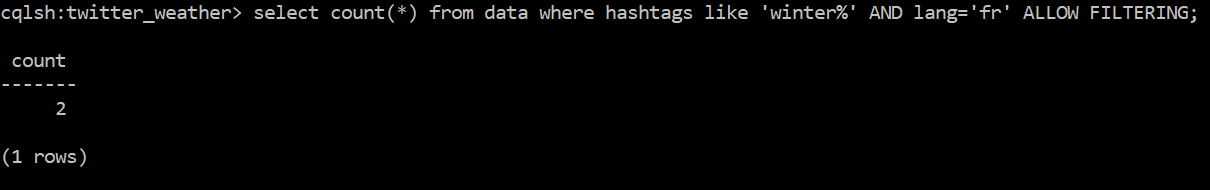
Query 2: From the weather\_forecast dataset, I found average day temperature for the weather containing the word “heavy”.

The query and output are:



From the twitter weather dataset, I counted the number of people that have the first hashtag as “winter” and the language of their tweets as French (fr).

The query and result are as follows:



Hence, there are just 2 such people.

Query 3: For the weather forecast dataset, I have found the rows that have weather as light rain and ordered the rows by morning temperature in descending order.

In order to use the ORDER BY function in Cassandra, it is required to have a table specifying the partitioning key and clustering column. The partitioning key will be the name of the column in the WHERE clause and the clustering column will be the name of the column that is used for ordering the data. For the purpose of running our query, I made a different table specifying weather as partitioning key and day as the clustering column as shown in the code below:

CREATE KEYSPACE weather\_forecast1

WITH REPLICATION = {

'class' : 'SimpleStrategy',

'replication\_factor' : 1

};

