**MapReduce**

**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]));

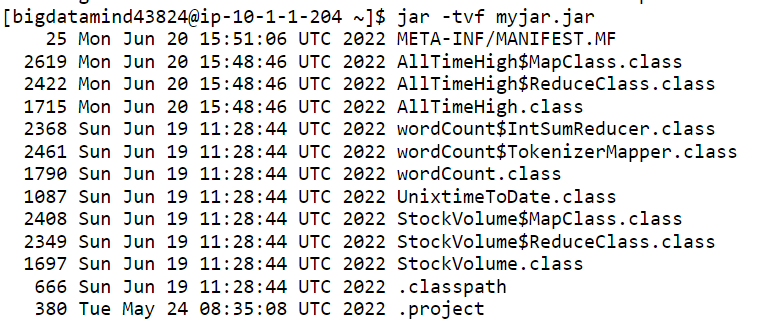
FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

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

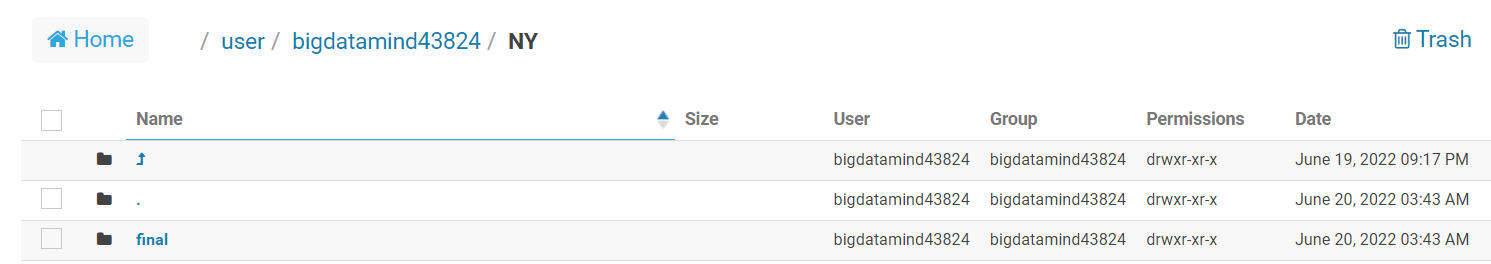
}

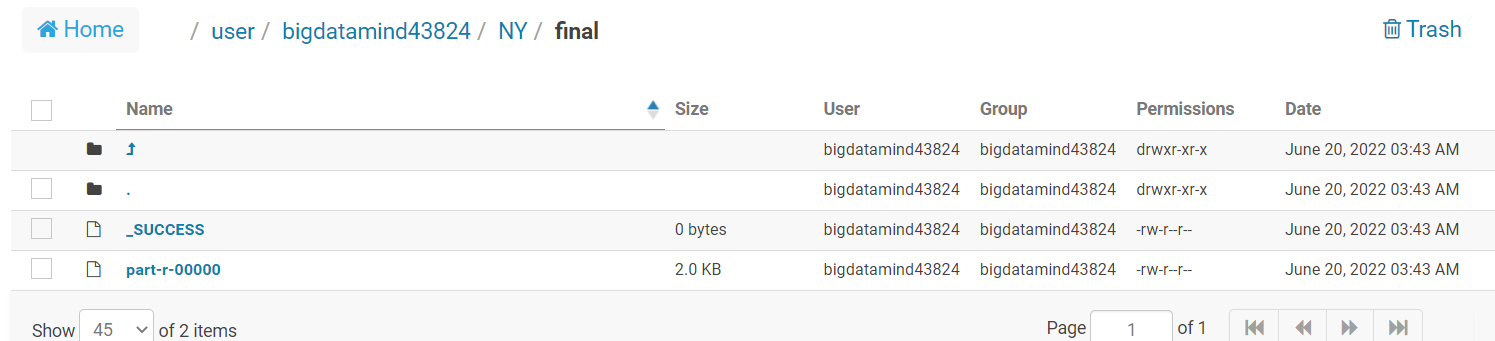
}

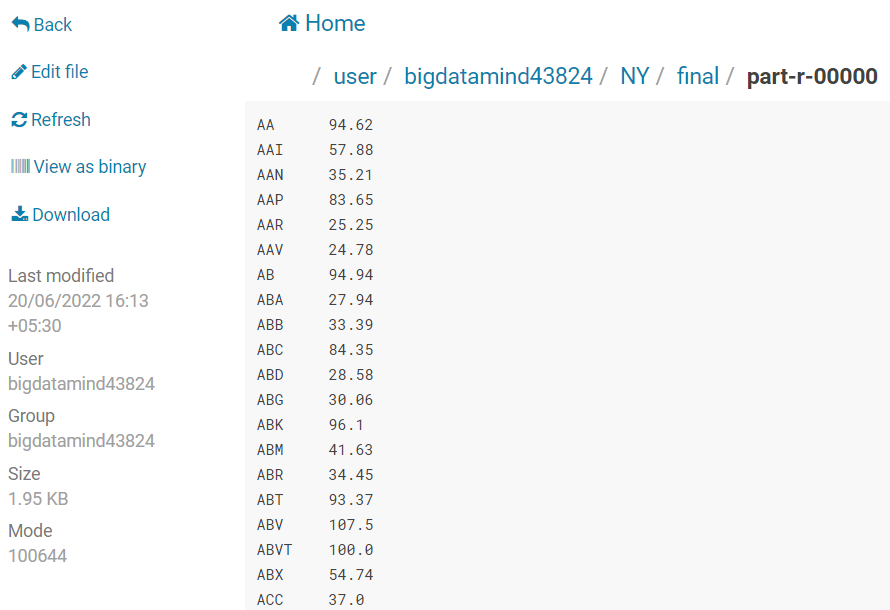
[bigdatamind43824@ip-10-1-1-204 ~]$ jar -tvf myjar.jar

