PB

Principles of Big Data Management Phase-2

Fall 2016

Submitted by: (Team 11)

Moulika Chadalavada (16234180)
Sampath Gattu (16231927)
Lakshmi Nikitha Kona (16231555)
Prathyusha P (16231926)

Table of Contents

1. Introduction

1.1 About Twitter

Twitter is a massive social networking service that enables users to send data read short 140 character messages called ‘tweets’. More than 140 million active users publish over 400 million tweets every day. Twitter was created in March 2006 by Jack Dorsey, Evan Williams, Biz Stone and Noah Glass and launched in July 2006. Twitters speed and ease of publication have made it an important communication medium for people from all walks of life. This stream of messages from a variety of users contains information on an array of topics, including conventional new stories, events of local interest, opinions, real-time events. Twitter APIs provide access to tweets from a time range, from a user, with a keyword.

1.2 Proposed Project

We choose ‘Diseases’ as our topic to do big data analysis. Based on twitter tweets, we predicted some interesting analysis on Diseases using thousands of tweets tweeted by different people. First we collected the tweets from twitter API based on some key words related to Disease. After that, we analyzed the data that we have collected. By using the analysis, we written some interesting SQL queries useful to give a proper result for the analysis.

Here we used Spark to processing the twitter data. Because it has many advantages like

- Speed: Run Programs up to 100x faster than Hadoop Map reduce in memory.
- Ease of Use: Write applications quickly in Java, Scala, Python, R.
- Generality: Combine SQL, streaming, and complex analytics.
- Runs Everywhere: Spark runs on Hadoop, Mesos, standalone, or in the cloud

2. System Requirements

2.1 Software Requirements

- Python 3.5
- JDK 1.8
- Scala 2.11.8
- Intellij IDEA
- Apache Spark

2.2 Language Requirements

- Python
- Scala
- SQL

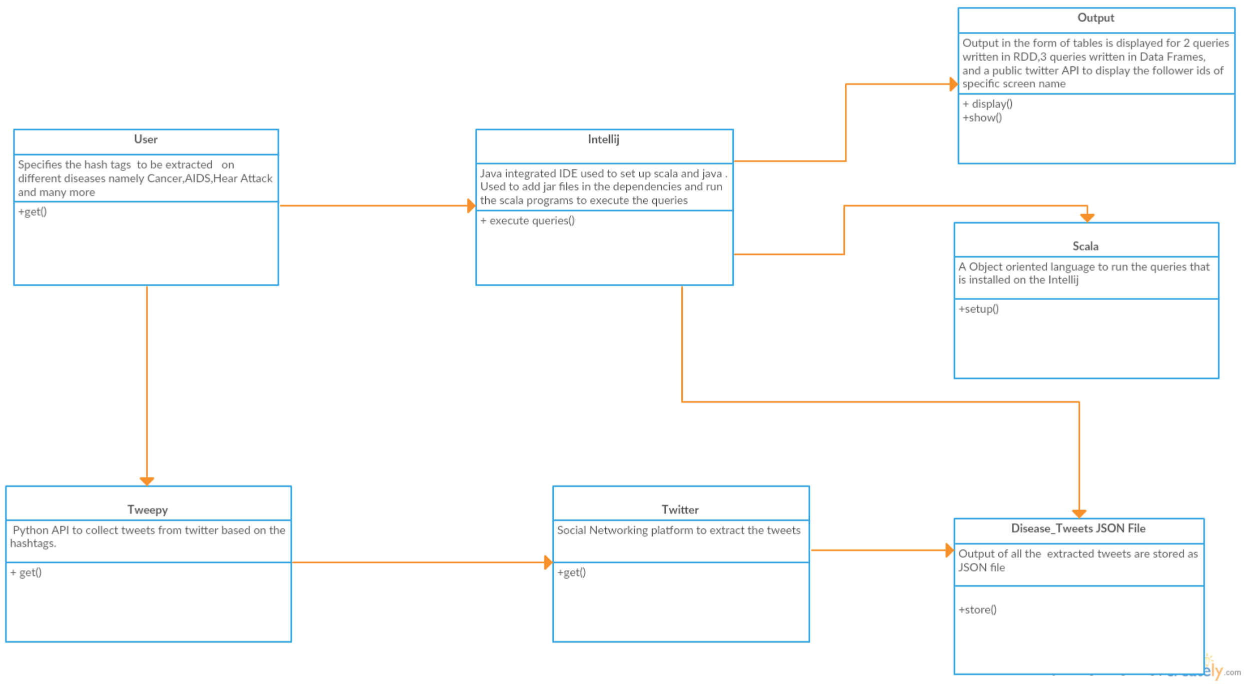
3. About the Theme

We have selected the theme diseases, which provides the analysis on the twitter tweets based on the queries. Initially tweets were collected using hashtags related to disease types such as Heart Stroke, Cancer, Chicken Pox etc. Using this data, the queries are written in Scala which gives analysis on which disease more tweets were done, which user made more number of tweets on which disease, which state people in USA are more involved in tweeting about the diseases, top languages using which tweets were posted. The extracted data is displayed in tables. Based on this data visualization is done in next phase.

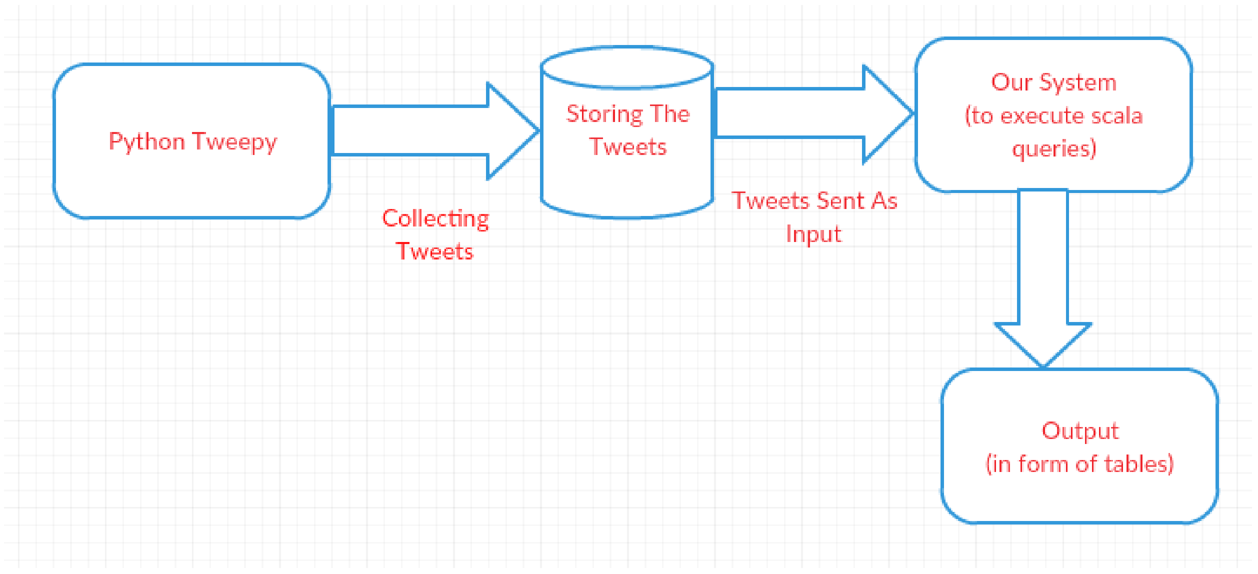
4. System Architecture

First we generated credential for accessing twitter. By using these credentials, we wrote a python program to collect twitter tweets based on keywords related to food. Tweets were stored in a text file in a JSON format. We will give these JSON file to SQL queries for analysis with Spark, Intellij with Scala program with queries.

UML Diagram


Output
Output in the form of tables is displayed for 2 queries written in RDD,3 queries written in Data Frames, and a public twitter API to display the follower ids of specific screen name
+ display()
+show()
User
Specifies the hash tags to be extracted on different diseases namely Cancer,AIDS,Hear Attack and many more
+get()
Intellij
Java integrated IDE used to set up scala and java . Used to add jar files in the dependencies and run the scala programs to execute the queries
+ execute queries()
Scala
A Object oriented language to run the queries that is installed on the Intellij
+setup()
Tweepy
Python API to collect tweets from twitter based on the hashtags.
+ get()
Twitter
Social Networking platform to extract the tweets
+get()
Disease_Tweets JSON File
Output of all the extracted tweets are stored as JSON file
+store()

System Architecture



5. Collecting Twitter Tweets

5.1 Generating Access Tokens

First we generated keys for accessing twitter API. For this we need to register our application by using, http://apps.twitter.com. We generated access token, access token secret, consumer key, consumer secret

ACCESS_TOKEN = "1974127951-N1QRXNCsVCywXl67BU1Wx7VlJ1fw2TlScZDY07s"
ACCESS_SECRET = "zLLfZ4ZF9BxvgzjXAkjJFAVFOL4P1i0fLSaZzc6LrnqZJ"
CONSUMER_KEY = "RfqrQLUtEAvwqSmNpGjZydmOn"
CONSUMER_SECRET = "1XdUvxMDfcL5eE89oAo7ZO8UESv6H7HacNa9B0Y7cGFa5MiPjo"

5.2 Streaming Twitter Tweets

After that we have written a python program for streaming twitter tweets. As our theme is related to 'Diseases' we used few hashtags such heartstroke,cancer,malaria,braintumour,chickenpox etc. From twitter, we have streamed almost 100000 instances. Tweets were stored in JSON (JavaScript Object Notation) format in a file.

