HADOOP Assignement

Masters' in Business Analytics & Big Data

Apache Hive

Group F

Mariana Narvaez Cubillos, Nisrine Ferahi, Jonas Hellevang, Daniel Bilitewski,
Rabiga Shangereyeva, Guillermo Chacon Clericus

Prof. Jorge Centeno

1. **Create a database named 2019o1_team_f.**

   CREATE DATABASE 2019o1_team_f;

2. **Select the database you just created so that all the tables you are going to create belong to that database.**

   USE 2019o1_team_f;

3. **Create an external table named sentiment_dictionary with the files provided.**

   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'
   STORED AS TEXTFILE
   LOCATION '/user/jonas.hellevang/hive/sentiment_dictionary'

   LOAD DATA LOCAL INPATH '/data/home/jonas.hellevang/group-assignment-resources/sentiment_dictionary/dictionary.tsv' into table sentiment_dictionary;

4. Create an external table named tweets_json with the files provided.

```
CREATE EXTERNAL TABLE TWEETS_JSON( CREATED_AT STRING,
ID BIGINT,
ID_STR STRING,
TEXT STRING,
SOURCE STRING,
TRUNCATED BOOLEAN,
IN_REPLY_TO_STATUS_ID BIGINT,
IN_REPLY_TO_STATUS_ID_STR STRING,
IN_REPLY_TO_USER_ID BIGINT,
IN_REPLY_TO_USER_ID_STR STRING,
IN_REPLY_TO_SCREEN_NAME STRING,
ENTITIES STRUCT<HASHTAGS: ARRAY <STRUCT<TEXT:STRING>>>,
`USER`
STRUCT<ID:BIGINT,ID_STR:STRING,NAME:STRING,SCREEN_NAME:STRING,LOCATION:STRING
,URL:STRING,DESCRIPTION:STRING,PROTECTED:BOOLEAN,VERIFIED:BOOLEAN,FOLLOWERS_
COUNT:INT,FRIENDS_COUNT:INT,LISTED_COUNT:INT,FAVOURITES_COUNT:INT,STATUSES_C
OUNT:INT,UTC_OFFSET:INT,TIME_ZONE:STRING,GEO_ENABLED:BOOLEAN,LANG:STRING>,
COORDINATES STRUCT <COORDINATES:ARRAY<FLOAT>>,
PLACE
STRUCT<COUNTRY:STRING,COUNTRY_CODE:STRING,FULL_NAME:STRING,NAME:STRING,PLA
CE_TYPE:STRING,URL:STRING>, QUOTED_STATUS_ID BIGINT,
QUOTED_STATUS_ID_STR STRING,
IS_QUOTE_STATUS BOOLEAN,
QUOTE_COUNT INT,
REPLY_COUNT INT,
RETWEET_COUNT INT,
FAVORITE_COUNT INT,
FAVORITED BOOLEAN,
RETWEETED BOOLEAN,
POSSIBLY_SENSITIVE BOOLEAN,
FILTER_LEVEL STRING,
LANG STRING
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION '/user/jonas.hellevang/hive/tweets_json';
LOAD DATA LOCAL INPATH '/data/home/jonas.hellevang/group-assignment-
resources/crypto-tweets' into table tweets_json;
```

5. Write a query that returns the total number of tweets in table tweets_json. Annotate both the number of records and the amount of seconds that it took.

```
SELECT COUNT(*) AS total FROM tweets_json;

OK
total
9639
Time taken: 22.404 seconds, Fetched: 1 row(s)
```

6. **Create 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. **Insert all rows from tweets_json into tweets_orc.**

   INSERT INTO TABLE tweets_orc SELECT * FROM tweets_json;

8. **Write a query that returns the total number of tweets in table tweets_orc. Annotate both the number of records and the amount of seconds that it took.**

   SELECT COUNT(*) as total FROM tweets_orc;

   OK
   total
   9639
   Time taken: 0.046 seconds, Fetched: 1 row(s)

9. **Verify that both tables contain the same number of tweets. which of the queries was faster?**

   Both tables have 9639 rows, but query number 8 for table tweets_orc was faster than the query for table tweets_json.