****

[bigdatamind43824@ip-10-1-1-204 ~]$ hadoop jar myjar.jar AllTimeHigh NYSE.csv NY/final

****

****

****

**Hive**

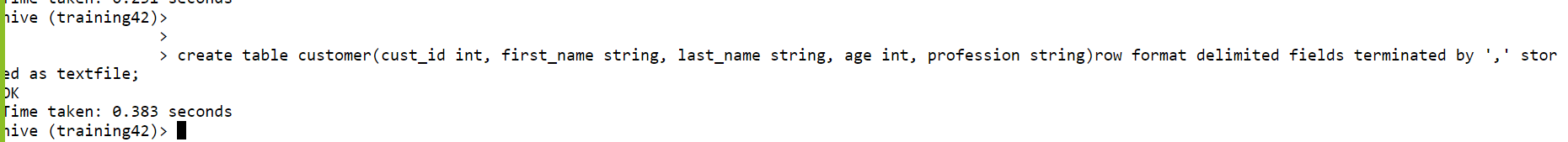
**1) Write a program to find the count of customers for each profession.**

hive (training42)>

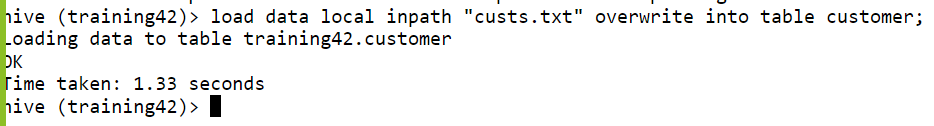
>

> create table customer(cust\_id int, first\_name string, last\_name string, age int, profession string)row format delimited fields terminated by ',' stor

