**RDD operations in Spark Core**

**// reading data into rdd**

**val textFile = sc.textFile("hdfs://localhost: path")**

**// mapping each row as cols seperated with tab**

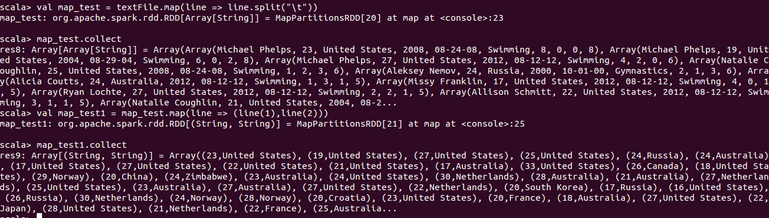
**val map\_test = textFile.map(line => line.split("\t"))**

**// printing the above rdd**

**map\_text.collect**

**// selecting 2,3 col values**

**val =map\_text1 = map\_text.map(line => (line(1),line(2)))**



**// filtering wheather the col 3 having india as value**

**val fil = map\_text.filter(line => line(2).contains("india"))**

**fil.collect**



**// getting count of medals own by each unique country**

**val map\_text2 = map\_text.map(line =>line(2),line(9).toInt))**

**map\_text2.reduceByKey(\_+\_).collect**



**// Here in this scenario, we have taken a pair of Country and total medals columns as key and value and we are performing reduceByKey operation on the RDD.**

**// We have got the final result as country and the total number of medals won by each country in all the Olympic games.**

**// count is used to return number of elements in the rdd**

**// selecting first and third col values**

**val map\_text2 = map\_text.map(line => line(0),line(2)))**

**map\_text2.count**

**//you can see that there are 8618 records in the RDD map\_test1.**

**CountByValue:  
countByValue**is used to count the number of occurrences of the elements in the RDD. Below is the sample demonstration of the above scenario.

**Val map.text1 =map\_text.map(line => line(2),line(5)))**

**map\_text1.countByValue**



In the above scenario, we have taken a pair of Country and Sport. By performing countByValue action we have got the count of each country in a particular sport.

**Reduce:**Below is the sample demonstration of the above scenario where we have taken the total number of medals column in the dataset and loaded into the RDD map\_test1. On this RDD we are performing reduce operation. Finally, we have got that there is a total of **9529** medals declared as the winners in Olympic.

val map\_text3 = map\_text.map(line => line(9).toInt))

**map\_text3.reduce((a,b) => a+b)**

**>> res: Int =9529**

**Take:  
take**will display the number of records we explicitly specify. Below is the sample demonstration of the above scenario.

**val map\_text4 = map\_text.map(line => line(2),line(9).toInt))**

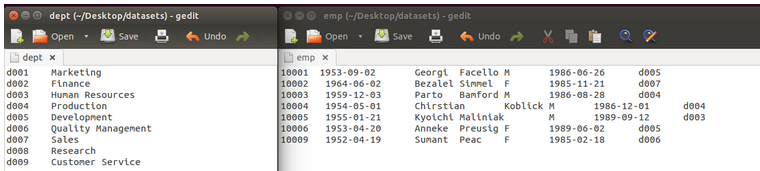
**map.text4.reduceByKey(\_+\_).collect**

**// above command gives all the records in the rdd, to print only first 2 records we use below command**

**map\_text4.reduceByKey(\_+\_).take(2)**

**Advance RDD Operations**

Here, we have taken two datasets, **dept** and **emp**,



#### RDD’s holding Objects:

Here, by using the **case class,** we will declare one object and will pass this case class as parameter to the RDD. You can refer to the below screen shot for the same.

**val dpt = sc.textFile(“hdfs://local: path”)**

**case class dept(id:String,name:String)**

#### val split\_dpt = dpt.map(line=> line.split(“\t”)).map(line => (dept(line(0),line(1))))

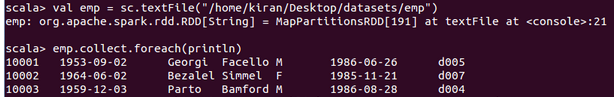
#### split\_dpt.collect

#### 

#### Foreach:

The foreach operationis used to iterate every element in the spark RDD. You can refer to the below screen shot for the same.

**emp.collect.foreach(println)**



In the above screen shot, you can see that every element in the spark RDD emp are printed in a separate line.

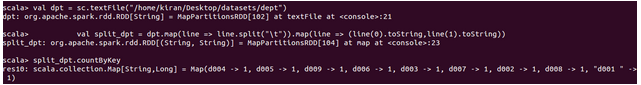
#### CountByKey:

The CountByKEy operation returns the number of elements present for each key. You can refer to the below screenshot for the same.

Val dpt = sc.textFile(“hdfs://local:path”)

Val split\_dpt =dpt.map(line => line.split(“\t”)).map(line => (line(0).toString, line(1).toString))

Split\_dpt.countByKey



Here, we have loaded the dataset and split the records by using tab as delimiter and created the pair as DeptNo and DeptName. Then, we have performed CountByKey operation and the result is as displayed.

#### SaveAsTextFile:

The SaveAsTExtFile operation stores the result of the RDD in a text File in the given output path. You can refer to the below screenshot for the same.

#### split\_dpt.saveAsTextFile(“hdfs://local: path”)

#### Keys:

The Keys operationis used to print all the keys in the RDD pair. You can refer to the below screen shot for the same.

