Spark :

* Only one SC can run per JVM
* spark.driver.allowMultipleContexts = true. , We can change but it is not recommended as we get unexpected results
* Creating sqlContext as it is not available as part of Spark 2.4.5
* val sqlContext=new org.apache.spark.sql.SQLContext(sc)
* import sqlContext.implicits.\_
* GroupBykey does not have combiner

**Reading Json file :**

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

import sqlContext.implicits.\_

**val orderjson=sqlContext.read.json("D:\\study\_material\\scalapractise\\csvfiles\\data-master (1)**[**\\data-master\\retail\_db\_json\\orders**](file:///\\data-master\\retail_db_json\\orders)**")**

**orderjson.printSchema**

**orderjson.select(“order\_id”)**

val orderjson=sqlContext.load("D:\\study\_material\\scalapractise\\csvfiles\\data-master (1)[\\data-master\\retail\_db\_json\\orders","json](file:///\\data-master\\retail_db_json\\orders%22,%22json)")

**Spark-Scala Programme**

import org.apache.spark.SparkContext  
import org.apache.spark.SparkConf  
  
object order{  
 def main (args:Array[String]): Unit = {  
  
 val inputfile="D:\\study\_material\\scalapractise\\csvfiles\\wordcount"  
 val outfile="D:\\study\_material\\scalapractise\\csvfiles\\outwordcount"  
 val conf =new SparkConf().setMaster("local[\*]").setAppName("wordcount")  
 val sc=new SparkContext(conf)  
 val input=sc.textFile(inputfile)  
 val words=input.flatMap( x => x.split(" "))  
 val wordscount=words.map( x => (x,1)).reduceByKey(\_+\_)  
 wordscount.coalesce(1).saveAsTextFile(outfile)  
 }  
}

**Denfining SQLCcontext variable**

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

**1.Fetch the the data part and convert to Integer**

**val orders=sc.textFile("D:\\study\_material\\scalapractise\\csvfiles\\orders")**

**orders.map( x => x.split(",")(1).substring(0,10).replace("-","").toInt).take(10)**

**orders.map{ x=> val y=x.split(","); val z=y(1).split(" ")(0).replace("-","").toInt;z}.distinct.take(10).foreach(println)**

**Assignment 1:**

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

OutPut: (1,20130715)

Solution: orders.map( x => { val o=x.split(","); (o(0).toInt,o(1).substring(0,10).replace("-","").toInt) }).foreach(println)

orders.map{ x=> val y=x.split(",");val date\_part=y(1).substring(0,10).replace("-","").toInt;(y(0),date\_part)}.take(10).foreach(println)

**Exercise 2: Word Count**

word\_rdd.flatMap( \_.split(" ")).map( (\_,1)).reduceByKey(\_+\_).collect()

**Example 3 :**

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

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

**OutPut**

* Status wise count

( orders.map( x => { val o=x.split(","); (o(3),1) } ).reduceByKey(\_+\_).collect())

* Find records where status is closed and Completed and in July Month

orders.filter( x => { val o=x.split(","); (o(3)=="CLOSED" || o(3)=="COMPLETE") && ( o(1).contains("2013-09"))}).foreach(println)

**Example 4**

Join Two rdd and find the orderdate and and order price for order id

val orders=sc.textFile("D:\\study\_material\\scalapractise\\csvfiles\\orders")

val ordersitems=sc.textFile("D:\\study\_material\\scalapractise\\csvfiles\\order\_items")

val orderjoin=orders.map( x => { val o=x.split(","); (o(0).toInt,o(1).substring(0,10).replace("-","").toInt)})

val orderitemsjoin=ordersitems.map( x => { val o=x.split(","); (o(1).toInt,o(4).toFloat) } )

output: val joinout=orderjoin.join(orderitemsjoin)

(58423,(20130820,499.95))

(53179,(20140625,399.96))

(53179,(20140625,399.98))

Example 5:

Joins:

val ordersmap=orders.map( x => (x.split(",")(0).toInt,x))

val ordersitemmap=ordersitems.map( x => ( x.split(",")(1).toInt,x))

val orderleftjoin=ordersmap.leftOuterJoin(ordersitemmap)

ordersitemmap.filter( x => x.\_1==65722).foreach(println)

val orderleftjoin=ordersmap.leftOuterJoin(ordersitemmap)