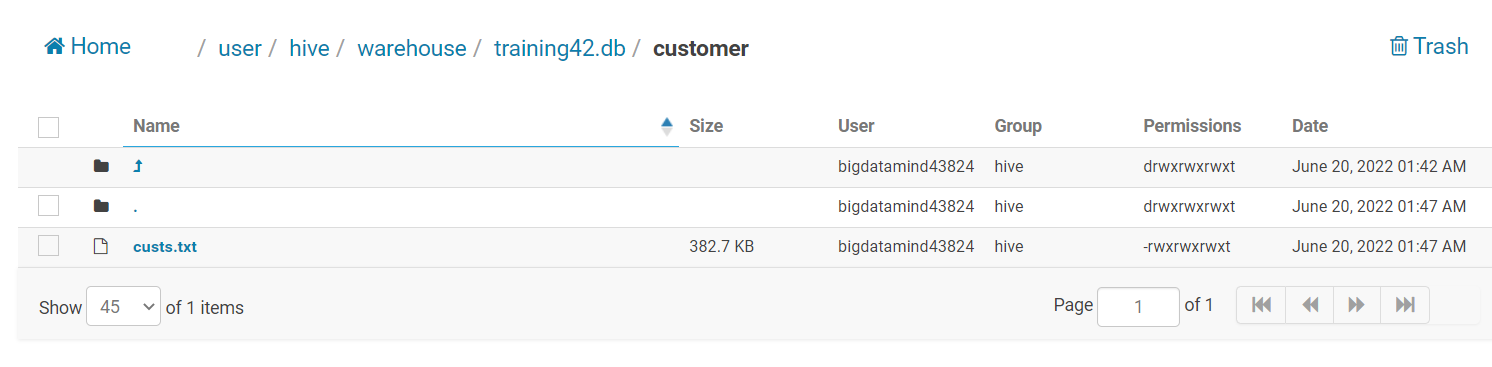
ed as textfile;



hive (training42)> load data local inpath "custs.txt" overwrite into table customer;

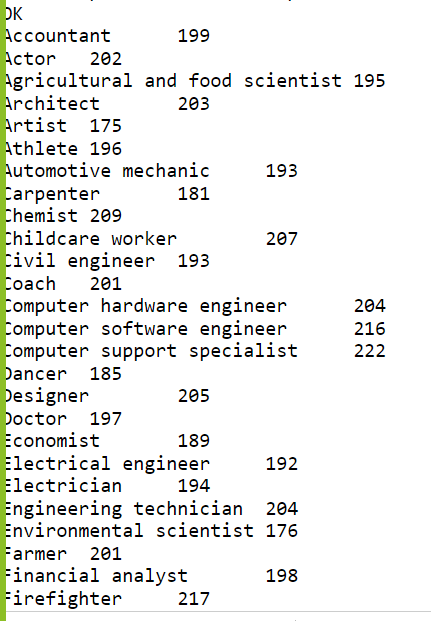


Data loaded in cutomer folder



Query

select profession, count(profession) as count from customer group by profession;



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

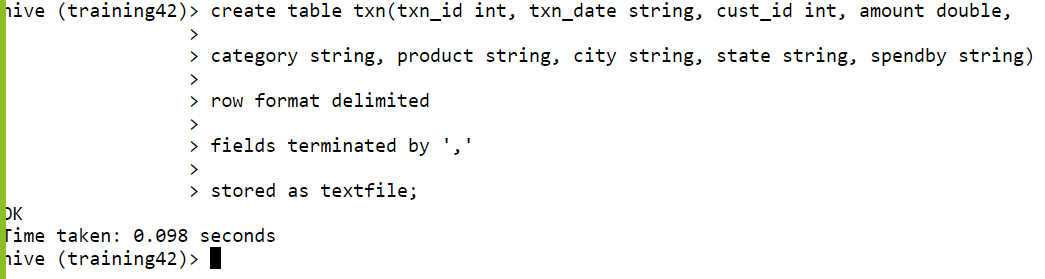
create table txn(txn\_id int, txn\_date string, cust\_id int, amount double,

category string, product string, city string, state string, spendby string)

