**SHELL COMMAND**

----------------------------------------------------------------------------------

get shell command and change the permissions to

-rw-rw-r-- ==>> drwxrwxr-x

chmod 764 **sqoop.sh** --->>only owner can excute

chmod 774 **sqoop.sh** --->>others can also excute

first view shell command

how to run = **./sqoop.sh**

**----------------------------------------------------------------------------------------------**

**NUMBER OF ORDERS BY STATUS**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

ordersMap = ordersRDD.map(lambda rec: (rec.split(",")[3], 1))

**-------------------------------------------**

**------We Can Use Any Of This Three Methords------**

**-------------------------------------------**

for i in ordersMap.countByKey().items(): print(i) It expects a K,V pair

for i in ordersMap.reduceByKey(lambda x,y: x+y).collect: print(i)

ordersByStatus = ordersMap.**groupByKey().map(lambda t: (t[0], sum(t[1])))**

---------------------------------------------------------------------------------------------------

ordersMap = ordersRDD.map(lambda rec: (rec.split(",")[3], rec)) **[[[or we need to pass entire record]]]**

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**ordersByStatus = ordersMap.aggregateByKey(0, lambda acc, value: acc+1, lambda acc, value: acc+value)//revenue per day**

**ordersByStatus = ordersMap.combineByKey(lambda value: 1, lambda acc, value: acc+1, lambda acc, value: acc+value)**

==========================================================

**LOADING DATA**

----------------------------------------------------------------------------------------------------

dataRDD **= sc.textFile**("/user/cloudera/sqoop\_import/departments")

for line in dataRDD.collect():

print(line)

)(or)( **print(dataRDD.count())**

**=========================================================**

**SAVING**

**==========================================================**

dataRDD.**saveAsTextFile**("/user/cloudera/pyspark/departments")

dataRDD.**saveAsSequenceFile**("/user/cloudera/pyspark/departmentsSeq")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

dataRDD.map(lambda x: (None, x)).saveAsSequenceFile("/user/cloudera/pyspark/departmentsSeq")

dataRDD.map(lambda x: tuple(x.split(",", 1))).saveAsSequenceFile("/user/cloudera/pyspark/departmentsSeq")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**READING FILE**

------------------------------------------------------------------------------------------------

**Sc.sequrnceFile**(“/user/cloudera/pyspark/departmentsSeq”)

==========================================================

**Writing data in json format**

----------------------------------------------------------------------------------------------------

departmentsData.**toJSON().saveAsTextFile**("/user/cloudera/pyspark/departmentsJson")

===========================================================

**WORD COUNT**

----------------------------------------------------------------------------------------------------

data = sc.textFile("/user/cloudera/wordcount.txt")

dataFlatMap = data**.flatMap(lambda x: x.split(" "))**

dataMap = dataFlatMap**.map(lambda x: (x, 1))**

dataReduceByKey = dataMap**.reduceByKey(lambda x,y: x + y)------->> there will be comma between x and y**

dataReduceByKey.**saveAsTextFile**("/user/cloudera/wordcountoutput")

===========================================================

**JOINING DATA SETS**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

orderItemsRDD = sc.textFile("/user/cloudera/sqoop\_import/order\_items")

ordersParsedRDD = ordersRDD**.map(lambda rec: (int(rec.split(",")[0]), rec))**

orderItemsParsedRDD = orderItemsRDD.**map(lambda rec: (int(rec.split(",")[1]), rec))**

ordersJoinOrderItems = orderItemsParsedRDD**.join**(ordersParsedRDD)

===========================================================

**NOTE OBSORVE FLOAT WE ONLY GIVE LAMBDA FUNCTION ONE TIME**

----------------------------------------------------------------------------------------------------

revenuePerOrderPerDay = ordersJoinOrderItems.map(lambda t: (t[1][1].split(",")[1], float(t[1][0].split(",")[4])))

===========================================================

**TOP CUSTOMER**

**-----------------------------------------------------------------------------------------**

revenuePerDayPerCustomerMap = revenuePerDayPerCustomer**.map(lambda rec: (rec[0][0], (rec[0][1], rec[1])))**

topCustomerPerDaybyRevenue = revenuePerDayPerCustomerMap.**reduceByKey(lambda x, y: (x if x[1] >= y[1] else y))**

**GET ORDER COUNT PER DAY**

----------------------------------------------------------------------------------------------------

**ordersPerDay = ordersJoinOrderItems.map(lambda rec: rec[1][1].split(",")[1] + "," + str(rec[0])).distinct()**

ordersPerDayParsedRDD = ordersPerDay.map(lambda rec: (rec.split(",")[0], 1))

totalOrdersPerDay = ordersPerDayParsedRDD.reduceByKey(lambda x, y: x + y)

===========================================================

**GET REVENUE PER DAY FROM JOINED DATA**

----------------------------------------------------------------------------------------------------

totalRevenuePerDay = revenuePerOrderPerDay.reduceByKey( \

lambda x, y: x + y **) ------->> there will be comma between x and y**

===========================================================

**JOINING ORDER COUNT PER DAY AND REVENUE PER DAY**

----------------------------------------------------------------------------------------------------

finalJoinRDD = totalOrdersPerDay.join(totalRevenuePerDay)

for data in finalJoinRDD.take(5):

print(data)

===========================================================

**JOINING USING HIVE**

----------------------------------------------------------------------------------------------------

from pyspark.sql import HiveContext

sqlContext = HiveContext(sc)

sqlContext.sql("set spark.sql.shuffle.partitions=10");

joinAggData = sqlContext.sql("select o.order\_date, round(sum(oi.order\_item\_subtotal), 2), \

count(distinct o.order\_id) from orders o join order\_items oi \

on o.order\_id = oi.order\_item\_order\_id \

group by o.order\_date order by o.order\_date")

for data in joinAggData.collect():

print(data)

===========================================================

**SUM**

----------------------------------------------------------------------------------------------------

COUNT NUMBER OF RECORDS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

ordersRDD.**count()**

++++++++++++++++++++++++++++++++++++++++++++++++++++++

TOTAL(ADDING if we have 1

2

3 as data set this code will give you 6 as output

orderItemsRDD = sc.textFile("/user/cloudera/sqoop\_import/order\_items")

orderItemsMap = orderItemsRDD.map(lambda rec: float(rec.split(",")[4]))

for i in orderItemsMap.take(5):

print i

orderItemsReduce = orderItemsMap.**reduce**(lambda rev1, rev2: rev1 + rev2**)---->>observe just REDUCE not REDUCE BY KEY**

===========================================================

**MAX PRICED PRODUCT**

----------------------------------------------------------------------------------------------------

productsRDD = sc.textFile("/user/cloudera/sqoop\_import/products")

productsMap = productsRDD.map(lambda rec: rec)

**productsMap.reduce(lambda rec1, rec2: (rec1 if((rec1.split(",")[4] != "" and rec2.split(",")[4] != "") and float(rec1.split(",")[4]) >= float(rec2.split(",")[4])) else rec2))**

===========================================================

**AVERAGE**

----------------------------------------------------------------------------------------------------

revenue = sc.textFile("/user/cloudera/sqoop\_import/order\_items").map(lambda rec: float(rec.split(",")[4])).reduce(lambda rev1, rev2: rev1 + rev2)

totalOrders = sc.textFile("/user/cloudera/sqoop\_import/order\_items").map(lambda rec: int(rec.split(",")[1])).distinct().count()

===========================================================

**NUMBER OF ORDERS BY ORDER DATE AND ORDER STATUS**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

ordersMapRDD = ordersRDD.map(lambda rec: ((rec.split(",")[1], rec.split(",")[3]), **1))**

ordersByStatusPerDay = ordersMapRDD.reduceByKey(lambda v1, v2: v1+v2)

for i in ordersByStatusPerDay.collect():

print(i)

===========================================================

**TOTAL REVENUE PER DAY**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

orderItemsRDD = sc.textFile("/user/cloudera/sqoop\_import/order\_items")

ordersParsedRDD = ordersRDD.map(lambda rec: (rec.split(",")[0], rec))

orderItemsParsedRDD = orderItemsRDD.map(lambda rec: (rec.split(",")[1], rec))

ordersJoinOrderItems = orderItemsParsedRDD.join(ordersParsedRDD)

ordersJoinOrderItemsMap = ordersJoinOrderItems.map(lambda t: (t[1][1].split(",")[1], float(t[1][0].split(",")[4])))

revenuePerDay = ordersJoinOrderItemsMap.reduceByKey(lambda acc, value: acc + value)

for i in revenuePerDay.collect(): print(i)

===========================================================

**REVENUE PER DAY PER ORDER**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

orderItemsRDD = sc.textFile("/user/cloudera/sqoop\_import/order\_items")

ordersParsedRDD = ordersRDD.map(lambda rec: (rec.split(",")[0], rec))

orderItemsParsedRDD = orderItemsRDD.map(lambda rec: (rec.split(",")[1], rec))

ordersJoinOrderItems = orderItemsParsedRDD.join(ordersParsedRDD)

ordersJoinOrderItemsMap = ordersJoinOrderItems.map(lambda t: ((t[1][1].split(",")[1], t[0]), float(t[1][0].split(",")[4])))

revenuePerDayPerOrder = ordersJoinOrderItemsMap.reduceByKey(lambda acc, value: acc + value)

revenuePerDayPerOrderMap = revenuePerDayPerOrder.map(lambda rec: (rec[0][0], rec[1]))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

revenuePerDay = revenuePerDayPerOrderMap.combineByKey( \

lambda x: (x, 1), \

lambda acc, revenue: (acc[0] + revenue, acc[1] + 1), \

lambda x, y: (round(x[0] + y[0], 2), x[1] + y[1]) )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

revenuePerDay = revenuePerDayPerOrderMap.aggregateByKey( \

(0, 0), \

lambda acc, revenue: (acc[0] + revenue, acc[1] + 1), \

lambda x, y: (round(x[0] + y[0], 2), x[1] + y[1])

===========================================================

**AVERAGE REVENUE PER DAY**

----------------------------------------------------------------------------------------------------

avgRevenuePerDay = revenuePerDay.map(lambda x: (x[0], x[1][0]/x[1][1])

===========================================================

**CUSTOMER ID WITH MAX REVENUE**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

orderItemsRDD = sc.textFile("/user/cloudera/sqoop\_import/order\_items")

ordersParsedRDD = ordersRDD.map(lambda rec: (rec.split(",")[0], rec))

orderItemsParsedRDD = orderItemsRDD.map(lambda rec: (rec.split(",")[1], rec))

ordersJoinOrderItems = orderItemsParsedRDD.join(ordersParsedRDD)

ordersPerDayPerCustomer = ordersJoinOrderItems.map(lambda rec: ((rec[1][1].split(",")[1], rec[1][1].split(",")[2]), float(rec[1][0].split(",")[4])))

revenuePerDayPerCustomer = ordersPerDayPerCustomer.reduceByKey(lambda x, y: x + y)

revenuePerDayPerCustomerMap = revenuePerDayPerCustomer.map(lambda rec: (rec[0][0], (rec[0][1], rec[1])))

topCustomerPerDaybyRevenue = revenuePerDayPerCustomerMap**.reduceByKey(lambda x, y: (x if x[1] >= y[1] else y))**

**#Using regular function**

def findMax(x, y):

if(x[1] >= y[1]):

return x

else:

return y

topCustomerPerDaybyRevenue = revenuePerDayPerCustomerMap.reduceByKey(lambda x, y: findMax(x, y))

**FITER**

----------------------------------------------------------------------------------------------------

ordersRDD = sc.textFile("/user/cloudera/sqoop\_import/orders")

for i in ordersRDD.filter(lambda x: x.split(",")[3] == "COMPLETE").take(5): print(i)

output = 3,2013-07-25 00:00:00.0,12111,COMPLETE we get only completed orders

dataRDD.**saveAsTextFile**("/user/cloudera/wordcountoutput")

for i in ordersRDD.filter(lambda x: "PENDING" in x.split(",")[3]).take(5): print(i)