CREATE TABLE weather\_forecast1.data2 (

city text,

created\_at date,

day decimal,

min decimal,

max decimal,

night decimal,

eve decimal,

morn decimal,

forcast2\_pressure decimal,

forcast2\_humidity decimal,

forcast2\_speed decimal,

forcast2\_deg decimal,

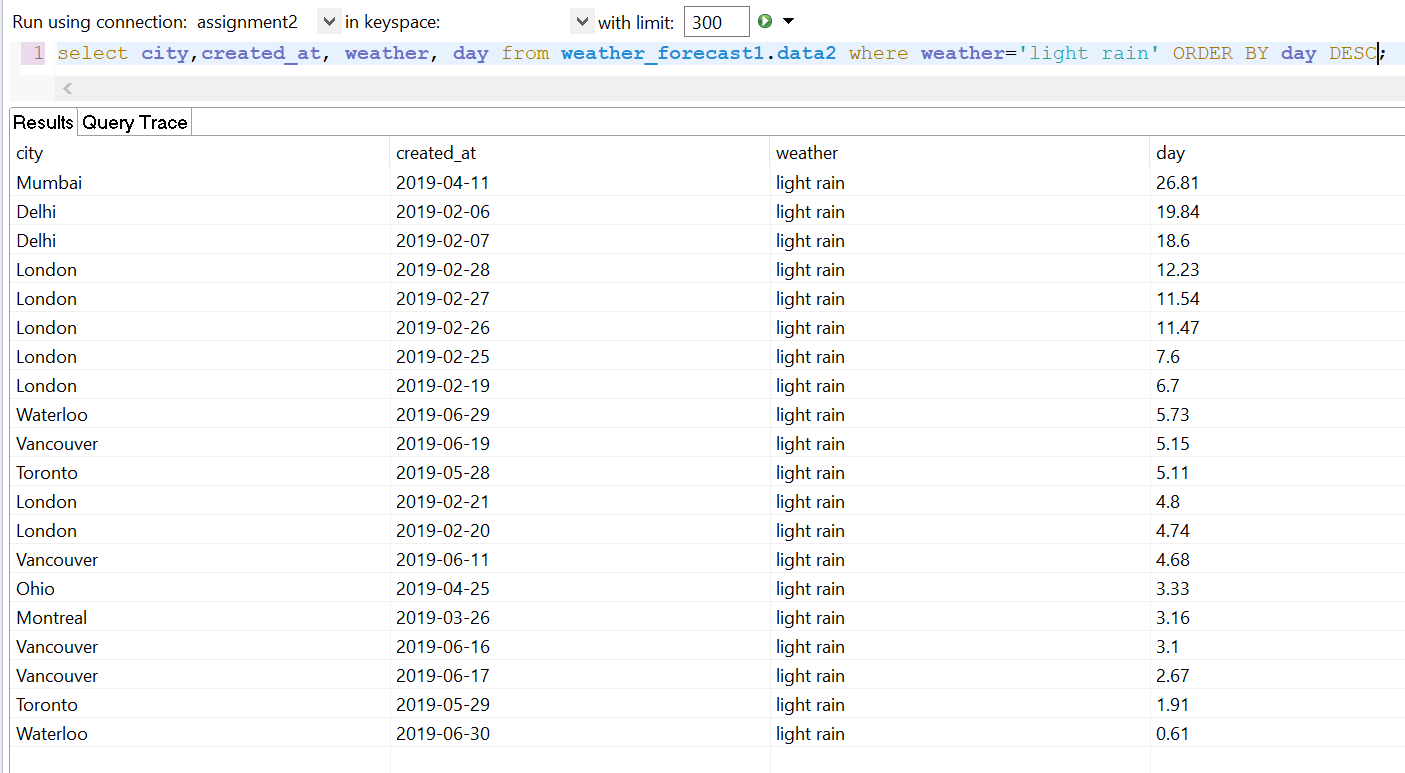
forcast2\_clouds decimal,

weather text,

primary key (weather, day)

);

The query and results are as follows:



For the twitter weather dataset, I have filtered the tweets where language is English and arranged them according to display text width in ascending order.

Here too, I made a different table specifying language as partitioning key and display\_text\_width as the clustering column as shown below:

CREATE KEYSPACE twitter\_weather1

WITH REPLICATION = {

'class' : 'SimpleStrategy',

'replication\_factor' : 1

};