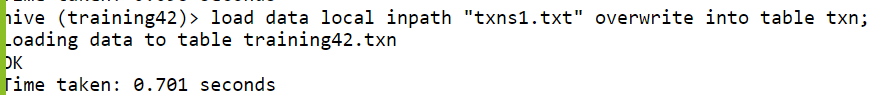
row format delimited

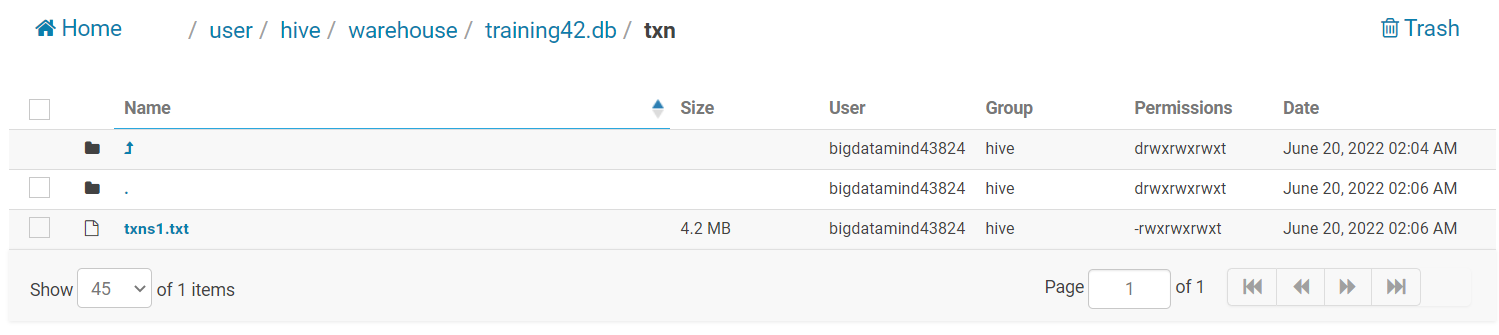
fields terminated by ','

stored as textfile;

****

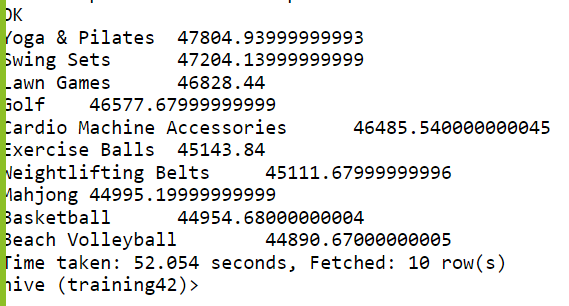
hive (training42)> load data local inpath "txns1.txt" overwrite into table txn;

****

****

Query

hive (training42)> select product , sum(amount) as max\_sales from txn group by product order by max\_sales DESC limit 10;



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

hive (training42)> create table txnPartByCat(txn\_id int, txn\_date string, cust\_id int,

>

> amount double, product string, city string, state string, spendby string)

>

> partitioned by (category string)

>

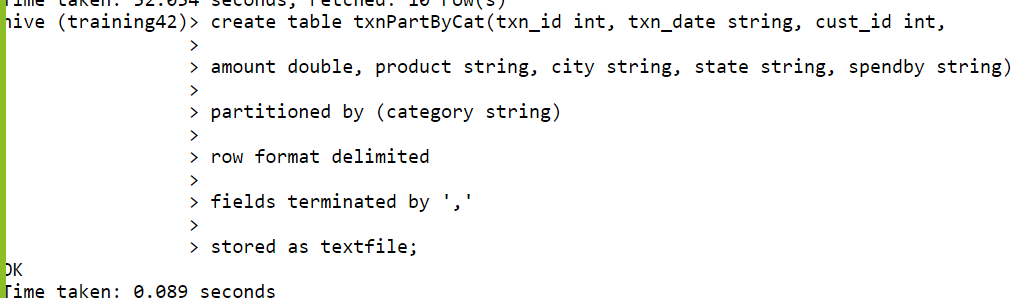
> row format delimited

>

> fields terminated by ','

>

> stored as textfile;



hive (training42)> desc txnPartbyCat;

