

Relational Data Analysis With Pig And Spark

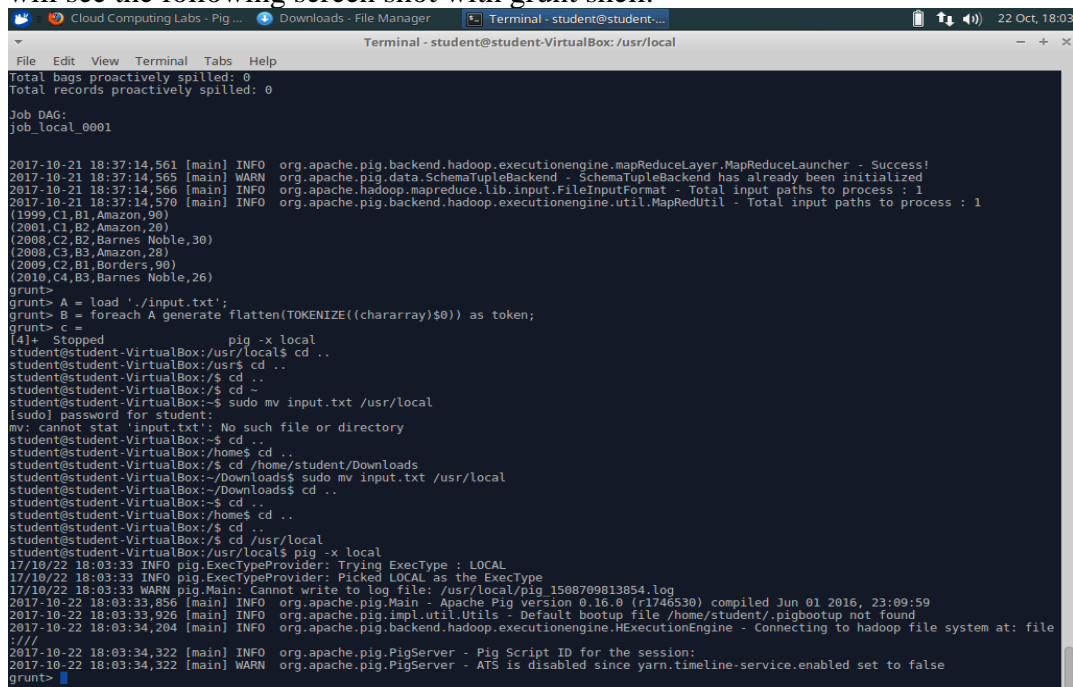
SUBHADRA TUMMALAPALLY

(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.



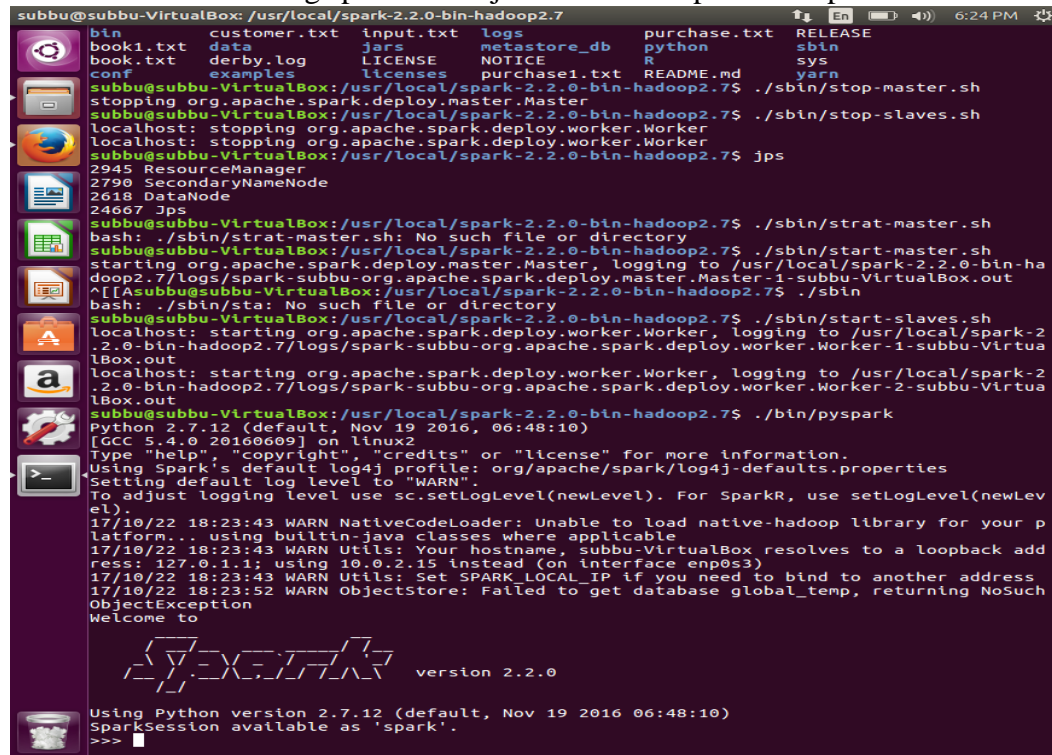
```
Cloud Computing Labs - Pig ... Downloads - File Manager Terminal - student@student-... 22 Oct, 18:03
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0001
2017-10-21 18:37:14,561 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2017-10-21 18:37:14,565 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-10-21 18:37:14,566 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-10-21 18:37:14,570 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(1999,C1,B1,Amazon,90)
(2001,C1,B2,Amazon,20)
(2008,C2,B2,Barnes Noble,30)
(2008,C3,B3,Amazon,28)
(2009,C2,B1,Borders,90)
(2010,C4,B3,Barnes Noble,26)
grunt>
grunt> A = load './input.txt';
grunt> B = foreach A generate flatten(TOKENIZE((chararray)$0)) as token;
grunt> c =
[4]+ Stopped pig -x local
student@student-VirtualBox:~/usr/local$ cd ..
student@student-VirtualBox:~/usr$ cd ..
student@student-VirtualBox:/$ cd ..
student@student-VirtualBox:/$ cd -
student@student-VirtualBox:~$ sudo mv input.txt /usr/local
[sudo] password for student:
mv: cannot stat 'input.txt': No such file or directory
student@student-VirtualBox:~/home$ cd ..
student@student-VirtualBox:/$ cd /home/student/Downloads
student@student-VirtualBox:~/Downloads$ sudo mv input.txt /usr/local
student@student-VirtualBox:~/Downloads$ cd ..
student@student-VirtualBox:/$ cd ..
student@student-VirtualBox:~/home$ cd ..
student@student-VirtualBox:/$ cd ..
student@student-VirtualBox:/$ cd /usr/local
student@student-VirtualBox:~/usr/local$ pig -x local
17/10/22 18:03:33 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
17/10/22 18:03:33 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
17/10/22 18:03:33 WARN pig.Main: Cannot write to log file: /usr/local/pig/1508709813854.log
2017-10-22 18:03:33,856 [main] INFO org.apache.pig.Main - Apache Pig version 0.16.0 (r1746530) compiled Jun 01 2016, 23:09:59
2017-10-22 18:03:33,926 [main] INFO org.apache.pig.impl.util.Utils - Default bootstrap file /home/student/.pigbootstrap not found
2017-10-22 18:03:34,204 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///
2017-10-22 18:03:34,322 [main] INFO org.apache.pig.PigServer - Pig Script ID for the session:
2017-10-22 18:03:34,322 [main] WARN org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
grunt>
```

