import java.io.{File, PrintWriter}

import javax.servlet.http.HttpServletRequest

import org.apache.spark.sql.Row

import org.apache.spark.sql.types.{StringType, StructField, StructType}

import org.apache.spark.{SparkConf, SparkContext}

//import org.apache.spark.sql.\_

/\*\*

\* Created by Swetcha on 11/6/2016.

\*/

object Queries{

def main(args: Array[String]): Unit = {

System.setProperty("hadoop.home.dir","C:\\Users\\sange\\Downloads\\hadooponwindows-master\\hadooponwindows-master")

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

val sc = new SparkContext(sparkConf)

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val lines= sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets1.txt")

val df=sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")

// val tel=lines.filter(x=>x.contains("tel"))

// val tam=lines.filter(z=>z.contains("tamil"))

// val kan=lines.filter(z=>z.contains("kannada"))

// val mal=lines.filter(z=>z.contains("malayalam"))

// val hin=lines.filter(z=>z.contains("hindi"))

// val pun=lines.filter(z=>z.contains("punjabi"))

// val bho=lines.filter(z=>z.contains("bhojpuri"))

// val ben=lines.filter(z=>z.contains("bengali"))

// val kor=lines.filter(z=>z.contains("korean"))

// val eng=lines.filter(y=>y.contains("eng"))

// val fre=lines.filter(z=>z.contains("french"))

// val jap=lines.filter(z=>z.contains("japanese"))

// val chi=lines.filter(z=>z.contains("chinese"))

// println(tel.count()+"!"+tam.count()+"!"+kan.count()+"!"+mal.count()+"!"+hin.count()+"!"+pun.count()+"!"+bho.count()+"!"+ben.count()+"!"+kor.count()+"!"+eng.count()+"!"+fre.count()+"!"+jap.count()+"!"+chi.count())

// df.registerTempTable("dftable")

df.createOrReplaceTempView("dftable")

// val que= sqlContext.sql("select user.location,user.followers\_count from dftable")

// que.show()

//

//

// val horror=sqlContext.sql("select user.name,user.followers\_count from dftable where text LIKE '%horror%'")

// val horrorcount=horror.count()

// val comedy=sqlContext.sql("select user.name,user.followers\_count as count from dftable where text LIKE '%comedy%'")

// val comedycount=comedy.count()

// val thriller=sqlContext.sql("select user.name,user.followers\_count as count from dftable where text LIKE '%thriller%'")

// val thrillercount=thriller.count()

// val animated=sqlContext.sql("select user.name,user.followers\_count as count from dftable where text LIKE '%animated%'")

// val animatedcount=animated.count()

// println("ANima Movies %s".format(animatedcount))

// println("Come Movies %s".format(comedycount))

// println("Thr Movies %s".format(thrillercount))

// println("Horr Movies %s".format(horrorcount))

//

//

// if(horrorcount>comedycount){

// if (horrorcount>thrillercount){

// if(horrorcount>animatedcount){

// println("Horror Movies are searched maximum")

// }else{

// println("Animated Movies are searched maximum")

// }

// }else if(thrillercount>animatedcount){

// println("Thriller Movies are searched maximum")

// }

// }else if(comedycount>thrillercount){

// if(comedycount>animatedcount){

// println("Comedy Movies are searched maximum")

// }

//

//

//

// val name : Option[String]= request getParameter "text"

// val hashtags= sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\hashtags.txt")

// val hashtagdf=hashtags.toDF().withColumnRenamed("\_corrupt\_record","hashfiles")

//

// hashtagdf.createOrReplaceTempView("dftab")

// val hashtagsquery=sqlContext.sql("select t.text as text,d.hashfiles as hashtags from dftable t JOIN dftab d on t.text like" +

// " CONCAT('%',d.hashfiles,'%')")

// hashtagsquery.show()

// val fwriter = new PrintWriter(new File("newFile.txt"))

// val active=sqlContext.sql("select user.name,max(user.followers\_count) as count, user.favourites\_count AS fav\_count " +

// "from dftable where user.followers\_count>100 group by user order by user.favourites\_count desc")

// active.show()

// printToFile(new File("active.csv")) { p => active.collect().foreach(p.println)

// }

// hashtagsquery.show()

