# HADOOP
## FINAL GROUP PROJECT

## GROUP E

**Patricia Llull Sperandio**

**Anna Koenig**

**Gotham Nair**

**Tommaso Mercaldo**

**Cyril Gebara**

**Tina Marie Sodal**

#1 A database called 2019o2_team_e was created using
create database 2019o2_team_e;

```
2019o1_team_a
2019o1_team_b
2019o1_team_c
2019o1_team_d
2019o1_team_e
2019o1_team_f
2019o1_team_g
2019o1_team_h
```

#2 use 2019o2_team_e;

#3

- To create an external table for sentiment_dictionary we used the below query

*create external table sentiment_dictionary (type string, length int, word string, word_type string, stemmed string, polarity string)row format delimited fields terminated by '\t' collection items terminated by',' stored as textfile location '/user/cyril.gebara/hive/sentiment_dictionary';*

- To load the data of dictionary.tsv into sentiment_dictionary, we used

*load data local inpath '/data/home/cyril.gebara/tweets/dictionary.tsv' into table sentiment_dictionary;*

#4

- To create an external table for tweets_json we used the below query, keep in mind that the hashtag entity and place struct were included later in the managed table tweets_raw_json since we will not use them now

*create external table tweets_json(created_at string, id_str string, text string, truncated boolean,*
*`user`*
*struct<id_str:string,screen_name:string,description:string,followers_count:int,utc_offset:int,time_zone:string,geo_enabled:boolean,lang:string>,*
*coordinates struct <coordinates:array<float>>,lang string) row format serde 'org.apache.hive.hcatalog.data.JsonSerDe' stored as textfile location '/user/cyril.gebara/hive/tweets_json';*

- To load the data of tweets.1 into tweets_json, we used

*load data local inpath '/data/home/cyril.gebara/tweets/tweets.1.json' into table tweets_json;*

We used the above query to load all tweets (e.g: tweets.2.json, tweets.3.json…)

# 5 Below is a query that returns the total number of tweets in table tweets_json

Select count(*) from tweets_json;

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 11.29 sec   HDFS Read: 52968008 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 11 seconds 290 msec
OK
count
9639
Time taken: 22.391 seconds, Fetched: 1 row(s)
```

#6 Below is a query that creates a managed table tweets_orc with same schema as tweets_json but stored in orc format

*create table tweets_orc like tweets_json stored as orc;*

*#7* Below is a query that inserts all rows of tweets_json into tweets_orc

*Insert into tweets_orc select * from tweets_json;*

*#8* Below is a query that returns the total number of tweets in table tweets_orc

```
hive> select count(*) as count from tweets_orc;
OK
count
9639
Time taken: 0.032 seconds, Fetched: 1 row(s)
```

#9 The Time difference (#5,#8) of running the queries is =22.391-0.032= 22.359 seconds which means that orc format tables give much faster results than json format tables, as we can see the number of tweets are the same for both (9639)

#10 Below is a query that returns the total number of users with geolocation enabled from table tweets_orc.

*select count( `user`.geo_enabled) from tweets_orc where `user`.geo_enabled=true;*