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.



```
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7
bin      customer.txt  input.txt  logs      purchase.txt  RELEASE
book1.txt data      jars      metastore_db  python      sbtln
book.txt  derby.log  LICENSE   NOTICE    R           sys
conf      examples  licenses  purchase1.txt  README.md   yarn

subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./sbin/stop-master.sh
stopping org.apache.spark.deploy.master.Master
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./sbin/stop-slaves.sh
localhost: stopping org.apache.spark.deploy.worker.Worker
localhost: stopping org.apache.spark.deploy.worker.Worker
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ jps
2945 ResourceManager
2790 SecondaryNameNode
2618 DataNode
24667 Jps
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./sbin/start-master.sh
bash: ./sbin/start-master.sh: No such file or directory
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./sbin/start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark-2.2.0-bin-hadoop2.7/logs/spark-subbu-org.apache.spark.deploy.master.Master-1-subbu-VirtualBox.out
^[[subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./sbin
bash: ./sbin/sta: No such file or directory
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./sbin/start-slaves.sh
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark-2.2.0-bin-hadoop2.7/logs/spark-subbu-org.apache.spark.deploy.worker.Worker-1-subbu-VirtualBox.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark-2.2.0-bin-hadoop2.7/logs/spark-subbu-org.apache.spark.deploy.worker.Worker-2-subbu-VirtualBox.out
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7$ ./bin/pyspark
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license()" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/10/22 18:23:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/10/22 18:23:43 WARN Utils: Your hostname, subbu-VirtualBox resolves to a loopback address: 127.0.0.1; using 10.0.2.15 instead (on interface enp0s3)
17/10/22 18:23:43 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/10/22 18:23:52 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Welcome to
      _ _ _ _ _
     / _ _ _ _ \   version 2.2.0
    / _ _ _ _ \
   / _ _ _ _ \
  / _ _ _ _ \
 / _ _ _ _ \
/_ _ _ _ _ \

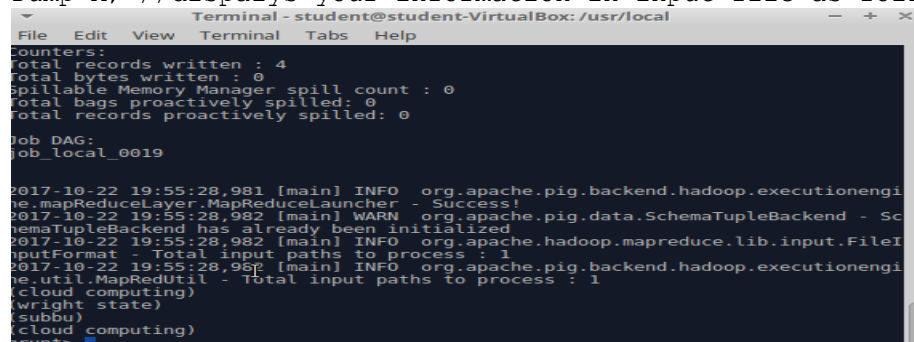
Using Python version 2.7.12 (default, Nov 19 2016 06:48:10)
SparkSession available as 'spark'.
>>>
```

