Relational Data Analysis With Pig And Spark

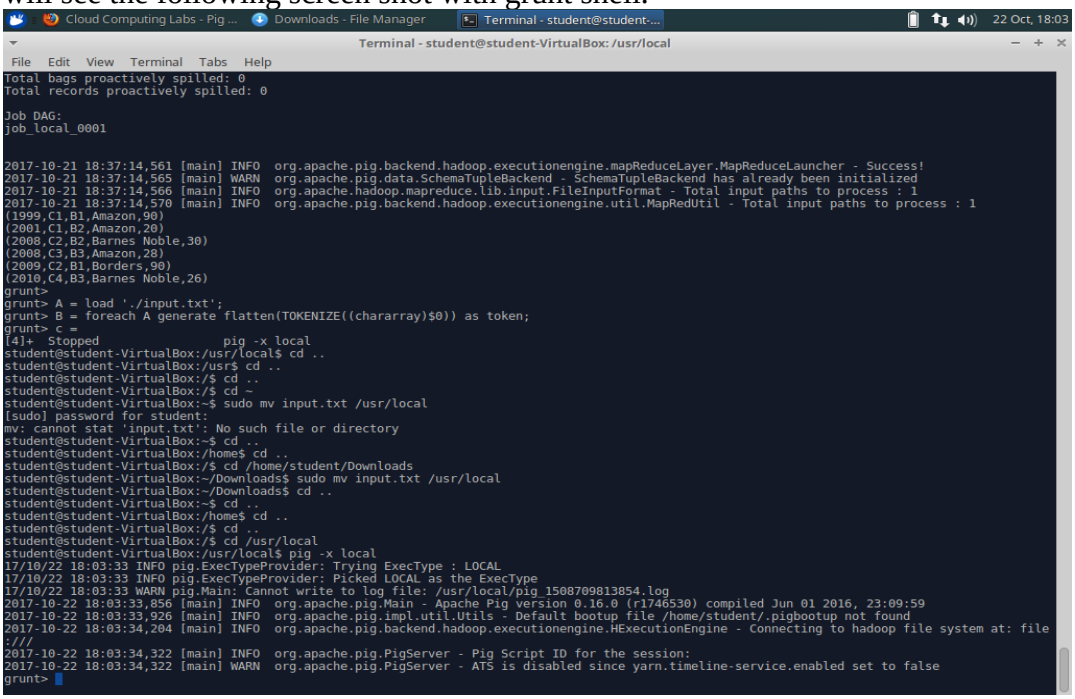
SUBHADRA TUMMALAPALLY

(U00863954)

INSTALLATION STEPS OF PIG :

Follow the below steps to install pig:

1. Download latest version of pig from pig.apache.org
2. Untar downloaded pig file by giving following command: *sudo tar -xvf pig-0.16.0*
3. Move untared file to /usr/local and open bashrc by giving sudo nano ~/.bashrc give java path and pig home path
4. Then do source bashrc.
5. Then go to /usr/local and give pig -x local if your pig successfully got installed then you will see the following screen shot with grunt shell.

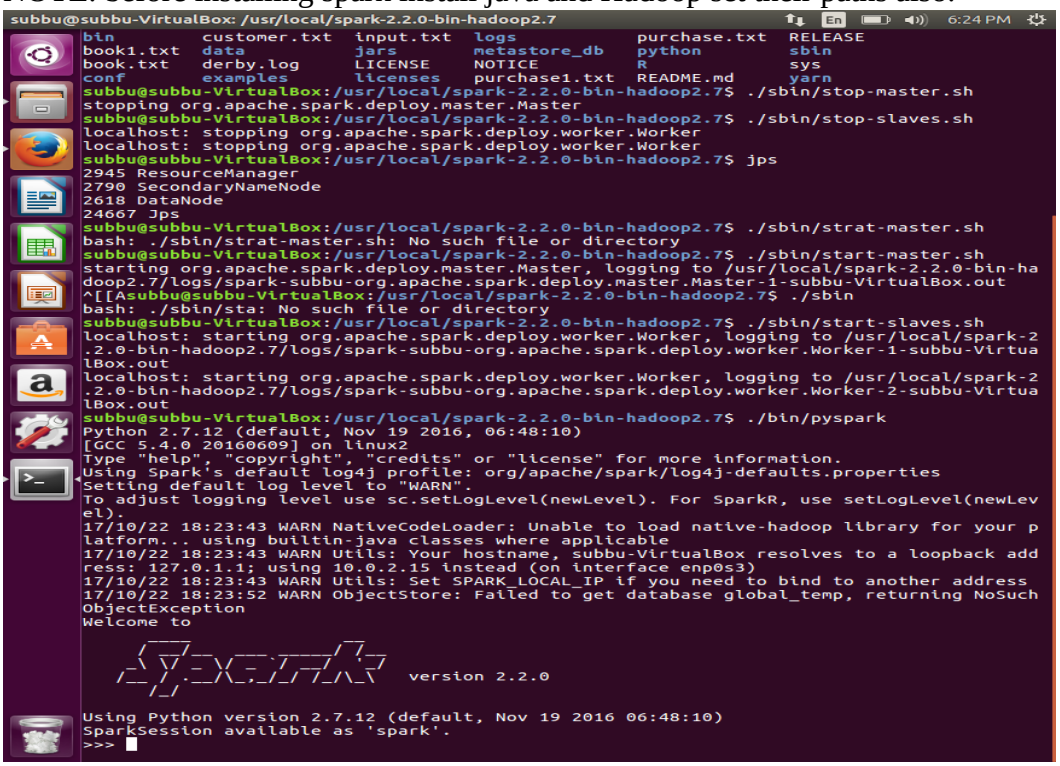


INSTALLATION STEPS OF SPARK:

Follow below steps to install spark:

1. Download latest version of spark from spark.apache.org and recent version of scala
2. Untar downloaded spark file by giving following command: *sudo tar -xvf spark-2.2.0-bin-hadoop2.7 and untar scala also: sudo tar -xvf scalaversion*
3. Move untared files to /usr/local and go to /usr/local/sparkversion and open bashrc by giving sudo nano ~/.bashrc and set scala path and spark path.
4. Then do source bashrc and go to /usr/local/spark-2.2.0-bin-hadoop2.7 and start master node and slaves node as mentioned in spark material in pilot. Then give ./bin/pyspark the following screen shot will appear if spark installed correctly.