Val emp = sc.textFile(“hdfs”://local)

Import org.apache.spark.rdd.RDD

Val pairs:RDD[String, Int)] = emp.map(line => line.split(“\t”)).map(line => (line(2).toString,line(0).toInt))

pairs.keys.collect



#### Values:

The Values operationis used to print all the values in the RDD pair. You can refer to the below screen shot for the same.

Val pairs:RDD[String, Int)] = emp.map(line => line.split(“\t”)).map(line => (line(2).toString,line(0).toInt))

pairs.values.collect



#### SortByKey:

The SortByKey operation returns the RDD that contains the key value pairs sorted by Keys. SortByKey accepts arguments true/false. ‘False’ will sort the keys in descending order and ‘True’ will sort the keys in ascending order. You can refer to the below screen shot for the same.l

Val pairs:RDD[String, Int)] = emp.map(line => line.split(“\t”)).map(line => (line(2).toString,line(0).toInt))

pairs.sortByKey(false).collect

pairs.sortByKey(true).collect



**Union:**

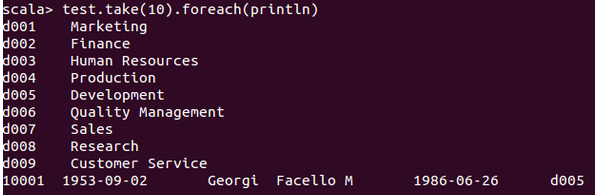
The Union operation results in an RDD which contains the elements of both the RDD’s. You can refer to the below screen shot to see how the Union operation performs.

val dpt = sc.textFile(“hdfs://local: path”)

val emp = sc.textFile(“hdfs://local: path”)

**val test = dpt.union(emp)**

**test.take(10).foreach(println)**



Here, we have created two RDDs and loaded the two datasets into them. We have performed Union operation on them, and from the result, you can see that both the datasets are combined and have printed the first 10 records of the newly obtained spark RDD. Here the 10th record is the first record of the second dataset.

#### Intersection:

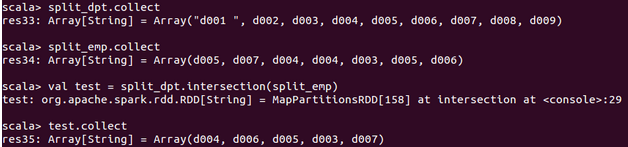
Intersection returns the elements of both the RDD’s. Refer the below screen shot to know how to perform intersection.

**Val split\_dpt = dpt.map(line => line.split(“\t”)).map(line =>line(0) )**

**Val split\_emp = emp.map(line => line.split(“\t”)).map(line =>line(6))**

**val test = split\_dpt.intersection(split\_emp)**

**test.collect**



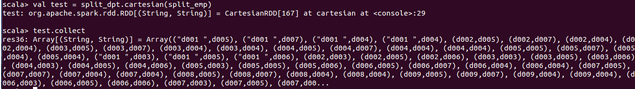
Here we have split the datasets by using tab delimiter and have extracted the 1st column from the first dataset and the 7th column from the second dataset. We have also performed intersection on the datasets and the result is as displayed.

#### Cartesian:

The Cartesian operationwill return the RDD containing the Cartesian product of the elements contained in both the RDDs. You can refer to the below screen shot for the same.

**val test = split\_dpt.cartesian(split\_emp)**

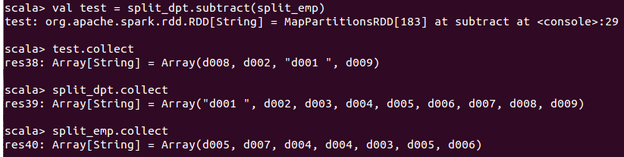
**test.collect**



Here we have split the datasets by using tab delimiter and have extracted 1st column from the first dataset and 7th column from the second dataset. Then, we have performed the Cartesian operation on the RDDs and the results are displayed.

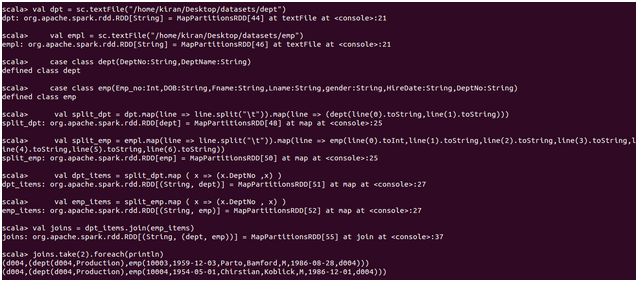
#### Subtract:

The Subtract operation will remove the common elements present in both the RDDs. You can refer to the below screen shot for the same.



#### Join:

The Join operation is used to join two RDDs. The default Join will be Inner join. You can refer to the below screen shot for the same.



Here, we have taken two case classes for the two datasets and have created two RDDs with the two datasets as the common element as key and the rest of the contents as value and have performed Join operation on the RDDs and the result is as displayed on the screen.

#### RighOuterJoin:

The RightOuterJoin operation returns the joined elements of both the RDDs, where the key must be present in the first RDD. You can refer to the below screen shot for the same.



Here, we have taken two case classes for the two datasets and have created two spark RDDs with the two datasets as the common element as key and the rest of the contents as values and we have performed rightOuterJoin operation on the RDDs and the result is as displayed on the screen.

#### LeftOuterJoin:

The LeftOuterJoin operation returns the joined elements of both the RDDs, where the key must be present in the second RDD. You can refer to the below screen shot for the same.