ANSWERS OF QUESTION 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; // displays your information in input file as follows.



```
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Counters:
Total records written : 4
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0019

2017-10-22 19:55:28,981 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapreduce_layer.MapReduceLauncher - Success!
2017-10-22 19:55:28,982 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-10-22 19:55:28,982 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-10-22 19:55:28,982 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapReduceUtil - Total input paths to process : 1
(cloud computing)
(wright state)
(subbu)
(cloud computing)
input
```

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 \$0 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.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 19:56:49,169 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-10-22 19:56:49,170 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-10-22 19:56:49,171 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(ccloud)
(computing)
(wright)
(state)
(subbu)
(ccloud)
(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.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 19:58:21,839 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2017-10-22 19:58:21,840 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2017-10-22 19:58:21,840 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(ccloud,((ccloud),(ccloud)))
(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-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_local_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 18:06:17,790 [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
duceTime
job_local_0001  1  1  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
(ccloud,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.

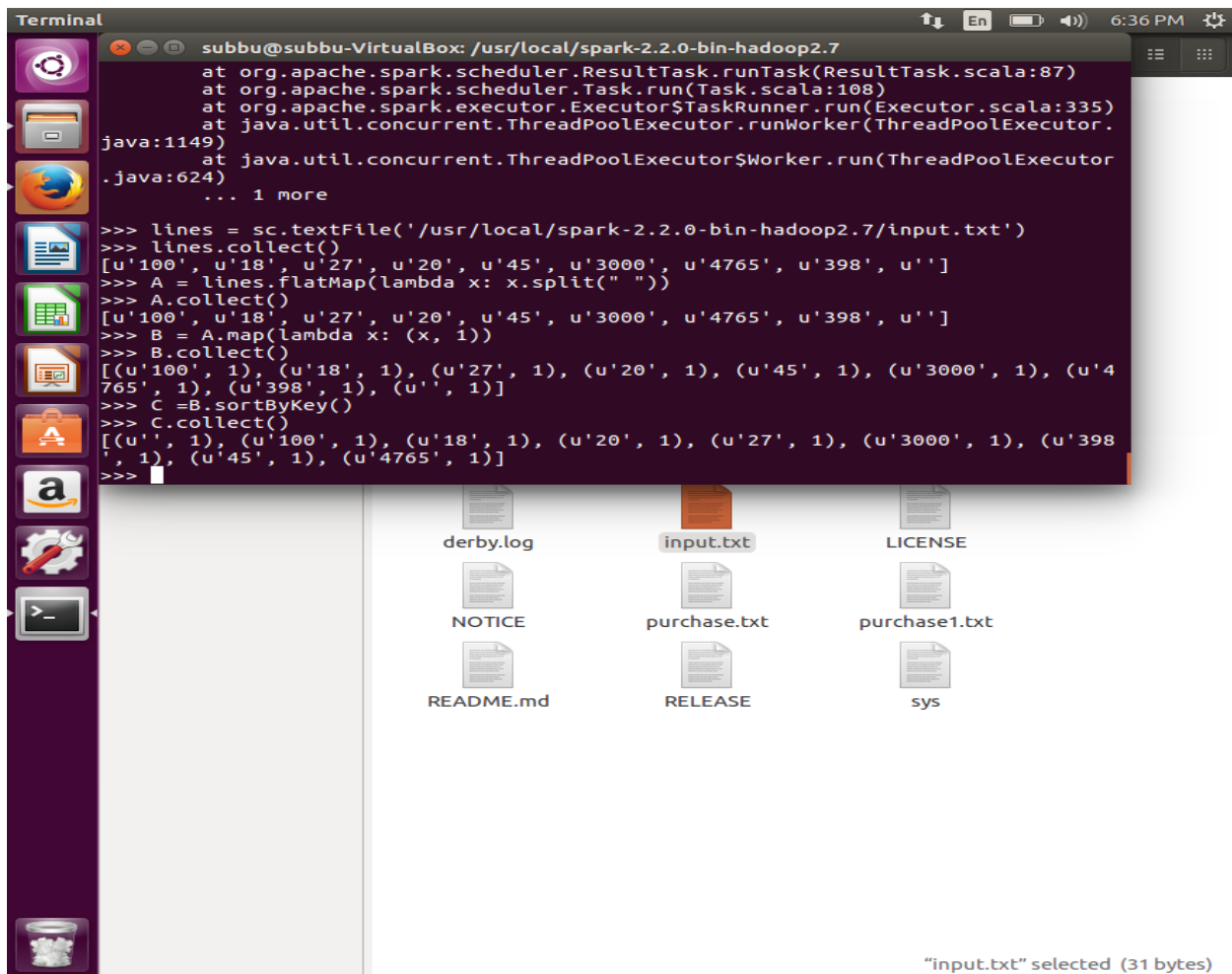
The code in 1.2 works as follows:

lines = spark.read.text(sys.argv[1]).rdd.map(**lambda** r: r[0])// loading input from the input location and lambda is used to initialise variable to name rows in input file r[0] displays first tuple in input file.

A = lines.flatMap(**lambda** x: x.split(' '))// splitting information which is stored in line and stored 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 displaying by giving c.collect() which is shown below.