10. **Write a query that returns the total number of users with geolocation enabled from table tweets_orc.**

    SELECT COUNT(*) AS count FROM tweets_orc WHERE `user`.geo_enabled = TRUE

    OK
    count
    1828

11. Write a query that returns the total number of tweets per language from table tweets_orc.

```
SELECT COUNT(*) AS count, lang FROM tweets_orc GROUP BY lang
```

OK

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. Write a query that returns the top 10 users with more tweets published from table tweets_orc.

```
SELECT `user`.name, COUNT(*) AS count_tweets FROM tweets_orc GROUP BY `user`.name
ORDER BY count_tweets DESC LIMIT 10

OK
user.name       count_tweets
Crypto_easymoney        268
Crypto Airdrops 93
CryptoVols              85
Automatski              83
Jonathan Pipeston       63
CoinCaps.ai             43
Ruth Diaz               39
BetFast.com             36
datajobsdata            35
Isak-Erik               31
```

13. Write a query that returns the geoname latitude, longitude and timezone of the tweet place by joining geonames and tweets_orc.

First, we created the table geonames:

```
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/jonas.hellevang/hive/geonames';

LOAD DATA LOCAL INPATH '/data/home/jonas.hellevang/cities15000.txt' into table
geonames;

SELECT DISTINCT o.id, g.latitude, g.longitude, g.timezone, o.place.name, g.name FROM
tweets_orc o LEFT JOIN geonames g  ON UPPER(o.place.country_code) =
UPPER(g.country_code) WHERE g.latitude IS NOT NULL AND ((UPPER(o.place.name) LIKE
UPPER(g.name)) OR (UPPER(g.name) LIKE UPPER(o.place.name)));
```

14. Write a query that returns the total count, total distinct count, maximum, minimum, average, standard deviation and percentiles 25th, 50th, 75th of hashtags in tweets from table tweets_orc:

To get the total number of hashtags in all tweets:

```
SELECT SUM(hashtags_per_tweet) AS total_hashtags
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;

OK
total_hashtags
10045
Time taken: 20.51 seconds, Fetched: 1 row(s)
```

Maximum:

```
SELECT MAX(hashtags_per_tweet) AS max_hashtags
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;

OK
max_hashtags
20
Time taken: 20.474 seconds, Fetched: 1 row(s)
```

Minimum:

```
SELECT MIN(hashtags_per_tweet) AS min_hashtags
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;

OK
min_hashtags
0
Time taken: 21.429 seconds, Fetched: 1 row(s)
```

Average:

```
SELECT AVG(hashtags_per_tweet) AS average_hashtags
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;

average_hashtags
1.0421205519244734
Time taken: 21.597 seconds, Fetched: 1 row(s)
```

Standard Deviation:

```
SELECT STDDEV (hashtags_per_tweet) AS SD_hashtags FROM
(SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;
```

```
sd_hashtags
1.9609979403555953
Time taken: 20.371 seconds, Fetched: 1 row(s)
```

Percentiles:

```
SELECT percentile (hashtags_per_tweet, 0.25) AS percentile
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;
```

```
SELECT percentile (hashtags_per_tweet, 0.5) AS percentile
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;
```

```
SELECT percentile (hashtags_per_tweet, 0.75) AS percentile
FROM (SELECT (SIZE(entities.hashtags)) AS hashtags_per_tweet
FROM tweets_orc
)tweets_orc;
```

In order to count the distinct hashtags in the tweet we had to use a table created for question 15 to get an array of hashtags per tweet.

Number of different hashtags:

```
SELECT COUNT(DISTINCT hashtag) FROM hashtag_word;
```

```
OK
number_of_unique_hashtags
1871
Time taken: 20.611 seconds, Fetched: 1 row(s)
```

15. Write a query that returns the top 10 more popular hashtags from table tweets_orc:

1. Create table with only array of hashtags per tweet:

```
CREATE TABLE hashtags AS
SELECT id AS id, entities.hashtags.text AS words
FROM tweets_json;
```

2. Create a new table to extract each hashtag in a new row. So, we need to split each word inside the array as a new row:

```
CREATE TABLE hashtag_word AS
SELECT id AS id, hashtag
FROM hashtags LATERAL VIEW explode (words) w AS hashtag;
```

3. Select top 10 more popular hashtags

```
SELECT UPPER(hashtag), COUNT (hashtag) AS total_count
FROM hashtag_word
GROUP BY UPPER(hashtag)
ORDER BY total_count DESC LIMIT 10;
```