// println("Printing the Common hashtags data of Hastags Data and Our Twitter Data")

// val rdd = sc.parallelize(List(hashtagdf,))

// map(p=>p.List(hashtagdf.collectAsList()),List(quee.collectAsList()))

// val quee=sqlContext.sql("select text from dftable")

// println(List(hashtagdf.collectAsList()))

// val horror=lines.filter(line=>line.contains("#horror")).count()

// val comedy=lines.filter(line=>line.contains("#comedy")).count()

// val thriller=lines.filter(line=>line.contains("#thriller")).count()

// val action=lines.filter(line=>line.contains("#action")).count()

// println("Horror Movies %s".format(horror))

// println("Comedy Movies %s".format(comedy))

// println("Thriller Movies %s".format(thriller))

// println("Action Movies %s".format(action))

// val rdd1 = sc.parallelize(List(horror,comedy,thriller,action))

// rdd1.collect().foreach(println)

/\*val jan=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Jan%' group by user")

val feb=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Feb%' group by user")

val mar=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Mar%' group by user")

val apr=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Apr%' group by user")

val may=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%May%' group by user")

val jun=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Jun%' group by user")

val jul=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Jul%' group by user")

val aug=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Aug%' group by user")

val sep=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Sep%' group by user")

val oct=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Oct%' group by user")

val nov=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Nov%' group by user")

val dec=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time," +

"Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Dec%' group by user")

val jancount=jan.count()

val febcount=feb.count()

val marcount=mar.count()

val aprcount=apr.count()

val maycount=may.count()

val juncount=jun.count()

val julcount=jul.count()

val augcount=aug.count()

val sepcount=sep.count()

val octcount=oct.count()

val novcount=nov.count()

val deccount=dec.count()

val firstquarter=jancount+febcount+marcount

val secondquarter=aprcount+maycount+juncount

val thirdquarter=julcount+augcount+sepcount

val fourthquarter=octcount+novcount+deccount

println("First Quarter Of Users %s" .format(firstquarter))

println("Second Quarter Of Users %s" .format(secondquarter))

println("Third Quarter Of Users %s" .format(thirdquarter))

println("Fourth Quarter Of Users %s" .format(fourthquarter))

\*/

/\*

if(firstquarter>secondquarter){

if(firstquarter>thirdquarter){

if(firstquarter>fourthquarter){

println("First Quarter Users are more than others")

}else{

println("Fourth Quarter Users are more than others")

}

}else if(thirdquarter>fourthquarter){

println("Third Quarter Users are more than others")

}

}else if(secondquarter>thirdquarter){

if(secondquarter>fourthquarter){

println("Second Quarter Users are more than others")

}

}

\*/

/\*val horror=sqlContext.sql("select user.name,user.followers\_count from dftable where text LIKE '%horror%'")

val horrorcount=horror.count()

println(horrorcount)

val comedy=sqlContext.sql("select user.name,user.followers\_count as count from dftable where text LIKE '%comedy%'")

val comedycount=comedy.count()

val thriller=sqlContext.sql("select user.name,user.followers\_count as count from dftable where text LIKE '%thriller%'")

val thrillercount=thriller.count()

val animated=sqlContext.sql("select user.name,user.followers\_count as count from dftable where text LIKE '%animated%'")

val animatedcount=animated.count()

if(horrorcount>comedycount){

if (horrorcount>thrillercount){

if(horrorcount>animatedcount){

println("Horror Movies are searched maximum")

}else{

println("Animated Movies are searched maximum")

}

}else if(thrillercount>animatedcount){

println("Thriller Movies are searched maximum")

}

}else if(comedycount>thrillercount){

if(comedycount>animatedcount){

println("Comedy Movies are searched maximum")