The image shows a terminal window titled "Terminal" with the user "subbu" at "subbu-VirtualBox". The terminal displays the execution of Spark code. The code reads a file "input.txt" from "/usr/local/spark-2.2.0-bin-hadoop2.7/", splits it into words, maps each word to a count of 1, sorts the results by key, and finally collects the output. The output shows a list of words and their counts, sorted in ascending order. Below the terminal, a file explorer view shows the contents of the "spark" folder, including files like "derby.log", "input.txt", "LICENSE", "NOTICE", "purchase.txt", "purchase1.txt", "README.md", "RELEASE", and "sys". The "input.txt" file is selected, and a status bar at the bottom indicates "input.txt" selected (31 bytes).

```
at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:87)
at org.apache.spark.scheduler.Task.run(Task.scala:108)
at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:335)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.
java:1149)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor
.java:624)
... 1 more

>>> lines = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/input.txt')
>>> lines.collect()
[u'100', u'18', u'27', u'20', u'45', u'3000', u'4765', u'398', u'']
>>> A = lines.flatMap(lambda x: x.split(" "))
>>> A.collect()
[u'100', u'18', u'27', u'20', u'45', u'3000', u'4765', u'398', u'']
>>> B = A.map(lambda x: (x, 1))
>>> B.collect()
[(u'100', 1), (u'18', 1), (u'27', 1), (u'20', 1), (u'45', 1), (u'3000', 1), (u'4
765', 1), (u'398', 1), (u'', 1)]
>>> C = B.sortByKey()
>>> C.collect()
[(u'', 1), (u'100', 1), (u'18', 1), (u'20', 1), (u'27', 1), (u'3000', 1), (u'398
', 1), (u'45', 1), (u'4765', 1)]
>>>
```

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.

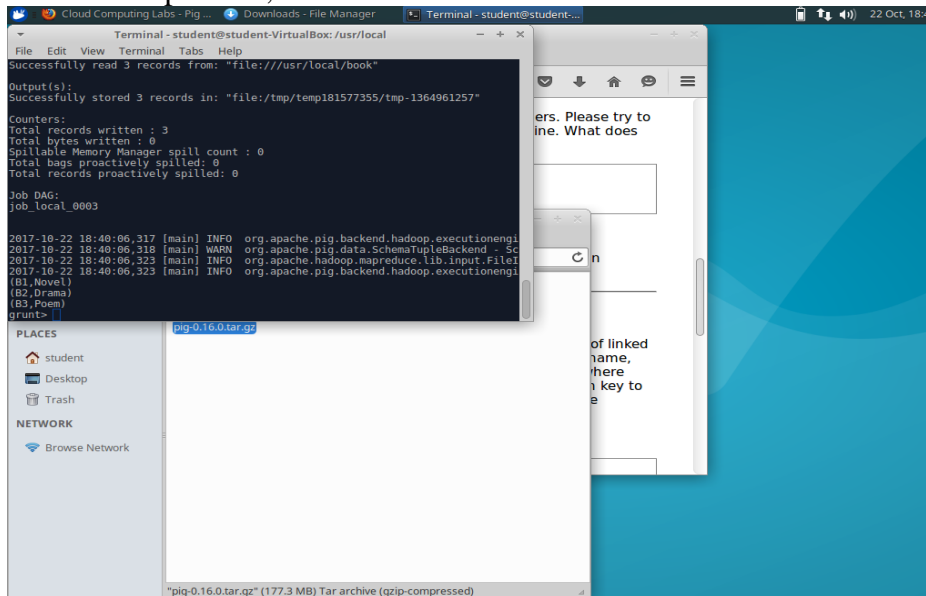
The code for each line in explanation is below:

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;

```
Cloud Computing Labs - Pig ... Downloads - File Manager Terminal - student@
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0004
2017-10-22 18:41:43,549 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 18:41:43,550 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
HEMA TupleBackend has already been initialized
2017-10-22 18:41:43,558 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 18:41:43,559 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(1999,C1,B1,Amazon,90)
(2001,C1,B2,Amazon,20)
(2008,C2,B2,Barnes Noble,30)
(2008,C3,B3,Amazon,20)
(2009,C2,B1,Borders,90)
(2010,C4,B3,Barnes Noble,26)
grunt>
```

Grunt>A = group create2 by SName;

Grunt> dump A;

```
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Counters:
Total records written : 3
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0026
2017-10-22 21:09:36,736 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 21:09:36,737 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
HEMA TupleBackend has already been initialized
2017-10-22 21:09:36,738 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 21:09:36,738 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(Amazon,{(1999,C1,B1,Amazon,90),(2001,C1,B2,Amazon,20),(2008,C3,B3,Amazon,28)})
(Borders,{(2009,C2,B1,Borders,90)})
(Barnes Noble,{(2008,C2,B2,Barnes Noble,30),(2010,C4,B3,Barnes Noble,26)})
grunt>
```

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

Grunt> dump B;

```
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Counters:
Total records written : 3
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0027
2017-10-22 21:14:00,887 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 21:14:00,888 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
HEMA TupleBackend has already been initialized
2017-10-22 21:14:00,888 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 21:14:00,888 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(Amazon,138)
(Borders,90)
(Barnes Noble,56)
grunt>
```


2.1.SPARKRDD:

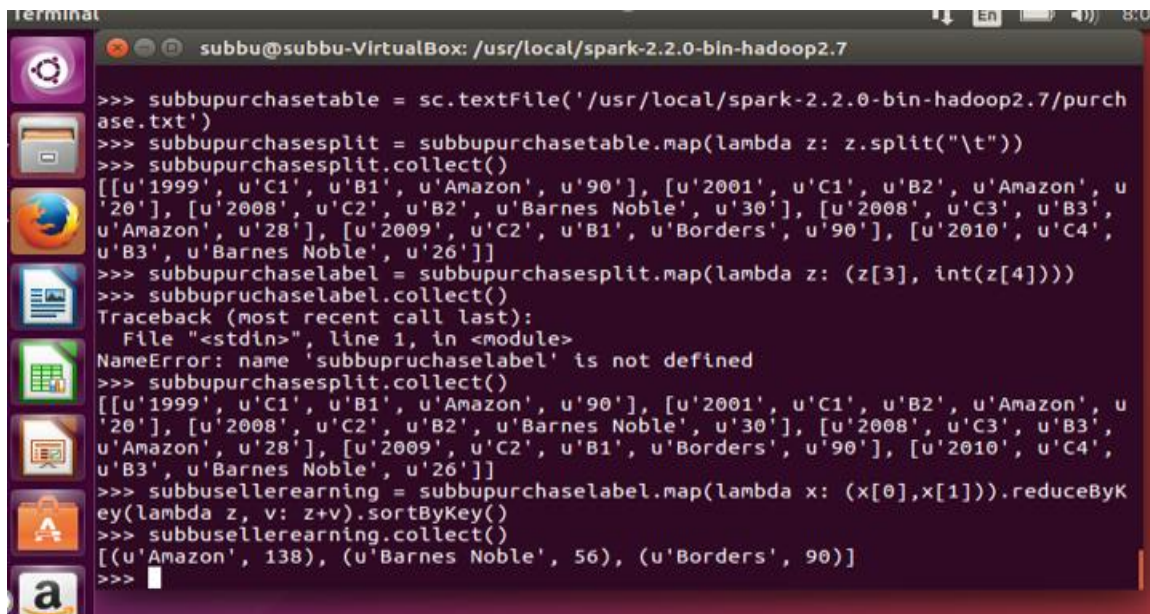
#EXPLANATION:

1. Load input file from its location using spark context as SC.
2. Split the output of 1st step into different tuples.
3. By using collect() display output of split operation
4. Display required tuples in your table and initialise 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 tuples of 1st step into separate tuples
>>>subbupurchasesplit.collect()//displays split output.
>>>subbupurchaselabel = 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:



```
Terminal
subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7

>>> subbupurchasetable = sc.textFile('/usr/local/spark-2.2.0-bin-hadoop2.7/purchase.txt')
>>> subbupurchasesplit = subbupurchasetable.map(lambda z: z.split("\t"))
>>> subbupurchasesplit.collect()
[[('1999', 'C1', 'B1', 'Amazon', '90'), ('2001', 'C1', 'B2', 'Amazon', '20'), ('2008', 'C2', 'B2', 'Barnes Noble', '30'), ('2008', 'C3', 'B3', 'Amazon', '28'), ('2009', 'C2', 'B1', 'Borders', '90'), ('2010', 'C4', 'B3', 'Barnes Noble', '26')]
>>> subbupurchaselabel = subbupurchasesplit.map(lambda z: (z[3], int(z[4])))
>>> subbupurchaselabel.collect()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'subbupurchaselabel' is not defined
>>> subbupurchasesplit.collect()
[[('1999', 'C1', 'B1', 'Amazon', '90'), ('2001', 'C1', 'B2', 'Amazon', '20'), ('2008', 'C2', 'B2', 'Barnes Noble', '30'), ('2008', 'C3', 'B3', 'Amazon', '28'), ('2009', 'C2', 'B1', 'Borders', '90'), ('2010', 'C4', 'B3', 'Barnes Noble', '26')]
>>> subbusellerearning = subbupurchaselabel.map(lambda x: (x[0], x[1])).reduceByKey(lambda z, v: z+v).sortByKey()
>>> subbusellerearning.collect()
[(u'Amazon', 138), (u'Barnes Noble', 56), (u'Borders', 90)]
>>>
```

(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:chararray,cid:chararray, bid:chararray, SName:chararray, Bprice:int); //loading and initializing data types of purchase table.
```

```
Grunt> dump create; //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; //joining filtered Amazon and minimum 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.
```

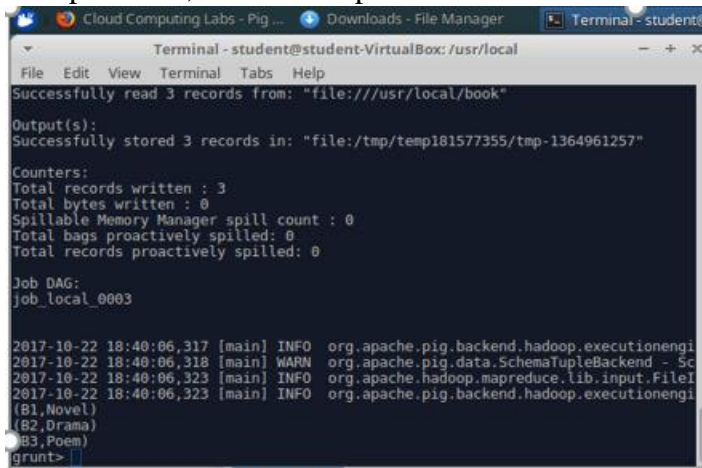
```
Grunt> dump joined;
```