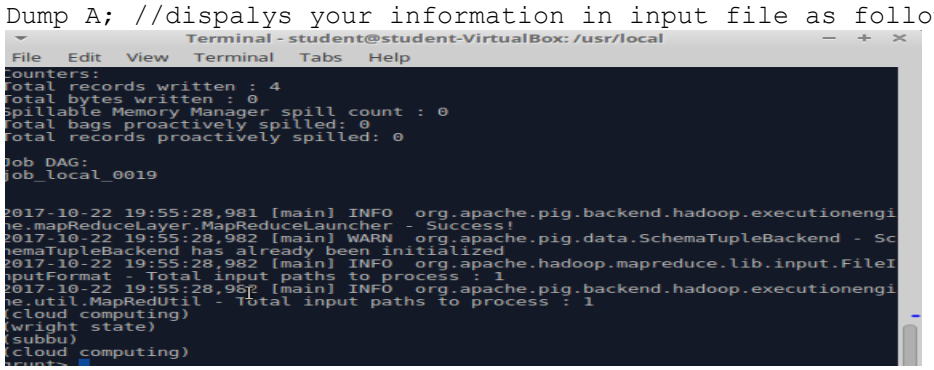
NOTE: before installing spark install java and Hadoop set their paths also.



ANSWERS OF QUSETION 1.1,1.2 & 2.1-2.3:

(1.1):

```
A = load './input.txt';// loads input from the location. Before doing this we
have to move input file to /usr/local where we are starting pig because
everything should be in same place to do.
Dump A; //dispalys your information in input file as follows.
```




```
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as token;// splits
the input stored in 'A' as follows and named that splitted information as
"token". Here foreach is defining each element stored in A GENERATE is
showing what we have to display or stores in B so condition we have to write
in GENERATE. Flatten(TOKENIZE) splitting input into parts $o displaying first
column of output and finally we are displaying splitted output as "token".
The screen shot of this command is below by giving DUMP B.
```

```
Terminal - student@student-VirtualBox: /usr/local
File  Edit  View  Terminal  Tabs  Help
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local_0020


2017-10-22 19:56:49,166 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 19:56:49,169 [main] WARN  org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2017-10-22 19:56:49,170 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 19:56:49,171 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(cloud)
(computing)
(wright)
(state)
(subbu)
(cloud)
(computing)
grunt>
```

C = **group** B **by** token;// grouping information in B by token so that it displays all similar words in one place which will be easy for count.

```
Terminal - student@student-VirtualBox: /usr/local
File  Edit  View  Terminal  Tabs  Help
Total records written : 5
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local_0021


2017-10-22 19:58:21,839 [main] INFO  org.apache.pig.backend.hadoop.executionen
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 19:58:21,839 [main] WARN  org.apache.pig.data.SchemaTupleBackend -
hemaTupleBackend has already been initialized
2017-10-22 19:58:21,840 [main] INFO  org.apache.hadoop.mapreduce.lib.input.Fil
nputFormat - Total input paths to process : 1
2017-10-22 19:58:21,840 [main] INFO  org.apache.pig.backend.hadoop.executionen
ne.util.MapRedUtil - Total input paths to process : 1
(cloud,{(cloud),(cloud)})
(state,{(state)})
(subbu,{(subbu)})
(wright,{(wright)})
(computing,{(computing),(computing)})
grunt>
```

D = **foreach** C **generate** group, COUNT(B); // here we are displaying count to each grouped word which is stored in 'C'.

```
Cloud Computing Labs - Pig ...   Downloads - File Manager   Terminal - student@student-...   22 Oct, 18:06
Terminal - student@student-VirtualBox: /usr/local
File  Edit  View  Terminal  Tabs  Help
2017-10-22 18:06:14,426 [Thread-4] INFO  org.apache.hadoop.mapred.Task - Task attempt_local_0001_r_000000_0 is allowed to commit now
2017-10-22 18:06:14,426 [Thread-4] INFO  org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task 'attempt_local_0001_
r_000000_0' to file:/tmp/temp181577355/tmp1014032491
2017-10-22 18:06:14,720 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 50% complete
2017-10-22 18:06:14,720 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Running jobs are [job_loc
al_0001]
2017-10-22 18:06:17,313 [Thread-4] INFO  org.apache.hadoop.mapred.LocalJobRunner - reduce > reduce
2017-10-22 18:06:17,314 [Thread-4] INFO  org.apache.hadoop.mapred.Task - Task 'attempt_local_0001_r_000000_0' done.
2017-10-22 18:06:17,750 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - 100% complete
2017-10-22 18:06:17,761 [main] INFO  org.apache.pig.tools.pigstats.mapreduce.SimplePigStats - Script Statistics:

HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.0.4   0.16.0  student 2017-10-22 18:06:10      2017-10-22 18:06:17     GROUP_BY

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianR
educetime       Alias   Feature Outputs
job_local_0001  1       1       n/a     n/a     n/a     n/a     n/a     n/a     n/a     n/a     A,B,D,c GROUP_BY,COMBINER       file:/tmp/temp1
81577355/tmp1014032491,

Input(s):
Successfully read 4 records from: "file:///usr/local/input.txt"

Output(s):
Successfully stored 5 records in: "file:/tmp/temp181577355/tmp1014032491"

Counters:
Total records written : 5
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local_0001


2017-10-22 18:06:17,766 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 18:06:17,768 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-10-22 18:06:17,769 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-10-22 18:06:17,770 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(cloud,2)
(state,1)
(subbu,1)
(wright,1)
(computing,2)
grunt>
```

store D **into** `'./output'`; // we are storing information in D in output file.

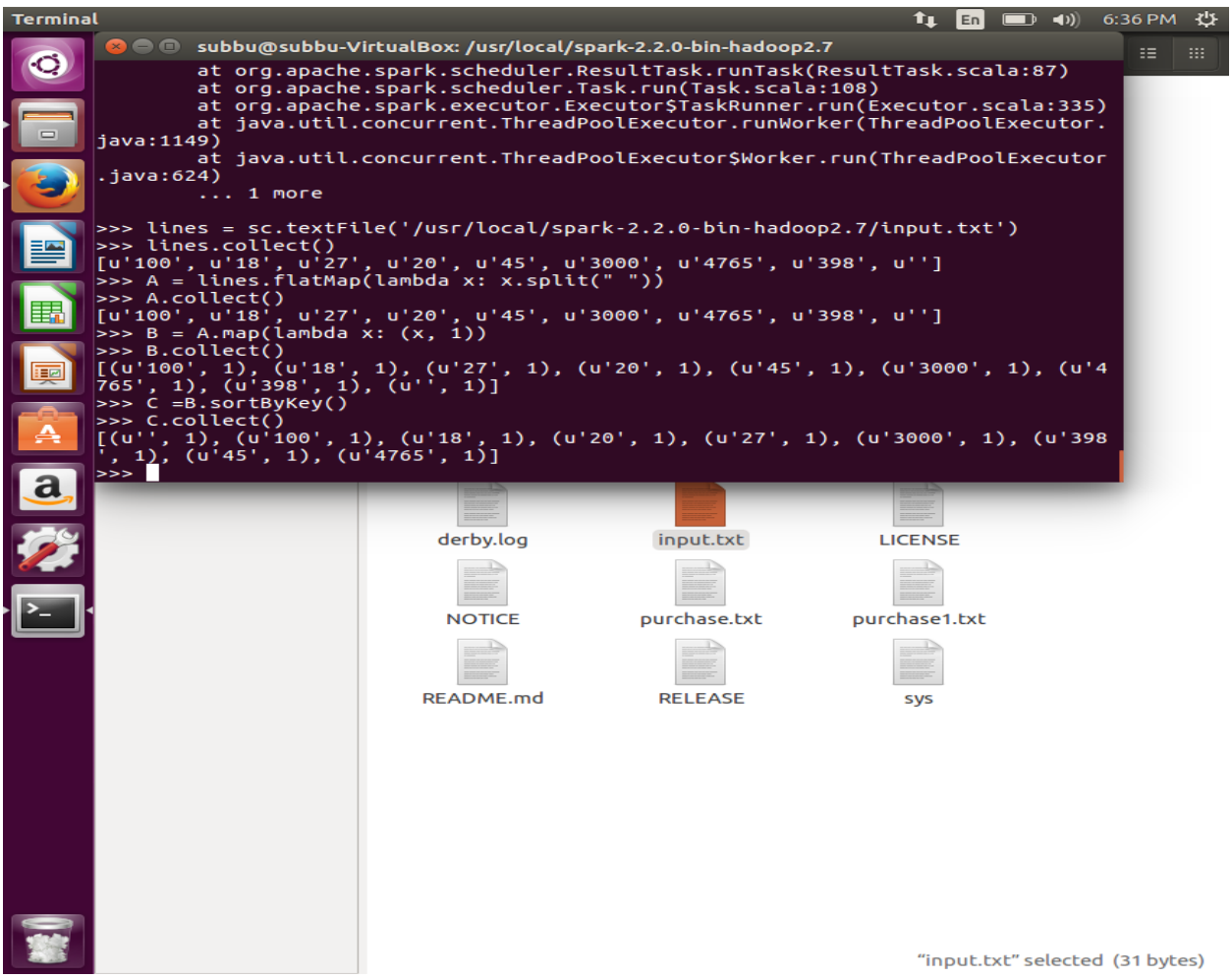
Finally the code in 1.1 displaying wordcount of given input file.