CREATE TABLE twitter\_weather1.data3 (

user\_id varchar,

created\_at varchar,

screen\_name text,

text\_all ascii,

source varchar,

display\_text\_width varchar,

hashtags varchar,

urls\_url varchar,

urls\_t\_co varchar,

urls\_expanded\_url varchar,

media\_url varchar,

media\_t\_co varchar,

media\_expanded\_url varchar,

media\_type varchar,

ext\_media\_url varchar,

ext\_media\_t\_co varchar,

ext\_media\_expanded\_url varchar,

mentions\_user\_id varchar,

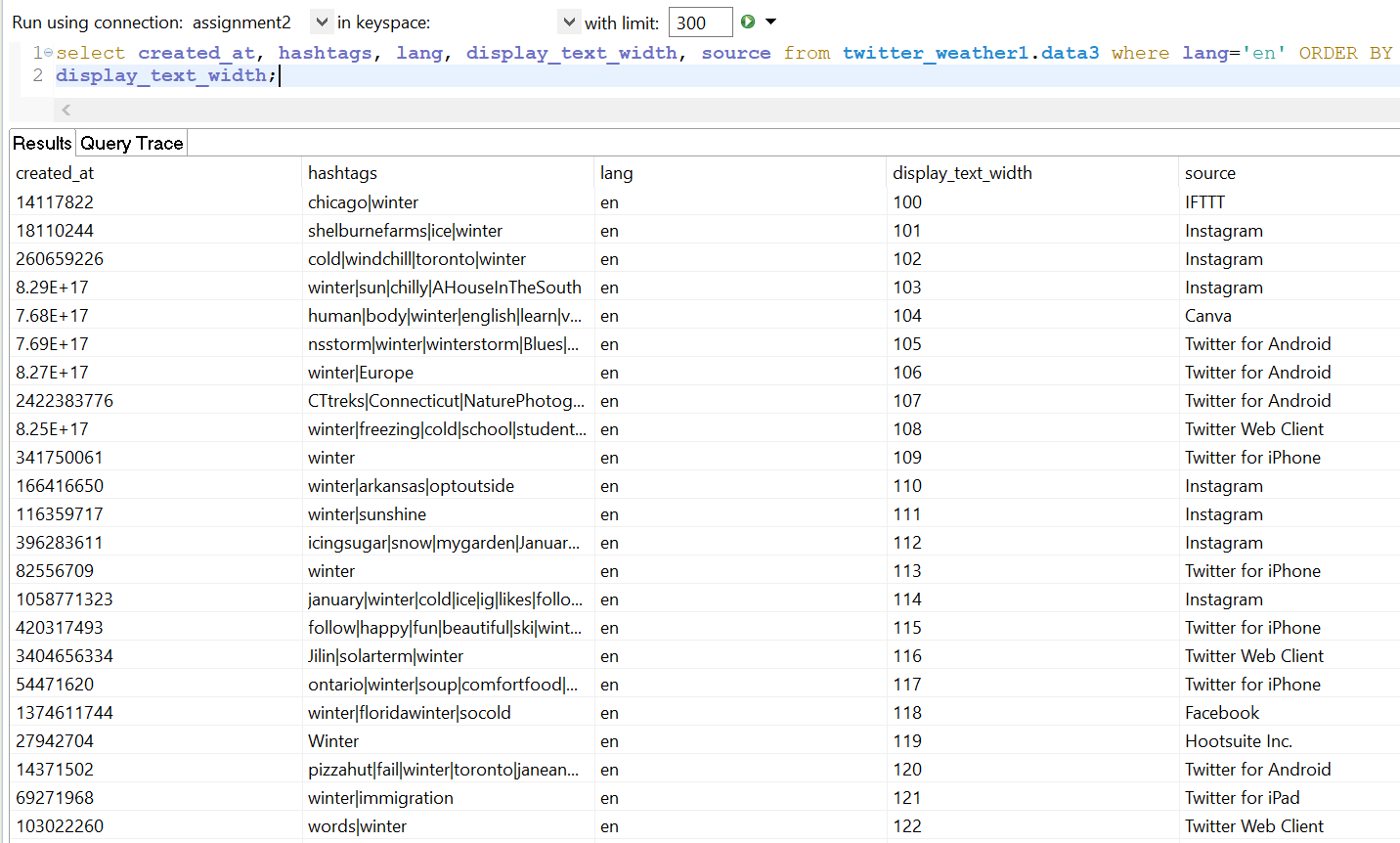
mentions\_screen\_name varchar,

lang text,

PRIMARY KEY (lang, display\_text\_width)

);

The query and output are:



Comments on complexity of code and query performance in Cassandra:

Regarding coding, the complexity of the code is somewhat moderate since CQL is primarily based on SQL and hence, the concepts used in writing SQL can be applied in writing CQL too. Regarding the performance of the query in Cassandra, Cassandra distributes data efficiently, allows almost linear scalability, writes data fast and provides almost constant data availability. The query performance can be seen in Datastax at the bottom bar where we get the output. All in all, as compared to MongoDB, the performance of Cassandra is better and efficient. But it is important to have expertise in Cassandra in order to harness maximum benefits and performance out of it.

Comments on searching alternate forms of the same word:

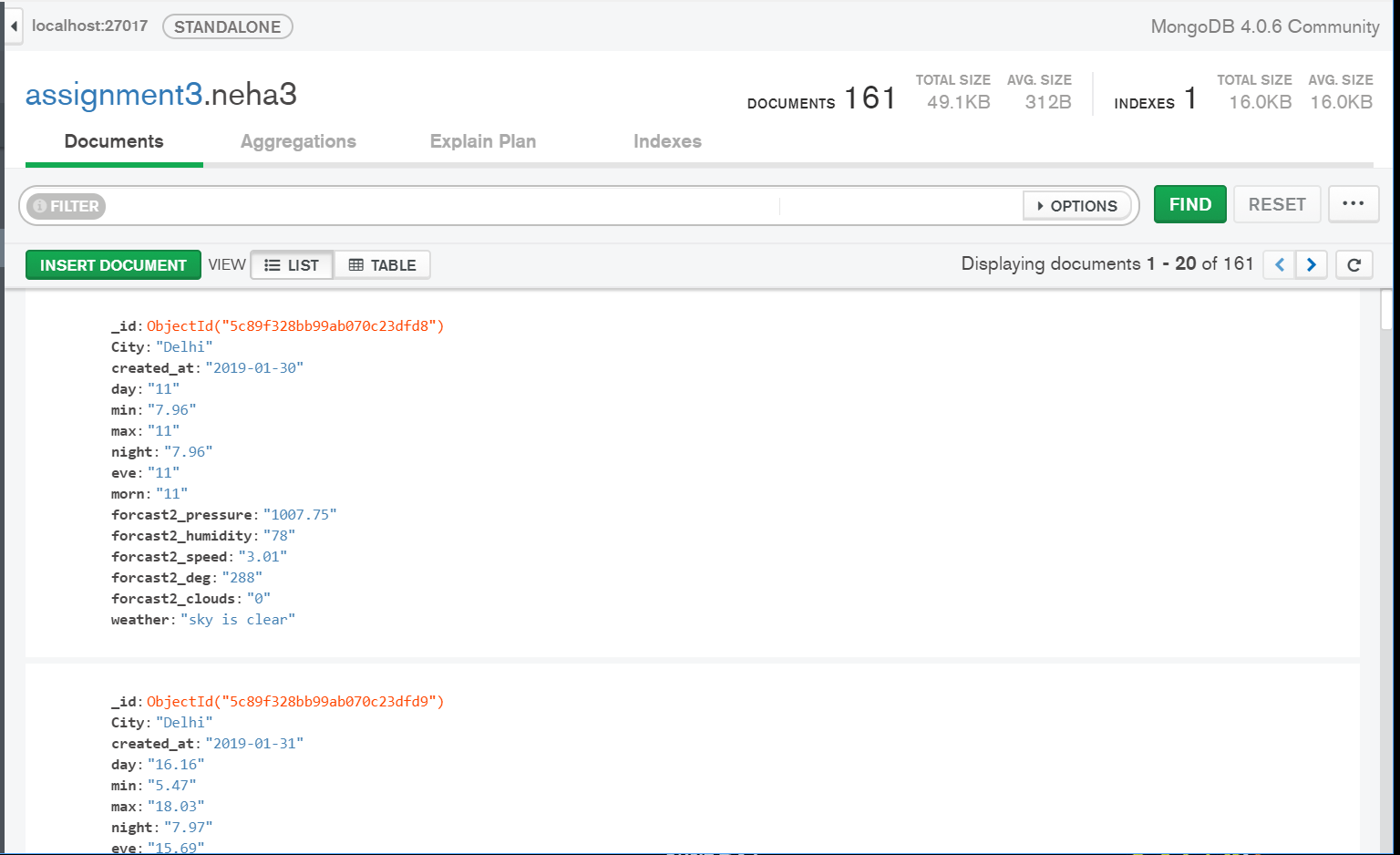
In Cassandra, to search for a word in an unstructured part of the document, it is required to index that column first. Then only keywords like CONTAINS and LIKE can be used in Cassandra. Here too, prior knowledge of regular expressions is required so that patterns of the words can be searched. Indexing on the column can be done by the following code:

CREATE CUSTOM INDEX fn\_p ON data (hashtags)  
USING 'org.apache.cassandra.index.sasi.SASIIndex' WITH OPTIONS = {  
'mode': 'CONTAINS',  
'analyzer\_class': 'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',  
'case\_sensitive': 'false'  
};

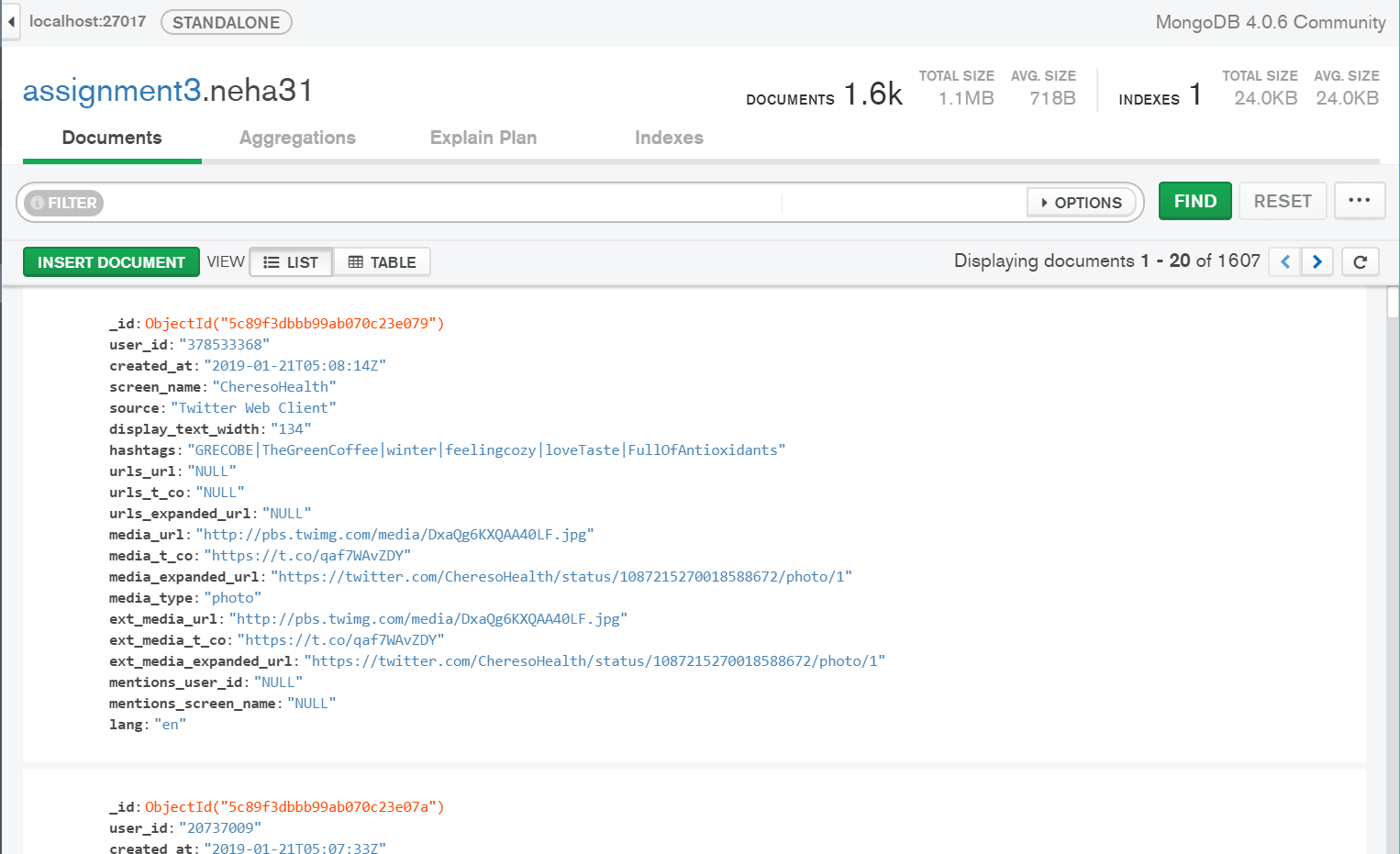
Loading the datasets on MongoDB and running queries on it:

I imported both the datasets into MongoDB using MongoDB Compass. I have made two connections for that. The two datasets look as follows:

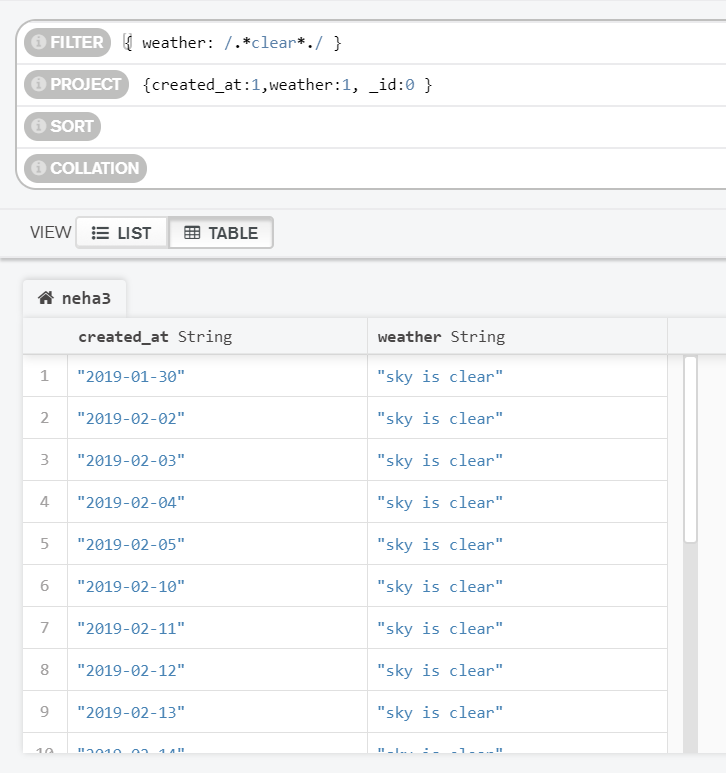
The weather\_forecast dataset:



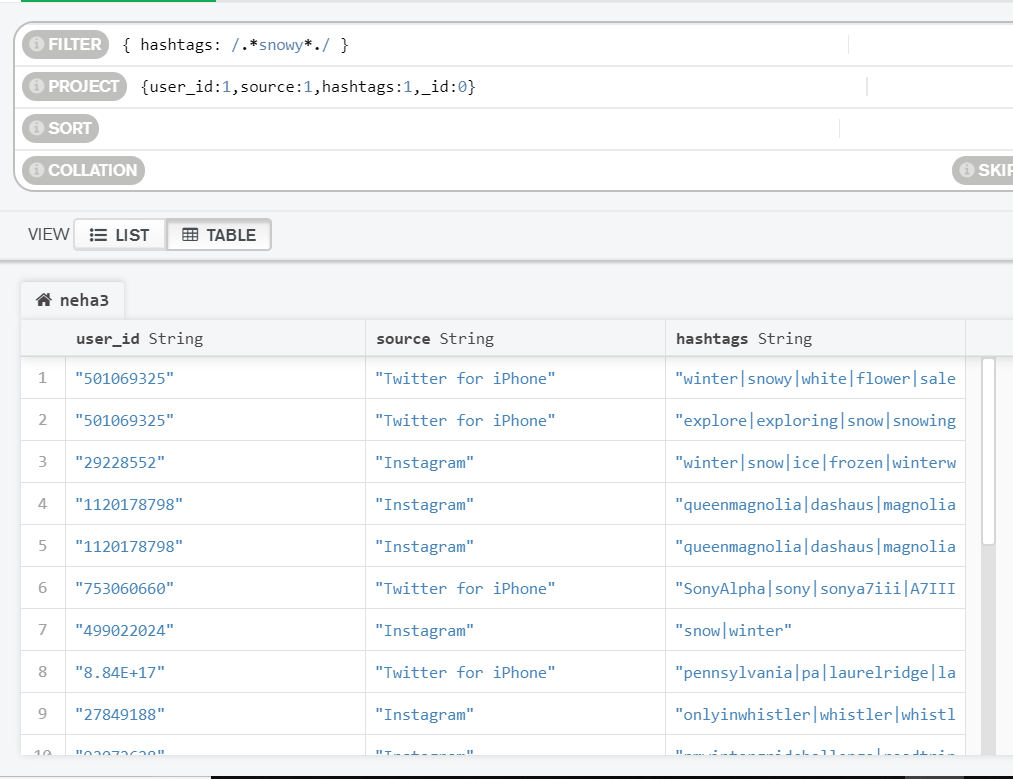
The twitter\_weather dataset looks like this:



Query 1: From the weather forecast dataset, we will display the dates that have the weather forecast containing the word ‘clear’.

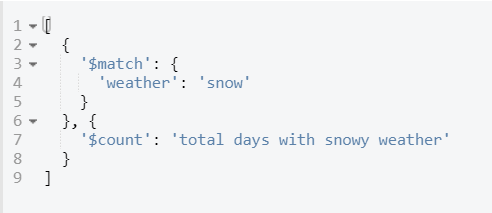


From the twitter weather dataset, we display the source and the user id of hashtags that contains the word “snowy”.

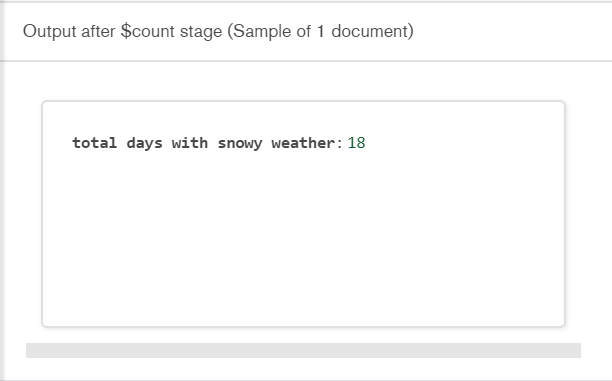


Query 2: Finding the total number of days that have weather as “snow”.

The query is as follows:



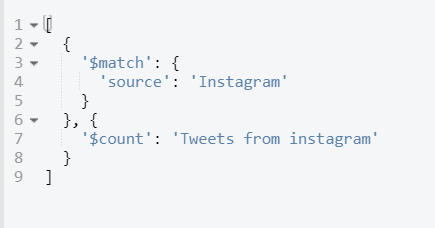
The output is:



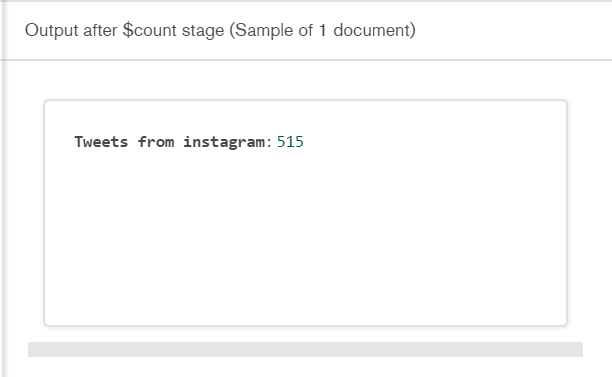
Hence, there are 18 such days.

In the twitter weather dataset, I have found out the number of documents that have tweets from Instagram.