```
1828
Time taken: 21.575 seconds,
```

#11 Below is  a query that returns the total number of tweets per language from table tweets_orc (followed by an example).

*select count(*) as count, lang  from tweets_orc group by lang;*

| count | lang |
|---|---|
| 10 | ar |
| 5 | ca |
| 5 | cs |
| 5 | cy |
| 12 | da |
| 59 | de |
| 8 | el |
| 8124 | en |
| 310 | es |
| 8 | et |
| 3 | eu |
| 1 | fi |
| 73 | fr |
| 10 | ht |
| 2 | hu |
| 33 | in |
| 1 | is |
| 36 | it |
| 144 | ja |
| 2 | ko |
| 3 | lt |
| 1 | lv |
| 22 | nl |
| 1 | no |
| 11 | pl |
| 99 | pt |
| 4 | ro |
| 12 | ru |
| 1 | sl |
| 3 | sv |
| 8 | th |
| 10 | tl |
| 163 | tr |
| 437 | und |
| 8 | vi |
| 5 | zh |

#12 Below is a query that returns the top 10 users with the most number of followers from table tweets_orc.

*select `user`.screen_name as Name, max(`user`.followers_count) as Followers  from tweets_orc group by `user`.screen_name order by Followers DESC limit 10;*

| name | followers |
|---|---|
| CNBC | 3320803 |
| em_com | 507521 |
| PureMichigan | 497994 |
| RichSimmondsZA | 485857 |
| Cointelegraph | 484874 |
| Vegas | 461802 |
| APompliano | 296365 |
| MichaelCMcKee | 288249 |
| Porn_Tubbe | 283785 |
| JoshRoomsburg | 270664 |

#13

- To create an external table for geonames we used the below query:

*create external table geonames(id bigint,name string,ascii_name string,alternate_names array<string>,latitude float,longitude float,feature_class string,feature_code string,country_code string,country_code2 array<string>,admin1_code string,admin2_code string,admin3_code string,admin4_code string,population bigint,elevation int,dem int,timezone string,modification_date date)row format delimited fields terminated by '\t' collection items terminated by ',' stored as textfile location '/user/cyril.gebara/hive/geonames';*

- To load the data of cities15000.txt into geonames, we used:

*load data local inpath '/data/home/cyril.gebara/cities15000.txt' into table geonames;*

- We created a new table tweets_raw_json in order to deal with the remaining questions: adding hashtag entity and place struct, and removing attributes that are not needed anymore.

```
create external table tweets_raw_json(entities
struct<hashtags:array<struct<text:string>>>,id_str string, text
string, place struct <country:string, full_name:string>) row format
serde 'org.apache.hive.hcatalog.data.JsonSerDe' stored as textfile
location '/user/cyril.gebara/hive/tweets_raw_json';
```

- As we have done before with tweets_orc, we created a managed table for tweets_raw_json which is called tweets_raw_orc and inserted values.

```
create table tweets_raw_orc like tweets_raw_json stored as orc

Insert into tweets_raw_orc select * from tweets_raw_json
```

- In order to join geonames with tweets_raw_orc, we wrote the below query that returns the geoname latitude, longitude and timezone of the tweet's country.

```
Select a.latitude, longitude, timezone from geonames as a inner join
tweets_raw_orc as b on upper(a.name)=upper(b.place.country);
```

#14

- We created a table called words(1) that includes only the id and the tweet's text from tweets_orc, then we splitted the text into words by creating a new table tweet_words(2) as shown below.

```
(1)create table words as select id_str as id, text as word from
tweets_orc;
(2)create table tweet_words as select id as id, splittedword from
words lateral view explode(split(lower(word), ' ') text_ex as
splittedword;

And this is how (2) looks like:
Select * from tweet_words;
```

```
1201223045634879488     box
1201223045634879488     or
1201223045634879488     a
1201223045634879488     video
1201223045634879488     game
1201223045634879488     or
1201223045634879488     a
1201223045634879488     book!
1201223053197352961     https://t.co/SOWnpiRMvh
Time taken: 0.029 seconds, Fetched: 129384 row(s)
```

- Below is a query that returns the total count of words in tweets:

```
hive> select count(*) as count from tweet_words;
OK
count
129384
```

- Below is a query that returns the distinct count of words in tweets:

*select count(distinct(splittedword)) as Distinctwords from tweet_words;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.65 sec   HDFS Read: 3527125 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 650 msec
OK
distinctwords
26013
Time taken: 21.598 seconds, Fetched: 1 row(s)
```

- To calculate the max number of hashtags, min number of hashtags, average number of hashtags, we created two tables, (1) hashtags that returns the ids and the full hashtag entity that are related to each tweet (as seen below )

(1)create table hashtags as select id_str as id, entities.hashtags.text as words from tweets_raw_orc;

```
1201223023933710338     ["racing","competition","NFT","cars"]
1201223024189546497     []
1201223025510830080     []
1201223031294697475     []
1201223035342249985     ["bitcoin"]
1201223036592103424     ["TREOS","marketplace","buy","TREOSMARKET"]
1201223045634879488     []
1201223053197352961     []
Time taken: 0.043 seconds, Fetched: 9639 row(s)
```

(2) Since we have many hashtags per tweet, we created the hashtag_word table that explodes the hashtag entity.