```
Grunt>illustrate joined;
```

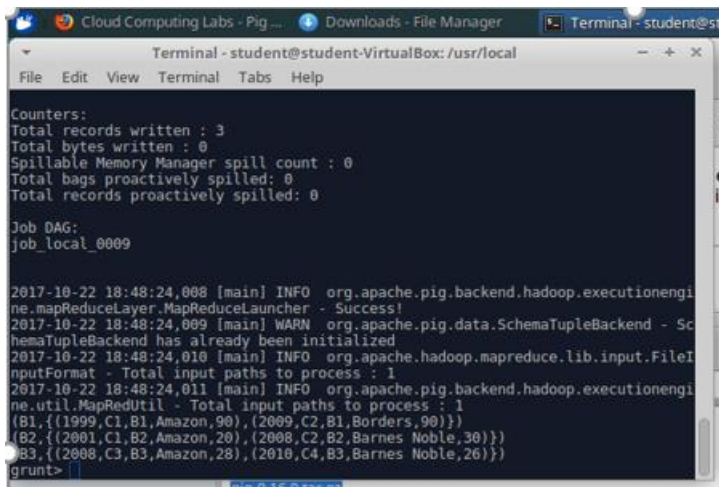
```
Grunt> store joined into "./output1";
```


DISPALYING OUTPUTS OF 2.2 stepwise:

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

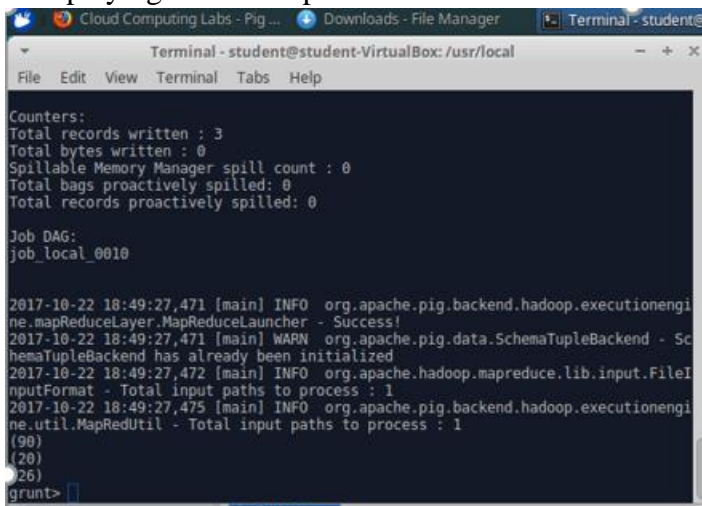


```
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Successfully read 3 records from: "file:///usr/local/book"
Output(s):
Successfully stored 3 records in: "file:/tmp/temp181577355/tmp-1364961257"
Counters:
Total records written : 3
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0003
2017-10-22 18:40:06,317 [main] INFO org.apache.pig.backend.hadoop.executionengi
2017-10-22 18:40:06,318 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
2017-10-22 18:40:06,323 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
2017-10-22 18:40:06,323 [main] INFO org.apache.pig.backend.hadoop.executionengi
(B1,Novel)
(B2,Drama)
(B3,Poem)
grunt>
```



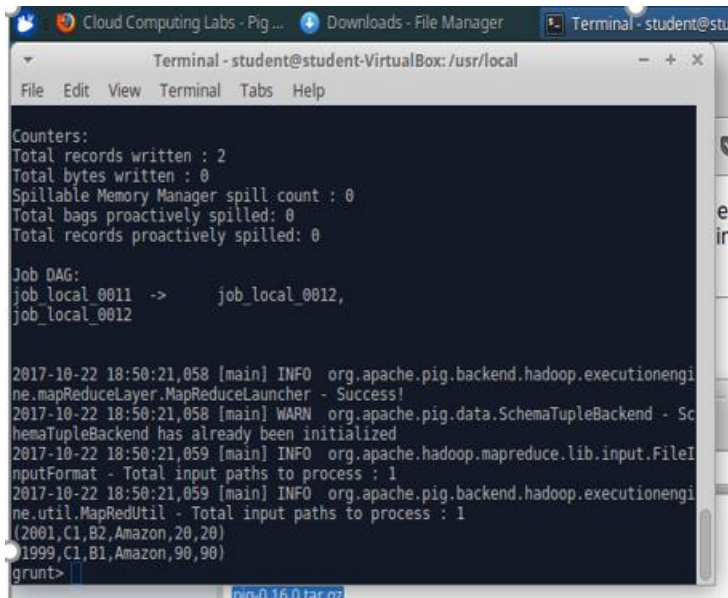
```
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Counters:
Total records written : 3
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0009
2017-10-22 18:48:24,008 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 18:48:24,009 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2017-10-22 18:48:24,010 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 18:48:24,011 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(B1,{(1999,C1,B1,Amazon,90),(2009,C2,B1,Borders,90)})
(B2,{(2001,C1,B2,Amazon,20),(2008,C2,B2,Barnes Noble,30)})
(B3,{(2008,C3,B3,Amazon,28),(2010,C4,B3,Barnes Noble,26)})
grunt>
```

3. displaying minimum price of all sellers