The code is:



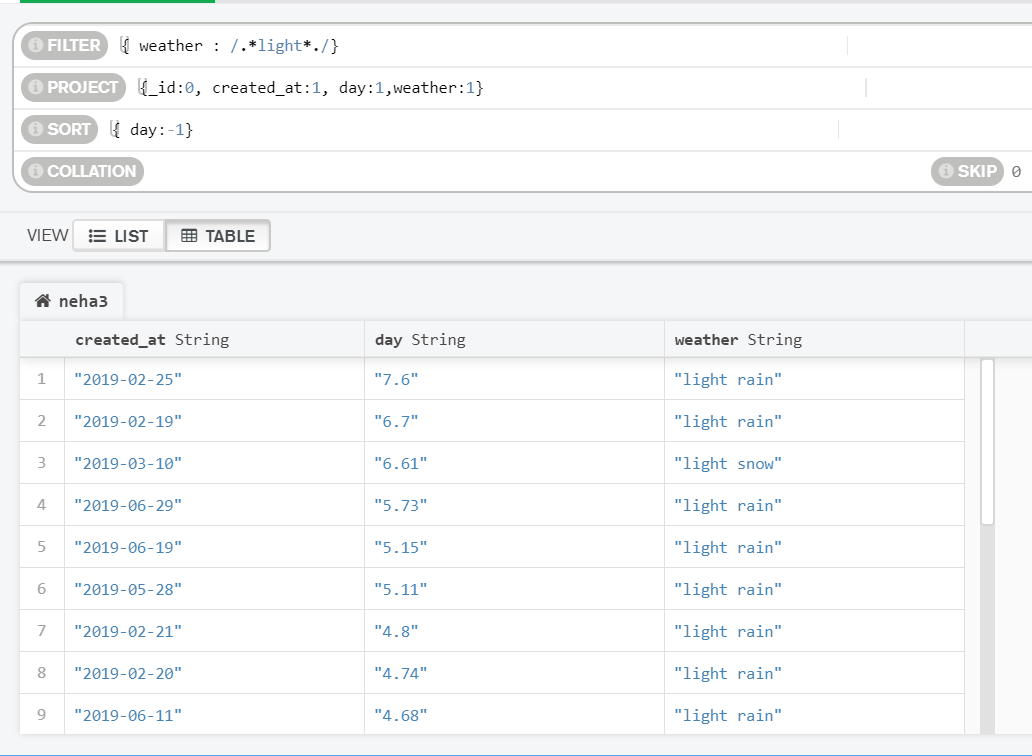
The output is:



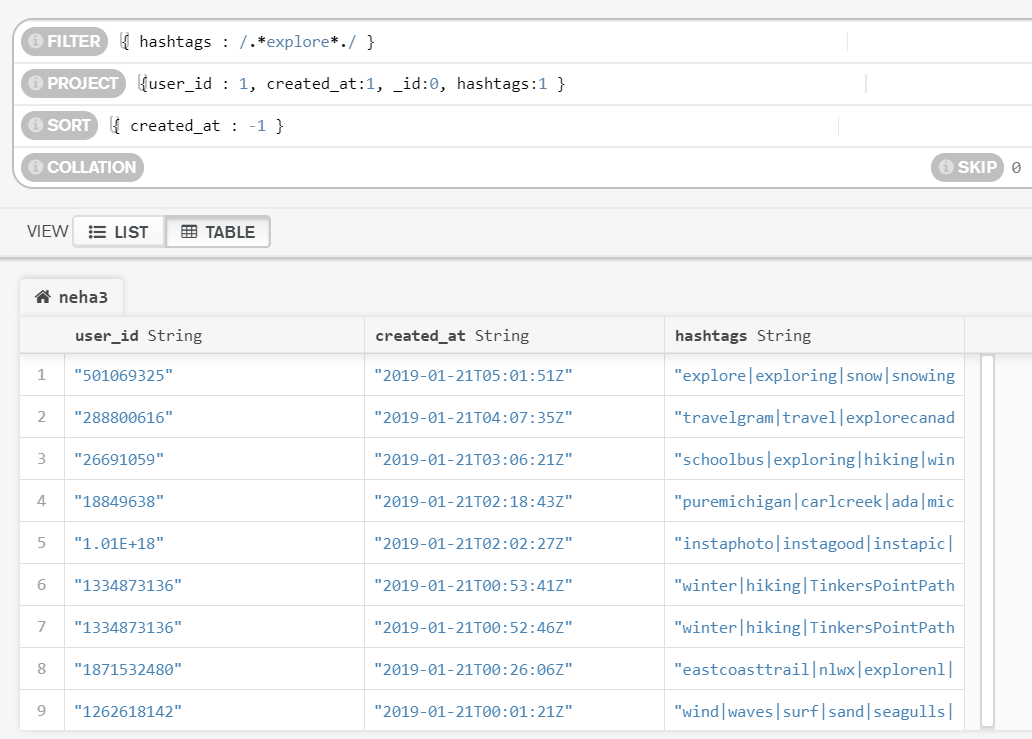
Hence, there are 515 tweets from Instagram.

