

This is the sample retail\_db data

Writing a application to find total sales for each day for CLOSED and COMPLETED orders.

For this we will use orders and orders\_items table

From orders , we will use orders\_date and from orders\_items we will use orders\_item\_subtotal

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

val conf=new SparkConf().setAppName(“Total Revenue Count”).setMaster(“local”)

val sc=new SparkContext(conf)

Reading data from orders and converting it to desired format

val orders=sc.textFile(C:\\Program Files\\Git\\MyGitFilesDirectory\\data\\retail\_db\\order\\part-00000”)

orders.take(5).foreach(println)

1,2013-07-25 00:00:00.0,11599,CLOSED

2,2013-07-25 00:00:00.0,256,PENDING\_PAYMENT

3,2013-07-25 00:00:00.0,12111,COMPLETE

4,2013-07-25 00:00:00.0,8827,CLOSED

5,2013-07-25 00:00:00.0,11318,COMPLETE

val ordersFilter=order.filter(rec=>(rec.split(‘,’)(3)==”COMPLETE” || rec.split(‘,’)(3)==”CLOSED”))

ordersfilter: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:16

val ordersfilterMap=ordersFilter.map(rec => (rec.split(‘,’)(0).toInt , rec.split(‘,’)(1)))

ordersfilterMap: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[5] at map at <console>:17

Now we will do the same with orders\_item and uses its order\_item\_subtotal

val ordersitem=sc.textFile(C:\\Program Files\\Git\\MyGitFilesDirectory\\data\\retail\_db\\order\_item\\part-00000”)

ordersitem.take(5).foreach(println)

1,1,957,1,299.98,299.98

2,2,1073,1,199.99,199.99

3,2,502,5,250.0,50.0

4,2,403,1,129.99,129.99

5,4,897,2,49.98,24.99

val ordersitemMap=ordersitem.map(rec => (rec.split(‘,’)(0).toInt , rec.split(‘,’)(4).toDouble))

ordersItemsMap: org.apache.spark.rdd.RDD[(Int, Double)] = MapPartitionsRDD[11] at map at <console>:16

Using Join

val ordersjoin=ordersfilterMap.join(ordersItemsMap)

ordersjoin: org.apache.spark.rdd.RDD[(Int, (String, Double))] = MapPartitionsRDD[31] at join at <console>:20

ordersjoin.take(5).foreach(println)

(65722,(2014-05-23 00:00:00.0,129.99))

(31037,(2014-02-02 00:00:00.0,179.97))

(34207,(2014-02-20 00:00:00.0,199.99))

(51620,(2014-06-13 00:00:00.0,129.99))

(46781,(2014-05-11 00:00:00.0,399.98))

ordersjoinMap=ordersjoin.map(rec => (rec.\_2))

ordersjoin.take(5).foreach(println)

(2014-05-23 00:00:00.0,129.99)

(2014-02-02 00:00:00.0,179.97)

(2014-02-20 00:00:00.0,199.99)

(2014-06-13 00:00:00.0,129.99)

(2014-05-11 00:00:00.0,399.98)

Now we need to group and calculate the total revenue generated on a single day

There are 3 ways

1. Use aggregateByKey
2. use reduceByKey
3. use groupByKey

val ordersjoinGBK = ordersjoinMap.groupByKey().map(rec => (rec.\_1 , rec.\_2.sum)).take(5).foreach(println)

(2013-10-05 00:00:00.0,20431.799999999985)

(2013-08-11 00:00:00.0,15212.209999999988)

(2013-09-28 00:00:00.0,14486.109999999988)

(2014-06-15 00:00:00.0,11968.36999999999)

(2014-05-17 00:00:00.0,14433.509999999982)

val ordersjoinRBK=ordersjoinMap.reduceByKey((agg,value)=>(agg+value)).take(5).foreach(println)

(2014-06-15 00:00:00.0,11968.36999999999)

(2013-10-05 00:00:00.0,20431.799999999985)

(2013-09-28 00:00:00.0,14486.109999999988)

(2014-05-17 00:00:00.0,14433.509999999982)

(2013-08-11 00:00:00.0,15212.209999999988)

val ordersjoinMapABK=ordersjoinMap.aggregateByKey(0.0)(

| (intAgg, intVal) => intAgg + intVal,

| (finAgg,finVal) => finAgg + finVal)

ordersjoinMapABK: org.apache.spark.rdd.RDD[(String, Double)] = ShuffledRDD[34] at aggregateByKey at <console>:23

Difference between groupByKey and aggregateByKey

groupByKey just groups the data while aggregateByKey also uses combiner and reducer ie.. it can use for purposes like add , count directly

Program with Path in/out path Validations

**import** com.typesafe.config.ConfigFactory  
**import** org.apache.spark.{SparkConf, SparkContext}  
**import** org.apache.hadoop.fs.\_  
  
**object** retial\_order {  
 **def** main(args: Array[String]): Unit = {  
  
 **val** props=ConfigFactory.*load*()  
 **val** conf = **new** SparkConf().  
 setMaster(props.getConfig(args(2)).getString(**"executionMode"**)).  
 setAppName(**"Total Revenue Collection"**)  
  
 **val** sc = **new** SparkContext(conf)  
  
 **val** inputBaseDir=args(0)  
 **val** outputBasedir=args(1)  
  
 **val** fs = FileSystem.*get*(sc.hadoopConfiguration)  
 **if**(!fs.exists(**new** Path(inputBaseDir)))  
 {  
 *println*(**"Base Directory Does not exists"**)  
 **return** }  
  
 **if**(fs.exists(**new** Path(outputBasedir)))  
 fs.delete(**new** Path(outputBasedir),**true**)  
  
  
  
  
 **val** ordersFilterMap = sc.textFile(inputBaseDir +**"\\orders\\part-00000"**).  
 filter(rec => (rec.split(**','**)(3) == **"CLOSED"** || rec.split(**','**)(3) == **"COMPLETE"**)).  
 map(rec => (rec.split(**','**)(0).toInt, rec.split(**','**)(1)))  
  
 **val** orderItemMap = sc.textFile(inputBaseDir+**"\\order\_items\\part-00000"**).  
 map(rec => (rec.split(**','**)(0).toInt, rec.split(**','**)(4).toDouble))  
  
 **val** totalrevenue=ordersFilterMap.join(orderItemMap).  
 map(rec=> (rec.\_2)).  
 groupByKey().  
 map(rec=>(rec.\_1,rec.\_2.sum)).  
 saveAsTextFile(outputBasedir)  
 }  
}