(2)*create table hashtag_word as select id as id, hashtag from hashtags lateral view explode(words) w as hashtag;*

```
1201223023933710338     racing
1201223023933710338     competition
1201223023933710338     NFT
1201223023933710338     cars
1201223035342249985     bitcoin
1201223036592103424     TREOS
1201223036592103424     marketplace
1201223036592103424     buy
1201223036592103424     TREOSMARKET
Time taken: 0.032 seconds, Fetched: 10045 row(s)
```

- To calculate the occurrences of the hashtag that was used the most (the most used hashtag), we wrote this query followed by its output:

*select hashtag, count(hashtag) as max from hashtag_word group by hashtag order by max DESC limit 1;*

```
MapReduce Total cumulative CPU time: 6 seconds 110 msec
Ended Job = job_1575415580694_1058
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.92 sec   HDFS Read: 294213 HDFS Write: 51590 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 6.11 sec   HDFS Read: 56890 HDFS Write: 12 SUCCESS
Total MapReduce CPU Time Spent: 13 seconds 30 msec
OK
hashtag max
Bitcoin 888
Time taken: 40.677 seconds, Fetched: 1 row(s)
```

- To calculate the occurrences of the hashtag that was used the least, we wrote this query followed by its output:

*select hashtag, count(hashtag) as min from hashtag_word group by hashtag order by min ASC limit 1;*

```
Total MapReduce CPU Time Spent: 14 seconds 610 msec
OK
hashtag min
16Days  1
Time taken: 44.536 seconds, Fetched: 1 row(s)
```

- To calculate the average number of occurrences for each hashtag (how many times on avg a hashtag is used), the reasoning behind this query was based on the fact that the total number of hashtags (not distinct) divided by the actual 'population' of hashtags (distinct) is the actual avg we needed. Below is the query followed by the output:

*select (count(*)/count(distinct id)) as avg from hashtag_word;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.43 sec   HDFS Read: 296125 HDFS Write: 17 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 430 msec
OK
avg
2.83197067944742
Time taken: 23.755 seconds, Fetched: 1 row(s)
```

- To calculate the standard deviation and the percentiles of the quantity of words in each tweet, we first created the following view that can permit us to operate the stddev and percentile functions:

*create view wordstweet_count as select count(UPPER(splittedword)) as count, id from tweet_words group by id;*

```
6       1201233430547222528
13      1201233431021142017
21      1201233434280124416
10      1201233434498289665
11      1201233438252109824
18      1201233440433221635
15      1201233441959895045
10      1201233443968954369
10      1201233450935758848
17      1201233452768645120
9       1201233464776900609
Time taken: 21.347 seconds, Fetched: 9638 row(s)
```

- To calculate the standard deviation, we used the below query followed by the output:

*select stddev(count) as std from wordstweet_count;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 8.47 sec   HDFS Read: 3527578 HDFS Write: 137 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 6.19 sec   HDFS Read: 6154 HDFS Write: 18 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 660 msec
OK
std
7.053984757370833
Time taken: 41.541 seconds, Fetched: 1 row(s)
```

- To calculate the 25th percentile of words in tweets, we used the below query followed by:

*select percentile(count,0.25) as perc025 from wordstweet_count;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.28 sec   HDFS Read: 3527915 HDFS Write: 256 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 7.15 sec   HDFS Read: 7025 HDFS Write: 4 SUCCESS
Total MapReduce CPU Time Spent: 17 seconds 430 msec
OK
perc025
7.0
Time taken: 45.085 seconds, Fetched: 1 row(s)
```

- To calculate the 50th percentile of words in tweets, we used the below query followed by:

*select percentile(count,0.5) as perc05 from wordstweet_count;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.49 sec   HDFS Read: 3527914 HDFS Write: 256 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 8.85 sec   HDFS Read: 7020 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 19 seconds 340 msec
OK
perc05
13.0
Time taken: 42.596 seconds, Fetched: 1 row(s)
```