Tweets Source Code: https://github.com/cmoulika009/Principles-of-Big-Data-Management/blob/master/PB%20Phase-2-%20Team%2011/Source%20Code/diseases.py

6. Analyzing Twitter Data

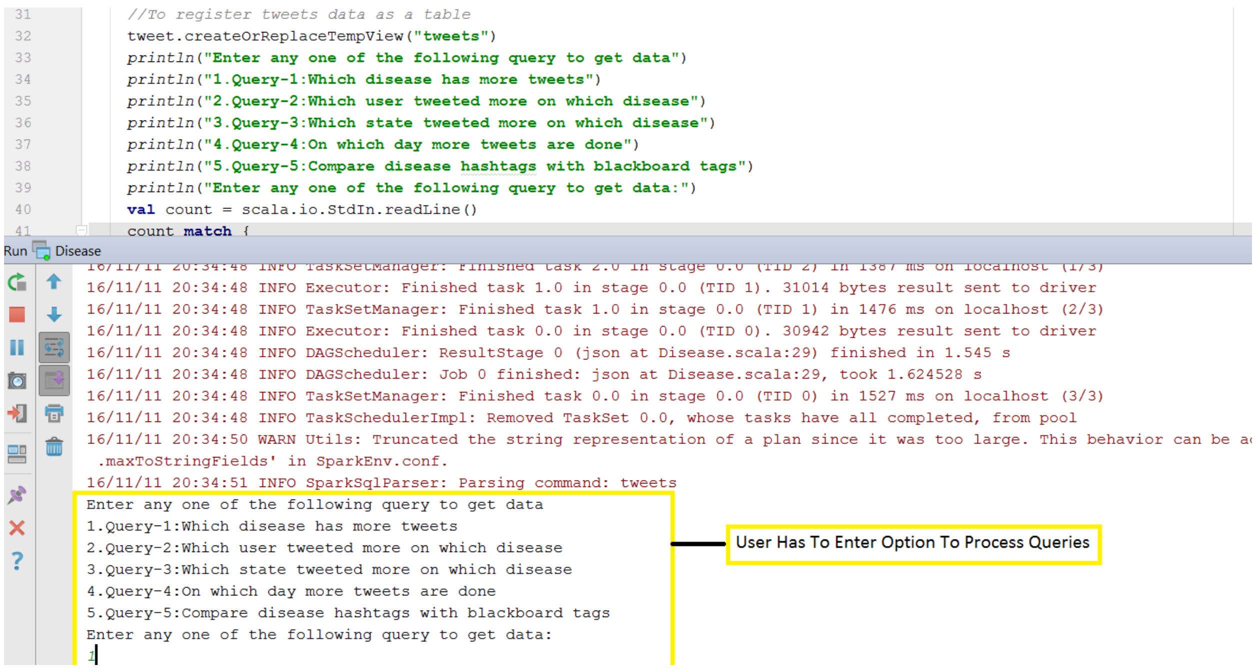
Queries Scala Source Code: https://github.com/cmoulika009/Principles-of-Big-Data-Management/tree/master/PB%20Phase-2-%20Team%2011/Source%20Code/PBPhase-2/src

Initially the twitter tweets json file is read in Scala and stored in temporary table.

```
val tweet = sqlContext.read.json
("C:\\Users\\nikky\\Desktop\\pbproject\\Disease_Tweets.json")

tweet.createOrReplaceTempView("tweets")
```

Using this temporary tweets data is analyzed by writing queries in Scala.

Switch Case is written to select the Query so that one query executed at a time as shown below. For example, if 1 is entered then Query-1 is executed.

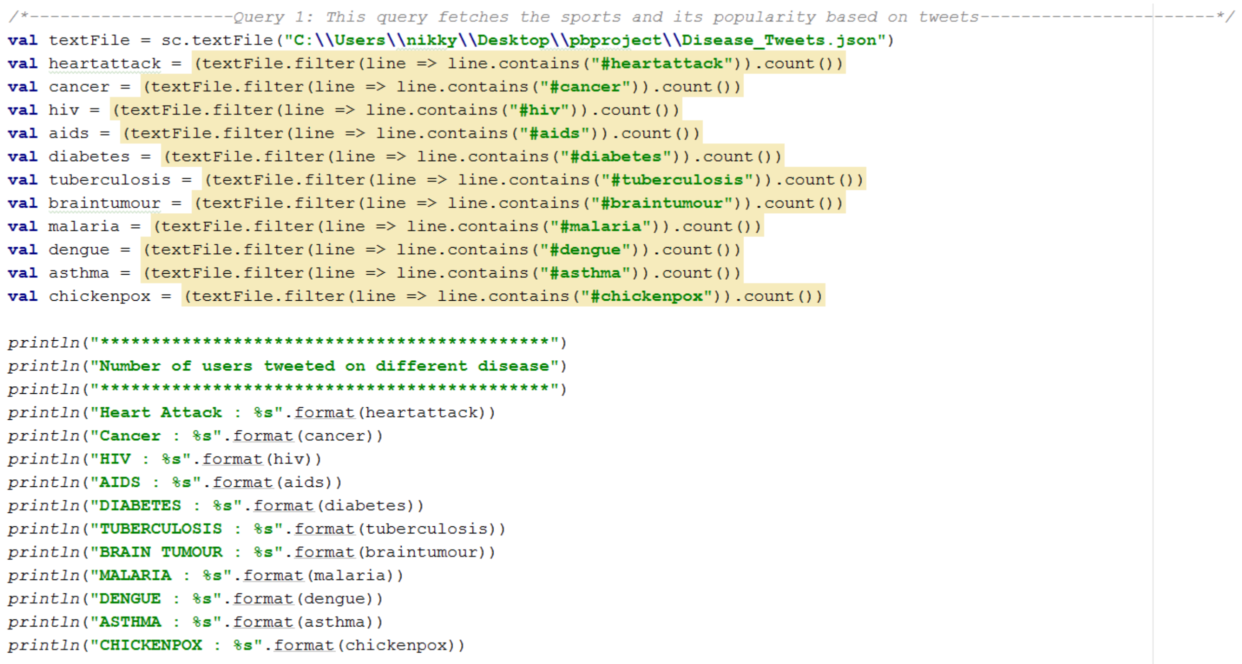


```
//To register tweets data as a table
tweet.createOrReplaceTempView("tweets")
println("Enter any one of the following query to get data")
println("1.Query-1:Which disease has more tweets")
println("2.Query-2:Which user tweeted more on which disease")
println("3.Query-3:Which state tweeted more on which disease")
println("4.Query-4:On which day more tweets are done")
println("5.Query-5:Compare disease hashtags with blackboard tags")
println("Enter any one of the following query to get data:")
val count = scala.io.StdIn.readLine()
count match {
```

```
16/11/11 20:34:48 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 1387 ms on localhost (1/3)
16/11/11 20:34:48 INFO Executor: Finished task 1.0 in stage 0.0 (TID 1). 31014 bytes result sent to driver
16/11/11 20:34:48 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 1476 ms on localhost (2/3)
16/11/11 20:34:48 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 30942 bytes result sent to driver
16/11/11 20:34:48 INFO DAGScheduler: ResultStage 0 (json at Disease.scala:29) finished in 1.545 s
16/11/11 20:34:48 INFO DAGScheduler: Job 0 finished: json at Disease.scala:29, took 1.624528 s
16/11/11 20:34:48 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 1527 ms on localhost (3/3)
16/11/11 20:34:48 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/11/11 20:34:50 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be a
 .maxToStringFields' in SparkEnv.conf.
16/11/11 20:34:51 INFO SparkSqlParser: Parsing command: tweets
Enter any one of the following query to get data
1.Query-1:Which disease has more tweets
2.Query-2:Which user tweeted more on which disease
3.Query-3:Which state tweeted more on which disease
4.Query-4:On which day more tweets are done
5.Query-5:Compare disease hashtags with blackboard tags
Enter any one of the following query to get data:
1
```