(1.2):

Before starting this code in pyspark store input file in spark folder from where we are starting pyspark.

The code in 1.2 works as follows:

```
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0])// loading input
from the input location and lambda is used intilalise variable to name rows
in input file r[0] dispalys first tuple in input file.
 A = lines.flatMap(lambda x: x.split(' '))// splitting information which is
stroed in line and stroing in A
 B = A.map(lambda x: (int(x), 1))//here we are mapping each word with count 1
int is using because all the elements in our input are numericals which is
integer type.
 C = B.sortByKey()// we are sorting the input in B in ascending order and
diplaying by giving c.collect() which is shown bellow.
```



This image is describing how each line is working by typing name.collect after each line we are showing the total performance of code here.

ANSWERS FROM 2.1-2.3

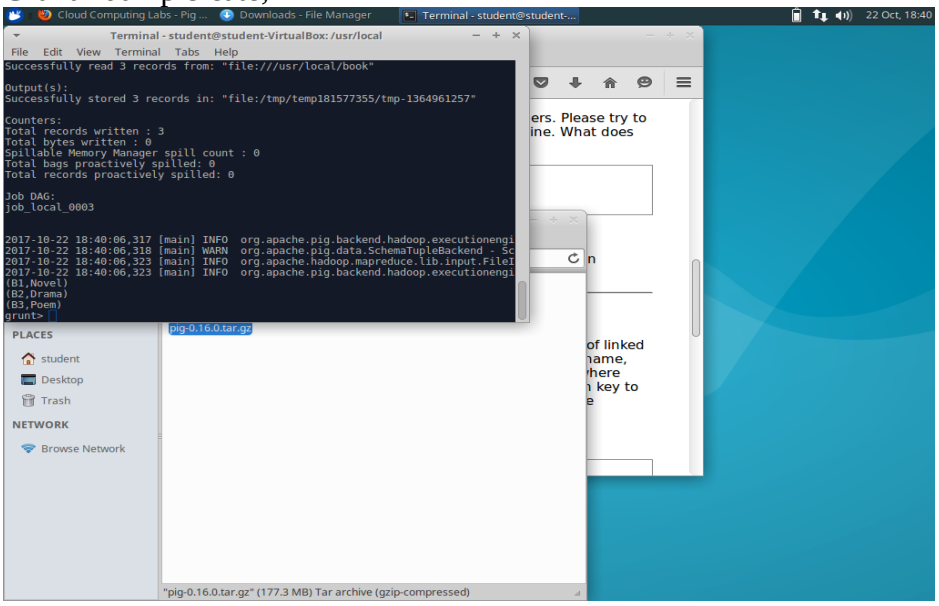
(2.1).PIG:
#EXPLANATION:

1. At first we have to load book and purchase tables by giving labels to them and naming their columns and initializing their data types.
2. Dump these tables to see the information in it.
3. Then group information in purchase table by seller name to each seller data at one place
4. Dump this table to see grouped information
5. Finally sum the grouped information by each book price using foreach and generate.
6. Dump 5th step to see final result i.e., how much each seller is earning.

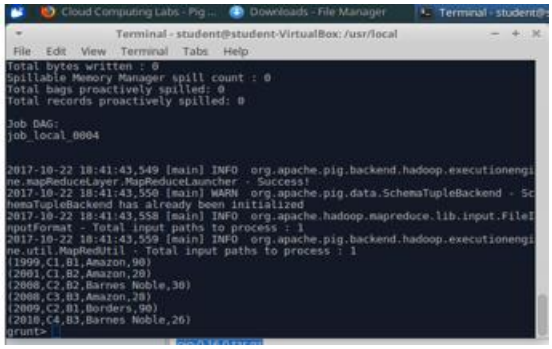
The code for each line in explanation is below:

CODE:

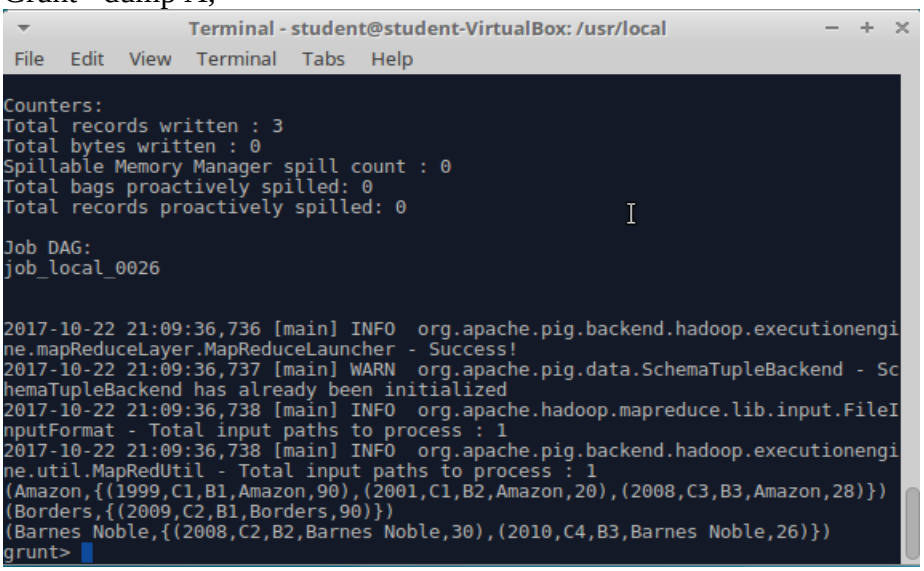
Grunt> create = LOAD ‘book’ as (bid : chararray, BName : chararray);//loading book table and initializing data types.
Grunt>create2 = LOAD ‘purchase’ as(year : chararray, cid : chararray, bid:chararray, BName: charaarray, Bprice: int);//loading purchase and initializing data types to each column
Grunt> dump create;



