**Q1. MapReduce**

**Problem Statement**

**Find all time High price for each stock**

**Commands:**

**import** java.io.\*;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.DoubleWritable;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.conf.\*;

**import** org.apache.hadoop.fs.\*;

**import** org.apache.hadoop.mapreduce.lib.input.\*;

**import** org.apache.hadoop.mapreduce.lib.output.\*;

**public** **class** AllTimeHigh {

**public** **static** **class** MapClass **extends** Mapper<LongWritable,Text,Text,DoubleWritable>

{

**private** Text stock\_id = **new** Text();

**private** DoubleWritable High = **new** DoubleWritable();

**public** **void** map(LongWritable key, Text value, Context context)

{

**try**{

String[] str = value.toString().split(",");

**double** high = Double.*parseDouble*(str[4]);

stock\_id.set(str[1]);

High.set(high);

//context.write(new Text(str[1]),new LongWritable(vol));

context.write(stock\_id, High);

}

**catch**(Exception e)

{

System.***out***.println(e.getMessage());

}

}

}

**public** **static** **class** ReduceClass **extends** Reducer<Text,DoubleWritable,Text,DoubleWritable>

{

**private** DoubleWritable result = **new** DoubleWritable();

**public** **void** reduce(Text key, Iterable<DoubleWritable> values,Context context) **throws** IOException, InterruptedException {

**double** maxValue=0;

**double** temp\_val=0;

**for** (DoubleWritable value : values) {

temp\_val = value.get();

**if** (temp\_val > maxValue) {

maxValue = temp\_val;

}

}

result.set(maxValue);

context.write(key, result);

//context.write(key, new LongWritable(sum));

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

//conf.set("name", "value")

//conf.set("mapreduce.input.fileinputformat.split.minsize", "134217728");

Job job = Job.*getInstance*(conf, "Highest Price for each stock");

job.setJarByClass(AllTimeHigh.**class**);

job.setMapperClass(MapClass.**class**);

//job.setCombinerClass(ReduceClass.class);

job.setReducerClass(ReduceClass.**class**);

job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(DoubleWritable.**class**);

FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

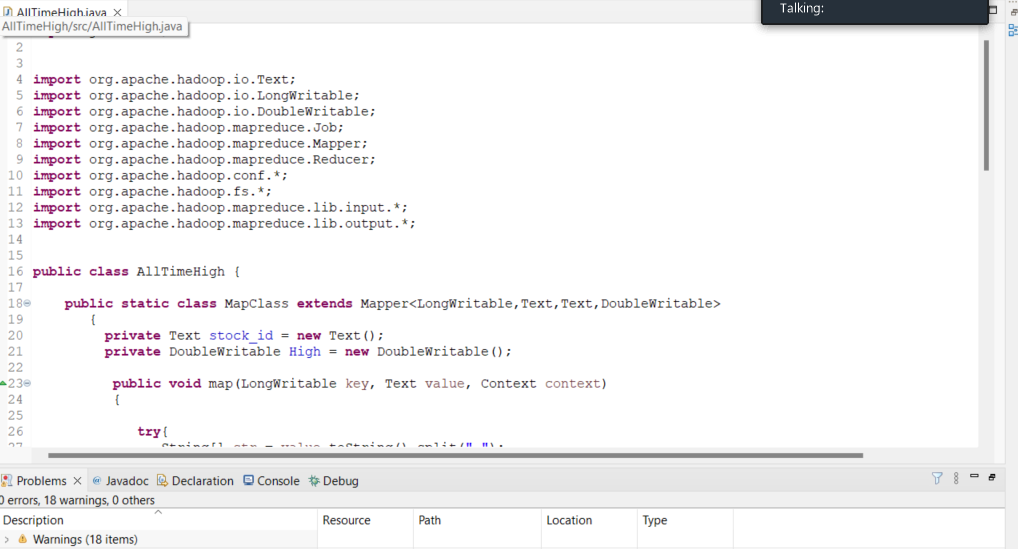
FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

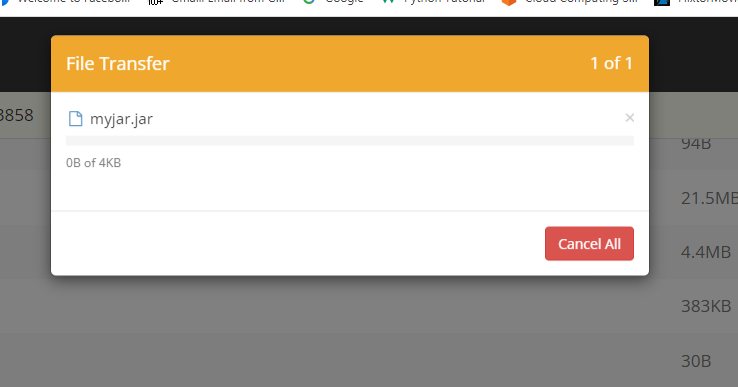
System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);