Query 3:

From the weather forecast dataset, I have sorted the documents according to their day temperature in descending order and filtered out the documents containing the word “light”.

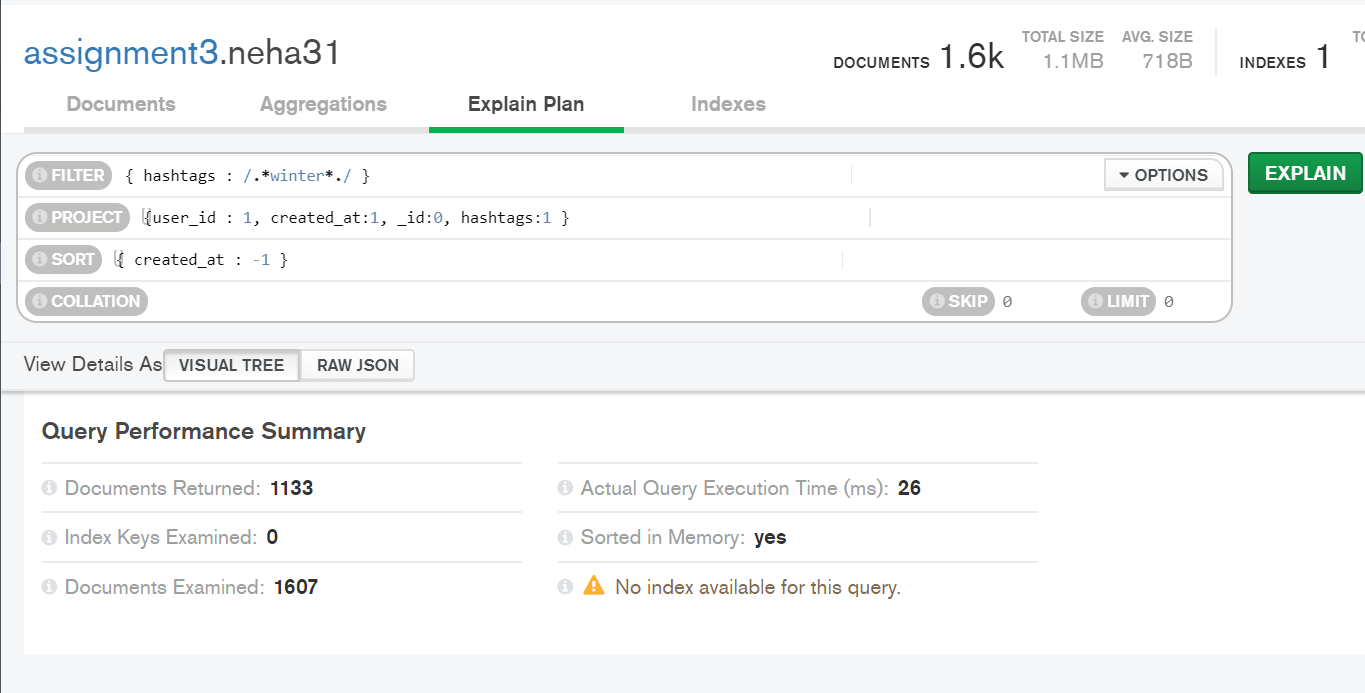


From the twitter weather dataset, I have filtered out the documents containing the hashtag “explore” and sorted them according to the date.



Comments on complexity of code and query performance in MongoDB Compass:

Writing a code in Compass is very easy since it has a very flexible GUI and required minimal extensive coding skills. Also, the query execution performance is also pretty fast. We can analyse the performance of the query in Compass by clicking on the explain plan tab. The example is:



We can see that it takes 26 ms for the above query to be executed by the MongoDB compass. In this way, we can see exactly what the query performance is.

Comments on searching alternate forms of the same word:

As we saw in query 1, we searched for rows having hashtag as “snowy” from our twitter weather dataset. The result of the query contains alternate forms of the word snowy like snow and snowing. This is because we specified in our query ‘/.\*snowy\*./’ which searches for different forms of the word. This is a very good and easy coding in MongoDB. Thus, for searching alternate forms, it is required to have prior knowledge of regular expressions so that specific patterns in the words can be queried.