```
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help
Counters:
Total records written : 3
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local_0010
2017-10-22 18:49:27,471 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 18:49:27,471 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2017-10-22 18:49:27,472 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 18:49:27,475 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(90)
(20)
(26)
grunt>
```

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



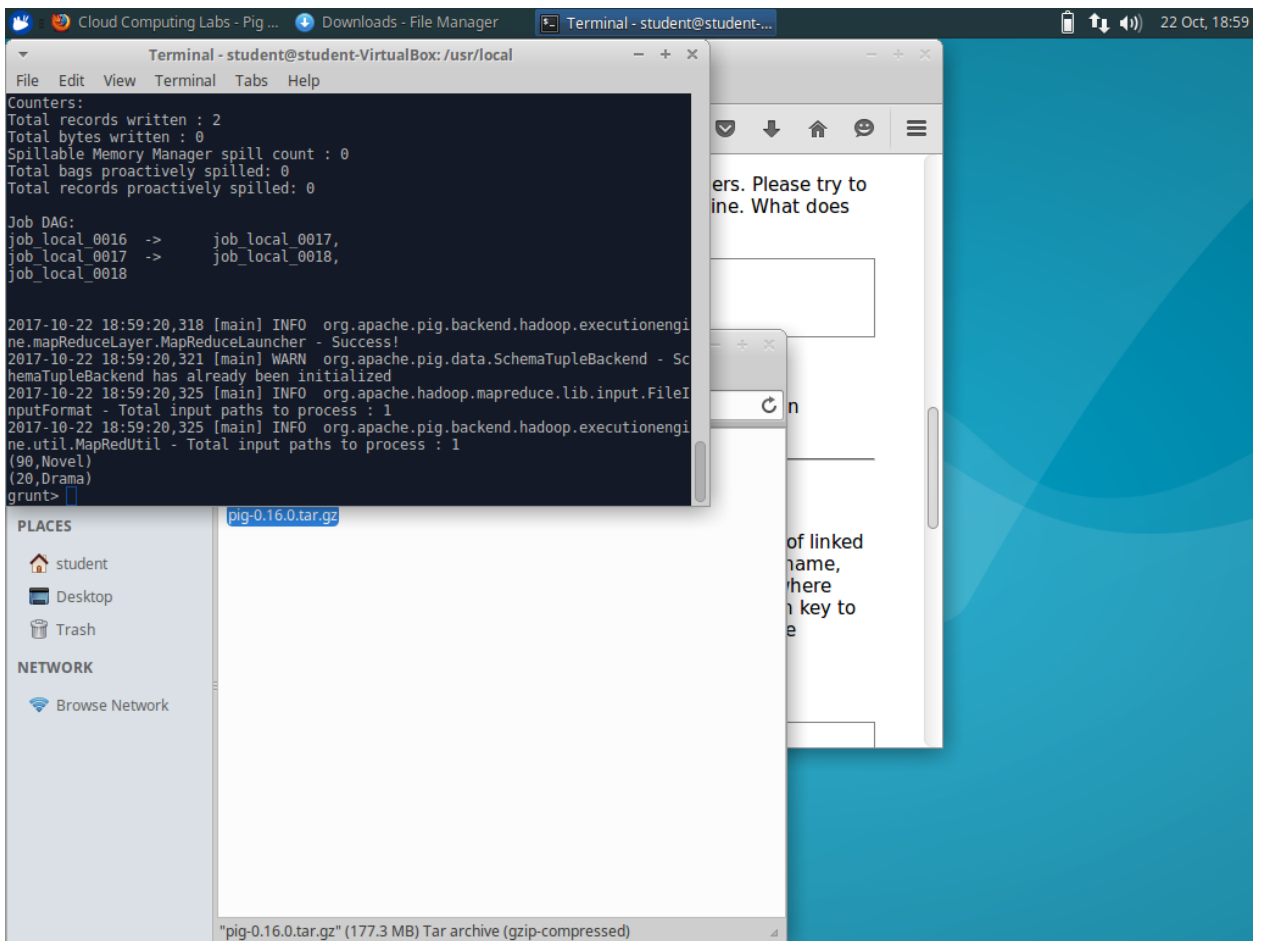
```
Cloud Computing Labs - Pig ... Downloads - File Manager Terminal - student@stud...
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help

Counters:
Total records written : 2
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local_0011 -> job_local_0012,
job_local_0012

2017-10-22 18:50:21,058 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 18:50:21,058 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2017-10-22 18:50:21,059 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 18:50:21,059 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(2001,C1,B2,Amazon,20,20)
(1999,C1,B1,Amazon,90,90)
grunt>
```

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



```
Cloud Computing Labs - Pig ... Downloads - File Manager Terminal - student@stud...
Terminal - student@student-VirtualBox: /usr/local
File Edit View Terminal Tabs Help

Counters:
Total records written : 2
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_local_0016 -> job_local_0017,
job_local_0017 -> job_local_0018,
job_local_0018