OUTPUT: 3,2013-07-25 00:00:00.0,12111,COMPLETE we get only completed orders

dataRDD.**saveAsTextFile**("/user/cloudera/wordcountoutput")

for i in ordersRDD.filter(lambda x: int(x.split(",")[0]) > 100).take(5): print(i)

OUTPUT: 101,2013-07-25 00:00:00.0,5116,CLOSED we get only >100 orders

dataRDD.**saveAsTextFile**("/user/cloudera/wordcountoutput")

for i in ordersRDD.filter(lambda x: int(x.split(",")[0]) > 100 or x.split(",")[3] in "PENDING").take(5): print(i)

OUTPUT: 21,2013-07-25 00:00:00.0,2711,PENDING order with pending we get orders with >100 &we get orders wth pending

112,2013-07-26 00:00:00.0,5375,CANCELED order >100

dataRDD.**saveAsTextFile**("/user/cloudera/wordcountoutput")

for i in dataRDD.filter(lambda line: int(line.split(",")[0]) < 1000 and ("PENDING" in line.split(",")[3] or line.split(",")[3] == ("CANCELLED"))).collect(): print(i)

for i in dataRDD.filter(lambda x: int(x.split(",")[0]) > 1000 and x.split(",")[3] != ("COMPLETE")).collect(): print(i)

OUTPUT:

68655,2014-07-13 00:00:00.0,84,PENDING\_PAYMENT

68657,2014-07-13 00:00:00.0,9211,CLOSED

except complete you will get all the orders above 1000

===========================================================

**SORTING AND RANKING**

----------------------------------------------------------------------------------------------------

orders = sc.textFile("/user/cloudera/sqoop\_import/orders")

for i in orders.map(lambda rec: (int(rec.split(",")[0]), rec)).**sortByKey().**collect(): print(i)

OUTPUT: (6, u'68882,2014-07-22 00:00:00.0,10000,ON\_HOLD')

.

.

(68883, u'68883,2014-07-23 00:00:00.0,5533,COMPLETE')

DATA will be sorted from small to big(assending order)

for i in orders.map(lambda rec: (int(rec.split(",")[0]), rec)).**sortByKey(False).**take(5): print(i)

OUTPUT: (68883, u'68882,2014-07-22 00:00:00.0,10000,ON\_HOLD')

.

.

(6, u'68883,2014-07-23 00:00:00.0,5533,COMPLETE')

DATA will be sorted from big to small(desending order)

for i in orders.map(lambda rec: (int(rec.split(",")[0]), rec)).**top(2):** print(i)

OUTPUT: will give you top 2 rec

(68883, u'68883,2014-07-23 00:00:00.0,5533,COMPLETE')

(68882, u'68882,2014-07-22 00:00:00.0,10000,ON\_HOLD')

for i in orders.map(lambda rec: (int(rec.split(",")[0]), rec)).**distinct().top(5):** print(i)

OUTPUT: will give you distinct top 2 rec

(68883, u'68883,2014-07-23 00:00:00.0,5533,COMPLETE')

(68882, u'68882,2014-07-22 00:00:00.0,10000,ON\_HOLD')

for i in orders.takeOrdered(5, lambda x: int(x.split(",")[0])): print(i)

or

for i in orders.map(lambda rec: (int(rec.split(",")[0]), rec)).**takeOrdered**(2, lambda x: x[0]): print(i)

OUTPUT: will give you first 2 rec

(1, u'1,2013-07-25 00:00:00.0,11599,CLOSED')

(2, u'2,2013-07-25 00:00:00.0,256,PENDING\_PAYMENT')

for i in orders.takeOrdered(5, lambda x**: -int**(x.split(",")[0])): print(i) watch there is minus

or

for i in orders.map(lambda rec: (int(rec.split(",")[0]), rec**)).takeOrdered**(5, lambda x: -x[0]): print(i)

OUTPUT: will give you lastrec to its cumilative upper records ex: if there are 1-10 numbers it will give you like 10 9 8

(68883, u'68883,2014-07-23 00:00:00.0,5533,COMPLETE')

(68882, u'68882,2014-07-22 00:00:00.0,10000,ON\_HOLD')

(68881, u'68881,2014-07-19 00:00:00.0,2518,PENDING\_PAYMENT')