- To calculate the 75th percentile of words in tweets, we used the below query followed by:

*select percentile(count,0.75) as perc075 from wordstweet_count;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.41 sec   HDFS Read: 3527915 HDFS Write: 256 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 6.19 sec   HDFS Read: 7025 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 16 seconds 600 msec
OK
perc075
19.0
Time taken: 41.874 seconds, Fetched: 1 row(s)
```

- To calculate the 100th percentile of words in tweets, we used the below query followed by:

*select percentile(count,1) as perc1 from wordstweet_count;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 10.01 sec   HDFS Read: 3527904 HDFS Write: 256 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 7.6 sec   HDFS Read: 7010 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 17 seconds 610 msec
OK
perc1
47.0
Time taken: 42.705 seconds, Fetched: 1 row(s)
```

- To make sure that the above query is correct, we calculated the maximum number of words in a tweet which should match the 100th percentile output (count of 47):

*select max(count) as max from wordstweet_count;*

```
max
47
Time taken: 43.768 seconds, Fetched: 1 row(s)
```

#15 Below is a query that returns the top 10 words with at least 4 letters from table  tweets_raw_orc

*select upper(splittedword) as distinctword,count(upper(splittedword)) as count from tweet_words group by splittedword having length(splittedword) > 4 order by count DESC limit 10;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 13.56 sec   HDFS Read: 3527415 HDFS Write: 659310 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 7.96 sec   HDFS Read: 664458 HDFS Write: 138 SUCCESS
Total MapReduce CPU Time Spent: 21 seconds 520 msec
OK
distinctword    count
BITCOIN 1156
#BITCOIN        612
#BITCOIN        547
STELLAR 340
BITCOIN 318
@CRYPTOEASYMONE2:       268
&AMP;   238
#CRYPTO 209
ABOUT   205
@JULIADAVISNEWS:        160
Time taken: 43.588 seconds, Fetched: 10 row(s)
```

#16

- To create a new table tweet_words_parquet we used the below query:

*Create table tweet_words_parquet like tweet_words stored as parquet;*

**Keeping in mind that in question #14 (2) the *tweet_words* table was created by normalizing the words to lower cases and using lateral view, we now create the new table based on this one in order to store it as a parquet.**

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 6.61 sec   HDFS Read: 3523688 HDFS Write: 889258 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 610 msec
OK
tweet_words.id  tweet_words.splittedword
Time taken: 16.72 seconds
```

*Insert into tweet_words_parquet select * from tweet_words;*

```
1201223045634879488     video
1201223045634879488     game
1201223045634879488     or
1201223045634879488     a
1201223045634879488     book!
1201223053197352961     https://t.co/SOWnpiRMvh
Time taken: 0.033 seconds, Fetched: 129384 row(s)
```

#17

- To create a parquet table called tweet_words_sentiment, we first created a view of polarity from sentiment_dictionary (as seen in (1) below) that displays the words in upper case with their respective polarity(0,1,-1) which was then left joined to tweet_words_parquet from question 16 for the final creation of tweet_words_sentiment (as seen in (2) below).

*(1)create view polarity as select upper(word) as word, case when polarity = 'positive' then 1 when polarity = 'negative' then -1 when polarity = 'neutral' then 0 end as polaritynum from sentiment_dictionary;*

```
YEAH    0
YEARN   1
YEARNING        1
YEARNINGLY      1
YELP    -1
YEP     1
YES     1
YOUTHFUL        1
ZEAL    1
ZEALOT  -1
ZEALOUS -1
ZEALOUSLY       -1
ZENITH  1
ZEST    1
Time taken: 0.058 seconds, Fetched: 8221 row(s)
```