2017-10-22 18:59:20,318 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2017-10-22 18:59:20,321 [main] WARN org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2017-10-22 18:59:20,325 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2017-10-22 18:59:20,325 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(90,Novel)
(20,Drama)
grunt>
```

Places: student, Desktop, Trash

Network: Browse Network

pig-0.16.0.tar.gz

"pig-0.16.0.tar.gz" (177.3 MB) Tar archive (gzip-compressed)

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 tuples and initialise the variables 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 tuples from Amazon by initialising 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 Filtered 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 tuples of 1st step into separate tuples
>>>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 tuples
>>>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.

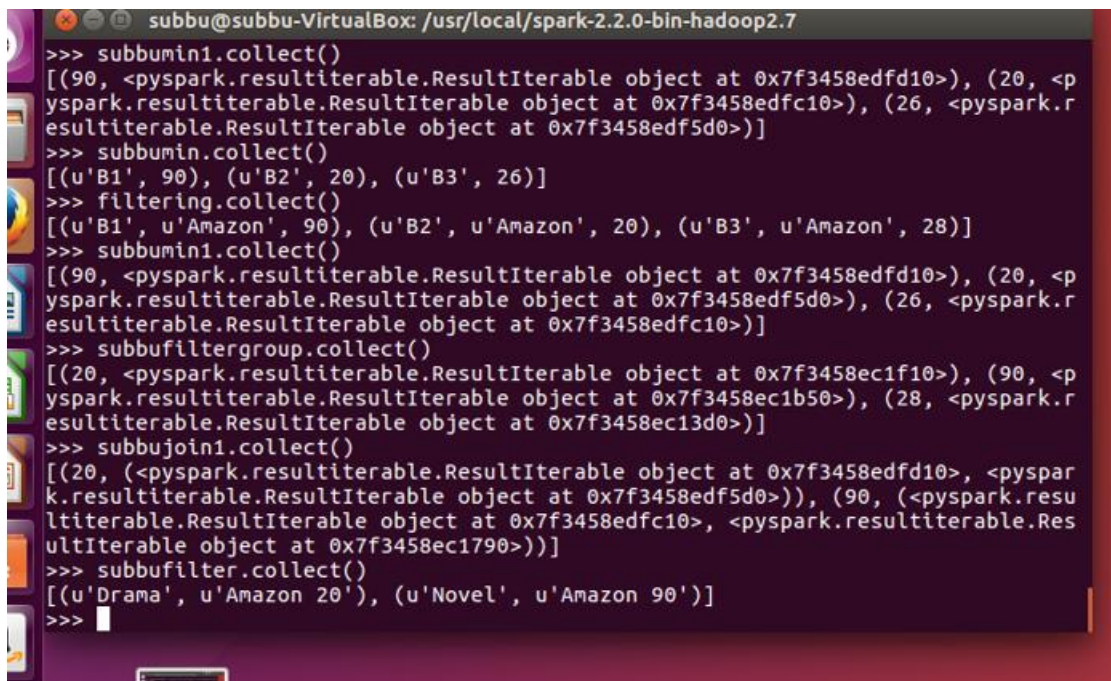
>>>subbugroup2 = subbubooklabel.groupBy(lambda v: v[0])//grouping info in book by book id
>>>Subbugroup2.collect()

>>>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:



```

subbu@subbu-VirtualBox: /usr/local/spark-2.2.0-bin-hadoop2.7
>>> subbumin1.collect()
[(90, <pyspark.resultiterable.ResultIterable object at 0x7f3458edfd10>), (20, <pyspark.resultiterable.ResultIterable object at 0x7f3458edfc10>), (26, <pyspark.resultiterable.ResultIterable object at 0x7f3458edf5d0>)]
>>> subbumin.collect()
[(u'B1', 90), (u'B2', 20), (u'B3', 26)]
>>> filteringsubbu.collect()
[(u'B1', u'Amazon', 90), (u'B2', u'Amazon', 20), (u'B3', u'Amazon', 28)]
>>> subbumin1.collect()
[(90, <pyspark.resultiterable.ResultIterable object at 0x7f3458edfd10>), (20, <pyspark.resultiterable.ResultIterable object at 0x7f3458edf5d0>), (26, <pyspark.resultiterable.ResultIterable object at 0x7f3458edfc10>)]
>>> subbufiltergroup.collect()
[(20, <pyspark.resultiterable.ResultIterable object at 0x7f3458ec1f10>), (90, <pyspark.resultiterable.ResultIterable object at 0x7f3458ec1b50>), (28, <pyspark.resultiterable.ResultIterable object at 0x7f3458ec13d0>)]
>>> subbujoin1.collect()
[(20, (<pyspark.resultiterable.ResultIterable object at 0x7f3458edfd10>, <pyspark.resultiterable.ResultIterable object at 0x7f3458edf5d0>)), (90, (<pyspark.resultiterable.ResultIterable object at 0x7f3458edfc10>, <pyspark.resultiterable.ResultIterable object at 0x7f3458ec1790>))]
>>> subbufilter.collect()
[(u'Drama', u'Amazon 20'), (u'Novel', u'Amazon 90')]
>>>

```

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 variables to them.
3. Initialize variables and display required tuples from joined table using map().
4. Create a bag for customer harry using set in that filter harry and map () required tuples.
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 tuples of 1st step into separate tuples
>>>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 tuples from customer table.
>>>subbucpjoin = subbucustomerlabel.join(subbupurchaselabel)//joining customer and purchase
tables.
>>>subbucpjoin1=subbucpjoin.map(lambda g: g[1]).reduceByKey(lambda g, f: g+"
+f")//displaying tuple one and concatenating to stuples as one.
>>>subbusplitcustomer=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
details 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 tuples
>>>subbucustomersplit = subbucustomertable.map(lambda f: f.split("\t"))//splitting into tuples
>>>subbupurhcasetable = subbupurchasesplit.map(lamda x: Row(custId=x[1], bid=x[2],
Name=x[3]))//taking required tuples 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(subbupurchasetable)// 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
>>> 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|
+-----+

>>>
```