|  |
| --- |
| #Introduction |
|  |

|  |
| --- |
| #spark-shell |
|  |

|  |
| --- |
| #pyspark |
|  |

|  |
| --- |
| #SparkContext |
|  |

|  |
| --- |
| #SQLContext |
|  |

|  |
| --- |
| #HiveContext |
|  |

|  |
| --- |
| #spark-sql (only latest version) |
|  |

|  |
| --- |
| #JDBC |
|  |

|  |
| --- |
| #To connect to remote database using jdbc |
|  |

|  |
| --- |
| #It works only from spark 1.3.0 or later |
|  |

|  |
| --- |
| #Either you need to run pyspark with driver-class-path or set environment variable with os.environ |
|  |

|  |
| --- |
| pyspark --driver-class-path /usr/share/java/mysql-connector-java.jar |
|  |

|  |
| --- |
| os.environ['SPARK\_CLASSPATH'] = "/usr/share/java/mysql-connector-java.jar" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| from pyspark.sql import SQLContext |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| sqlContext = SQLContext(sc) |
|  |

|  |
| --- |
| jdbcurl = "jdbc:mysql://quickstart.cloudera:3306/retail\_db?user=retail\_dba&password=cloudera" |
|  |

|  |
| --- |
| df = sqlContext.load(source="jdbc", url=jdbcurl, dbtable="departments") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for rec in df.collect(): |
|  |

|  |
| --- |
| print(rec) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| df.count() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ############################################################################## |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Developing simple scala based applications for spark |
|  |

|  |
| --- |
| #Save this to a file with py extension |
|  |

|  |
| --- |
| from pyspark import SparkContext, SparkConf |
|  |

|  |
| --- |
| conf = SparkConf().setAppName("pyspark") |
|  |

|  |
| --- |
| sc = SparkContext(conf=conf) |
|  |

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

|  |
| --- |
| for line in dataRDD.collect(): |
|  |

|  |
| --- |
| print(line) |
|  |

|  |
| --- |
| dataRDD.saveAsTextFile("/user/cloudera/pyspark/departmentsTesting") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Run using this command |
|  |

|  |
| --- |
| #master local will run in spark native mode |
|  |

|  |
| --- |
| spark-submit --master local saveFile.py |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #master yarn will run in yarn mode |
|  |

|  |
| --- |
| spark-submit --master yarn saveFile.py |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ############################################################################## |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Load data from HDFS and storing results back to HDFS using Spark |
|  |

|  |
| --- |
| from pyspark import SparkContext |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| for line in dataRDD.collect(): |
|  |

|  |
| --- |
| print(line) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| print(dataRDD.count()) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| dataRDD.saveAsTextFile("/user/cloudera/pyspark/departments") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Object files are not available in python |
|  |

|  |
| --- |
| dataRDD.saveAsObjectFile("/user/cloudera/pyspark/departmentsObject") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #saveAsSequenceFile |
|  |

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

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

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

|  |
| --- |
| path="/user/cloudera/pyspark/departmentsSeq" |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| dataRDD.map(lambda x: tuple(x.split(",", 1))).saveAsNewAPIHadoopFile(path,"org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat",keyClass="org.apache.hadoop.io.Text",valueClass="org.apache.hadoop.io.Text") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #reading sequence file |
|  |

|  |
| --- |
| data = sc.sequenceFile("/user/cloudera/pyspark/departmentsSeq") |
|  |

|  |
| --- |
| data = sc.sequenceFile("/user/cloudera/pyspark/orders") |
|  |

|  |
| --- |
| data = sc.sequenceFile("/user/cloudera/pyspark/departmentsSeq", "org.apache.hadoop.io.IntWritable", "org.apache.hadoop.io.Text") |
|  |

|  |
| --- |
| for rec in data.collect(): |
|  |

|  |
| --- |
| print(rec) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| from pyspark.sql import HiveContext |
|  |

|  |
| --- |
| sqlContext = HiveContext(sc) |
|  |

|  |
| --- |
| depts = sqlContext.sql("select \* from departments") |
|  |

|  |
| --- |
| for rec in depts.collect(): |
|  |

|  |
| --- |
| print(rec) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| sqlContext.sql("create table departmentsSpark as select \* from departments") |
|  |

|  |
| --- |
| depts = sqlContext.sql("select \* from departmentsSpark") |
|  |

|  |
| --- |
| for rec in depts.collect(): |
|  |

|  |
| --- |
| print(rec) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #We can run hive INSERT, LOAD and any valid hive query in Hive context |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Make sure you copy departments.json to HDFS |
|  |