(2)Create table tweet_words_sentiment stored as parquet as select a.id as id, a.splittedword as word, Coalesce(b.polaritynum,0) as polarity from tweet_words_parquet as a left join polarity as b on upper(a.splittedword) = upper(b.word);

```
MapReduce Jobs Launched:
Stage-Stage-4: Map: 1   Cumulative CPU: 8.74 sec   HDFS Read: 897168 HDFS Write: 918344 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 740 msec
OK
id      word    polarity
Time taken: 22.2 seconds
1201223036592103424     just    1
1201223036592103424     just    1
1201223036592103424     around  0
1201223036592103424     the     0
1201223036592103424     corner...       0
1201223045634879488     RT      0
1201223045634879488     @PaulWHauser:   0
1201223045634879488     And     0
1201223045634879488     it's    0
1201223045634879488     NOT     0
1201223045634879488     based   0
1201223045634879488     off     0
1201223045634879488     a       0
1201223045634879488     cereal  0
1201223045634879488     box     0
1201223045634879488     or      0
1201223045634879488     a       0
1201223045634879488     video   0
1201223045634879488     game    -1
1201223045634879488     or      0
1201223045634879488     a       0
1201223045634879488     book!   0
1201223053197352961     https://t.co/SOWnpiRMvh 0
Time taken: 0.031 seconds, Fetched: 134318 row(s)
```

*# 18* To create a table tweets_sentiment(2 below), we first created a view sentiment_test(1 below) from tweets_words_sentiment (created in question #17(2))that provides the id and sum of polarities of each ID.

*(1)Create view sentiment_test as select id, SUM(polarity) as sum from tweet_words_sentiment group by id;*

```
1201233434498289665     1
1201233438252109824     0
1201233440433221635     0
1201233441959895045     5
1201233443968954369     -3
1201233450935758848     0
1201233452768645120     0
1201233464776900609     0
Time taken: 21.416 seconds, Fetched: 9638 row(s)
```

*(2)Create table tweets_sentiment stored as parquet as select id, sum, case when sum < 0 then 'negative' when sum > 0 then 'positive' when sum = 0 then 'neutral' end as polarity from sentiment_test;*

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 13.11 sec   HDFS Read: 308446 HDFS Write: 230914 SUCCESS
Total MapReduce CPU Time Spent: 13 seconds 110 msec
OK
id      sum     polarity
Time taken: 23.512 seconds
```

```
1201233419826552832     -1      negative
1201233423551094787     1       positive
1201233429855113217     2       positive
1201233430547222528     0       neutral
1201233431021142017     0       neutral
1201233434280124416     0       neutral
1201233434498289665     1       positive
1201233438252109824     0       neutral
1201233440433221635     0       neutral
1201233441959895045     5       positive
1201233443968954369     -3      negative
1201233450935758848     0       neutral
1201233452768645120     0       neutral
1201233464776900609     0       neutral
Time taken: 0.034 seconds, Fetched: 9638 row(s)
```

#19

- To write a query that returns the hourly evolution of sentiment of tweets with the hashtag LTC or Litecoin we first create a view table called hourhashtag (1) that includes only the hour of the created_at attribute, id, and polarity. Then we created another view distinct_hourhashtag that filters by distinct IDs because could have happened that one tweet contains both Litecoin and LTC at the same time (2)

*(1)Create view hourhashtag as select substr(a.created_at,12,2) as hour, a.id_str as id, b.polarity as polarity from (tweets_json as a inner join hashtag_word as c on a.id_str = c.id) inner join tweets_sentiment as b on a.id_str = b.id where c.hashtag = 'LTC' or c.hashtag = 'Litecoin';*

```
19      1201214582070677504     neutral
19      1201214582070677504     neutral
19      1201214667969966087     neutral
19      1201215256661516288     positive
19      1201215905004507138     neutral
19      1201217601747202048     neutral
19      1201219843418140672     neutral
19      1201221939265716232     neutral
19      1201221997935681538     neutral
19      1201221997935681538     neutral
19      1201222690343927809     neutral
Time taken: 22.148 seconds, Fetched: 114 row(s)
```

(2) *Create view distinct_hourhashtag as select distinct(id), polarity, hour from hourhashtag;*

As you can see from the below output, the ids are all distinct ( count is 98 rather than 114 from hourhashtag)