}

}\*/

/\*val tel=lines.filter(x=>x.contains("tel"))

val tam=lines.filter(z=>z.contains("tamil"))

val kan=lines.filter(z=>z.contains("kannada"))

val mal=lines.filter(z=>z.contains("malayalam"))

val hin=lines.filter(z=>z.contains("hindi"))

val pun=lines.filter(z=>z.contains("punjabi"))

val bho=lines.filter(z=>z.contains("bhojpuri"))

val ben=lines.filter(z=>z.contains("bengali"))

val kor=lines.filter(z=>z.contains("korean"))

val eng=lines.filter(y=>y.contains("eng"))

val fre=lines.filter(z=>z.contains("french"))

val jap=lines.filter(z=>z.contains("japanese"))

val chi=lines.filter(z=>z.contains("chinese"))

val south=mal.union(kan).union(tam).union(tel).count()

val foreign=eng.union(kor).union(fre).union(jap).union(chi).count()

val north=hin.union(pun).union(bho).union(ben).count()

println("South Indian Movies %s".format(south))

println("North Indian Movies %s".format(north))

println("Foreign Movies %s".format(foreign))\*/

val hashtags= sqlContext.read.json("C:\\Users\\sange\\Desktop\\Pavan\\hashtags.txt")

val hashtagdf=hashtags.toDF().withColumnRenamed("\_corrupt\_record","name")

val dftab= hashtagdf.registerTempTable("dftab")

hashtagdf.collectAsList()

val rdd = sc.parallelize(List(hashtagdf,))

map(p=>p.List(hashtagdf.collectAsList()),List(quee.collectAsList()))

val quee=sqlContext.sql("select text from dftable")

// println(List(hashtagdf.collectAsList()))

/\* val hashquery=sqlContext.sql("select t.text as text,d.name as hashtags from dftable t JOIN dftab d on t.text like CONCAT('%',d.name,'%')")

hashquery.show()

// Query1 most searched movies

/\*val query5=

sqlContext.sql("select user.name,user.favourites\_count from dftable where text LIKE '%horror%'")

val query6=

sqlContext.sql("select user.favourites\_count,place.country from dftable where user.location='London'and place.country is not null ")

val queryjoin=query5.join(query6,"favourites\_count")

queryjoin.show()\*/

/\*

val jan=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time,Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Jan%'")

val jancount=jan.count()

val feb=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time,Count(\*) as count from dftable where SUBSTRING(user.created\_at,5,3) like '%Feb%'")

val febcount=feb.count()\*/

// println("count %s".format(que))

// val us=sqlContext.sql("select Count(\*) as count from dftable where user.location like'%US%' group by count")

// us.show()

// val uscount=us.count()

// val ind=sqlContext.sql("select user.screen\_name,user.time\_zone, count(\*) as count from dftable where user.location like'%IND%' group by user")

// val indcount=ind.count()

// val uk=sqlContext.sql("select user.screen\_name,user.time\_zone, count(\*) as count from dftable where user.location like'%UK%' group by user")

// val ukcount=uk.count()

// val jap=sqlContext.sql("select user.screen\_name,user.time\_zone, count(\*) as count from dftable where user.location like'%Jap%' group by user")

// val japcount=jap.count()

// val taiwan=sqlContext.sql("select user.screen\_name,user.time\_zone, count(\*) as count from dftable where user.location like'%Taiwan%' group by user")

// val taiwancount=taiwan.count()

// val rdds=unionAll

// val spar=sc.parallelize(List(us,ind,uk))

// val ndsm= sc.parallelize(List(jap,taiwan))

// val uni=spar.union(ndsm)

// val abc=sqlContext.sql("SELECT user.name AS Name,user.followers\_count AS Followers\_Count

// FROM dftable WHERE user.verified=true ORDER BY user.followers\_count DESC LIMIT 10")

// abc.show()

// val fwriter = new PrintWriter(new File("newFile.txt"))

// val count = lines.flatMap(line=>line.split(" ")).map(word=>(word,1)).reduceByKey(\_ + \_)

// count.foreach(println)

// printToFile(new File("exampleout.txt")) { p => count.collect().foreach(p.println)

// }

// val jsonRDD = sc.wholeTextFiles("C:\\Users\\sange\\Desktop\\Pavan\\data2.json")

// val namesJson = sqlContext.read.json(jsonRDD)

// jsonRDD[String]=jsonRDD.map(p=>p.id)

// val myPartition=lines.map(\_.split(",")).filter(line=>line.contains("#movies")).countByValue()

// myPartition.collect().foreach(println)

/\* val hor=lines.filter(line=>line.contains("#horror")).count()

// println("count %s" .format(met))

val com=lines.filter(line=>line.contains("#comedy")).count()

val thri=lines.filter(line=>line.contains("#thriller")).count()

val que=lines.filter(line=>line.contains("#action")).count()