Find those records which are not available in

orderleftjoin.filter( x => x.\_2.\_2==None).take(10).foreach( println)

**Example Finding minimum element by using reduce function**

Input: val o=List(3,7,9,2)

Solution: o.reduce( (x,y) => { if ( x<y) x else y })

**Example** :**Finding maximum revenue from orderitems**

ordersitems.map( x => x.split(",")(4).toFloat).reduce( (x,y) => { if ( x<y) x else y } )

**Example : Summing revenue by order id and Compact buffer**

val ordersitems=sc.textFile("D:\\study\_material\\scalapractise\\csvfiles\\order\_items")

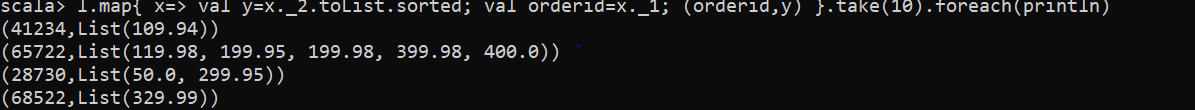
val l=ordersitems.map{ x=> val o=x.split(","); (o(1).toInt,o(4).toFloat)}.groupByKey()

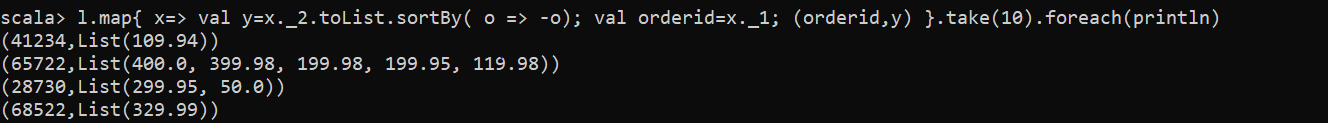
l.map{ x => val orderid=x.\_1; val y=x.\_2.toList.sum ; (orderid,y) }.take(10).foreach(println)

orderitems.map{ x=> val y=x.split(","); (y(1).toInt,y(2).toFloat)}.groupByKey.map{ x=> val orderid=x.\_1; val agg=x.\_2.toList.min; (orderid,agg)}.take(10)

**Example : sort the element in values**

l.map{ x => val orderid=x.\_1; val y=x.\_2.toList.sum ; (orderid,y) }.take(10).foreach(println)





l.flatMap{ x => x.\_2.toList.sortBy( o=> -o).map( rec => (x.\_1,rec)) }.take(10).foreach(println)

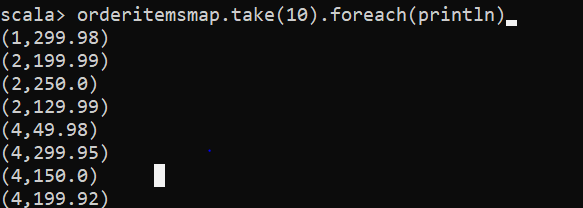
If we put minus then it reverse the sorting

**Example Finding minimum element for every ordered**

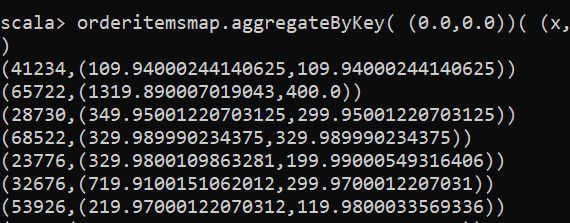
ordersitems.map{ x => val o=x.split(","); ( o(1).toInt,o(4).toFloat) }.reduceByKey{ (x,y) => if( x>y) x else y}.take(10).foreach(println)

Example : Aggregate by logic

orderitemsmap.aggregateByKey( (0.0,0) ) ( (x,y) => ( x.\_1+y,x.\_2+1),(x,y)=>(x.\_1+y.\_1,x.\_2+y.\_2))

Input: 

Output:



Solution: **orderitemsmap.aggregateByKey( (0.0,0.0))( (x,y) => ( x.\_1+y, if( y > x.\_2) y else x.\_2),(x,y) => (x.\_1+y.\_1, if(x.\_2>y.\_2) x.\_2 else y.\_2 )).take(10).foreach(println)**

**OutPut of green Highlight is input to Yellow highlighted function**

**Example : Sort the data by a key**

val products=sc.textFile("D:\\study\_material\\scalapractise\\csvfiles\\products")

val productsmap=products.map{ x=> val o=x.split(","); (o(1).toInt,x)}

productsmap.sortByKey(false).take(10).foreach(println)

***CompositeKey Sorting***

products.filter( x => x.split(",")(4) !="").map{ x=> val o=x.split(","); ( (o(1).toInt,o(4).toFloat),x) }.sortByKey()

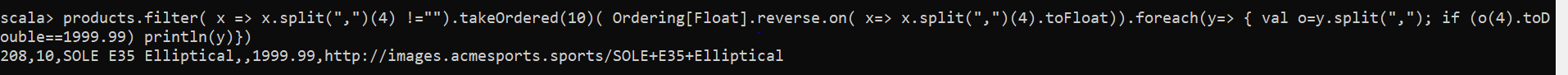
products.filter( x=> x.split(",")(4)!="").map{ x=> val y=x.split(","); ((y(1).toInt,-y(4).toFloat),x)}.sortByKey().take(100).foreach(println)

Example 11 : Ranking

products.filter( x => x.split(",")(4) !="").takeOrdered(10)( Ordering[Float].reverse.on( x=> x.split(",")(4).toFloat)).foreach(y=> { val o=y.split(","); if (o(4).toDouble==1999.99) println(y)})

products.filter{ x=> val y=x.split(","); y(4)!=""}.takeOrdered(10)(Ordering[Float].reverse.on( x=> x.split(",")(4).toFloat)).foreach{ x=> val y=x.split(","); if(y(4).toDouble==1799.99) println(x) }

takeOrdered allow us to take elements by passing the function as second argument.As you can we passed second currying function in Green Highlighted



Example : Reducebykey(\_++\_)

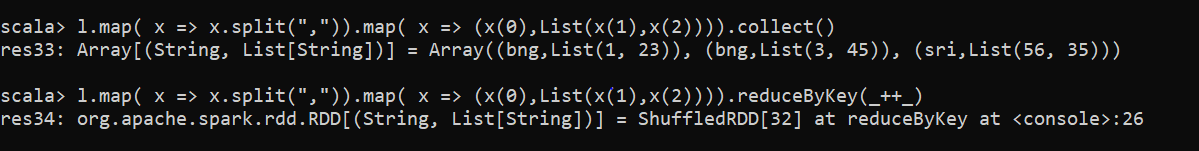
**Input:**

bng,1,23

bng,3,45

sri,56,35

**Output**: Array[(String, List[String])] = Array((sri,List(56, 35)), (bng,List(1, 23, 3, 45)))



Example 10:

By Key Ranking - Converting (K, V) pairs into (K, Iterable[V]) using groupByKey

val productgroup=products.filter{ x=> val y=x.split(","); y(4)!=""}.map{ x=> val y=x.split(","); (y(1).toInt,x)}.groupByKey()

Sorting products data by price

products.filter{ x=> x.split(",")(4)!=""}.map{ x=> val y=x.split(","); (y(4).toFloat,x)}.sortBy( x=> -x.\_1).map{ x=> x.\_2}.take(10).foreach(println)

Example 11: **Finding top prices from every product category:**

val products = sc.textFile("/public/retail\_db/products")

val productsMap = products.

filter(product => product.split(",")(4) != "").

map(product => (product.split(",")(1).toInt, product))

val productsGroupByCategory = productsMap.groupByKey

def getTopNPricedProducts(productsIterable: Iterable[String], topN: Int): Iterable[String] = {

val productPrices = productsIterable.map(p => p.split(",")(4).toFloat).toSet

val topNPrices = productPrices.toList.sortBy(p => -p).take(topN)

val productsSorted = productsIterable.toList.sortBy(product => -product.split(",")(4).toFloat)

val minOfTopNPrices = topNPrices.min

val topNPricedProducts = productsSorted.takeWhile(product => product.split(",")(4).toFloat >= minOfTopNPrices)

topNPricedProducts

}

val top3PricedProductsPerCategory = productsGroupByCategory.flatMap(rec => getTopNPricedProducts(rec.\_2, 3))

Note :We can top three prices for every product category