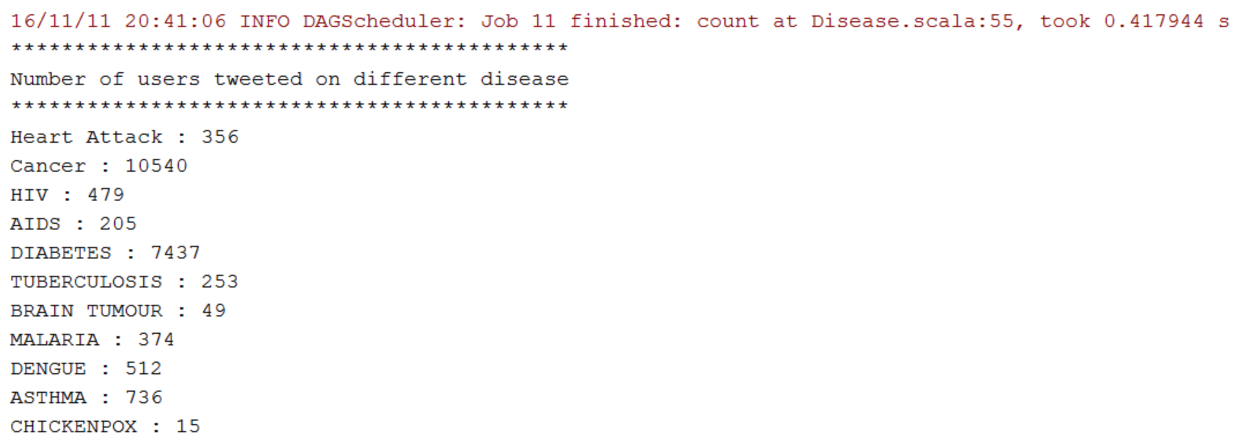
6.1 RDD Query 1

As we collected tweets on various diseases, in the first query we are fetching the diseases and its tweets count in the file. This query is written using RDD, where we are fetching the count of diseases using hashtags using **filter** and the count is printed further.



```
/*--------------------Query 1: This query fetches the sports and its popularity based on tweets-----------------------*/
val textFile = sc.textFile("C:\\Users\\nikky\\Desktop\\pbproject\\Disease_Tweets.json")
val heartattack = (textFile.filter(line => line.contains("#heartattack")).count())
val cancer = (textFile.filter(line => line.contains("#cancer")).count())
val hiv = (textFile.filter(line => line.contains("#hiv")).count())
val aids = (textFile.filter(line => line.contains("#aids")).count())
val diabetes = (textFile.filter(line => line.contains("#diabetes")).count())
val tuberculosis = (textFile.filter(line => line.contains("#tuberculosis")).count())
val braintumour = (textFile.filter(line => line.contains("#braintumour")).count())
val malaria = (textFile.filter(line => line.contains("#malaria")).count())
val dengue = (textFile.filter(line => line.contains("#dengue")).count())
val asthma = (textFile.filter(line => line.contains("#asthma")).count())
val chickenpox = (textFile.filter(line => line.contains("#chickenpox")).count())

println("*********************************************")
println("Number of users tweeted on different disease")
println("*********************************************")
println("Heart Attack : %s".format(heartattack))
println("Cancer : %s".format(cancer))
println("HIV : %s".format(hiv))
println("AIDS : %s".format(aids))
println("DIABETES : %s".format(diabetes))
println("TUBERCULOSIS : %s".format(tuberculosis))
println("BRAIN TUMOUR : %s".format(braintumour))
println("MALARIA : %s".format(malaria))
println("DENGUE : %s".format(dengue))
println("ASTHMA : %s".format(asthma))
println("CHICKENPOX : %s".format(chickenpox))
```

Output of query-1: Tweet count on different diseases

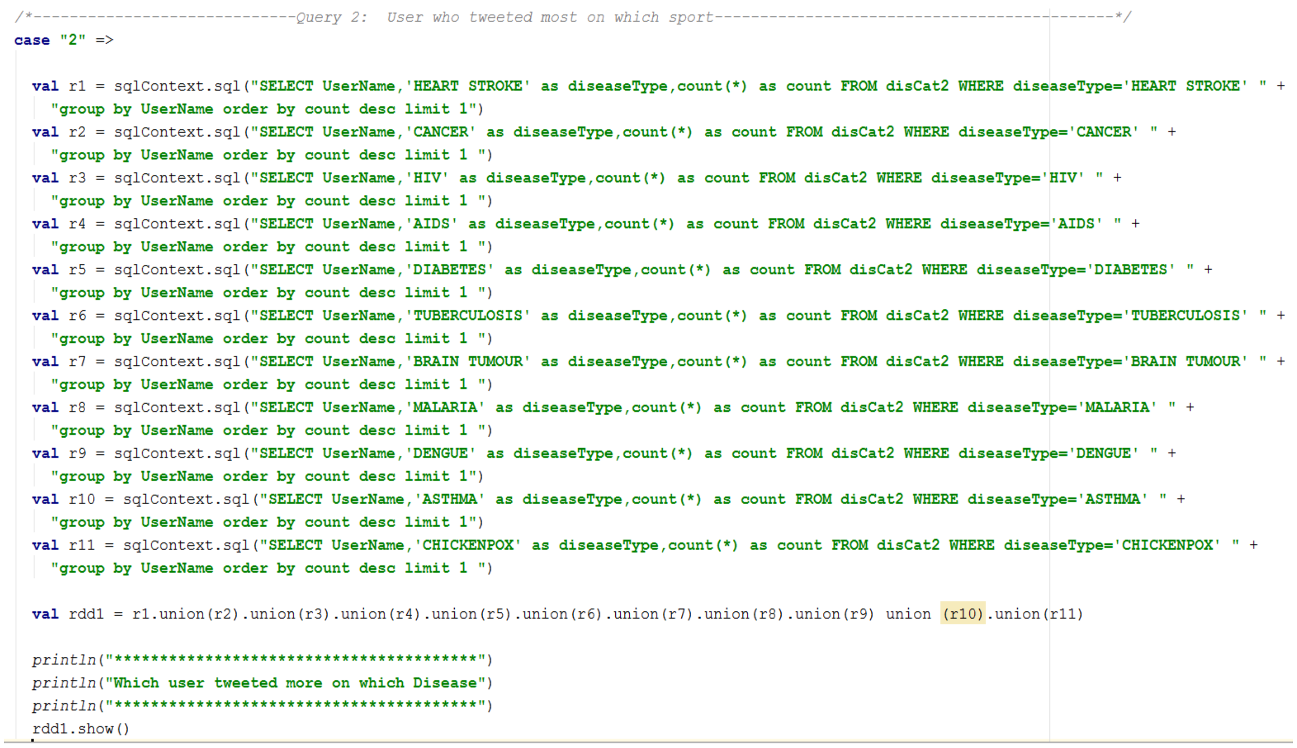


```
16/11/11 20:41:06 INFO DAGScheduler: Job 11 finished: count at Disease.scala:55, took 0.417944 s
********************************************
Number of users tweeted on different disease
********************************************
Heart Attack : 356
Cancer : 10540
HIV : 479
AIDS : 205
DIABETES : 7437
TUBERCULOSIS : 253
BRAIN TUMOUR : 49
MALARIA : 374
DENGUE : 512
ASTHMA : 736
CHICKENPOX : 15
```

Query Analysis: According to the query, more tweets are done on Cancer and least on Chickenpox