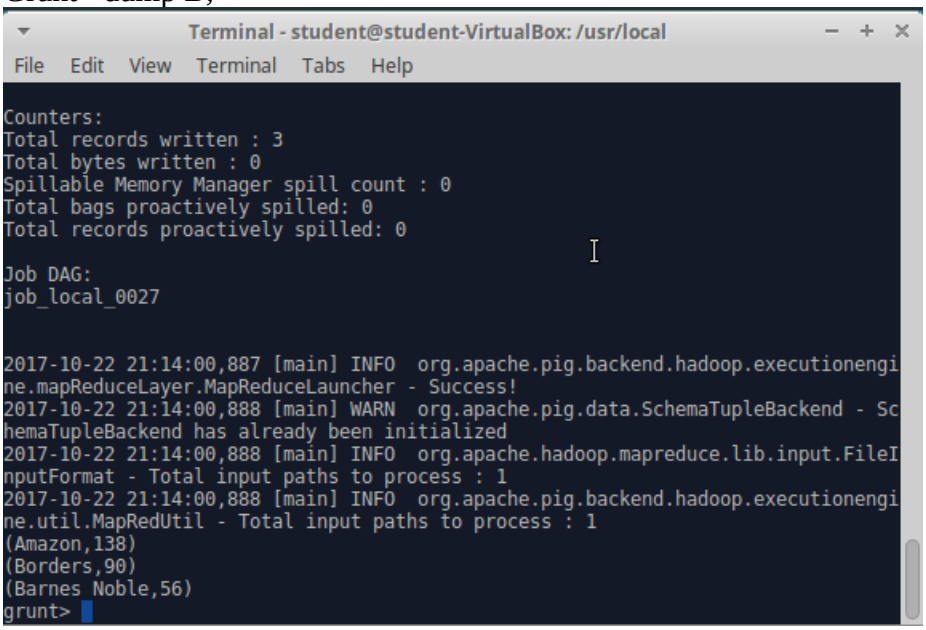
Grunt>dump create2;



Grunt>A = group create2 by SName;
Grunt> dump A;



Grunt>B= foreach A generate group, SUM(create2.Bprice);
Grunt> dump B;



2.1.SPARKRDD:

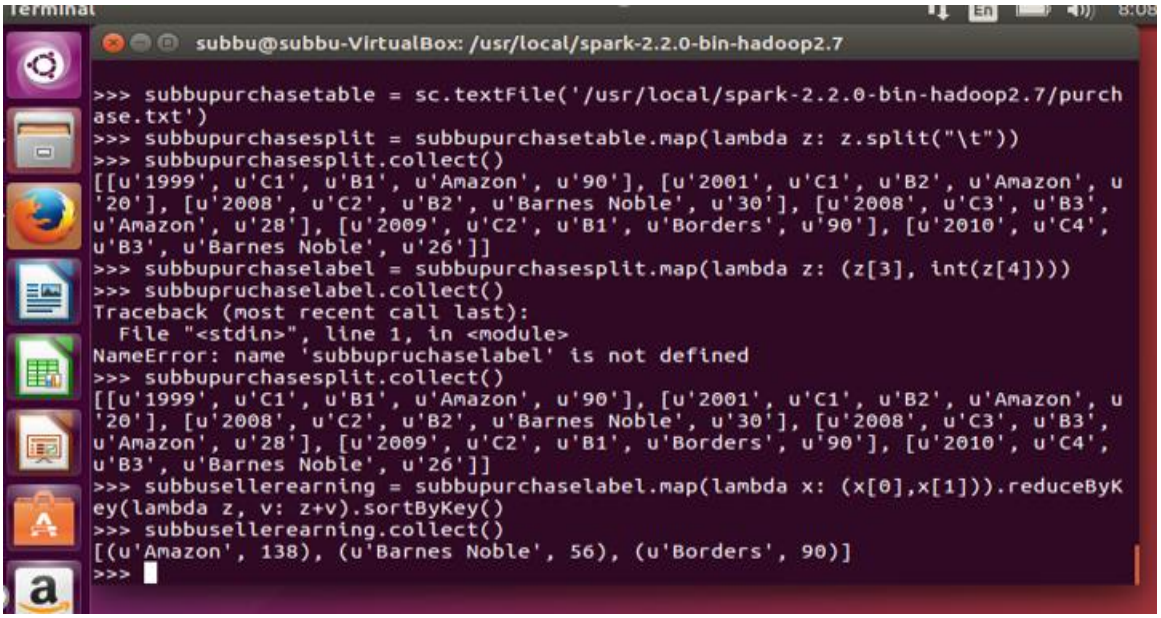
#EXPLANATION:

1. Load input file from its location using spark context as SC.
2. Split the output of 1st step into different tupples.
3. By using collect() display output of split operation
4. Display required tuples in your table and intialise data variables to them using map and lambda and display them using collect()
5. By using map function generate (key,value) to output of 4th step and sum them by using reduceByKey() to display total earnings of each seller. And by using collect() display final result.

#CODE:

```
>>> subbupurchasetable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/purchase.txt')
                                                    //loading table from its location using 'sc'.
>>>subbupurchasesplit = subbupurchasetable.map(lambda z: z.split("\t"))
                                                    //splitting data tupples of 1st step into separate tupples
>>>subbupurchasesplit.collect()//displays split output.
>>>subbupurhcaselabel = subbupurchasesplit.map(lambda z: (z[3],int(z[4]))
                                                    //initializing data variables to required tuples of 3rd step.
>>>subbupurchaselabel.collect() // displays outcome of 4th step
>>>subbusellerearnings = subbupurchaselabel.map(lambda x: (x[0],x[1])).reduceByKey(lambda z, v: z+v).sortByKey()
                                                    //here we are mapping based on(key, values) and generating sum to each seller by using reduceByKey and sorting the final outcome using sortByKey().
>>>subbusellerearnings.collect() // dispalys output of above step.
```

:here I'm showing output of entire code above after executed in pyspark console window:



(2.2.PIG):

#EXPLANATION:

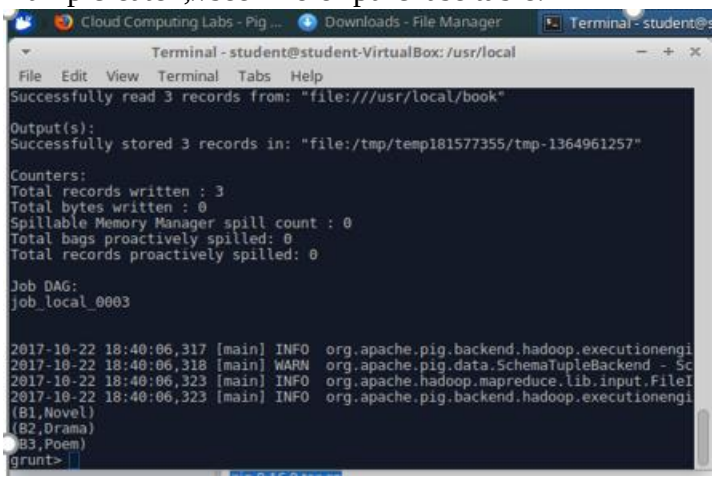
1. Load book and purchase tables from their location and label them with some name and dump both tables to see the information in it.
2. Filter the purchase table by seller Amazon to see all Amazon info as separate table. And dump to see the result of filtered.
3. Name each column of filtered table with same labels used before in purchase table.
4. Group the information in purchase table by book id.
5. After grouping generate minimum price of each seller by book price.
6. Join filtered amazon table and generated minimum price for purchase table.
7. Now join 6th step and book tables by book id.
8. Now display only book name and book id for joined tables in 7th step using foreach and generate functions.
9. Dump the table to see final result.
10. Store the information.
11. Illustrate it to see entire performance.

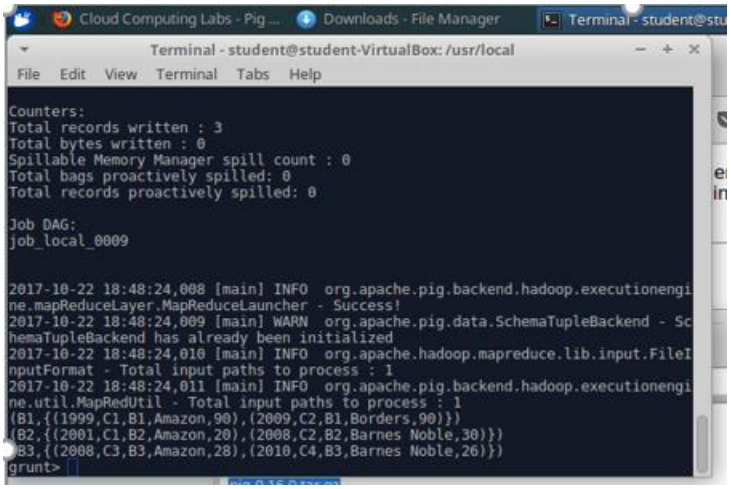
#CODE:

```
Grunt>create = load 'book'  as(bid:chararray, BName : chararray);//loading and initializing

Grunt> create2 = load 'purchase' as(year:chararraycid:chararray, bid:chararray, SName:chararray,
Bprice:int); //loading and initializing data types of purchase table.

Grunt> dump crerate; //see input in book table.

Grunt> dump create2; //see input in purchase table.

Grunt> step1 = filter purchase by SName == "Amazon"; //filtering Amazon seller.

Grunt> step2 = foreach step1 generate year, cid, bid, SName,Bprice;//naming tuples in Amazon.

Grunt>step3 = group create2 by bid; //grouping purchase table by book id.

Grunt>step4 = foreach step3 generate MIN(create2.price) as Bprice; // calculating minimum price
                                                                    Of each book.

Grunt>step5= join step2 by Bprice, step4 by $0; //goining filterd Amazon and minimu price table.

Grunt>step6 = join step5 by step2::bid, create bid; //joining step5 by book id of filtered Amazon
                                                        And book table.

Grunt>joined = foreach step6 generate step2::Bprice, create::BName // displaying result as Book
                                                                    Name and price.

Grunt> dump joined;

Grunt>illustrate joined;

Grunt> store joined into "./output1";
```

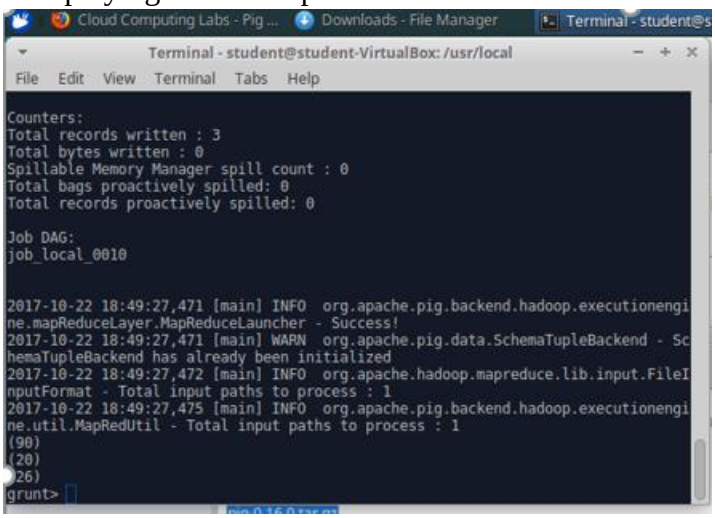
DISPALYING OUTPUTS OF 2.2 stepwise:

1. Dump create; // see info of book table.
2. Dump create2;//see info of purchase table.

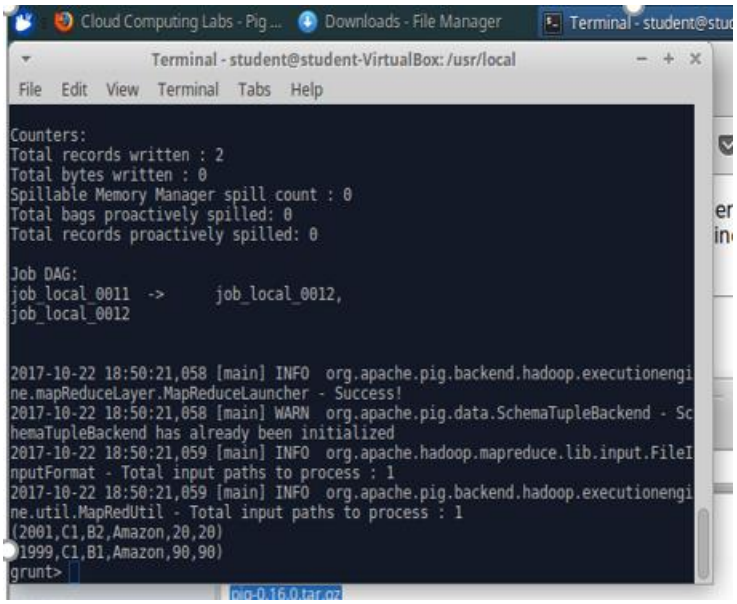




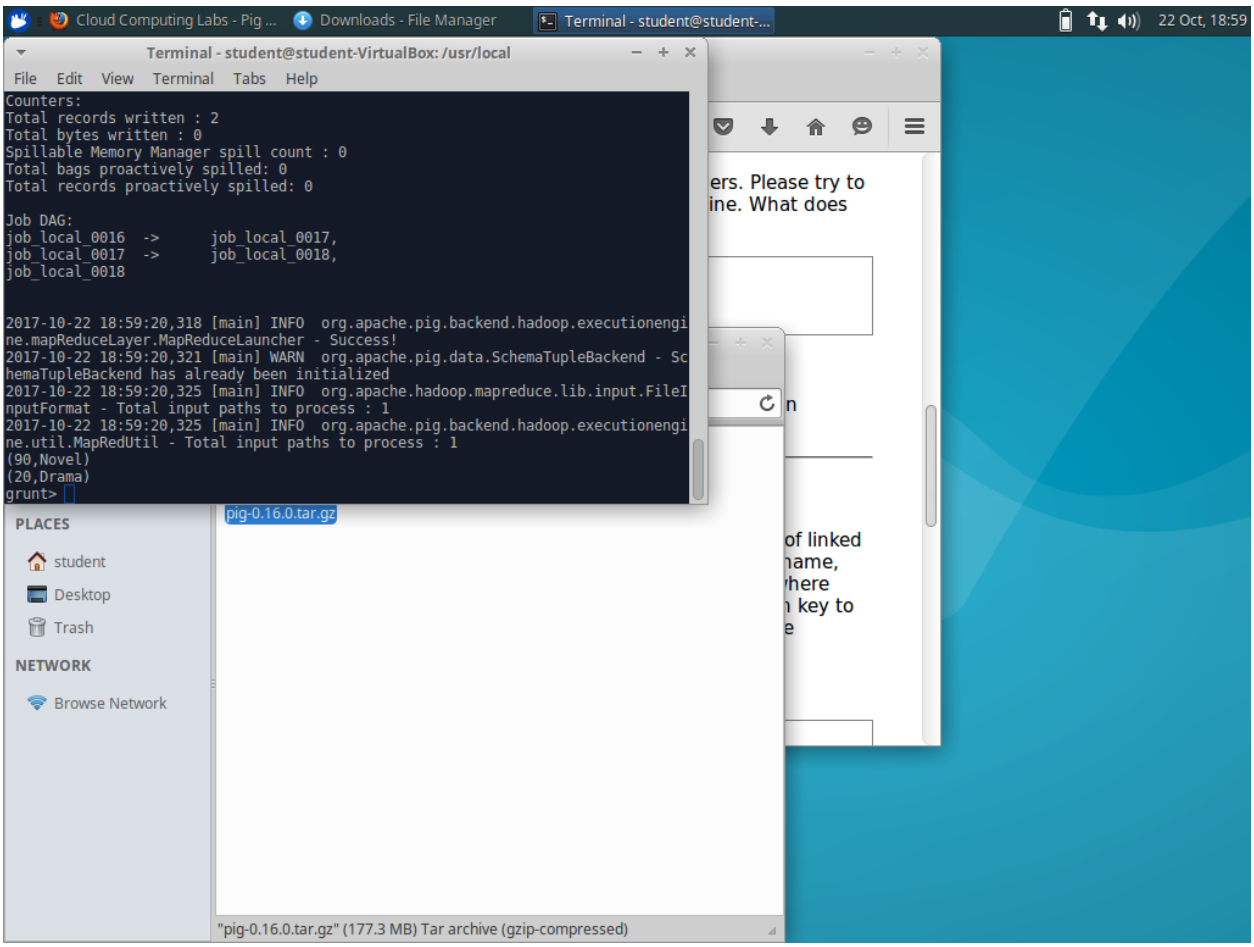
3.dispalying minimum price of all sellers



4.displaying results of joined table amazon and minimum price of each seller.



5.Showing final results after combining 4th result with book and displaying book name and id as results.



2.2.SPARK:

#EXPLANATION:

1. First load the book and purchase tables using spark context as sc.textFile and split the data of both tables into different tupples and intialise the varibles to required tuples using map and display them using collect().
2. Generate minimum price of all books among all sellers using reduceByKey and sort the output by using sortByKey().
3. Now filter Amazon seller from purchase table using filter().
4. Display required tupples from Amazon by intialsing variables to it using map().
5. Group the minimum price result by book id.
6. Group filtered Amazon table by book id
7. Now join both grouped minimum price and Filetered Amazon and display result using collect().
8. Now group information in book table by book id
9. And join book table and result of 7th step to see final result by using collect()

#CODE:

```
>>> subbupurchasetable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/purchase.txt')
                                                    //loading table from its location using 'sc'.
>>>subbupurchasesplit = subbupurchasetable.map(lambda z: z.split("\t"))
                                                    //splitting data tupples of 1st step into separate tupples
>>>subbupurchasesplit.collect()//displays split output.
>>>subbupurhcaselabel = subbupurchasesplit.map(lambda z: (z[2],z[3],int(z[4]))
                                                    //initializing data variables to required tuples of 3rd step.
>>>subbupurchaselabel.collect() // displays outcome of 4th step
>>>subbubooktable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/book.txt')// load book.

>>>subbubooksplit = subbubooktable.map(lambda v: v.split("\t")//splitting data in book

>>>subbubooklabel = subbubooksplit.map(lambda v: (v[0],v[1]))//initializing variables to tupples

>>>subbubooklabel.collect()

>>>subbumin = subbupurchaselabel.map(lambda x: (x[0],x[2])).reudceByKey(lambda v.f: min(v,f)).sortByKey() // generating minimum price of all books among all sellers and sorting the out come.

>>>subbumin.collect()

>>>filtering=subbupurchaselabel.filter(lambda e: (e[2]=="Amzon"))//filtering Amazon data into one table.

>>>filtering.collect()

>>>filteringsubbu = filtering.map(lambda f: (f[0],f[1],f[2]))//initializing variables to Amazon table.
```

>>>filteringsubbu.collect()

>>>subbumin1 = subbumin.groupBy(lambda x: x[0]) //grouping minimum price of book table by book id.

>>>subbufiltergroup = filteringsubbu.groupBy(lambda d: d[0])//grouping amazon table by book id.

>>>subbujoin1 = subbumin1.join(subbufiltergroup)//joining grouped minimum price table and grouped amazon table.

>>>subbujoin1.collect() // displaying the output of subbujoin1.

>>>subbubooklabel.collect()// see data in book table once.

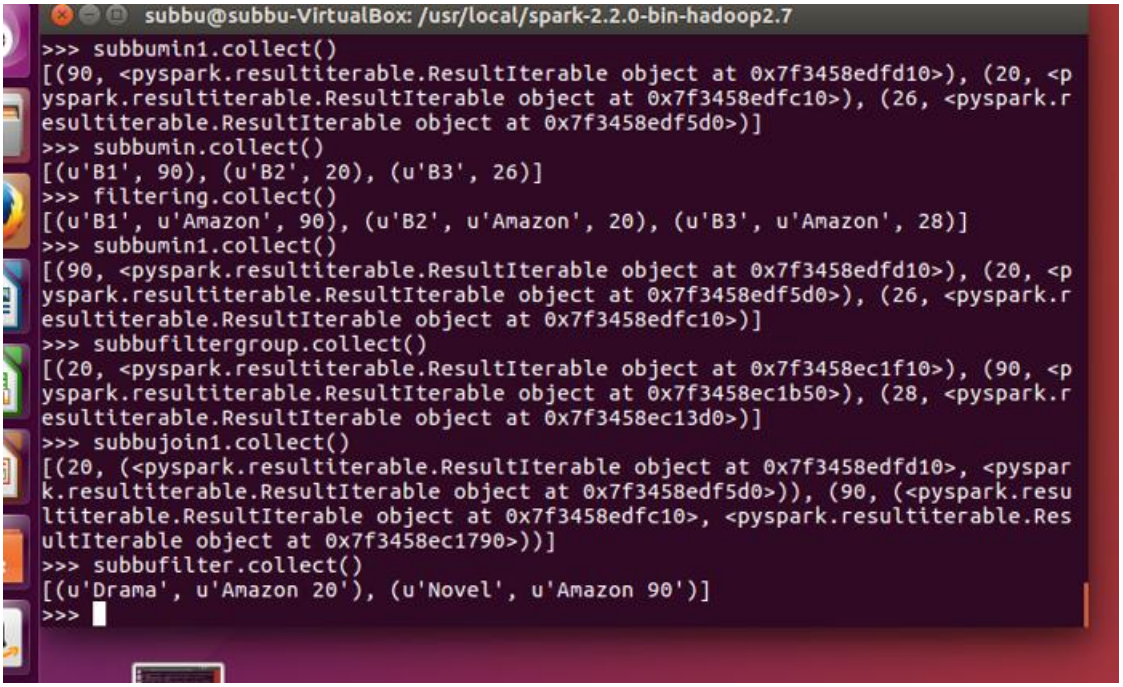
>>>subbugroup2 = subbubooklabel.groupBy(lambda v: v[0])//grouping info in book by book id

>>>Subbugroup2.collect()

>>>subbufilter = subbujoin1.join(subbugroup2)//joining result of subbujoin1 and grouped book table

>>>subbufilter.collect() // dispalys final output in subbufilter.

HERE IS THE OUTPUT FOR ABOVE CODE:



2.3.SPARKRDD:

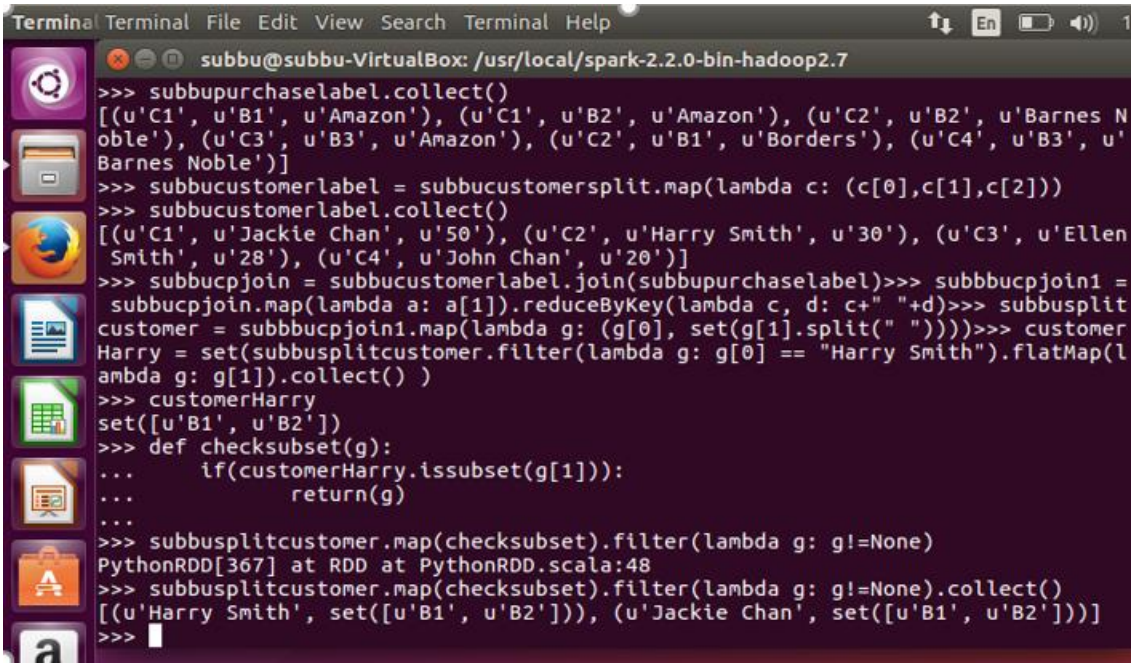
#EXPLANATION:

1. Load the purchase table and customer table initialize the variables after splitting data in them and use collect() to see information in them.
2. Join purchase table and customer table after initializing varibles to them.
3. Initialize variables and display required tupples from joined table using map().
4. Create a bag for customer harry using set in that filter harry and map () required tupples.
5. Write a condition to compare harry table and joined table to display common data
6. Compare and display common person who is having same books as harry. Display result by collect()

#CODE:

```
>>> subbupurchasetable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/purchase.txt')
                                                        //loading table from its location using 'sc'.
>>>subbupurchasesplit = subbupurchasetable.map(lambda z: z.split("\t"))
                                                        //splitting data tupples of 1st step into separate tupples
>>>subbupurchasesplit.collect()//displays split output.
>>>subbupurhcaselabel = subbupurchasesplit.map(lambda z: (z[1],z[2],z[3]))
                                                        //initializing data variables to required tuples of 3rd step.
>>>subbupurchaselabel.collect() // displays outcome of 4th step
>>>subbucustomertable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/customer.txt')
>>>subbucustomersplit = subbucustomertable.map(lambda c: c.split("\t"))//splitting data in
customer table
>>>subbucustomerlabel = subbucustomersplit.map(lambda c: (c[0],c[1],c[2]))//displaying
required tupples from customer table.
>>>subbucpjoin = subbucustomerlabel.join(subbupurchaselabel)//joining customer and purchase
tables.
>>>subbucpjoin1=subbucpjoin.map(lambda g: g[1]).reduceByKey(lambd g, f: g+"
"+f)//displaying tuple one and concatenating to stupples as one.
>>>subbuplitcustomer=subbucpjoin1.map(lambda v: (v[0],set(v[1].split(" ")))// split the data
>>>customerHarry=set(subbusplitcustomer.filter(lambda x:x[0]=="Harry
Smith").flatmap(lambda a: a[1]).collect()//filtering harrysmith from customer table and making
datails into one bag.
>>>customerHarry
>>> def checksubset(g);
…       if(customerHarry.issubset(g[1]))
…                   return (g)
…
>>> subbusplitcustomer.map(checksubset).filter(lambda g: g!=None).collect() // entire steps
from def checksubset to this step comparing match with filtered Harry Smith table and
displaying result who is having similar set of books to him
```

HERE IS THE OUTPUT SCREENSHOTS OF ENTIRE ABOVE PROGRAM:



```
Terminal Terminal File Edit View Search Terminal Help
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7
>>> subbupurchaselabel.collect()
[(u'C1', u'B1', u'Amazon'), (u'C1', u'B2', u'Amazon'), (u'C2', u'B2', u'Barnes N
oble'), (u'C3', u'B3', u'Amazon'), (u'C2', u'B1', u'Borders'), (u'C4', u'B3', u'
Barnes Noble')]
>>> subbucustomerlabel = subbucustomersplit.map(lambda c: (c[0],c[1],c[2]))
>>> subbucustomerlabel.collect()
[(u'C1', u'Jackie Chan', u'50'), (u'C2', u'Harry Smith', u'30'), (u'C3', u'Ellen
 Smith', u'28'), (u'C4', u'John Chan', u'20')]
>>> subbucpjoin = subbucustomerlabel.join(subbupurchaselabel)>>> subbbucpjoin1 =
 subbucpjoin.map(lambda a: a[1]).reduceByKey(lambda c, d: c+" "+d)>>> subbusplit
customer = subbbucpjoin1.map(lambda g: (g[0], set(g[1].split(" "))))>>> customer
Harry = set(subbusplitcustomer.filter(lambda g: g[0] == "Harry Smith").flatMap(l
ambda g: g[1]).collect() )
>>> customerHarry
set([u'B1', u'B2'])
>>> def checksubset(g):
...     if(customerHarry.issubset(g[1])):
...             return(g)
...
>>> subbusplitcustomer.map(checksubset).filter(lambda g: g!=None)
PythonRDD[367] at RDD at PythonRDD.scala:48
>>> subbusplitcustomer.map(checksubset).filter(lambda g: g!=None).collect()
[(u'Harry Smith', set([u'B1', u'B2'])), (u'Jackie Chan', set([u'B1', u'B2']))]
>>>
```

2.3.SPARKSQL:

#EXPALANTION:

1. First we have import the packages of spl using import* to pyspark shell.
2. Next load input files of purchase and customer using sc.textFile after that label all required tuples using row(). And create DataFrame to it and display Data Frame using collect()
3. Then create temporary name to DataFRame using createOrReplaceTempView() and display this table using show().
4. Join customer table and purchase table and create temporary name to it. And display it by show().
5. Filter joined table where age>=28 condition.
6. Create dataframe to it and give name to this filtered table by CreateOrReplaceTempView() and display it by using .show()
7. write nested sql command for filtered table and previously joined table to display common result by comparing with Harry Smith by doing group BY seller name.
8. finally display the results using .show()

#CODE:

```
>>>from pyspark.sql import*

>>> subbupurchasetable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/purchase.txt')

                                                        //loading table from its location using 'sc'.
>>>subbucustomertable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/customer.txt')
                                                        //loading purchase table using sc.tetxFile
>>>subbupurchasesplit = subbupurchasetable.map(lambda g: g.split("\t"))//splitting tupples
>>>subbucustomersplit = subbucustomertable.map(lambda f: f.split("\t"))//splitting into tupples
>>>subbupurhcaselable = subbupurchasesplit.map(lamda x: Row(custId=x[1], bid=x[2],
Name=x[3]))//taking required tupples and naming them
>>>subbucustomerlabel = subbucustomerspli.map(lambda f: Row(custId=f[0], CName=f[1],
Cage=f[2])) // taking required data and naming them
>>> subbuDf=sqlContext.createDataFrame(subbucustomerlabel) // creating data frame
>>>subbuDf.createOrReplaceTempView("subbucustomer") // naming data frame
>>>subbuDf1 = sqlContext.createDataFrame(subbupurchaselabel)// creating data frame to
purchase.
>>>subbuDf1.createOrReplaceTempView("subbupurchase") // naming data frame of purchase
>>>subbuDf.show()// display table of customer Data Frame
>>>subbuDf1.show()// display table of purchase data Frame
>>>subbujoin = spark.sql("select * from subbupurchase d JOIN Subbucustomer f ON d.custId =
f.custId") // joining customer dataframe and purchase DataFrame using Customer Id's
>>>subbujoin.createOrReplaceTempView("resulttable")// naming joined table
>>>subbujoin.show()//shows joined tables
>>>subbufilterjoin = subbujoin.filter(subbujoin['age']>=28)// filtering joined table by age
>>>subbufilterjoin.createOrReplaceTempView("result1table")//naming filtered table
>>>subbufilterjoin.show()
>>>spark.sql("select CName from resulttable WHERE bid IN(select bid from result1table where
CName=\"Harry Smith\") group by CName).show()    // displays final result after compring data
                                                        With Harry Smith.
```

HERE IS THE SCREEN SHOT OF FINAL OUTPUT FOR ABOVE CODE:

sss

```
Terminal File Edit View Search Terminal Help                    En   4:58 PM
>>> subbufilterjoin.show()
+------------+---+---+---+---+-----------+
|       SNmae|bid|cid|age|cid|       name|
+------------+---+---+---+---+-----------+
|      Amazon| B3| C3| 28| C3|Ellen Smith|
|      Amazon| B1| C1| 50| C1|Jackie Chan|
|      Amazon| B2| C1| 50| C1|Jackie Chan|
|Barnes Noble| B2| C2| 30| C2|Harry Smith|
|     Borders| B1| C2| 30| C2|Harry Smith|
+------------+---+---+---+---+-----------+

>>> spark.sql("select name from resulttable WHERE bid IN(select bid from result1table WH
ERE name=\"HARRY SMITH\") group by name").show()
+----+
|name|
+----+
+----+

>>> subbufilterjoin=subbujoin.filter(subbujoin['name'] =\"Harry Smith\")
  File "<stdin>", line 1
    subbufilterjoin=subbujoin.filter(subbujoin['name'] =\"Harry Smith\")
                                                                       ^
SyntaxError: unexpected character after line continuation character
>>> spark.sql("select name from resulttable WHERE bid IN(select bid from result1table WH
ERE name=\"Harry Smith\") group by name").show()
+-----------+
|       name|
+-----------+
|Harry Smith|
|Jackie Chan|
+-----------+

>>> spark.sql("select name from resulttable WHERE bid IN(select bid from result1table WH
ERE name!=\"Harry Smith\") group by name").show()
+-----------+
|       name|
+-----------+
|Harry Smith|
|Ellen Smith|
|  John Chan|
|Jackie Chan|
+-----------+

>>> spark.sql("select name from resulttable WHERE bid IN(select bid from result1table WH
ERE name=\"Harry Smith\") group by name").show()
+-----------+
|       name|
+-----------+
|Harry Smith|
|Jackie Chan|
+-----------+

>>>
```