}

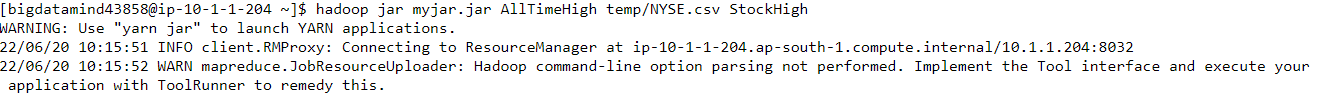
}

[bigdatamind43858@ip-10-1-1-204 ~]$ hadoop jar myjar.jar AllTimeHigh temp/NYSE.csv StockHigh





Output



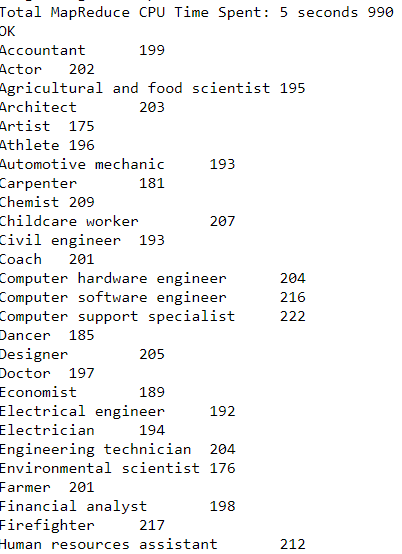


**1) Write a program to find the count of customers for each profession.**

Command:

hive (dbda2022)> select profession,count(custno) from customer group by profession;

output:

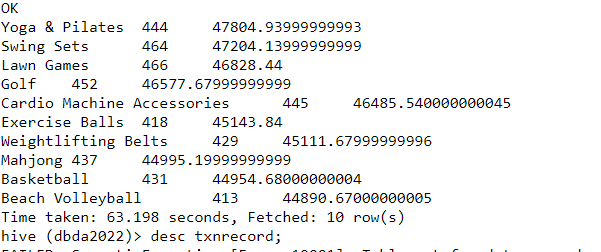


**2) Write a program to find the top 10 products sales wise**

Command:

hive (dbda2022)> select product,count(product),sum(amount) as total from txnrecords group by product order by total desc limit 10;

output:



**3) Write a program to create partiioned table on category**

**Command:**

set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.dynamic.partition=true;

create table txnrecsByCatp(txnno INT, txndate STRING, custno INT, amount DOUBLE,

product STRING, city STRING, state STRING, spendby STRING)

partitioned by (category STRING)

row format delimited

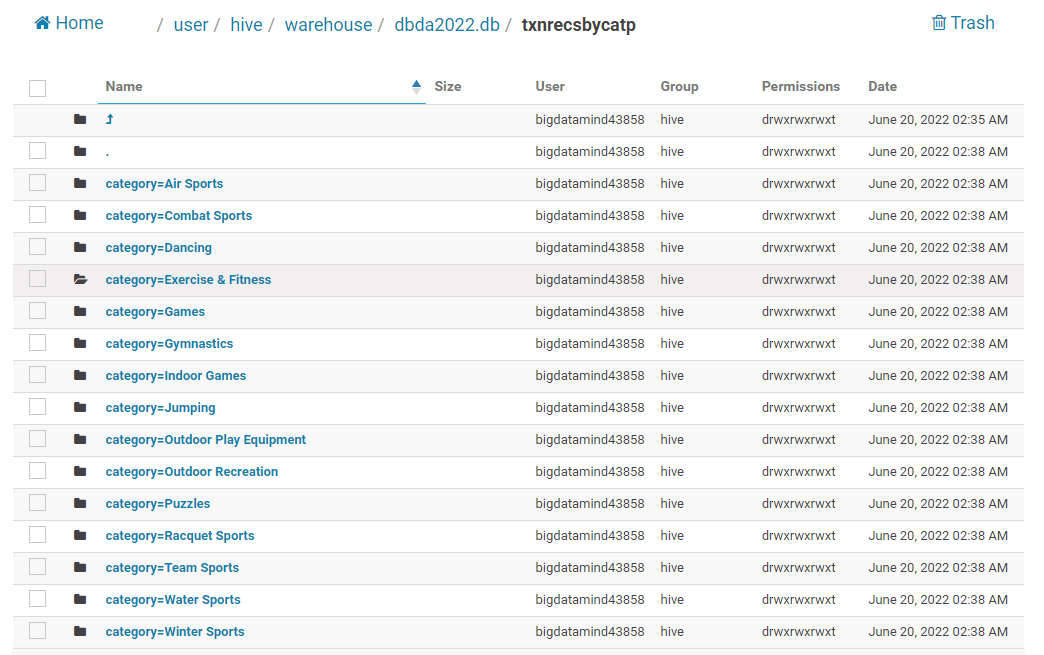
fields terminated by ','

stored as textfile;