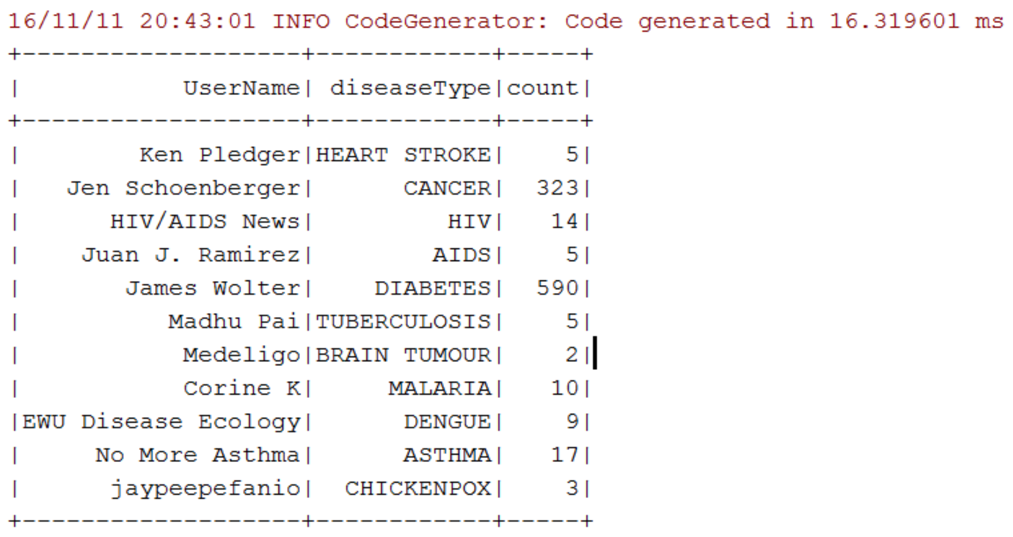
6.2 RDD Query 2

In this query the we are fetching maximum count of tweets done by a user on particular disease. This query is written using RDD. Initially for each disease the top tweeted user is fetched and **UNION** RDD is used to club all the diseases.



```
/*-----------------------------Query 2:  User who tweeted most on which sport--------------------------------------------*/
case "2" =>

  val r1 = sqlContext.sql("SELECT UserName,'HEART STROKE' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='HEART STROKE' " +
    "group by UserName order by count desc limit 1")
  val r2 = sqlContext.sql("SELECT UserName,'CANCER' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='CANCER' " +
    "group by UserName order by count desc limit 1 ")
  val r3 = sqlContext.sql("SELECT UserName,'HIV' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='HIV' " +
    "group by UserName order by count desc limit 1 ")
  val r4 = sqlContext.sql("SELECT UserName,'AIDS' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='AIDS' " +
    "group by UserName order by count desc limit 1 ")
  val r5 = sqlContext.sql("SELECT UserName,'DIABETES' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='DIABETES' " +
    "group by UserName order by count desc limit 1 ")
  val r6 = sqlContext.sql("SELECT UserName,'TUBERCULOSIS' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='TUBERCULOSIS' " +
    "group by UserName order by count desc limit 1 ")
  val r7 = sqlContext.sql("SELECT UserName,'BRAIN TUMOUR' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='BRAIN TUMOUR' " +
    "group by UserName order by count desc limit 1 ")
  val r8 = sqlContext.sql("SELECT UserName,'MALARIA' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='MALARIA' " +
    "group by UserName order by count desc limit 1 ")
  val r9 = sqlContext.sql("SELECT UserName,'DENGUE' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='DENGUE' " +
    "group by UserName order by count desc limit 1")
  val r10 = sqlContext.sql("SELECT UserName,'ASTHMA' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='ASTHMA' " +
    "group by UserName order by count desc limit 1")
  val r11 = sqlContext.sql("SELECT UserName,'CHICKENPOX' as diseaseType,count(*) as count FROM disCat2 WHERE diseaseType='CHICKENPOX' " +
    "group by UserName order by count desc limit 1 ")

  val rdd1 = r1.union(r2).union(r3).union(r4).union(r5).union(r6).union(r7).union(r8).union(r9) union (r10).union(r11)

  println("*****************************************")
  println("Which user tweeted more on which Disease")
  println("*****************************************")
  rdd1.show()
```

Output of query-2: Tweet count on different diseases for top Users

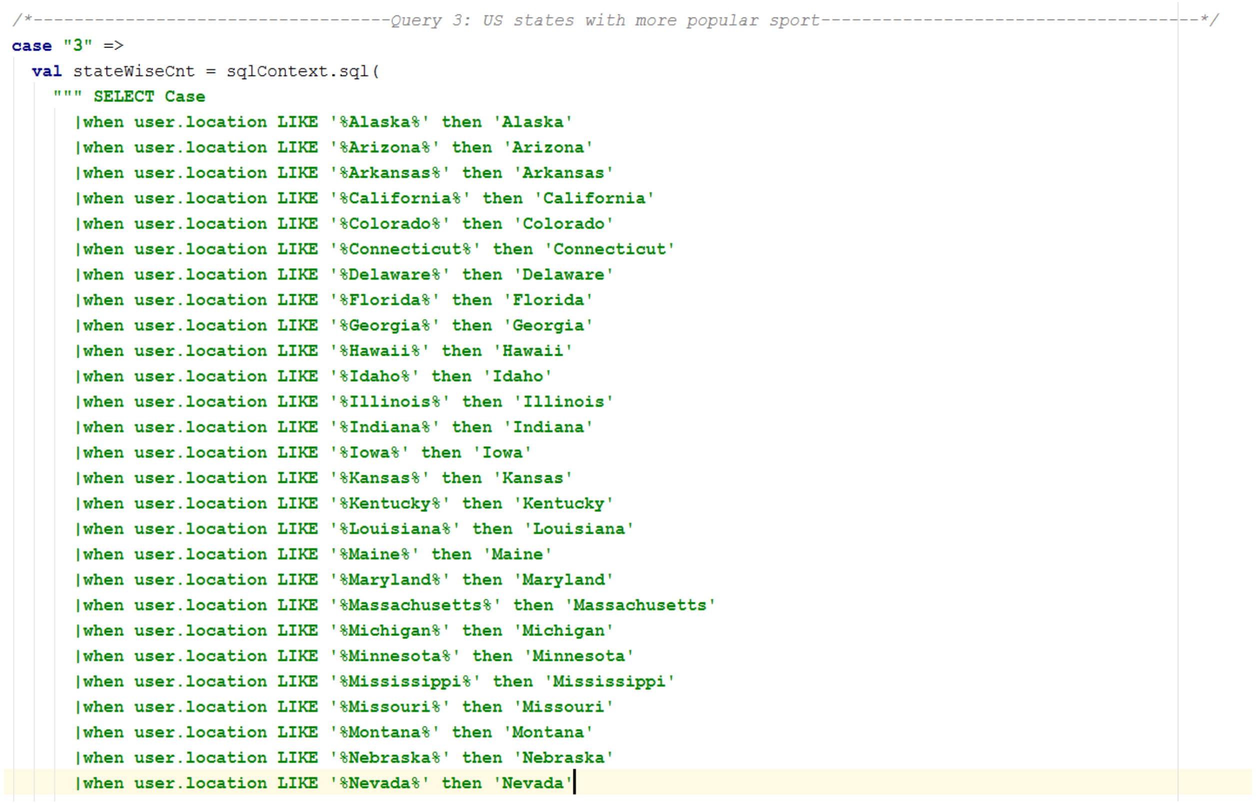


```
16/11/11 20:43:01 INFO CodeGenerator: Code generated in 16.319601 ms
+-------------------+------------+-----+
|           UserName| diseaseType|count|
+-------------------+------------+-----+
|        Ken Pledger|HEART STROKE|    5|
|    Jen Schoenberger|      CANCER|  323|
|      HIV/AIDS News|         HIV|   14|
|     Juan J. Ramirez|        AIDS|    5|
|       James Wolter|    DIABETES|  590|
|          Madhu Pai|TUBERCULOSIS|    5|
|           Medeligo|BRAIN TUMOUR|    2|
|           Corine K|     MALARIA|   10|
|EWU Disease Ecology|      DENGUE|    9|
|     No More Asthma|      ASTHMA|   17|
|      jaypeepefanio|  CHICKENPOX|    3|
+-------------------+------------+-----+
```

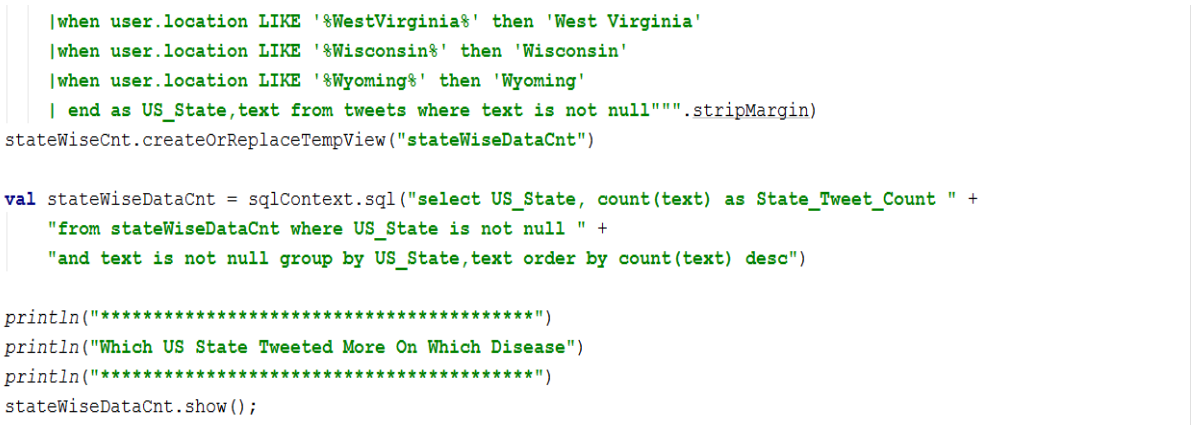
Query Analysis: According to the query, James Wolter tweeted 590 tweets on Diabetes disease which is highest when compared to others.

6.3 Data Frame Query 3

In this query the top state that tweeted more on diseases is fetched. First the location in tweets are fetched from tweets file and count is displayed for US States as shown below.

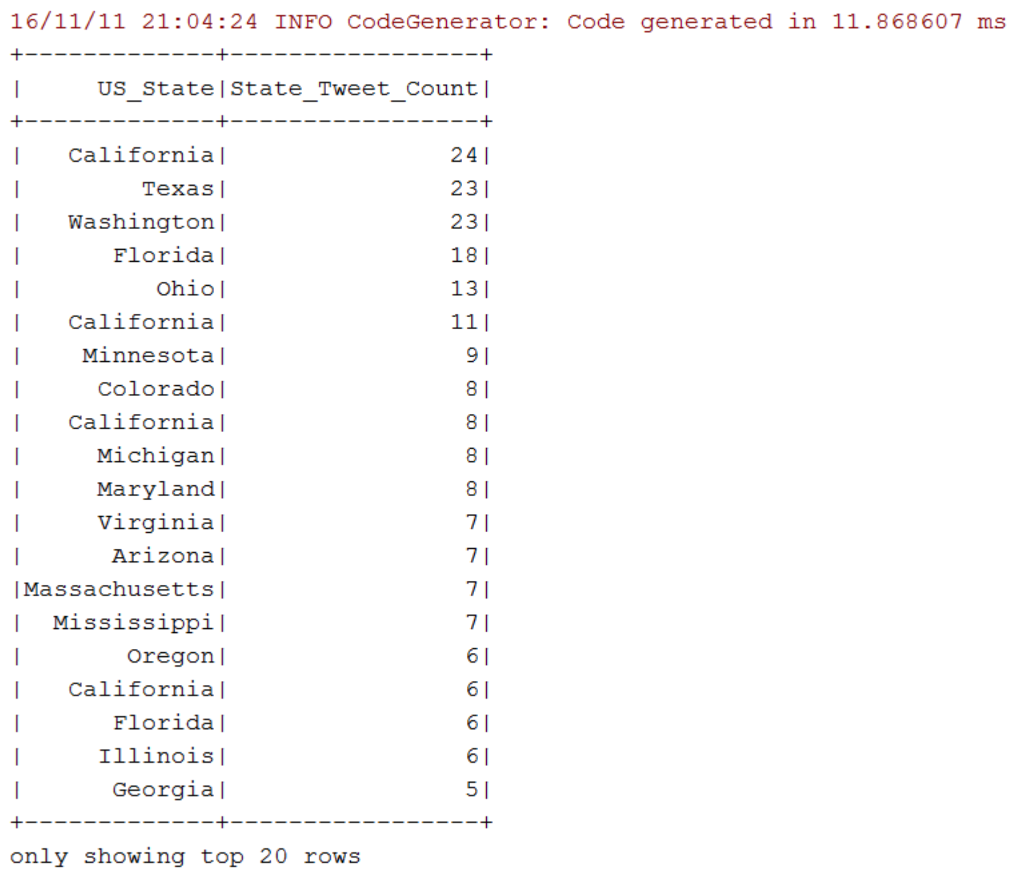


```
/*-----------------------------------Query 3: US states with more popular sport-------------------------------------*/
case "3" =>
  val stateWiseCnt = sqlContext.sql(
    """ SELECT Case
      |when user.location LIKE '%Alaska%' then 'Alaska'
      |when user.location LIKE '%Arizona%' then 'Arizona'
      |when user.location LIKE '%Arkansas%' then 'Arkansas'
      |when user.location LIKE '%California%' then 'California'
      |when user.location LIKE '%Colorado%' then 'Colorado'
      |when user.location LIKE '%Connecticut%' then 'Connecticut'
      |when user.location LIKE '%Delaware%' then 'Delaware'
      |when user.location LIKE '%Florida%' then 'Florida'
      |when user.location LIKE '%Georgia%' then 'Georgia'
      |when user.location LIKE '%Hawaii%' then 'Hawaii'
      |when user.location LIKE '%Idaho%' then 'Idaho'
      |when user.location LIKE '%Illinois%' then 'Illinois'
      |when user.location LIKE '%Indiana%' then 'Indiana'
      |when user.location LIKE '%Iowa%' then 'Iowa'
      |when user.location LIKE '%Kansas%' then 'Kansas'
      |when user.location LIKE '%Kentucky%' then 'Kentucky'
      |when user.location LIKE '%Louisiana%' then 'Louisiana'
      |when user.location LIKE '%Maine%' then 'Maine'
      |when user.location LIKE '%Maryland%' then 'Maryland'
      |when user.location LIKE '%Massachusetts%' then 'Massachusetts'
      |when user.location LIKE '%Michigan%' then 'Michigan'
      |when user.location LIKE '%Minnesota%' then 'Minnesota'
      |when user.location LIKE '%Mississippi%' then 'Mississippi'
      |when user.location LIKE '%Missouri%' then 'Missouri'
      |when user.location LIKE '%Montana%' then 'Montana'
      |when user.location LIKE '%Nebraska%' then 'Nebraska'
      |when user.location LIKE '%Nevada%' then 'Nevada'
```



```
    |when user.location LIKE '%WestVirginia%' then 'West Virginia'
    |when user.location LIKE '%Wisconsin%' then 'Wisconsin'
    |when user.location LIKE '%Wyoming%' then 'Wyoming'
    | end as US_State,text from tweets where text is not null""".stripMargin)
stateWiseCnt.createOrReplaceTempView("stateWiseDataCnt")

val stateWiseDataCnt = sqlContext.sql("select US_State, count(text) as State_Tweet_Count " +
    "from stateWiseDataCnt where US_State is not null " +
    "and text is not null group by US_State,text order by count(text) desc")

println("*******************************************")
println("Which US State Tweeted More On Which Disease")
println("*******************************************")
stateWiseDataCnt.show();
```

Output of query-3: US States that tweeted more on Diseases

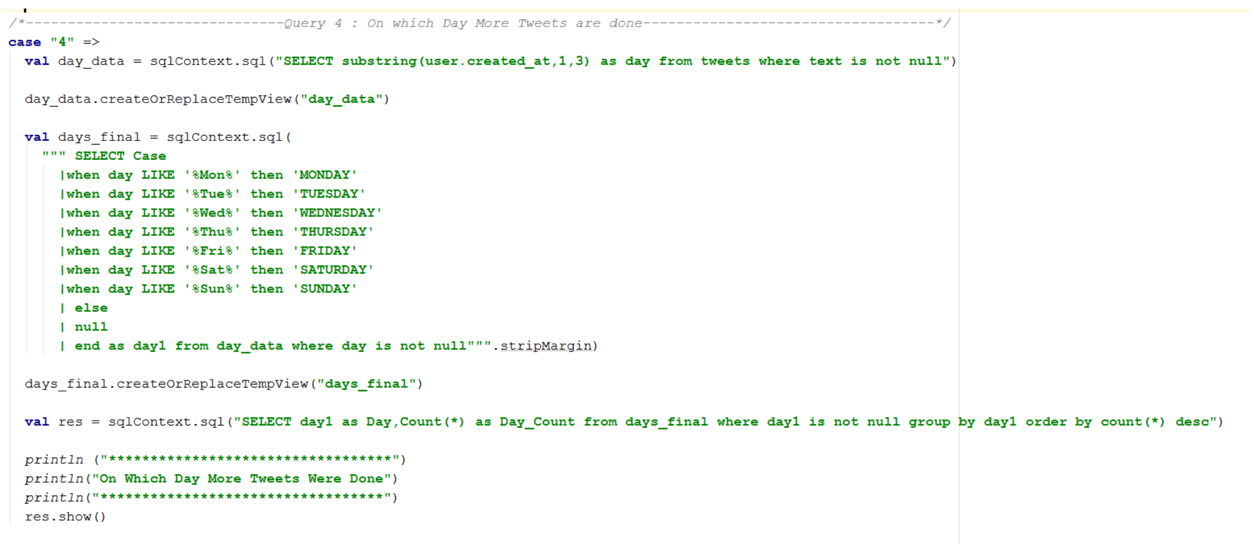


```
16/11/11 21:04:24 INFO CodeGenerator: Code generated in 11.868607 ms
+-------------+-----------------+
|     US_State|State_Tweet_Count|
+-------------+-----------------+
|   California|               24|
|        Texas|               23|
|   Washington|               23|
|      Florida|               18|
|         Ohio|               13|
|   California|               11|
|    Minnesota|                9|
|     Colorado|                8|
|   California|                8|
|     Michigan|                8|
|     Maryland|                8|
|     Virginia|                7|
|      Arizona|                7|
|Massachusetts|                7|
|  Mississippi|                7|
|       Oregon|                6|
|   California|                6|
|      Florida|                6|
|     Illinois|                6|
|      Georgia|                5|
+-------------+-----------------+
only showing top 20 rows
```

Query Analysis: According to the query, California state tweeted more on diseases when compared to other US states.