OK

txn\_id int

txn\_date string

cust\_id int

amount double

product string

city string

state string

spendby string

category string

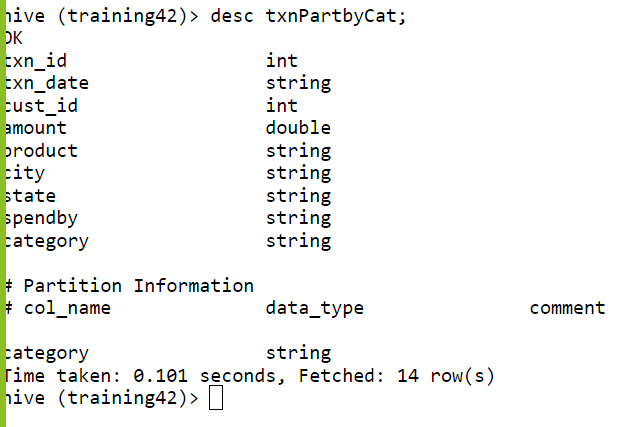
# Partition Information

# col\_name data\_type comment

category string

Time taken: 0.101 seconds, Fetched: 14 row(s)

hive (training42)>

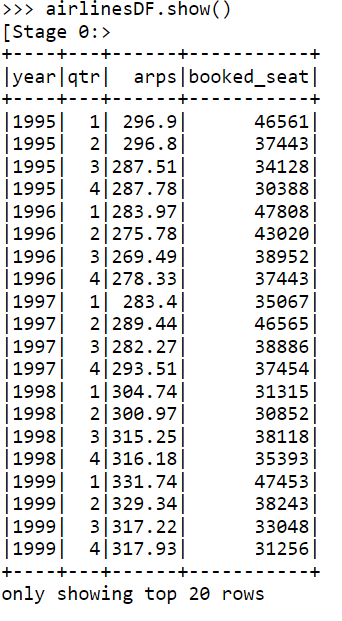


Pyspark

>>> from pyspark.sql.types import StructType, StringType, IntegerType, DoubleType, LongType

>>> schema = StructType().add("year", StringType(), True).add("qtr", StringType(), True).add("arps", DoubleType(),True).add("booked\_seat", LongType(),True)

>>> airlinesDF =spark.read.format("csv").option("header", "TRUE").schema(schema).load("/user/bigdatamind43824/airlines.csv")

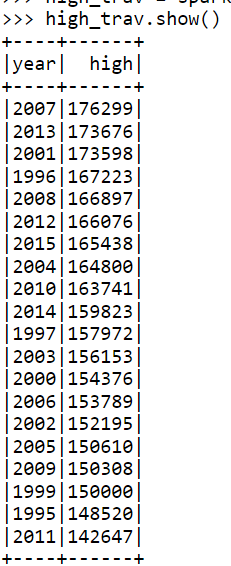


>>> airlinesDF.registerTempTable("airlines")

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

>>> high\_trav = spark.sql("select year, sum(booked\_seat) as high from airlines group by year order by high desc")

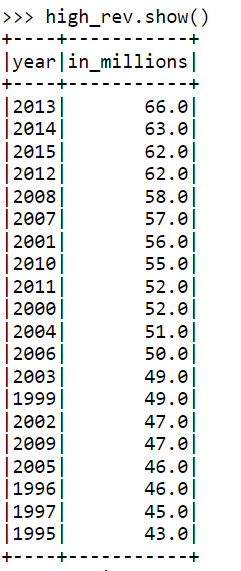
>>> high\_trav.show()



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

>>> high\_rev = spark.sql("select year, round(sum(arps \* booked\_seat)/1000000) as in\_millions from airlines group by year order by in\_millions desc")

>>> high\_rev.show()



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

>>> highrev = spark.sql("select (year,qtr) as period, sum(arps\*booked\_seat)/1000000 as rev\_in\_million from airlines group by period order by rev\_in\_million desc limit 1

")

>>> highrev.show()