|  |
| --- |
| #create departments.json on Linux file system |
|  |

|  |
| --- |
| {"department\_id":2, "department\_name":"Fitness"} |
|  |

|  |
| --- |
| {"department\_id":3, "department\_name":"Footwear"} |
|  |

|  |
| --- |
| {"department\_id":4, "department\_name":"Apparel"} |
|  |

|  |
| --- |
| {"department\_id":5, "department\_name":"Golf"} |
|  |

|  |
| --- |
| {"department\_id":6, "department\_name":"Outdoors"} |
|  |

|  |
| --- |
| {"department\_id":7, "department\_name":"Fan Shop"} |
|  |

|  |
| --- |
| {"department\_id":8, "department\_name":"TESTING"} |
|  |

|  |
| --- |
| {"department\_id":8000, "department\_name":"TESTING"} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #copying to HDFS (using linux command line) |
|  |

|  |
| --- |
| hadoop fs -put departments.json /user/cloudera/pyspark |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| from pyspark import SQLContext |
|  |

|  |
| --- |
| sqlContext = SQLContext(sc) |
|  |

|  |
| --- |
| departmentsJson = sqlContext.jsonFile("/user/cloudera/pyspark/departments.json") |
|  |

|  |
| --- |
| departmentsJson.registerTempTable("departmentsTable") |
|  |

|  |
| --- |
| departmentsData = sqlContext.sql("select \* from departmentsTable") |
|  |

|  |
| --- |
| for rec in departmentsData.collect(): |
|  |

|  |
| --- |
| print(rec) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Writing data in json format |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Validating the data |
|  |

|  |
| --- |
| hadoop fs -cat /user/cloudera/pyspark/departmentsJson/part\* |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ############################################################################## |
|  |

|  |
| --- |
| # Developing word count program |
|  |

|  |
| --- |
| # Create a file and type few lines and save it as wordcount.txt and copy to HDFS |
|  |

|  |
| --- |
| # to /user/cloudera/wordcount.txt |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| 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) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| dataReduceByKey.saveAsTextFile("/user/cloudera/wordcountoutput") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for i in dataReduceByKey.collect(): |
|  |

|  |
| --- |
| print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ############################################################################## |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Join disparate datasets together using Spark |
|  |

|  |
| --- |
| # Problem statement, get the revenue and number of orders from order\_items on daily basis |
|  |

|  |
| --- |
| 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) |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| # 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 total1, total2: total1 + total2 \ |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for data in totalRevenuePerDay.collect(): |
|  |

|  |
| --- |
| print(data) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Joining order count per day and revenue per day |
|  |

|  |
| --- |
| finalJoinRDD = totalOrdersPerDay.join(totalRevenuePerDay) |
|  |

|  |
| --- |
| for data in finalJoinRDD.take(5): |
|  |

|  |
| --- |
| print(data) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # 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) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Using spark native sql |
|  |

|  |
| --- |
| from pyspark.sql import SQLContext, Row |
|  |

|  |
| --- |
| sqlContext = SQLContext(sc) |
|  |

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| ordersMap = ordersRDD.map(lambda o: o.split(",")) |
|  |

|  |
| --- |
| orders = ordersMap.map(lambda o: Row(order\_id=int(o[0]), order\_date=o[1], \ |
|  |

|  |
| --- |
| order\_customer\_id=int(o[2]), order\_status=o[3])) |
|  |

|  |
| --- |
| ordersSchema = sqlContext.inferSchema(orders) |
|  |

|  |
| --- |
| ordersSchema.registerTempTable("orders") |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| orderItemsMap = orderItemsRDD.map(lambda oi: oi.split(",")) |
|  |

|  |
| --- |
| orderItems = orderItemsMap.map(lambda oi: Row(order\_item\_id=int(oi[0]), order\_item\_order\_id=int(oi[1]), \ |
|  |

|  |
| --- |
| order\_item\_product\_id=int(oi[2]), order\_item\_quantity=int(oi[3]), order\_item\_subtotal=float(oi[4]), \ |
|  |

|  |
| --- |
| order\_item\_product\_price=float(oi[5]))) |
|  |

|  |
| --- |
| orderItemsSchema = sqlContext.inferSchema(orderItems) |
|  |

|  |
| --- |
| orderItemsSchema.registerTempTable("order\_items") |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| 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) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ############################################################################## |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Calculate aggregate statistics (e.g., average or sum) using Spark |
|  |

|  |
| --- |
| #sum |
|  |

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

|  |
| --- |
| ordersRDD.count() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| 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) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Get max priced product from products table |
|  |

|  |
| --- |
| #There is one record which is messing up default , delimiters |
|  |

|  |
| --- |
| #Clean it up (we will see how we can filter with out deleting the record later) |
|  |

|  |
| --- |
| hadoop fs -get /user/cloudera/sqoop\_import/products |
|  |

|  |
| --- |
| #Delete the record with product\_id 685 |
|  |

|  |
| --- |
| hadoop fs -put -f products/part\* /user/cloudera/sqoop\_import/products |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #pyspark script to get the 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)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #avg |
|  |

|  |
| --- |
| 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 status |
|  |