```
1201229915217154055     neutral 20
1201229924767584258     positive        20
1201230356571013120     neutral 20
1201230435747061762     positive        20
1201230703821635584     positive        20
1201230735320838144     positive        20
1201231211299033089     positive        20
1201231437896114177     positive        20
1201231520502972416     neutral 20
1201231795578011649     positive        20
1201232009533644800     neutral 20
1201232010741633025     positive        20
1201233337966374913     positive        20
1201233366487449600     neutral 20
Time taken: 29.592 seconds, Fetched: 98 row(s)
```

- Finally to get the hourly evolution of tweet's sentiment for tweets containing the hashtag LTC or Litecoin, we wrote the below query followed by the output, keeping in mind that the were only positive and neutral (no negatives) tweet's polarities with the presence of LTC and Litecoin hashtags.

```
select t1.hour as hour ,sum(t1.positive) as positives,sum(t2.neutral) as neutrals from ( select hour,count(*) as positive from distinct_hourhashtag where polarity='positive' group by hour) t1 join ( select hour,count(*) as neutral from distinct_hourhashtag where polarity='neutral' group by hour ) t2 ON t1.hour=t2.hour group by t1.hour;
```

```
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 13.14 sec   HDFS Read: 52975490 HDFS Write: 159 SUCCESS
Stage-Stage-9: Map: 1  Reduce: 1   Cumulative CPU: 12.24 sec   HDFS Read: 52975691 HDFS Write: 159 SUCCESS
Stage-Stage-3: Map: 1  Reduce: 1   Cumulative CPU: 6.33 sec   HDFS Read: 4505 HDFS Write: 159 SUCCESS
Stage-Stage-10: Map: 1  Reduce: 1   Cumulative CPU: 4.86 sec   HDFS Read: 4505 HDFS Write: 159 SUCCESS
Stage-Stage-13: Map: 1   Cumulative CPU: 2.7 sec   HDFS Read: 5037 HDFS Write: 162 SUCCESS
Stage-Stage-5: Map: 1  Reduce: 1   Cumulative CPU: 5.71 sec   HDFS Read: 6195 HDFS Write: 25 SUCCESS
Total MapReduce CPU Time Spent: 44 seconds 980 msec
OK
hour    positives       neutrals
18      4       12
19      43      19
20      11      9
Time taken: 142.766 seconds, Fetched: 3 row(s)
```

As you can notice, we disregarded year, month and day since all of the tweet dates are the same day 2019-12-01.

Since we were concerned of misunderstanding question 14, we decided to also provide you with the total count and distinct count of hashtags, just in case you meant by this. In order to do so, we used hashtag_word (#14,2).

```
select count(*) from hashtag_word;
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 8.41 sec   HDFS Read: 52244 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 8 seconds 410 msec
OK
_c0
10045
Time taken: 22.447 seconds, Fetched: 1 row(s)
```

```
select count(distinct(hashtag))as count from hashtag_word;
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 7.36 sec   HDFS Read: 294900 HDFS Write: 5 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 360 msec
OK
count
1871
Time taken: 20.991 seconds, Fetched: 1 row(s)
```

```
        MapReduce Total cumulative CPU time: 6 seconds 110 msec
        Ended Job = job_1575415580694_1058
        MapReduce Jobs Launched:
        Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.92 sec   HDFS Read: 294213 HDFS Write: 51590 SUCCESS
        Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 6.11 sec   HDFS Read: 56890 HDFS Write: 12 SUCCESS
        Total MapReduce CPU Time Spent: 13 seconds 30 msec
        OK
        hashtag max
        Bitcoin 888
        Time taken: 40.677 seconds, Fetched: 1 row(s)
```

```
select (count(*)/count(distinct id)) as avg from hashtag_word;
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.43 sec   HDFS Read: 296125 HDFS Write: 17 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 430 msec
OK
avg
2.83197067944742
Time taken: 23.755 seconds, Fetched: 1 row(s)
```