// println("count %s".format(que))

val rdd = sc.parallelize(List(hor,com,thri,que))

rdd.collect().foreach(println)\*/

// val lit=lines.collect().foreach(println)

// val hashtags = sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\hashtags.txt")

// val nit=hashtags.collect().foreach(println)

// val res= hashtags.distinct().intersection(lines)

// println(res)

// val s=hashtags.map(\_.split("\n"))

// Query 2

/\*val active=sqlContext.sql("select user.name,max(user.followers\_count) as count, user.favourites\_count AS fav\_count

from dftable where user.followers\_count>100 group by user order by user.favourites\_count desc")

active.show()\*/

//query 3 followers count grouped by user location

//val place =sqlContext.sql("select user.location, max(user.followers\_count) AS count from dftable

// where user.followers\_count>100 group by user.location")

//place.show()

//query4

//val post=sqlContext.sql("select user.name, SUBSTRING(user.created\_at,5,3) AS month,SUBSTRING(user.created\_at,12,8) AS time from dftable

// where text LIKE '%telugu%' group by user")

//post.show()

//count.foreach(println)

//.map(\_.split(",")).map(attributes => (attributes(0), attributes(1).trim.toInt)).toDF()

//val schemaString = "contributors coordinates created\_at display\_text\_range entities extended\_entities extended\_tweet favorite\_count favorited filter\_level geo id id\_str in\_reply\_to\_screen\_name in\_reply\_to\_status\_id in\_reply\_to\_status\_id\_str in\_reply\_to\_user\_id in\_reply\_to\_user\_id\_str is\_quote\_status lang limit place possibly\_sensitive quoted\_status quoted\_status\_id quoted\_status\_id\_str retweeted retweeted\_status scopes source text timestamp\_ms truncated user"

//val schema = StructType(schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, true)))

//val rowRDD = rddMovies.map(\_.split(",")).map(p=> Row(p(0),p(1),p(2),p(3),p(4),p(5),p(6),p(7),p(8),p(9),p(10),p(11),p(12),p(13),p(14),p(15),p(16),p(17),p(18),p(19),p(20),p(21),p(22),p(23),p(24),p(25),p(26),p(27),p(28),p(29),p(30),p(31),p(32),p(33),p(34)))// val dfMovies = sqlContext.createDataFrame(rowRDD,schema)

// dfMovies.registerTempTable("movies")

// val movienames = sqlContext.sql("select id\_str from movies where id\_str= 790284109495107585")

// movienames.map(t=> "id:"+ t(0)).collect().foreach(println)

// df.show()

// df.printSchema()

// df.sqlContext.sql("Select \* from df")

// df.groupBy("retweet\_count")

// df.show()

// df.printSchema()

// val query1=sqlContext.sql("select user.id from dftable")

// query1.show()

// val lines = sc.textFile("C:\\Users\\sange\\Desktop\\Pavan\\tweets2.txt")

// var eachTweet = sqlContext.read.json(lines);

// val jsonRDD = sc.wholeTextFiles("C:\\Users\\sange\\Desktop\\Pavan\\data2.json").map(x => x.\_2)

// val namesJson = sqlContext.read.json(jsonRDD)

// namesJson.printSchema()

// namesJson.registerTempTable("tablename")

/\*val query1=sqlContext.sql("select user.name,created\_at from dftable where created\_at IN 'select created\_at from tablename'")

query1.show()

val q=sqlContext.sql("select user.name from tablename")

q.show()\*/

// val count = lines.flatMap(line=>line.split(" ")).map(word=>(word,1)).reduceByKey(\_ + \_)

// count.foreach(println)

// val lineLengths = lines.map(s => s.length)

// val totalLength = lineLengths.reduce((a, b) => a + b)

// lineLengths.persist()

// println(totalLength)

// val linecount=lines.map(\_.split(","))

// val jsons = lines.map(x=>x.split(",").filter(x => x(0) == "created\_at"))

// jsons.foreach(println)// linecount.flatMap(x => x(2).split(" ")).map((\_, 1)).take(5)

// val count=linecount.count()

// println(count)

}}

def printToFile(f: java.io.File)(op: java.io.PrintWriter => Unit) {

val p = new java.io.PrintWriter(f)

try { op(p) } finally { p.close() }

}

}