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

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

|  |
| --- |
| for i in ordersMap.countByKey().items(): print(i) |
|  |

|  |
| --- |
| #groupByKey is not very efficient |
|  |

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

|  |
| --- |
| ordersByStatus = ordersMap.reduceByKey(lambda acc, value: acc + value) |
|  |

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

|  |
| --- |
| ordersByStatus = ordersMap.aggregateByKey(0, lambda acc, value: acc+1, lambda acc, value: acc+value) |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| for recs in ordersByStatus.collect(): |
|  |

|  |
| --- |
| print(recs) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Number of orders by order date and order status |
|  |

|  |
| --- |
| #Key orderDate and orderStatus |
|  |

|  |
| --- |
| 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) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #average |
|  |

|  |
| --- |
| #average revenue per day |
|  |

|  |
| --- |
| #Parse Orders (key order\_id) |
|  |

|  |
| --- |
| #Parse Order items (key order\_item\_order\_id) |
|  |

|  |
| --- |
| #Join the data sets |
|  |

|  |
| --- |
| #Parse joined data and get (order\_date, order\_id) as key and order\_item\_subtotal as value |
|  |

|  |
| --- |
| #Use appropriate aggregate function to get sum(order\_item\_subtotal) for each order\_date, order\_id combination |
|  |

|  |
| --- |
| #Parse data to discard order\_id and get order\_date as key and sum(order\_item\_subtotal) per order as value |
|  |

|  |
| --- |
| #Use appropriate aggregate function to get sum(order\_item\_subtotal) per day and count(distinct order\_id) per day |
|  |

|  |
| --- |
| #Parse data and apply average logic |
|  |

|  |
| --- |
| 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 total1, total2: (round(total1[0] + total2[0], 2), total1[1] + total2[1]) \ |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| revenuePerDay = revenuePerDayPerOrderMap.aggregateByKey( \ |
|  |

|  |
| --- |
| (0, 0), \ |
|  |

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

|  |
| --- |
| lambda total1, total2: (round(total1[0] + total2[0], 2), total1[1] + total2[1]) \ |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for data in revenuePerDay.collect(): |
|  |

|  |
| --- |
| print(data) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| 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)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Using Hive Context |
|  |

|  |
| --- |
| from pyspark.sql import HiveContext |
|  |

|  |
| --- |
| hiveContext = HiveContext(sc) |
|  |

|  |
| --- |
| hiveContext.sql("set spark.sql.shuffle.partitions=10"); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| data = hiveContext.sql(" \ |
|  |

|  |
| --- |
| select \* from ( \ |
|  |

|  |
| --- |
| select o.order\_date, o.order\_customer\_id, sum(oi.order\_item\_subtotal) order\_item\_subtotal \ |
|  |

|  |
| --- |
| from orders o join order\_items oi \ |
|  |

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

|  |
| --- |
| group by o.order\_date, o.order\_customer\_id) q1 \ |
|  |

|  |
| --- |
| join \ |
|  |

|  |
| --- |
| (select q.order\_date, max(q.order\_item\_subtotal) order\_item\_subtotal \ |
|  |

|  |
| --- |
| from (select o.order\_date, o.order\_customer\_id, sum(oi.order\_item\_subtotal) order\_item\_subtotal \ |
|  |

|  |
| --- |
| from orders o join order\_items oi \ |
|  |

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

|  |
| --- |
| group by o.order\_date, o.order\_customer\_id) q \ |
|  |

|  |
| --- |
| group by q.order\_date) q2 \ |
|  |

|  |
| --- |
| on q1.order\_date = q2.order\_date and q1.order\_item\_subtotal = q2.order\_item\_subtotal \ |
|  |

|  |
| --- |
| order by q1.order\_date") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # This query works in hive |
|  |

|  |
| --- |
| select \* from (select q.order\_date, q.order\_customer\_id, q.order\_item\_subtotal, |
|  |

|  |
| --- |
| max(q.order\_item\_subtotal) over (partition by q.order\_date) max\_order\_item\_subtotal |
|  |

|  |
| --- |
| from (select o.order\_date, o.order\_customer\_id, sum(oi.order\_item\_subtotal) order\_item\_subtotal |
|  |

|  |
| --- |
| from orders o join order\_items oi |
|  |

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

|  |
| --- |
| group by o.order\_date, o.order\_customer\_id) q) s |
|  |

|  |
| --- |
| where s.order\_item\_subtotal = s.max\_order\_item\_subtotal |
|  |

|  |
| --- |
| order by s.order\_date; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| select \* from ( |
|  |

|  |
| --- |
| select o.order\_date, o.order\_customer\_id, sum(oi.order\_item\_subtotal) order\_item\_subtotal |
|  |

|  |
| --- |
| from orders o join order\_items oi |
|  |

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

|  |
| --- |
| group by o.order\_date, o.order\_customer\_id) q1 |
|  |

|  |
| --- |
| join |
|  |

|  |
| --- |
| (select q.order\_date, max(q.order\_item\_subtotal) order\_item\_subtotal |
|  |

|  |
| --- |
| from (select o.order\_date, o.order\_customer\_id, sum(oi.order\_item\_subtotal) order\_item\_subtotal |
|  |

|  |
| --- |
| from orders o join order\_items oi |
|  |

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

|  |
| --- |
| group by o.order\_date, o.order\_customer\_id) q |
|  |

|  |
| --- |
| group by q.order\_date) q2 |
|  |

|  |
| --- |
| on q1.order\_date = q2.order\_date and q1.order\_item\_subtotal = q2.order\_item\_subtotal |
|  |

|  |
| --- |
| order by q1.order\_date; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ########################################################################################## |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Filter data into a smaller dataset using Spark |
|  |

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

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| for i in ordersRDD.filter(lambda line: int(line.split(",")[0]) > 1000 and ("PENDING" in line.split(",")[3] or line.split(",")[3] == ("CANCELLED"))).take(5): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for i in ordersRDD.filter(lambda line: int(line.split(",")[0]) > 1000 and line.split(",")[3] != ("COMPLETE")).take(5): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Check if there are any cancelled orders with amount greater than 1000$ |
|  |

|  |
| --- |
| #Get only cancelled orders |
|  |

|  |
| --- |
| #Join orders and order items |
|  |

|  |
| --- |
| #Generate sum(order\_item\_subtotal) per order |
|  |

|  |
| --- |
| #Filter data which amount to greater than 1000$ |
|  |

|  |
| --- |
|  |
|  |

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

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

|  |
| --- |
|  |
|  |

|  |
| --- |
| ordersParsedRDD = ordersRDD.filter(lambda rec: rec.split(",")[3] in "CANCELED").map(lambda rec: (int(rec.split(",")[0]), rec)) |
|  |

|  |
| --- |
| orderItemsParsedRDD = orderItemsRDD.map(lambda rec: (int(rec.split(",")[1]), float(rec.split(",")[4]))) |
|  |

|  |
| --- |
| orderItemsAgg = orderItemsParsedRDD.reduceByKey(lambda acc, value: (acc + value)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ordersJoinOrderItems = orderItemsAgg.join(ordersParsedRDD) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for i in ordersJoinOrderItems.filter(lambda rec: rec[1][0] >= 1000).take(5): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| ########################################################## |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Write a query that produces ranked or sorted data using Spark |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Global 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) |
|  |

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

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

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

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

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

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

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #By key sorting and ranking |
|  |

|  |
| --- |
| def getAllSortByPrice(rec, bool): |
|  |

|  |
| --- |
| if(bool == False): |
|  |

|  |
| --- |
| x = sorted(rec[1], key = lambda k: -float(k.split(",")[4])) |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| x = sorted(rec[1], key = lambda k: float(k.split(",")[4])) |
|  |

|  |
| --- |
| return (y for y in x) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getAll(rec): |
|  |

|  |
| --- |
| return (x for x in rec[1]) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getFirstTwo(rec): |
|  |

|  |
| --- |
| x = [ ] |
|  |

|  |
| --- |
| ctr = 0 |
|  |

|  |
| --- |
| for i in rec[1]: |
|  |

|  |
| --- |
| if(ctr < 2): |
|  |

|  |
| --- |
| x.append(i) |
|  |

|  |
| --- |
| ctr = ctr + 1 |
|  |

|  |
| --- |
| return (y for y in x) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| def getTop(rec): |
|  |

|  |
| --- |
| x = [ ] |
|  |

|  |
| --- |
| max = 0 |
|  |

|  |
| --- |
| for i in rec[1]: |
|  |

|  |
| --- |
| prodPrice = float(i.split(",")[4]) |
|  |

|  |
| --- |
| if(prodPrice > max): |
|  |

|  |
| --- |
| max = prodPrice |
|  |

|  |
| --- |
| for j in rec[1]: |
|  |

|  |
| --- |
| if(float(j.split(",")[4]) == max): |
|  |

|  |
| --- |
| x.append(j) |
|  |

|  |
| --- |
| return (y for y in x) |
|  |

|  |
| --- |
|  |
|  |

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

|  |
| --- |
| productsMap = products.map(lambda rec: (rec.split(",")[1], rec)) |
|  |

|  |
| --- |
| productsGroupBy = productsMap.groupByKey() |
|  |

|  |
| --- |
| for i in productsGroupBy.collect(): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Get data sorted by product price per category |
|  |

|  |
| --- |
| #You can use map or flatMap, if you want to see one record per line you need to use flatMap |
|  |

|  |
| --- |
| #Map will return the list |
|  |

|  |
| --- |
| for i in productsGroupBy.map(lambda rec: sorted(rec[1], key=lambda k: float(k.split(",")[4]))).take(100): print(i) |
|  |

|  |
| --- |
| for i in productsGroupBy.map(lambda rec: sorted(rec[1], key=lambda k: float(k.split(",")[4]), reverse=True)).take(100): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #To get topN products by price in each category |
|  |

|  |
| --- |
| def getTopN(rec, topN): |
|  |

|  |
| --- |
| x = [ ] |
|  |

|  |
| --- |
| x = list(sorted(rec[1], key=lambda k: float(k.split(",")[4]), reverse=True)) |
|  |

|  |
| --- |
| import itertools |
|  |

|  |
| --- |
| return (y for y in list(itertools.islice(x, 0, topN))) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for i in productsMap.groupByKey().flatMap(lambda x: getTopN(x, 2)).collect(): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #To get topN priced products by category |
|  |

|  |
| --- |
| def getTopDenseN(rec, topN): |
|  |

|  |
| --- |
| x = [ ] |
|  |

|  |
| --- |
| topNPrices = [ ] |
|  |

|  |
| --- |
| prodPrices = [ ] |
|  |

|  |
| --- |
| prodPricesDesc = [ ] |
|  |

|  |
| --- |
| for i in rec[1]: |
|  |

|  |
| --- |
| prodPrices.append(float(i.split(",")[4])) |
|  |

|  |
| --- |
| prodPricesDesc = list(sorted(set(prodPrices), reverse=True)) |
|  |

|  |
| --- |
| import itertools |
|  |

|  |
| --- |
| topNPrices = list(itertools.islice(prodPricesDesc, 0, topN)) |
|  |

|  |
| --- |
| for j in sorted(rec[1], key=lambda k: float(k.split(",")[4]), reverse=True): |
|  |

|  |
| --- |
| if(float(j.split(",")[4]) in topNPrices): |
|  |

|  |
| --- |
| x.append(j) |
|  |

|  |
| --- |
| return (y for y in x) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| for i in productsMap.groupByKey().flatMap(lambda x: getTopDenseN(x, 2)).collect(): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| productsFlatMap = products.flatMap(lambda rec: (rec.split(",")[1], float(rec.split(",")[4]))) |
|  |

|  |
| --- |
| for i in productsMap.groupByKey().flatMap(lambda x: getFirstTwo(x)).collect(): print(i) |
|  |

|  |
| --- |
| for i in productsMap.groupByKey().flatMap(lambda x: getAllSortByPrice(x, True)).collect(): print(i) |
|  |

|  |
| --- |
| for i in productsMap.groupByKey().flatMap(getAll).collect(): print(i) |
|  |

|  |
| --- |
| for i in productsMap.groupByKey().flatMap(getTop).collect(): print(i) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Sorting using queries |
|  |

|  |
| --- |
| #Global sorting and ranking |
|  |

|  |
| --- |
| select \* from products order by product\_price desc; |
|  |

|  |
| --- |
| select \* from products order by product\_price desc limit 10; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #By key sorting |
|  |

|  |
| --- |
| #Using order by is not efficient, it serializes |
|  |

|  |
| --- |
| select \* from products order by product\_category\_id, product\_price desc; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Using distribute by sort by (to distribute sorting and scale it up) |
|  |

|  |
| --- |
| select \* from products distribute by product\_category\_id sort by product\_price desc; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #By key ranking (in Hive we can use windowing/analytic functions) |
|  |

|  |
| --- |
| select \* from (select p.\*, |
|  |

|  |
| --- |
| dense\_rank() over (partition by product\_category\_id order by product\_price desc) dr |
|  |

|  |
| --- |
| from products p |
|  |

|  |
| --- |
| distribute by product\_category\_id) q |
|  |

where dr <= 2 order by product\_category\_id, dr;