16. Create a table tweet_words in parquet format exploding the words in the tweets. Also normalize the words to lower case:

```
CREATE TABLE tweets_parquet_temp STORED AS parquet AS SELECT * FROM tweets_json;

CREATE TABLE tweet_words_temp AS SELECT id AS id, text AS text FROM
tweets_parquet_temp;

CREATE TABLE split_words AS SELECT id AS id, SPLIT(text,' ') AS words FROM
tweet_words_temp;

CREATE TABLE tweet_words AS SELECT id AS id, word FROM split_words LATERAL VIEW
EXPLODE(words) w AS word;

SELECT * FROM tweet_words LIMIT 200;
```

17. Create a table tweet_words_sentiment in parquet format as the result of a query that returns the polarity of each word by left joining tweet_words with sentiment_dictionary. The polarity for nonjoining words will be neutral(you can use coalesce function). Also codify the polarity:

When we select the DISTINCT polarity in sentiment dictionary table, we noticed that the data is not cleaned because we don't only have positive, negative, neutral. We also have m, both, y, n

We decided to replace y with positive, n with negative, both and m with neutral:

```
CREATE TABLE tweet_words_sentiment_temp STORED AS parquet AS
SELECT t.id AS id, t.word AS word,  COALESCE(s.polarity, 'neutral') AS polarity
FROM tweet_words t LEFT OUTER JOIN sentiment_dictionary s ON (UPPER(s.word) =
UPPER(t.word));

CREATE TABLE tweet_words_sentiment AS SELECT id AS id, word AS word,
REGEXP_REPLACE(REGEXP_REPLACE(REGEXP_REPLACE(REGEXP_REPLACE(REGEXP_REPLACE
(REGEXP_REPLACE(REGEXP_REPLACE (polarity, 'positive', '1'), 'negative','-1'), 'neutral','0'),
'both','0'), 'm','0'), 'y','1'), 'n','-1')AS polarity FROM tweet_words_sentiment_temp;
```

18. Create a table tweets_sentiment in parquet format as the result of a query that sums the polarity of every tweet so that:

sum(polarity) > 0 -> 'positive'
sum(polarity) < 0 -> 'negative'
sum(polarity) = 0 -> 'neutral'

```
CREATE TABLE tweets_sentiment STORED AS parquet AS
SELECT id AS id,
  CASE
  WHEN SUM(polarity) > 0 THEN 'positive'
  WHEN SUM(polarity) < 0 THEN 'negative'
  WHEN SUM(polarity) = 0 THEN 'neutral'
  END AS polarity FROM tweet_words_sentiment GROUP BY id;
```

19. Write a query that returns the hourly evolution of sentiment of tweets with hashtag EOS:

```
SELECT hour, SUM(positive) AS positive, SUM(negative) AS negative
FROM (SELECT CONCAT(SUBSTRING(created_at, 27,4),
  CASE
  WHEN SUBSTRING(o.created_at,5,3) = 'Jan' then '1'
  WHEN SUBSTRING(o.created_at,5,3) = 'Feb' then '2'
  WHEN SUBSTRING(o.created_at,5,3) = 'Mar' then '3'
  WHEN SUBSTRING(o.created_at,5,3) = 'Apr' then '4'
  WHEN SUBSTRING(o.created_at,5,3) = 'May' then '5'
  WHEN SUBSTRING(o.created_at,5,3) = 'Jun' then '6'
  WHEN SUBSTRING(o.created_at,5,3) = 'Jul' then '7'
  WHEN SUBSTRING(o.created_at,5,3) = 'Aug' then '8'
  WHEN SUBSTRING(o.created_at,5,3) = 'Sep' then '9'
  WHEN SUBSTRING(o.created_at,5,3) = 'Oct' then '10'
  WHEN SUBSTRING(o.created_at,5,3) = 'Nov' then '11'
  WHEN SUBSTRING(o.created_at,5,3) = 'Dec' then '12'
  END,SUBSTRING(o.created_at,9,2),SUBSTRING(created_at,12,2)) AS hour,
  CASE
  WHEN polarity = 'positive' then 1
  ELSE 0
  END AS positive,
  CASE
  WHEN polarity = 'negative' then 1
  ELSE 0
  END AS negative
  FROM tweets_sentiment s INNER JOIN tweets_orc o ON s.id = o.id INNER JOIN
  hashtag_word h ON o.id = h.id WHERE h.hashtag = 'EOS') AS subquery GROUP BY hour;
```

```
OK
hour            positive        negative
2019120118      2               0
2019120119      4               0
2019120120      2               0
```