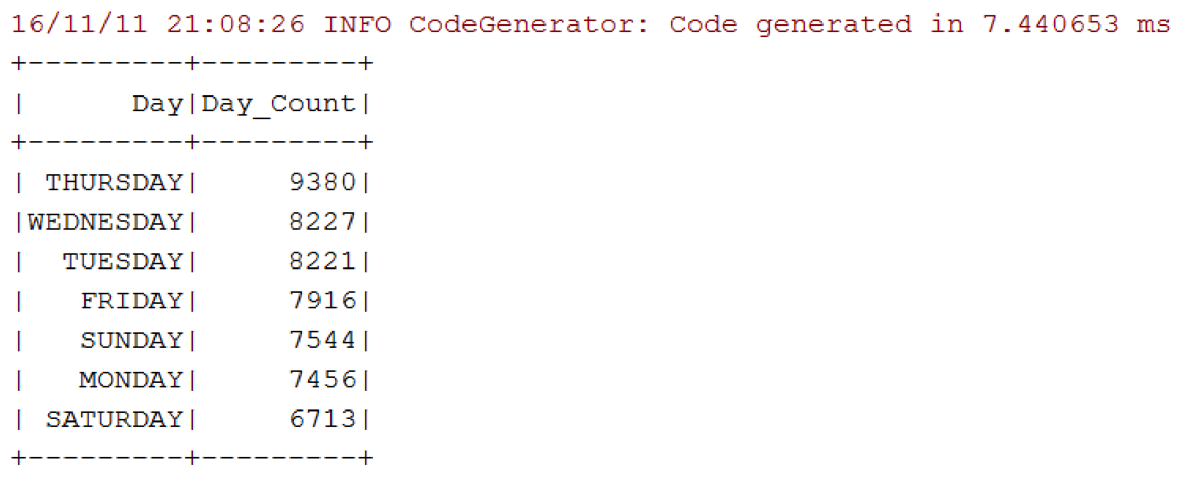
6.4 Data Frame Query 4

In this query, data is fetched based which day of week more tweets are done on Diseases. Initially **created_at** is fetched from tweets file and count of tweets is done on each week of day.



```
/*------------------------------Query 4 : On which Day More Tweets are done-----------------------------------*/
case "4" =>
  val day_data = sqlContext.sql("SELECT substring(user.created_at,1,3) as day from tweets where text is not null")

  day_data.createOrReplaceTempView("day_data")

  val days_final = sqlContext.sql(
    """ SELECT Case
      |when day LIKE '%Mon%' then 'MONDAY'
      |when day LIKE '%Tue%' then 'TUESDAY'
      |when day LIKE '%Wed%' then 'WEDNESDAY'
      |when day LIKE '%Thu%' then 'THURSDAY'
      |when day LIKE '%Fri%' then 'FRIDAY'
      |when day LIKE '%Sat%' then 'SATURDAY'
      |when day LIKE '%Sun%' then 'SUNDAY'
      | else
      | null
      | end as day1 from day_data where day is not null""".stripMargin)

  days_final.createOrReplaceTempView("days_final")

  val res = sqlContext.sql("SELECT day1 as Day,Count(*) as Day_Count from days_final where day1 is not null group by day1 order by count(*) desc")

  println ("***********************************")
  println("On Which Day More Tweets Were Done")
  println("***********************************")
  res.show()
```

Output of query-4: On which day, more tweets are done on diseases



```
16/11/11 21:08:26 INFO CodeGenerator: Code generated in 7.440653 ms
+---------+---------+
|      Day|Day_Count|
+---------+---------+
| THURSDAY|     9380|
|WEDNESDAY|     8227|
|  TUESDAY|     8221|
|   FRIDAY|     7916|
|   SUNDAY|     7544|
|   MONDAY|     7456|
| SATURDAY|     6713|
+---------+---------+
```

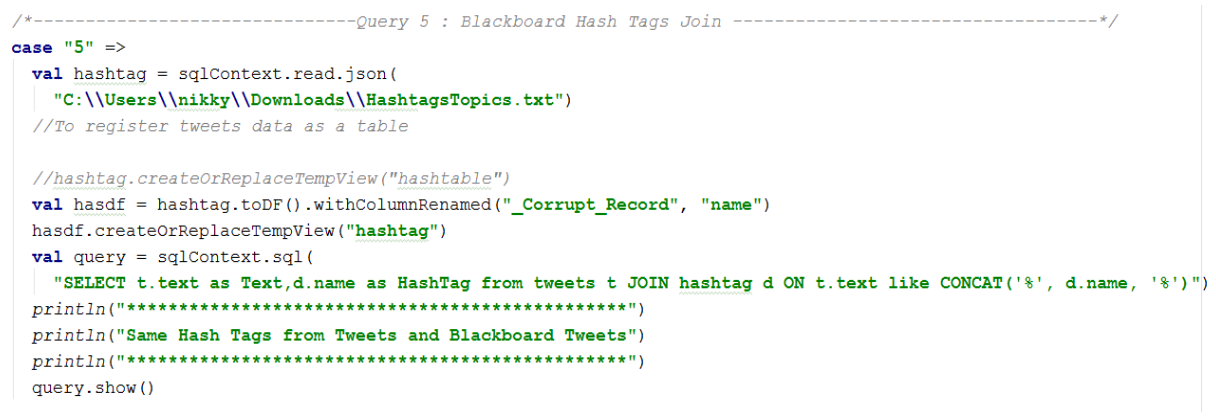
Query Analysis: According to the query, on Thursday more tweets are done and less tweets are done on Saturday

6.5 Blackboard Popular Hash Tags Query 5

In this query, we took popular hash tags text file from blackboard and performed **JOIN** operation with hash tags from diseases tweets file.

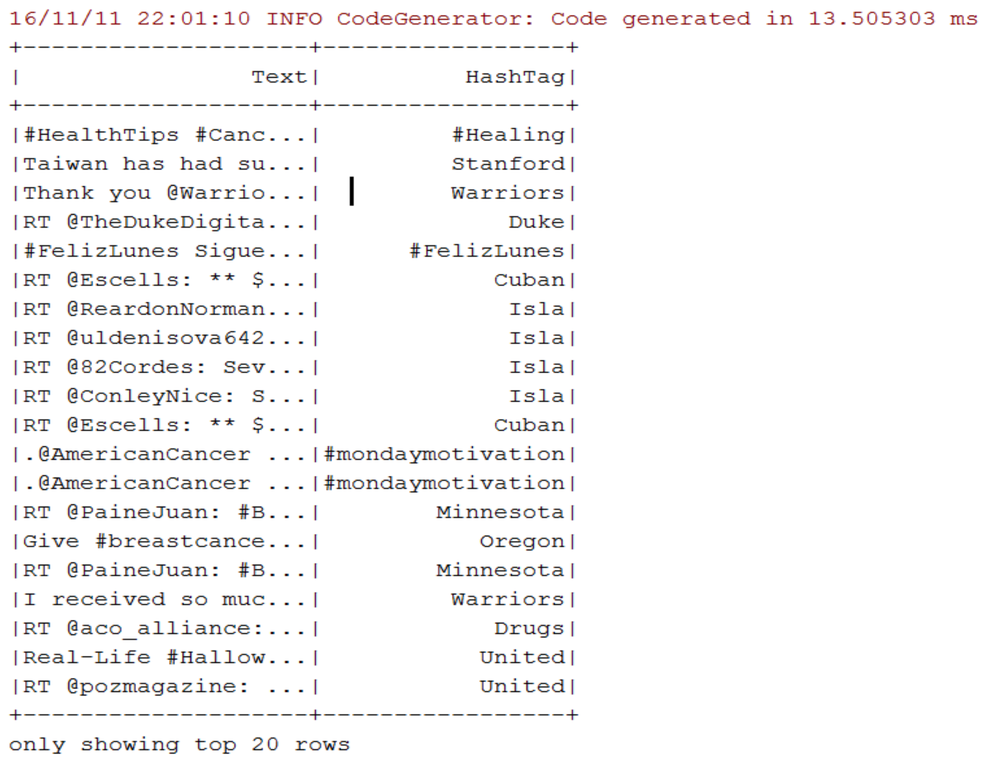
Blackboard Popular Hash Tags:

https://github.com/cmoulika009/Principles-of-Big-Data-Management/blob/master/PB%20Phase-2-%20Team%2011/Blackboard%20Tweets.txt



```
/*------------------------------Query 5 : Blackboard Hash Tags Join -----------------------------------*/
case "5" =>
  val hashtag = sqlContext.read.json(
    "C:\\Users\\nikky\\Downloads\\HashtagsTopics.txt")
  //To register tweets data as a table

  //hashtag.createOrReplaceTempView("hashtable")
  val hasdf = hashtag.toDF().withColumnRenamed("_Corrupt_Record", "name")
  hasdf.createOrReplaceTempView("hashtag")
  val query = sqlContext.sql(
    "SELECT t.text as Text,d.name as HashTag from tweets t JOIN hashtag d ON t.text like CONCAT('%', d.name, '%')")
  println("**************************************************")
  println("Same Hash Tags from Tweets and Blackboard Tweets")
  println("**************************************************")
  query.show()
```

Output of query-5: Which hash tags are common between popular hashtags and tags in tweets file



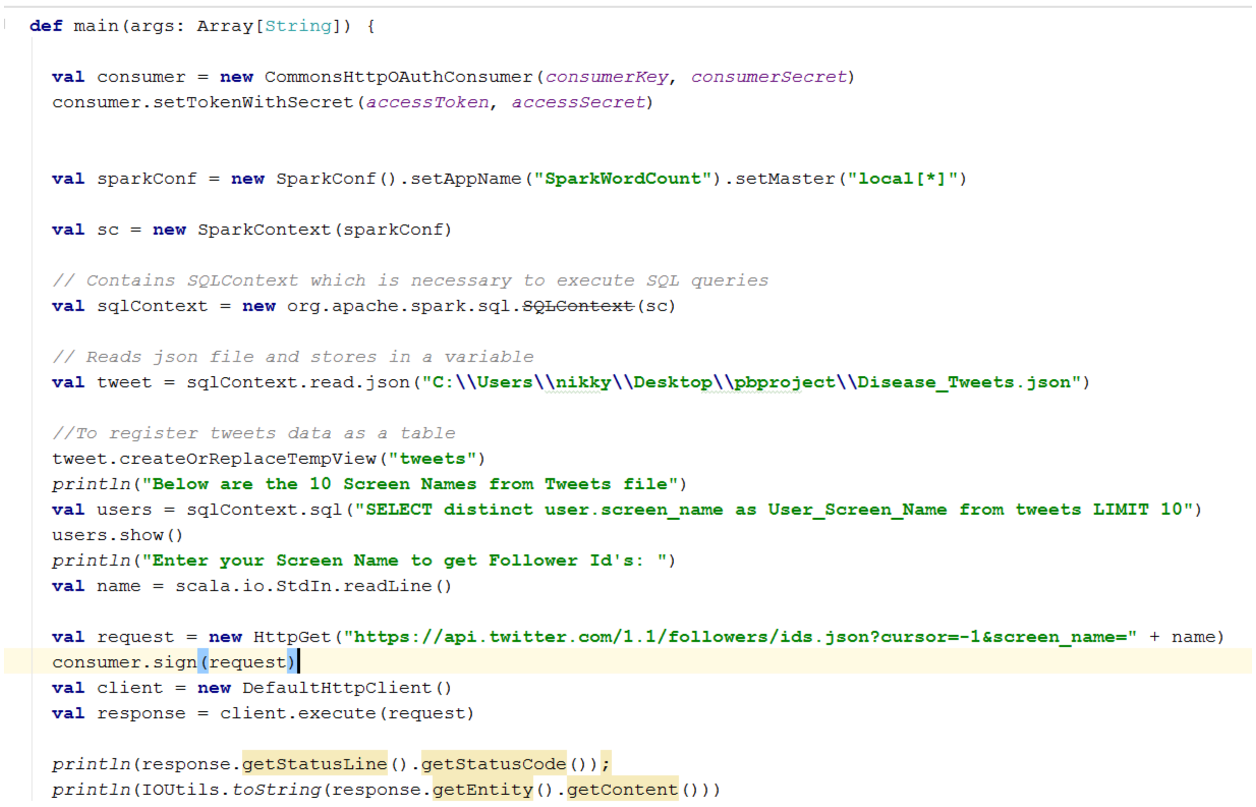
```
16/11/11 22:01:10 INFO CodeGenerator: Code generated in 13.505303 ms
+--------------------+-----------------+
|                Text|          HashTag|
+--------------------+-----------------+
|#HealthTips #Canc...|         #Healing|
|Taiwan has had su...|         Stanford|
|Thank you @Warrio...|         Warriors|
|RT @TheDukeDigita...|             Duke|
|#FelizLunes Sigue...|      #FelizLunes|
|RT @Escells: ** $...|            Cuban|
|RT @ReardonNorman...|             Isla|
|RT @uldenisova642...|             Isla|
|RT @82Cordes: Sev...|             Isla|
|RT @ConleyNice: S...|             Isla|
|RT @Escells: ** $...|            Cuban|
|.@AmericanCancer ...|#mondaymotivation|
|.@AmericanCancer ...|#mondaymotivation|
|RT @PaineJuan: #B...|        Minnesota|
|Give #breastcance...|           Oregon|
|RT @PaineJuan: #B...|        Minnesota|
|I received so muc...|         Warriors|
|RT @aco_alliance:...|            Drugs|
|Real-Life #Hallow...|           United|
|RT @pozmagazine: ...|           United|
+--------------------+-----------------+
only showing top 20 rows
```

6.6 Twitter API Query 6

Twitter Get Followers ids API is used. A query to display ten screen names from the tweets file is written. When the query is executed a table with ten screen names is displayed in the table.

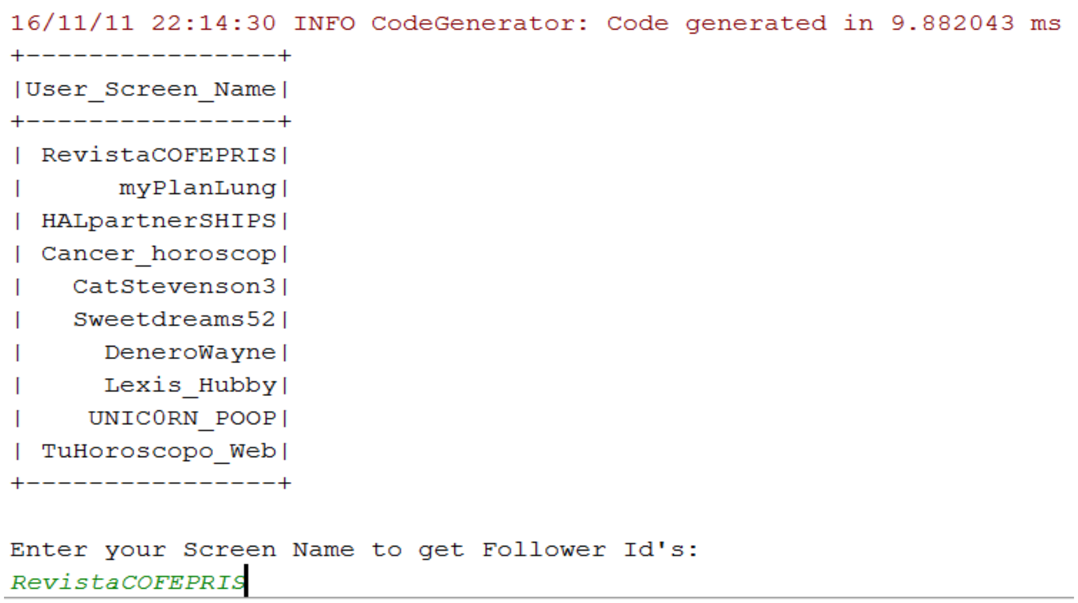
```
val request = new HttpGet("https://api.twitter.com/1.1/followers/ids.json?cursor=-
1&screen_name=" + name)
```

First the user is given a Choice to enter a screen name of his choice. Once the screen name has been inputted the follower's id



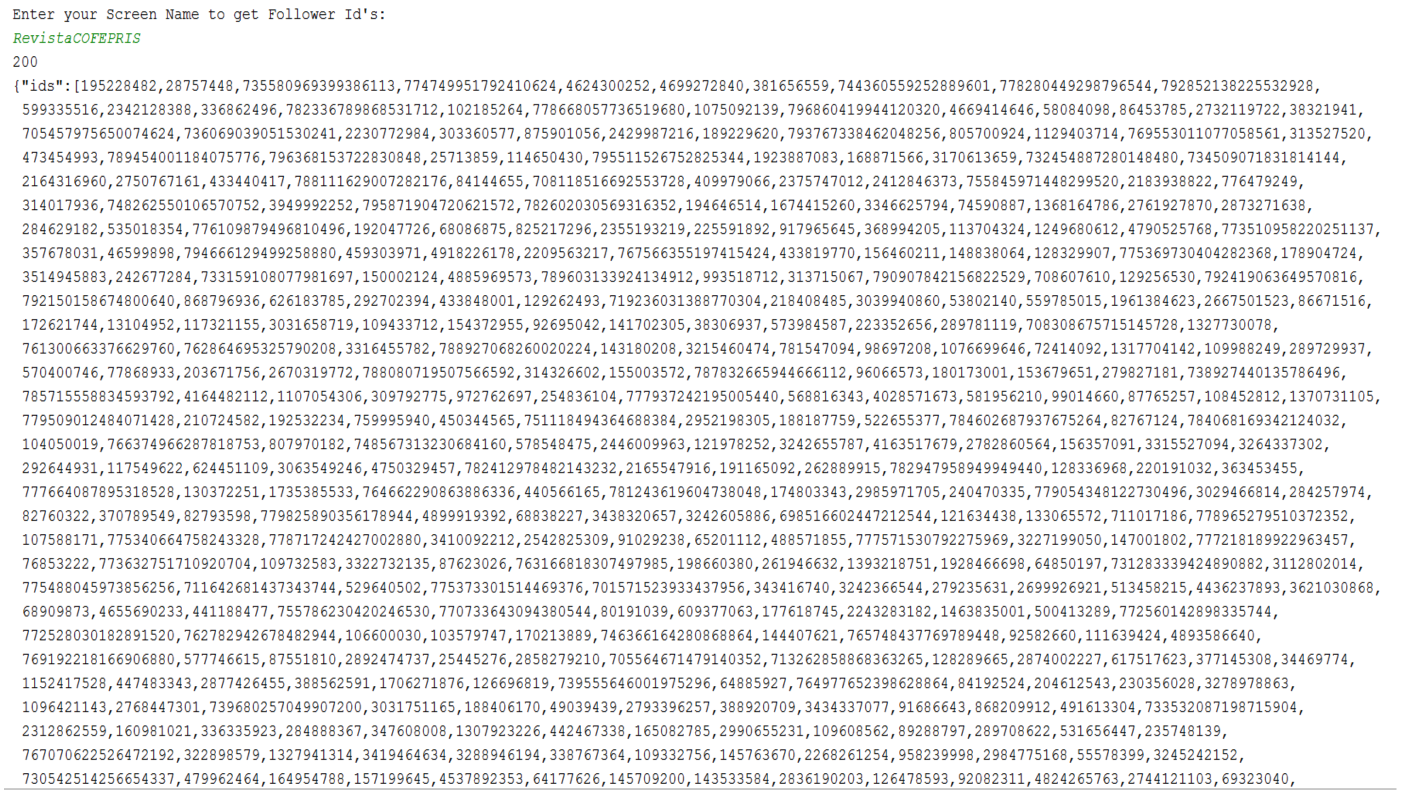
```
def main(args: Array[String]) {

  val consumer = new CommonsHttpOAuthConsumer(consumerKey, consumerSecret)
  consumer.setTokenWithSecret(accessToken, accessSecret)


  val sparkConf = new SparkConf().setAppName("SparkWordCount").setMaster("local[*]")

  val sc = new SparkContext(sparkConf)

  // Contains SQLContext which is necessary to execute SQL queries
  val sqlContext = new org.apache.spark.sql.~~SQLContext~~(sc)

  // Reads json file and stores in a variable
  val tweet = sqlContext.read.json("C:\\Users\\nikky\\Desktop\\pbproject\\Disease_Tweets.json")

  //To register tweets data as a table
  tweet.createOrReplaceTempView("tweets")
  println("Below are the 10 Screen Names from Tweets file")
  val users = sqlContext.sql("SELECT distinct user.screen_name as User_Screen_Name from tweets LIMIT 10")
  users.show()
  println("Enter your Screen Name to get Follower Id's: ")
  val name = scala.io.StdIn.readLine()

  val request = new HttpGet("https://api.twitter.com/1.1/followers/ids.json?cursor=-1&screen_name=" + name)
  consumer.sign(request)
  val client = new DefaultHttpClient()
  val response = client.execute(request)

  println(response.getStatusLine().getStatusCode());
  println(IOUtils.toString(response.getEntity().getContent()))
```

Output of query-6: Initially User Screen Name is displayed where single Screen Name is entered



```
16/11/11 22:14:30 INFO CodeGenerator: Code generated in 9.882043 ms
+----------------+
|User_Screen_Name|
+----------------+
| RevistaCOFEPRIS|
|      myPlanLung|
| HALpartnerSHIPS|
| Cancer_horoscop|
|   CatStevenson3|
|   Sweetdreams52|
|     DeneroWayne|
|     Lexis_Hubby|
|     UNICORN_POOP|
| TuHoroscopo_Web|
+----------------+

Enter your Screen Name to get Follower Id's:
RevistaCOFEPRIS
```

Once screen name **RevistaCOFEPRIS** is entered the follower id's are displayed as shown below



```
Enter your Screen Name to get Follower Id's:
RevistaCOFEPRIS
200
{"ids":[195228482,28757448,735580969399386113,774749951792410624,4624300252,4699272840,381656559,744360559252889601,778280449298796544,792852138225532928,
 599335516,2342128388,336862496,782336789868531712,102185264,778668057736519680,1075092139,796860419944120320,4669414646,58084098,86453785,2732119722,38321941,
 705457975650074624,736069039051530241,2230772984,303360577,875901056,2429987216,189229620,793767338462048256,805700924,1129403714,769553011077058561,313527520,
 473454993,789454001184075776,796368153722830848,25713859,114650430,795511526752825344,1923887083,168871566,3170613659,732454887280148480,734509071831814144,
 2164316960,2750767161,433440417,788111629007282176,84144655,708118516692553728,409979066,2375747012,2412846373,755845971448299520,2183938822,776479249,
 314017936,748262550106570752,3949992252,795871904720621572,782602030569316352,194646514,1674415260,3346625794,74590887,1368164786,2761927870,2873271638,
 284629182,535018354,776109879496810496,192047726,68086875,825217296,2355193219,225591892,917965645,368994205,113704324,1249680612,4790525768,773510958220251137,
 357678031,46599898,794666129499258880,459303971,4918226178,2209563217,767566355197415424,433819770,156460211,148838064,128329907,775369730404282368,178904724,
 3514945883,242677284,733159108077981697,150002124,4885969573,789603133924134912,993518712,313715067,790907842156822529,708607610,129256530,792419063649570816,
 792150158674800640,868796936,626183785,292702394,433848001,129262493,719236031388770304,218408485,3039940860,53802140,559785015,1961384623,2667501523,86671516,
 172621744,13104952,117321155,3031658719,109433712,154372955,92695042,141702305,38306937,573984587,223352656,289781119,708308675715145728,1327730078,
 761300663376629760,762864695325790208,3316455782,788927068260020224,143180208,3215460474,781547094,98697208,1076699646,72414092,1317704142,109988249,289729937,
 570400746,77868933,203671756,2670319772,788080719507566592,314326602,155003572,787832665944666112,96066573,180173001,153679651,279827181,738927440135786496,
 785715558834593792,4164482112,1107054306,309792775,972762697,254836104,777937242195005440,568816343,4028571673,581956210,99014660,87765257,108452812,1370731105,
 779509012484071428,210724582,192532234,759995940,450344565,751118494364688384,2952198305,188187759,522655377,784602687937675264,82767124,784068169342124032,
 104050019,766374966287818753,807970182,748567313230684160,578548475,2446009963,121978252,3242655787,4163517679,2782860564,156357091,3315527094,3264337302,
 292644931,117549622,624451109,3063549246,4750329457,782412978482143232,2165547916,191165092,262889915,782947958949949440,128336968,220191032,363453455,
 777664087895318528,130372251,1735385533,764662290863886336,440566165,781243619604738048,174803343,2985971705,240470335,779054348122730496,3029466814,284257974,
 82760322,370789549,82793598,779825890356178944,4899919392,68838227,3438320657,3242605886,698516602447212544,121634438,133065572,711017186,778965279510372352,
 107588171,775340664758243328,778717242427002880,3410092212,2542825309,91029238,65201112,488571855,777571530792275969,3227199050,147001802,777218189922963457,
 76853222,773632751710920704,109732583,3322732135,87623026,763166818307497985,198660380,261946632,1393218751,1928466698,64850197,731283339424890882,3112802014,
 775488045973856256,711642681437343744,529640502,775373301514469376,701571523933437956,343416740,3242366544,279235631,2699926921,513458215,4436237893,3621030868,
 68909873,4655690233,441188477,755786230420246530,770733643094380544,80191039,609377063,177618745,2243283182,1463835001,500413289,772560142898335744,
 772528030182891520,762782942678482944,106600030,103579747,170213889,746366164280868864,144407621,765748437769789448,92582660,111639424,4893586640,
 769192218166906880,577746615,87551810,2892474737,25445276,2858279210,705564671479140352,713262858868363265,128289665,2874002227,617517623,377145308,34469774,
 1152417528,447483343,2877426455,388562591,1706271876,126696819,739555646001975296,64885927,764977652398628864,84192524,204612543,230356028,3278978863,
 1096421143,2768447301,739680257049907200,3031751165,188406170,49039439,2793396257,388920709,3434337077,91686643,868209912,491613304,733532087198715904,
 2312862559,160981021,336335923,284888367,347608008,1307923226,442467338,165082785,2990655231,109608562,89288797,289708622,531656447,235748139,
 767070622526472192,322898579,1327941314,3419464634,3288946194,338767364,109332756,145763670,2268261254,958239998,2984775168,55578399,3245242152,
 730542514256654337,479962464,164954788,157199645,4537892353,64177626,145709200,143533584,2836190203,126478593,92082311,4824265763,2744121103,69323040,
```

7. References

https://apps.twitter.com

http://www.tutorialspoint.com

https://www.jetbrains.com/idea/