INSERT OVERWRITE TABLE txnrecsByCatp PARTITION(category) select txn.txnno, txn.txndate,txn.custno, txn.amount,txn.product,txn.city,txn.st

ate, txn.spendby, txn.category from txnrecords txn DISTRIBUTE By category;

**output:**

****

**QUESTION 3 [15 marks]**

1. **What was the highest number of people travelled in which year?**

>>> rdd=sc.textfile("/user/bigdatamind43858/temp/airlines.csv")

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'SparkContext' object has no attribute 'textfile'

>>> rdd=sc.textFile("/user/bigdatamind43858/temp/airlines.csv")

>>> header=rdd.first()

>>> rdd2=rdd.filter(lambda a : a!= header)

>>> rdd2.count()

84

>>> rdd.count()

85

>>> rdd3=rdd2.mao(lambda a : a.encode("ascii","ignore"))

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

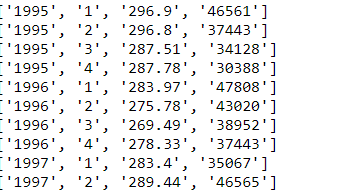
AttributeError: 'PipelinedRDD' object has no attribute 'mao'

>>> rdd3=rdd2.map(lambda a : a.encode("ascii","ignore"))

>>> rdd4=rdd3.map(lambda a : a.split(","))

>>> for i in rdd4.take(10):

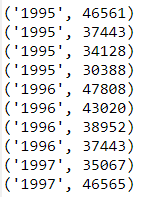
... print(i)



>>> q1=rdd4.map(lambda a:(a[0],int(a[3])))

>>> for i in q1.take(10):

... print(i)



>>> q12=q1.reduceByKey(lambda a,b: a+b)

>>> q13=q12.sortBy(lambda a:-a[1])

>>> q13.first()

Output:

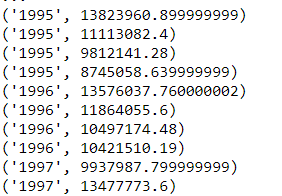


1. **Identifying the highest revenue generation for which year**

>>> q2=rdd4.map(lambda a:(a[0],float(a[2])\*int(a[3])))

>>> for i in q2.take(10):

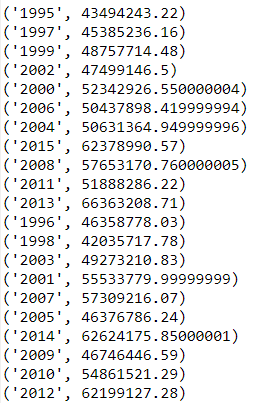
... print(i)



>>> q21=q2.reduceByKey(lambda a,b:a+b)

>>> for i in q21.collect():

... print(i)



>>> q22=q21.sortBy(lambda a:-a[1])

>>> q22.first()

Output:



1. **Identifying the highest revenue generation for which year and quarter (Common group)**

>>> q3=rdd4.map(lambda a:(a[0]+" "+a[1],float(a[2])\*int(a[3])))

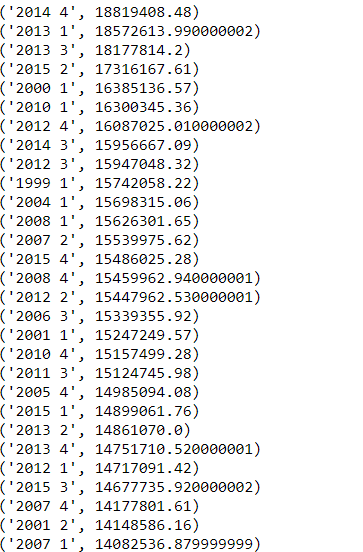
>>> q4=q3.reduceByKey(lambda a,b:a+b)

>>> q5=q4.sortBy(lambda a :-a[1])

>>> for i in q5.collect():

... print(i)

Output:



>>> q5.first()

Output:

