

1) ls: To display a list of the contents of a directory

```
[root@sandbox ~]# hadoop fs -ls /
```

```
Found 14 items
```

```
drwxr-xr-x      - root      hdfs      0 2019-10-14 09:08 /MR
drwxrwxrwx     - yarn      hadoop    0 2016-03-14 14:19 /app-logs
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:25 /apps
drwxr-xr-x     - yarn      hadoop    0 2016-03-14 14:19 /ats
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:50 /demo
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:19 /hdp
drwxr-xr-x     - mapred    hdfs      0 2016-03-14 14:19 /mapred
drwxrwxrwx     - mapred    hadoop    0 2016-03-14 14:19 /mr-history
drwxr-xr-x     - root      hdfs      0 2019-10-14 10:12 /myfiles
drwxr-xr-x     - root      hdfs      0 2019-10-14 08:52 /new
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:42 /ranger
drwxrwxrwx     - spark     hadoop    0 2019-02-21 11:51 /spark-history
drwxrwxrwx     - hdfs      hdfs      0 2016-03-14 14:31 /tmp
drwxr-xr-x     - hdfs      hdfs      0 2019-10-14 08:48 /user
```

2) mkdir: To Create a Directory in HDFS

```
[root@sandbox ~]# hadoop fs -mkdir new
```

```
[root@sandbox ~]# hadoop fs -ls /
```

```
Found 14 items
```

```
drwxr-xr-x      - root      hdfs      0 2019-10-14 09:08 /MR
drwxrwxrwx     - yarn      hadoop    0 2016-03-14 14:19 /app-logs
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:25 /apps
drwxr-xr-x     - yarn      hadoop    0 2016-03-14 14:19 /ats
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:50 /demo
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:19 /hdp
drwxr-xr-x     - mapred    hdfs      0 2016-03-14 14:19 /mapred
drwxrwxrwx     - mapred    hadoop    0 2016-03-14 14:19 /mr-history
drwxr-xr-x     - root      hdfs      0 2019-10-14 10:12 /myfiles
drwxr-xr-x     - root      hdfs      0 2019-10-14 08:52 /new
drwxr-xr-x     - hdfs      hdfs      0 2016-03-14 14:42 /ranger
drwxrwxrwx     - spark     hadoop    0 2019-02-21 11:51 /spark-history
drwxrwxrwx     - hdfs      hdfs      0 2016-03-14 14:31 /tmp
drwxr-xr-x     - hdfs      hdfs      0 2019-10-14 08:48 /user
```

3) put (or) copyFromLocal :copies the file or directory from the local file system to the HDFS

```
[root@sandbox ~]# cat > FirstFile
```

```
This is a Unix File created in Unix Local system which will be pushed to HDFS
```

```
^Z
```

```
[1] + Stopped          cat > FirstFile
```

```
[root@sandbox ~]# cat FirstFile
```

```
This is a Unix File created in Unix Local system which will be pushed to HDFS
```

```
[root@sandbox ~]# hadoop fs -put FirstFile new/
```

```
[root@sandbox ~]# hadoop fs -ls new/
```

```
Found 1 items
```

```
-rw-r--r-- 3 root hdfs 78 2019-10-31 10:03 new/FirstFile
```

```
[root@sandbox ~]# hadoop fs -cat new/FirstFile
```

```
This is a Unix File created in Unix Local system which will be pushed to HDFS
```

```
[root@sandbox local]# hadoop fs -copyFromLocal file1 new
```

```
[root@sandbox local]# hadoop fs -ls new
```

```
Found 2 items
```

```
-rw-r--r-- 3 root hdfs 78 2019-10-31 10:03 new/FirstFile
```

```
-rw-r--r-- 3 root hdfs 42 2019-10-31 10:21 new/file1
```

4) get (or) copyToLocal :copies the file or directory from the HDFS to Local File System

```
[root@sandbox ~]# hadoop fs -get new/FirstFile local/
```

```
[root@sandbox ~]# cd local
```

```
[root@sandbox local]# ls
```

```
FirstFile
```

```
[root@sandbox local]# cat FirstFile
```

```
This is a Unix File created in Unix Local system which will be pushed to HDFS
```

```
[root@sandbox ~]# hadoop fs -copyToLocal new/file1 newLocal/
```

```
[root@sandbox ~]# ls newLocal
```

```
file1
```

5) cp :copies the file or directory from one location to another in HDFS

```
[root@sandbox local]# hadoop fs -mkdir /latest
```

```
[root@sandbox local]# hadoop fs -cp new/FirstFile /latest
```

```
[root@sandbox local]# hadoop fs -ls /latest
```

```
Found 1 items
```

```
-rw-r--r-- 3 root hdfs 78 2019-10-31 10:16 /latest/FirstFile
```

```
[root@sandbox local]# hadoop fs -cat /latest/FirstFile
```

```
This is a Unix File created in Unix Local system which will be pushed to HDFS
```

6) mv :moves the file or directory from one location to another in HDFS

```
[root@sandbox local]# cat > dummy
```

```
This is a dummy file which is created in Unix
```

```
^Z
```

```
[2] + Stopped cat > dummy
```

```
[root@sandbox local]# cat dummy
```

```
This is a dummy file which is created in Unix
```

```
[root@sandbox local]# hadoop fs -put dummy new/
```

```
[root@sandbox local]# hadoop fs -mv new/dummy /latest/
```

```
[root@sandbox local]# hadoop fs -ls latest
```

```
[root@sandbox local]# hadoop fs -ls /latest
```

Found 2 items

```
-rw-r--r-- 3 root hdfs 78 2019-10-31 10:16 /latest/FirstFile
-rw-r--r-- 3 root hdfs 46 2019-10-31 10:18 /latest/dummy
```

7) **rm,rmdir** :removes the file or directory from the HDFS

```
[root@sandbox ~]# hadoop fs -ls new
```

Found 2 items

```
-rw-r--r-- 3 root hdfs 78 2019-10-31 10:03 new/FirstFile
-rw-r--r-- 3 root hdfs 42 2019-10-31 10:21 new/file1
```

```
[root@sandbox ~]# hadoop fs -rm new/file1
```

19/10/31 10:33:42 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 360 minutes, Emptier interval = 0 minutes.

Moved: 'hdfs://sandbox.hortonworks.com:8020/user/root/new/file1' to trash at:

hdfs://sandbox.hortonworks.com:8020/user/root/.Trash/Current

```
[root@sandbox ~]# hadoop fs -ls new
```

Found 1 items

```
-rw-r--r-- 3 root hdfs 78 2019-10-31 10:03 new/FirstFile
```

```
[root@sandbox ~]# hadoop fs -rmdir new
```

rmdir: `new': Directory is not empty

```
[root@sandbox ~]# hadoop fs -rm-r new
```

-rm-r: Unknown command

```
[root@sandbox ~]# hadoop fs -rm -r new
```

19/10/31 10:35:49 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 360 minutes, Emptier interval = 0 minutes.

Moved: 'hdfs://sandbox.hortonworks.com:8020/user/root/new' to trash at:

hdfs://sandbox.hortonworks.com:8020/user/root/.Trash/Current

8) **touchz** : To Create an empty file in HDFS

```
[root@sandbox ~]# hadoop fs -ls /rpk
```

Found 1 items

```
-rw-r--r-- 3 root hdfs 95 2021-01-18 09:42 /rpk/states.txt
```

```
[root@sandbox ~]# hadoop fs -touchz /rpk/empty.txt
```

```
[root@sandbox ~]# hadoop fs -ls /rpk
```

Found 2 items

```
-rw-r--r-- 3 root hdfs 0 2021-01-30 08:34 /rpk/empty.txt
-rw-r--r-- 3 root hdfs 95 2021-01-18 09:42 /rpk/states.txt
```

9) **count** : To Count the number of Directories, files and bytes under the path

```
[root@sandbox ~]# hadoop fs -count /rpk
```

```
1      2      95 /rpk
```

10) usage : Returns the Usage for an Individual command

```
[root@sandbox ~]# hadoop fs -usage rm
```

```
Usage: hadoop fs [generic options] -rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...
```

11) help: Displays help for given command or all commands if none is specified

```
[root@sandbox ~]# hadoop fs -help mkdir
```

```
-mkdir [-p] <path> ... :
```

Create a directory in specified location.

-p Do not fail if the directory already exists

12) du : command to check the file size

```
[root@sandbox ~]# hadoop fs -du -s /rpk/states.txt
```

```
95 /rpk/states.txt
```

13) appendToFile : Appends the contents of all given local files to the given destination file on HDFS

```
[root@sandbox ~]# cat > localfile
```

This is to illustrate appendToFile command

```
^Z
```

```
[1]+ Stopped cat > localfile
```

```
[root@sandbox ~]# hadoop fs -appendToFile localfile rpk/empty.txt
```

```
[root@sandbox ~]# hadoop fs -cat rpk/empty.txt
```

This is to illustrate appendToFile command

14) tail : shows the last 1KB of the given file

```
[root@sandbox ~]# hadoop fs -tail /rpk/states.txt
```

```
100,smith,AP
```

```
101,jones,TN
```

```
102,KIng,AP
```

```
103,ram,TN
```

```
104,sita,K
```

```
105,Lakshman,Kerala
```

```
106,aaa,Kerala
```

15) stat : Prints statistics about the file/directory

```
[root@sandbox ~]# hadoop fs -stat %b /rpk/states.txt
```

```
95
```

```
[root@sandbox ~]# hadoop fs -stat %g /rpk/states.txt
```

```
hdfs
```

```
[root@sandbox ~]# hadoop fs -stat %n /rpk/states.txt
```

```
states.txt
```

```
[root@sandbox ~]# hadoop fs -stat %o /rpk/states.txt
```

134217728

[root@sandbox ~]# hadoop fs -stat %r /rpk/states.txt

3

[root@sandbox ~]# hadoop fs -stat %u /rpk/states.txt

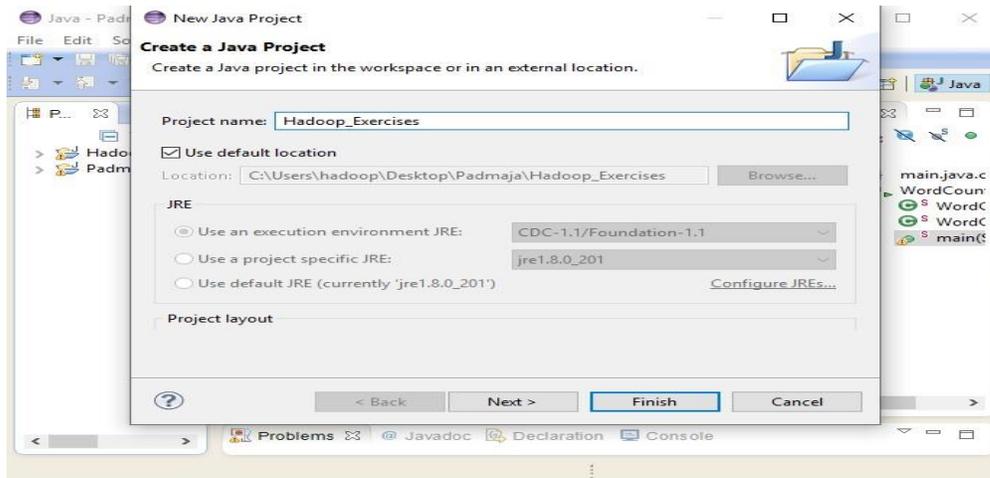
root

[root@sandbox ~]# hadoop fs -stat %y /rpk/states.txt

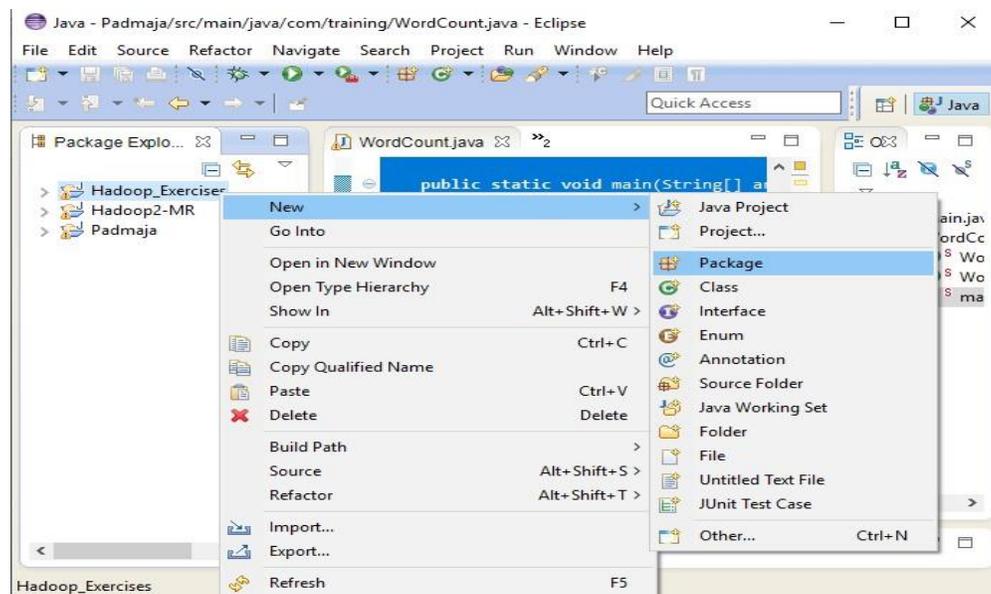
2021-01-18 09:42:33

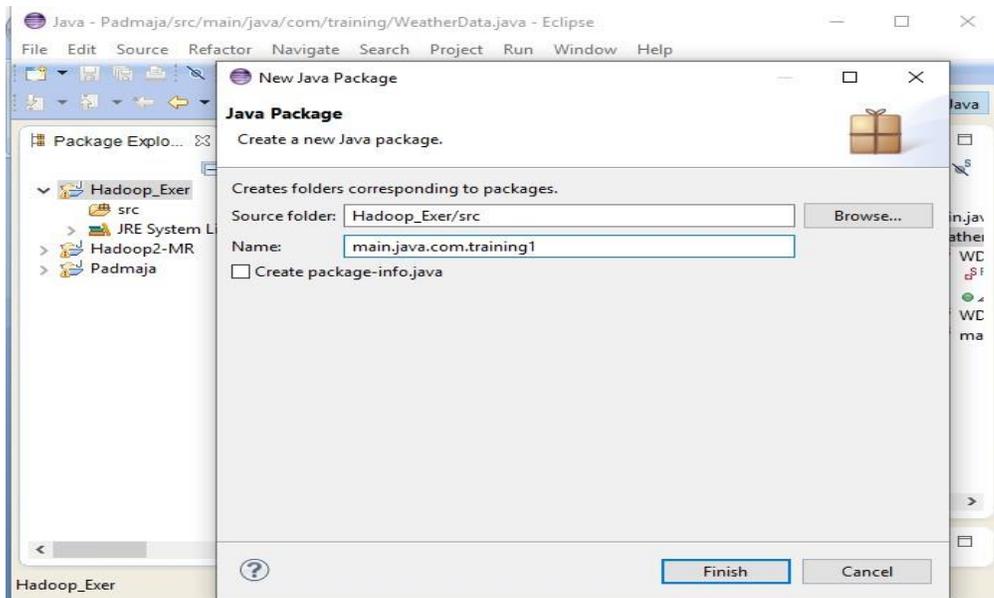
WordCount and Weather Data Analysis Program using MapReduce

Step 1: Right click on Package Explorer window, select new ->Java project to Create a Java Project called “Hadoop_Exer”

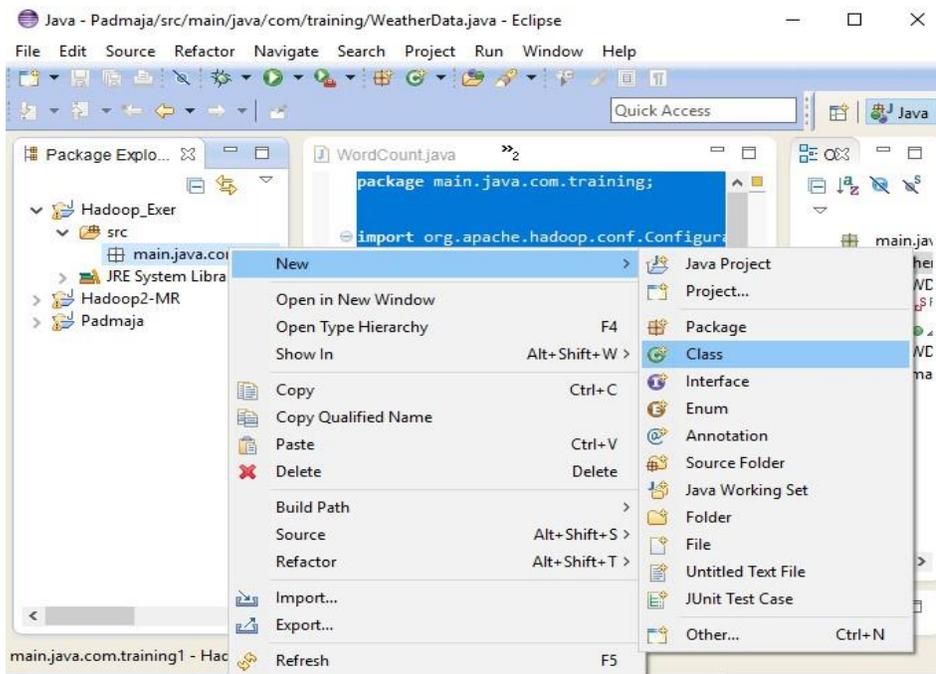


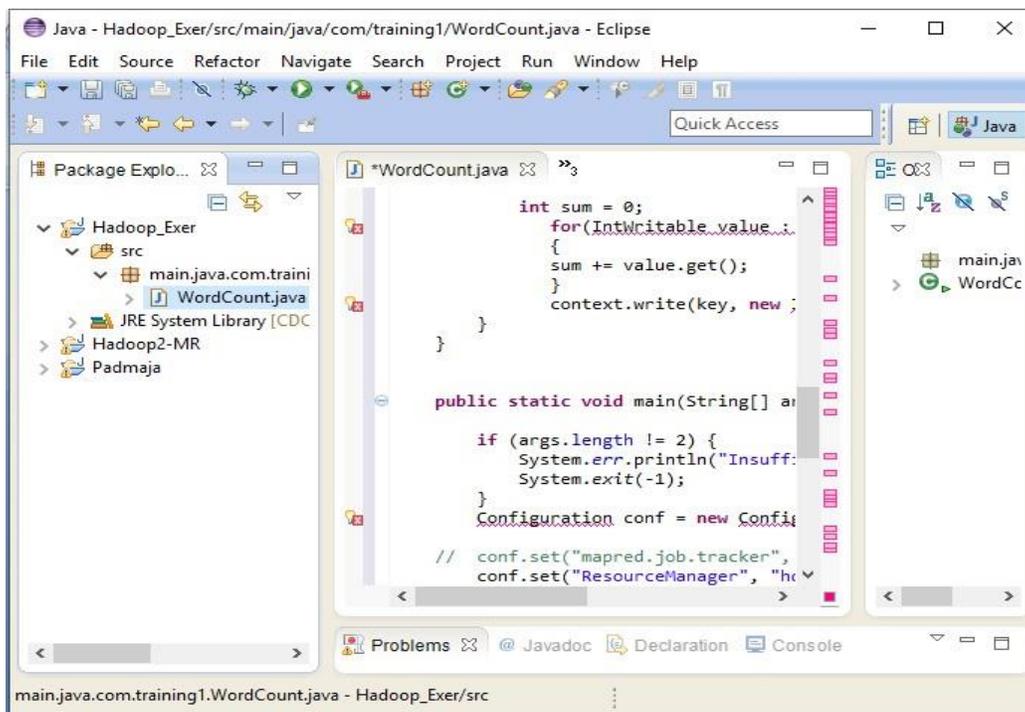
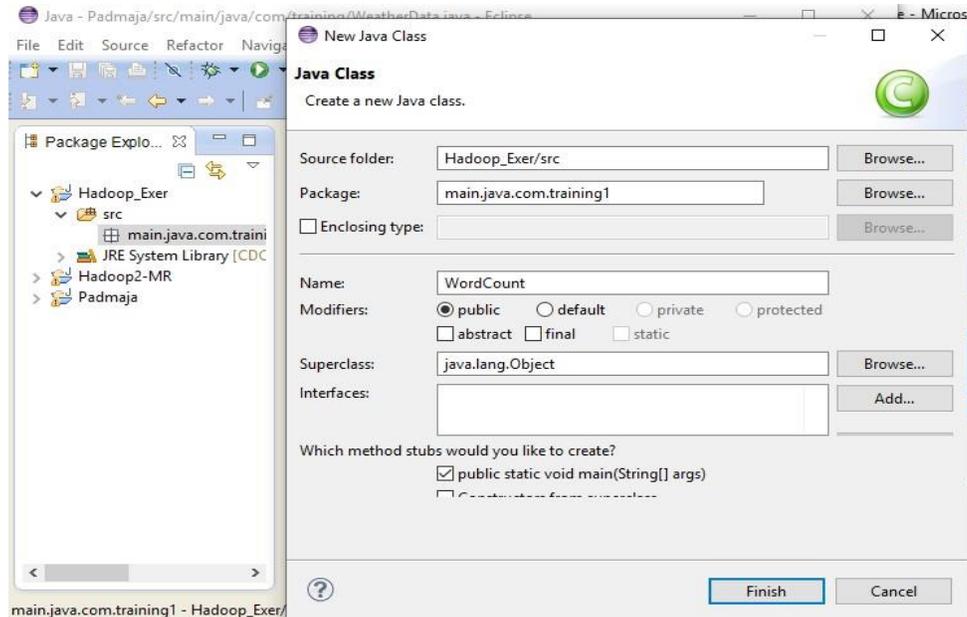
Step2 : Right click on Hadoop_Exer” Project , select new ->Package to Create a Package called “main.java.com.training “ in the Java Project “Hadoop_Exer”





Step 3: Right click on “main.java.com.training1 “ package , select new ->class to Create a Class called “WordCount” in package “main.java.com.training1” package

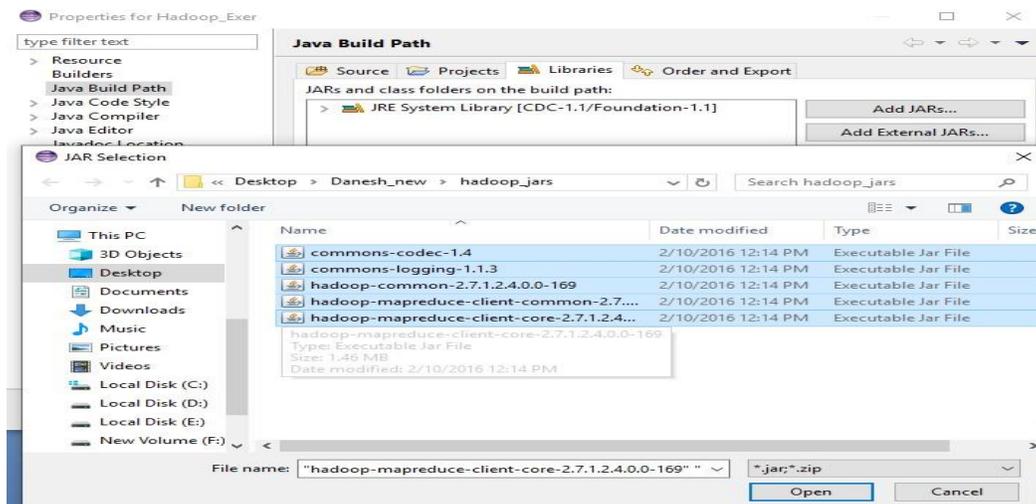
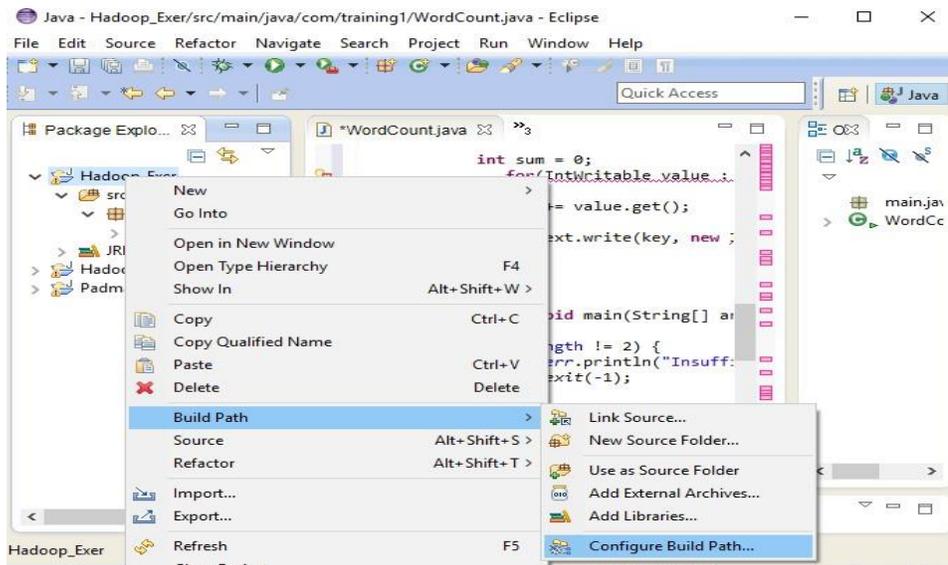




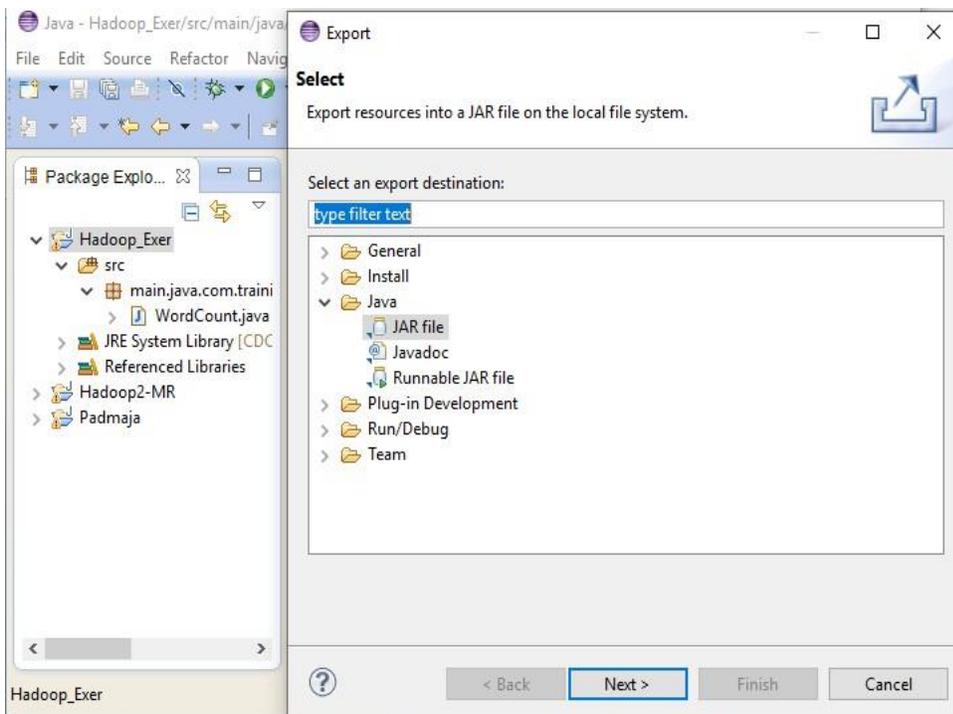
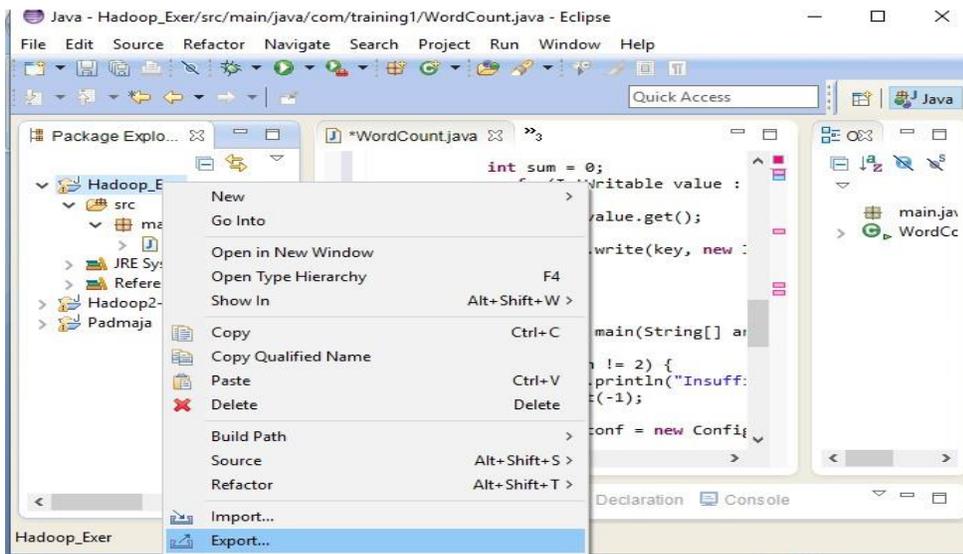
Step 4 : Right click on Hadoop_Exer” Project, select Build path ->Configure Build path,select Libraries tab, remove all Jar files and add External Jar files to execute the program

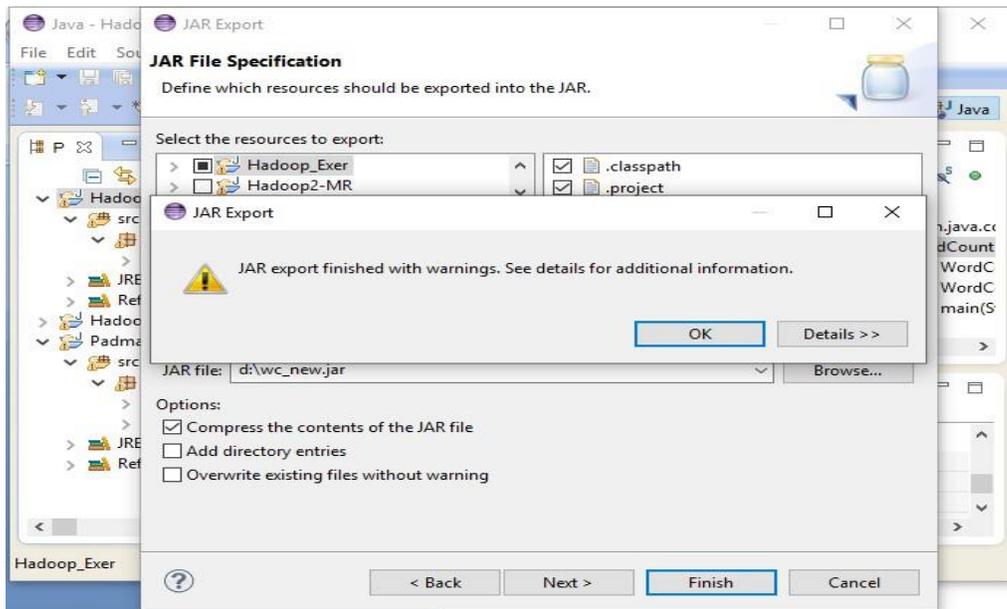
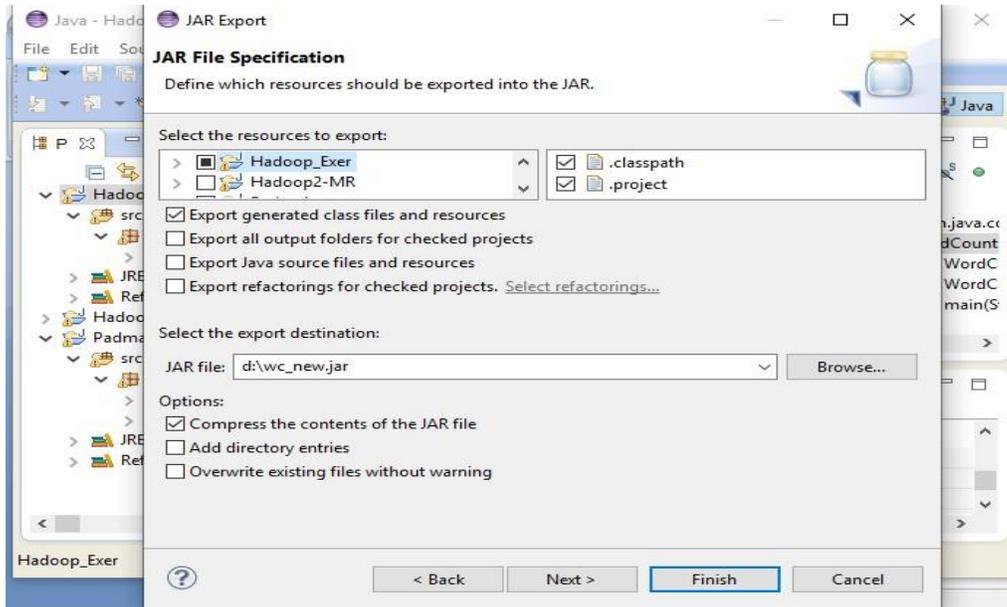
Step 4 : Right click on Hadoop_Exer” Project, select Build path ->Configure Build path,select Libraries tab, remove all Jar files and add External Jar files to execute the program

Step 4 : Right click on Hadoop_Exer” Project, select Build path ->Configure Build path,select Libraries tab, remove all Jar files and add External Jar files to execute the program

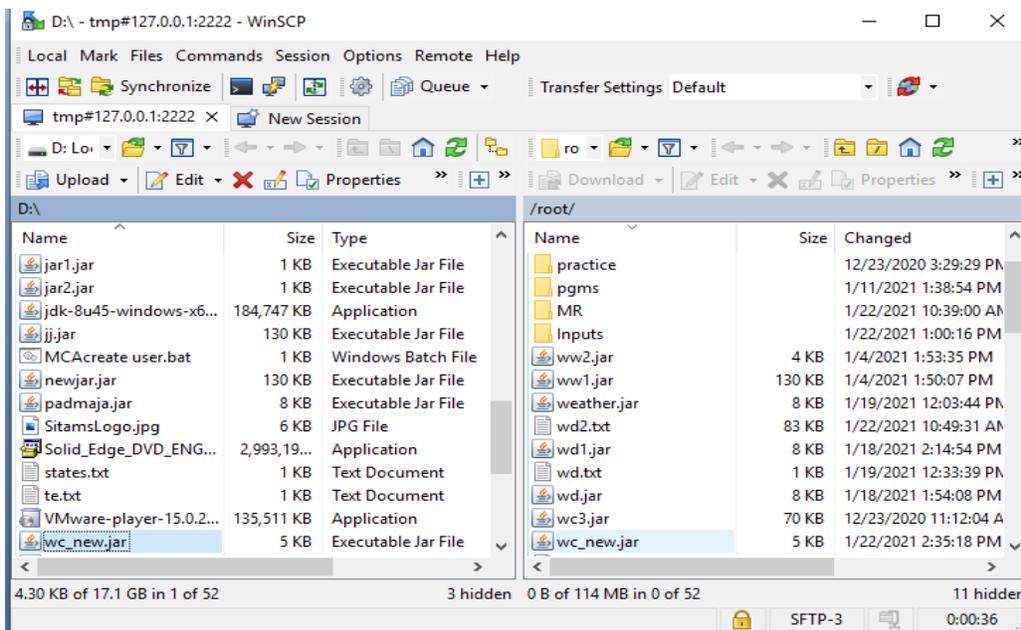
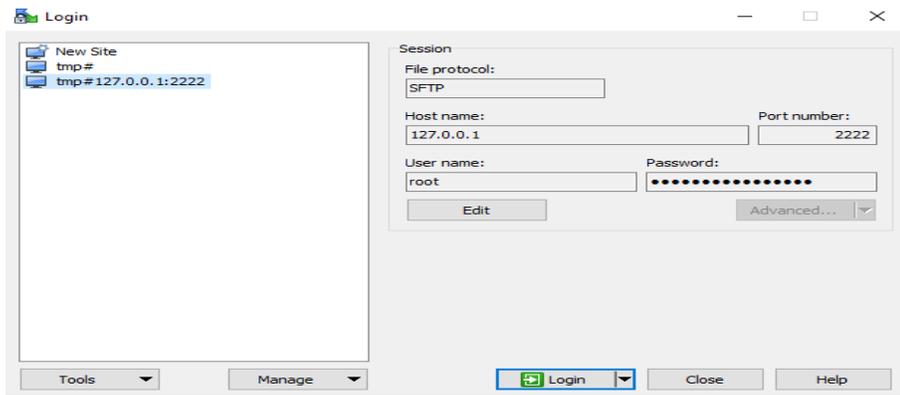


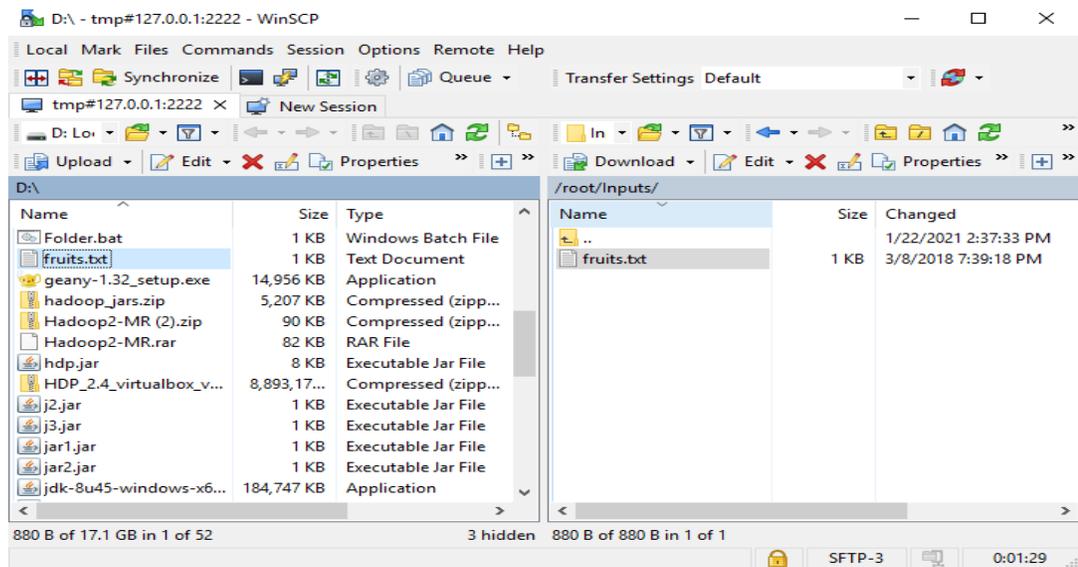
Step 5: select Export ->Java→jar , type the jar file with destination





Step 6: Open winscp , to transfer fruits.txt and wc_new.jar from windows to Unix. Drag the wc_new.jar from d:/ to unix's root directory





Step 7: create a directory called “Inputs” in unix and transfer the input file “fruits.txt” from root to Inputs directory

```
root@sandbox:~  
[root@sandbox ~]# mkdir Inputs  
[root@sandbox ~]#
```

Step 8: Create a directory called “inputFiles” in hadoop

```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -mkdir InputFiles  
[root@sandbox ~]#
```

Step 9: transfer fruits.txt which is in Inputs directory to hadoop's directory "InputFiles"

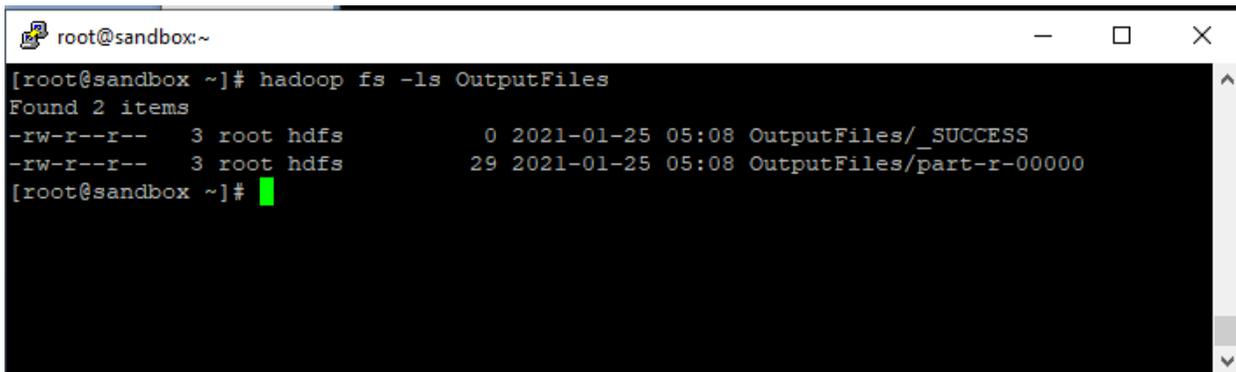
```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -put Inputs/fruits.txt InputFiles  
[root@sandbox ~]#
```

```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -ls InputFiles  
Found 1 items  
-rw-r--r--  3 root hdfs      880 2021-01-25 04:53 InputFiles/fruits.txt  
[root@sandbox ~]#
```

Step 9: type the below command to execute the code

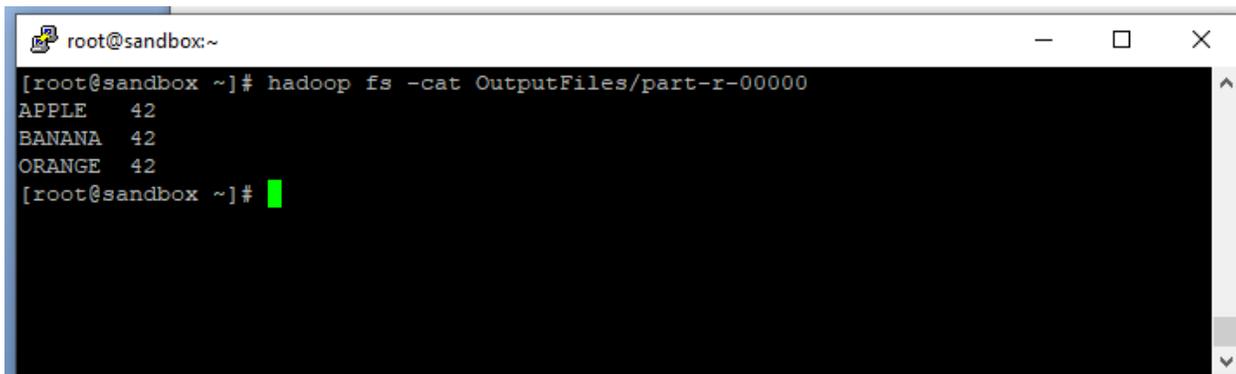
```
root@sandbox:~  
[root@sandbox ~]# hadoop jar wc_new.jar main.java.com.training1.WordCount InputFiles/fruits.txt OutputFiles  
WARNING: Use "yarn jar" to launch YARN applications.  
21/01/25 05:07:37 INFO impl.TimelineClientImpl: Timeline service address: http://sandbox.hortonworks.com:8188/ws/v1/timeline/  
21/01/25 05:07:37 INFO client.RMPProxy: Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050  
21/01/25 05:07:38 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
21/01/25 05:07:38 INFO input.FileInputFormat: Total input paths to process : 1
```

Step 10: check the output folder called "OutputFiles"



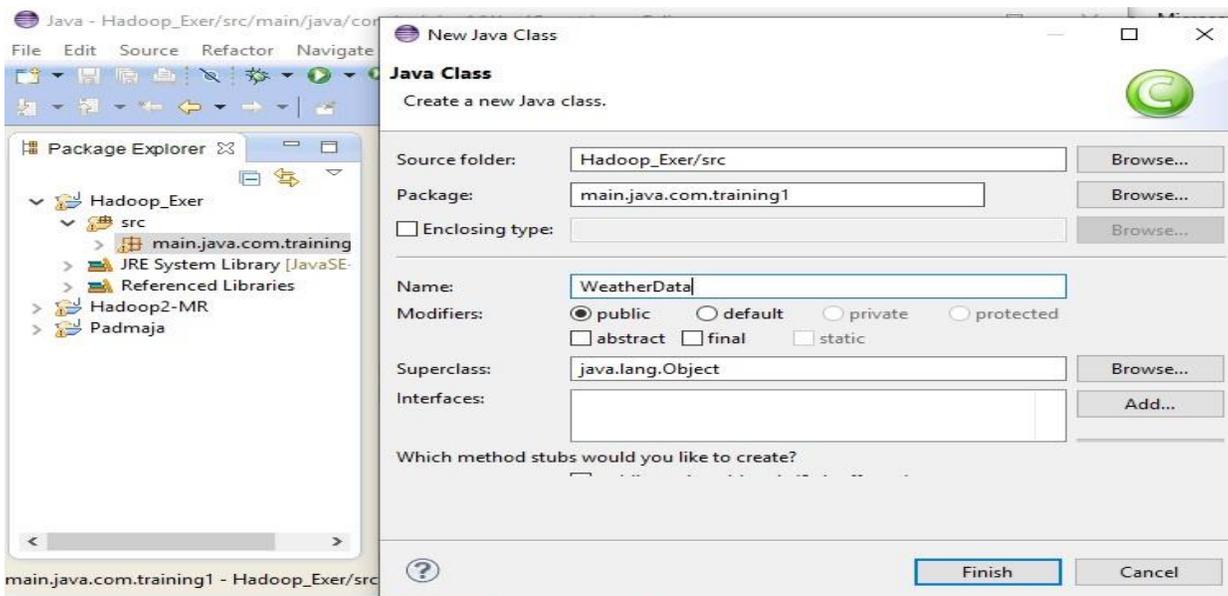
```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -ls OutputFiles  
Found 2 items  
-rw-r--r--  3 root hdfs      0 2021-01-25 05:08 OutputFiles/_SUCCESS  
-rw-r--r--  3 root hdfs     29 2021-01-25 05:08 OutputFiles/part-r-00000  
[root@sandbox ~]#
```

Step 11: Display the output

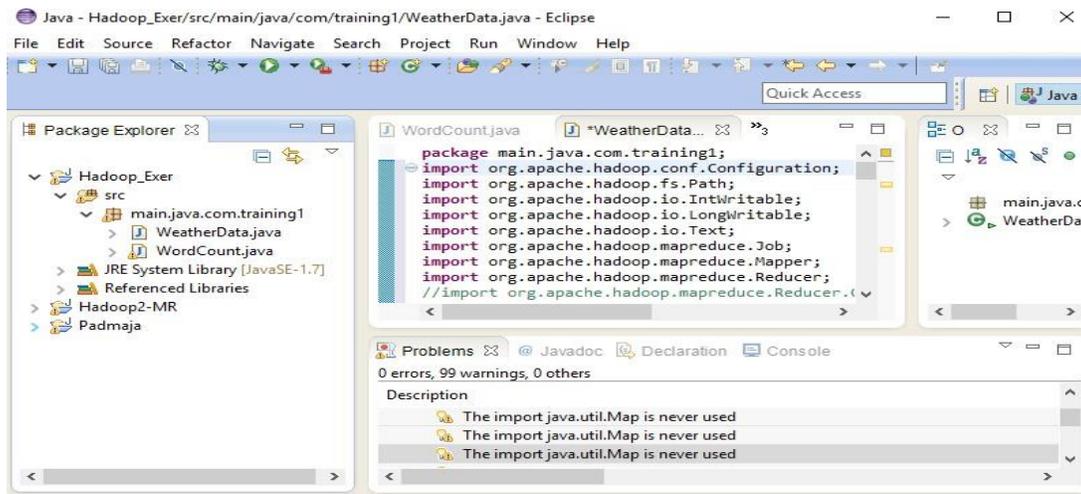


```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -cat OutputFiles/part-r-00000  
APPLE  42  
BANANA 42  
ORANGE 42  
[root@sandbox ~]#
```

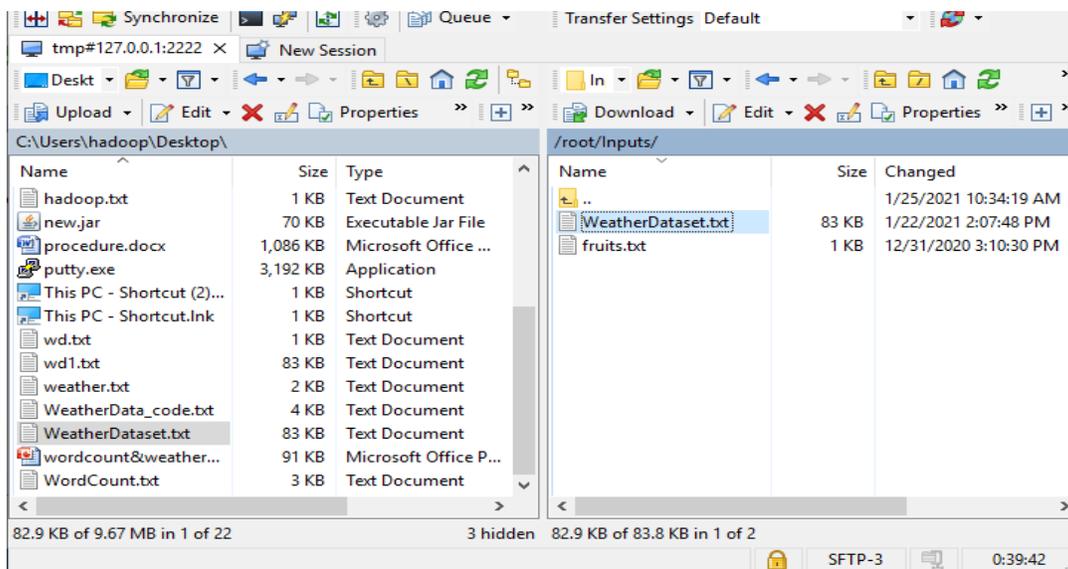
Step 11: create another java file called "WeatherData" in the same package as



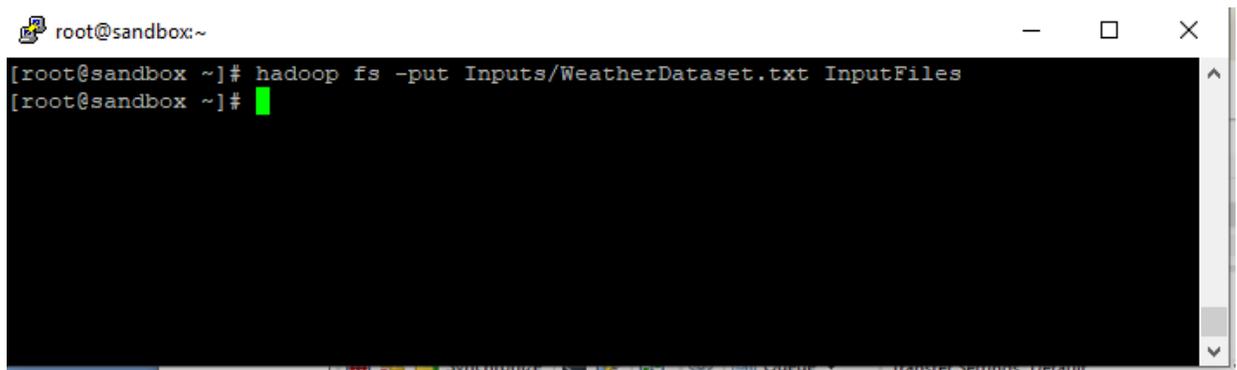
Step 12 : type the weather data analysis code



Step 13 : transfer the dataset from windows to unix folder “Inputs”



Step 14 : transfer Unix file “WeatherDataset.txt” from unix to Hadoop



Step 15 : Check the dataset existence in hadoop folder “InputFiles”

```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -ls InputFiles  
Found 2 items  
-rw-r--r--  3 root hdfs      84956 2021-01-25 05:21 InputFiles/WeatherDataset.txt  
-rw-r--r--  3 root hdfs       880 2021-01-25 04:53 InputFiles/fruits.txt  
[root@sandbox ~]#
```

Step 16 : execute the WeatherData program

```
root@sandbox:~  
[root@sandbox ~]# hadoop jar wc new.jar main.java.com.training1.WeatherData InputFiles/WeatherDataset.txt /Out1
```

Step 17 : Display the Results

```
root@sandbox:~  
[root@sandbox ~]# hadoop fs -ls /Out1  
Found 2 items  
-rw-r--r--  3 root hdfs          0 2021-01-29 09:06 /Out1/_SUCCESS  
-rw-r--r--  3 root hdfs       40 2021-01-29 09:06 /Out1/part-r-00000  
[root@sandbox ~]# hadoop fs -cat /Out1/part-r-00000  
1901  239  
1902   6  
1903  22  
1904 -17  
1905   0  
[root@sandbox ~]#
```

TO ILLUSTRATE THE PIG DATA PROCESSING OPERATORS**1) LOAD and STORE Data**

LOAD Operator
STORE Operator

LOAD Operator**2) Creating a file called sample**

```
[root@sandbox ~]# cat > sample
100 Ram MCA
101 Sita MCA
102 Lakshman MBA
103 Bharat MBA
104 Smith MCA
105 Jones MBA
```

3) Create a directory called Operators and move sample to operators

```
[root@sandbox ~]# mkdir operators
[root@sandbox ~]# mv sample operators
[root@sandbox ~]# ls operators
sample
```

4) Create a directory called hadoop_dir in hadoop

```
[root@sandbox ~]# hadoop fs -mkdir hadoop_dir
```

5) Transfer unix file called sample to hadoop directory

```
[root@sandbox ~]# hadoop fs -put operators/sample hadoop_dir
```

6) Goto Pig

```
[root@sandbox ~]# pig
grunt>
```

7) Load sample file which is in hadoop directory to Pig Relation

```
grunt> A = Load 'hadoop_dir/sample' using PigStorage('\t') as
(sno:int,sname:charArray,branch:charArray);
```

8) Display the contents of Relation A

```
grunt> dump A;
(100,Ram,MCA)
(101,Sita,MCA)
(102,LakshmanMBA,)
(103,Bharat,MBA)
```

(104,Smith,MCA)

(105,Jones,MBA)

STORE OPERATOR

- 1) Display the contents of States.txt

```
[root@sandbox ~]# cat states.txt
```

100,smith,AP

101,jones,TN

102,KIng,AP

103,ram,TN

104,sita,K

105,Lakshman,Kerala

106,aaa,Kerala

- 2) Create a Directory in Hadoop as rp1

```
[root@sandbox ~]# hadoop fs -mkdir rp1
```

- 3) Transfer the Unix File to Hadoop's Directory rp1

```
[root@sandbox ~]# hadoop fs -put states.txt rp1
```

- 4) Display the contents of Hadoop directory rp1's contents

```
[root@sandbox ~]# hadoop fs -ls rp1
```

Found 1 items

```
-rw-r--r-- 3 root hdfs      95 2021-01-28 05:16 rp1/states.txt
```

- 5) Load the file into the Pig Relation A and display the Relation content

```
grunt> A = Load 'rp1/states.txt' using PigStorage(',') as (sno:int,sname:chararray,states:chararray);
```

```
grunt> dump A;
```

(100,smith,AP)

(101,jones,TN)

(102,KIng,AP)

(103,ram,TN)

(104,sita,K)

(105,Lakshman,Kerala)

(106,aaa,Kerala)

- 5) Get only the AP employees and display the results

```
grunt> B = filter A by states == 'AP';
```

```
grunt> dump B;
```

(100,smith,AP)

(102,KIng,AP)

- 6) Store the Pig relation contents into Hadoop directory rp2

```
grunt> store B into 'rp2';
```

- 7) Check the Contents in Hadoop Directory

```
[root@sandbox ~]# hadoop fs -ls rp2
```

```
Found 2 items
```

```
-rw-r--r--  3 root hdfs      0 2021-01-28 05:20 rp2/_SUCCESS
```

```
-rw-r--r--  3 root hdfs    25 2021-01-28 05:20 rp2/part-m-0000
```

- 8) Display the file contents

```
[root@sandbox ~]# hadoop fs -cat rp2/part-m-00000
```

```
100  smith  AP
```

```
102  KIng   AP
```

2) FILTERING DATA

FILTER OPERATOR

- 1) To display only MCA Students using FILTER operator

```
grunt> B = FILTER A BY branch == 'MCA';
```

```
grunt> dump B;
```

```
(100,Ram,MCA)
```

```
(101,Sita,MCA)
```

```
(104,Smith,MCA)
```

- 2) To display only MBA Students using FILTER operator

```
grunt> C = FILTER A BY branch == 'MBA';
```

```
grunt> dump C;
```

```
(102,Lakshman,MBA)
```

```
(103,Bharat,MBA)
```

```
(105,Jones,MBA)
```

3) GROUPING AND JOINING DATA

JOIN Operator

GROUP Operator

COGROUP Operator

CROSS Operator

JOIN OPERATOR

1) Create a file called join1

```
[root@sandbox ~]# cat > join1
joe    2
hank   4
ali    0
eve    3
hank   2
```

2) Create another file called join2

```
[root@sandbox ~]# cat > join2
2      tie
4      coat
1      scarf
3      hat
```

3) Create a directory called h1 in hadoop

```
[root@sandbox ~]# hadoop fs -mkdir h1
```

3) Push both join1 and join2 to the h1 directory

```
[root@sandbox ~]# hadoop fs -put join1 h1
[root@sandbox ~]# hadoop fs -put join2 h1
```

4) Load join2 file as Pig Relation

```
grunt>A = LOAD 'h1/join2' USING PigStorage ('\t') AS (id:int,name:charArray);
grunt>DUMP A;
(2,tie)
(4,coat)
(1,scarf)
(3,hat)
```

5) Load join1 file as Pig Relation

```
grunt>B = LOAD 'h1/join1' USING PigStorage ('\t') AS (name:charArray,id:int);
grunt>DUMP B;
(joe,2)
(hank,4)
(ali,0)
(eve,3)
(hank,2)
```

6) Join the relations A and B

```
grunt>C = JOIN A BY $0, B BY $1;
```

```
grunt>DUMP C;
```

```
(2,tie,hank,2)
(2,tie,joe,2)
(3,hat,eve,3)
(4,coat,hank,4)
```

COGROUP OPERATOR:

```
grunt>D = COGROUP A BY $0, B BY $1;
```

```
grunt>DUMP D;
```

```
(0,{},{(ali,0)})
(1,{{(1,scarf)},{}})
(2,{{(2,tie)},{(hank,2),(joe,2)}})
(3,{{(3,hat)},{(eve,3)}})
(4,{{(4,coat)},{(hank,4)}})
```

GROUP OPERATOR**1) Load student1 file from hadoop into Pig Relation “Records”**

```
Records = LOAD '/pigfunctions/student1' using PigStorage('\t') as (name:charArray,
subject:charArray marks:int);
```

```
Dump Records;
```

2) To Group students Subjectwise

```
sub = GROUP Records by subject;
```

```
Dump sub;
```

Dump Records; // Displays the original records

```
(james, wp,78)
(smith, dsat,65)
(james, dsat,73)
```

```
(jones, unix,89)
```

```
(smith, unix,50)
```

```
(jones, unix,85)
```

Dump sub; // Displays subject wise marks

```
( wp, {(james, wp,78)})
```

```
( dsat, {(james, dsat,73),(smith, dsat,65)})
```

```
( unix, {(jones, unix,85),(smith, unix,50),(jones, unix,89)})
```

CROSS OPERATOR

1) Create a file called cross2

```
[root@sandbox ~]# cat>cross2
```

```
1 2
```

```
3 4
```

2) Create another file called cross1

```
[root@sandbox ~]# cat>cross1
```

```
a b c
```

```
e f g
```

3) Push both cross1 and cross2 into hadoop's directory h1

```
[root@sandbox ~]# hadoop fs -put cross h1
```

```
[root@sandbox ~]# hadoop fs -put cross1 h1
```

4) Load both cross1 and cross2 from hadoop's directory to Pig

```
grunt> G = LOAD 'h1/cross' USING PigStorage ('\t') AS (fname:int,lname:int);
```

```
DUMP G;
```

```
(1,2)
```

```
(3,4)
```

```
grunt> H = LOAD 'h1/cross1' USING PigStorage ('\t') AS (fname:chararray,lname:chararray);
```

```
DUMP H;
```

```
(a,b)
```

(e,f)

5) Apply Cross Operator

```
grunt> I = CROSS G,H;
```

```
DUMP I;
```

(3,4,e,f)

(3,4,a,b)

(1,2,e,f)

(1,2,a,b)

4) SORTING DATA

ORDER Operator

1) Create a File called “order”

```
[root@sandbox ~]# cat>order
```

```
104 ram mca
```

```
105 sita mca
```

```
102 lakshman mba
```

```
103 bharatha mba
```

2) Push the file into hadoop’s directory h1

```
[root@sandbox ~]# hadoop fs -put order h1
```

3) Load the Hadoop file into Pig

```
grunt> J = LOAD 'h1/order' USING PigStorage ('\t') AS (id:int,name:chararray,branch:chararray);
```

```
DUMP J;
```

(104,ram,mca)

(105,sita,mca)

(102,lakshman,mba)

(103,bharatha,mba)

4) Sort by id using ORDER operator

```
grunt> K = ORDER J BY $0;
```

```
DUMP K;
```

(102,lakshman,mba)

(103,bharatha,mba)

(104,ram,mca)

(105,sita,mca)

5) **COMBINING AND SPLITTING DATA**

UNION Operator for Combining the Pig Relations

SPLIT Operator for Splitting the Pig Relations

UNION OPERATOR

1) Create a file called “union”

```
[root@sandbox ~]# cat>union
```

```
2 3
```

```
1 2
```

```
2 4
```

2) Create another file called “union1”

```
[root@sandbox ~]# cat>union1
```

```
z x 8
```

```
w y 1
```

3) Push both union and union1 into hadoop’s directory “h1”

```
[root@sandbox ~]# hadoop fs -put union h1
```

```
[root@sandbox ~]# hadoop fs -put union1 h1
```

4) Load both union and union1 Hadoop’s files into Pig

```
grunt> L = LOAD 'h1/union' USING PigStorage ('\t') AS (id:int,id1:int);
```

```
DUMP L;
```

```
(2,3)
```

```
(1,2)
```

```
(2,4)
```

```
grunt> M = LOAD 'h1/union1' USING PigStorage ('\t') AS (fname:chararray,lname:chararray,id:int);
```

```
DUMP M;
```

```
(z,x,8)
```

```
(w,y,1)
```

5) Apply UNION Operator

```
grunt> N = UNION L,M;
```

```
DUMP N;
```

(z,x,8)
(w,y,1)
(2,3)
(1,2)
(2,4)

SPLIT OPERATOR:**1) Create a file called split**

```
[root@sandbox ~]# cat>split
100 smith 24 hyd
101 ivan 20 chennai
102 bayross 23 delhi
103 tornwhite 25 bay
104 leon 26 chennai
```

2) Push the file to Hadoop's Directory "h1"

```
[root@sandbox ~]# hadoop fs -put split h1
```

3) Load the file from Hadoop to Pig

```
grunt> O = LOAD 'h1/split' USING PigStorage ('\t') AS
(id:int,name:chararray,age:int,location:chararray);
```

```
DUMP O;
```

```
(100,smith,24,hyd)
(101,ivan,20,chennai)
(102,bayross,23,delhi)
(103,tornwhite,25,bay)
(104,leon,26,chennai)
```

4) Apply Split Operator

```
grunt> SPLIT O INTO Q IF age<=23,R if age>23;
```

```
DUMP Q;
```

```
(101,ivan,20,chennai)
(102,bayross,23,delhi)
```

```
DUMP R;
```

```
(100,smith,24,hyd)
(103,tornwhite,25,bay)
(104,leon,26,chennai)
```

LIMIT OPERATOR

- 1) To Display Top 2 records of Relation A

```
grunt> A = Load 'hadoop_dir/sample' using PigStorage('\t') as
```

```
(sno:int,sname:chararray,branch:chararray);
```

```
grunt> D = LIMIT A 2;
```

```
grunt> dump D;
```

```
(100,Ram,MCA)
```

```
(101,Sita,MCA)
```

Pig Code to find

- 1) **Total Marks obtained by each student**
- 2) **Average Obtained by each student and**
- 3) **Number of subjects taken by each student**

using Pig EVAL FUNCTIONS

\$ vi eval1.pig

```
Records = LOAD '/pigfunctions/student1' using PigStorage('\t')
as(name:charArray,subject:charArray,marks:int);
```

```
Dump Records;
```

```
Name = GROUP Records by name;
```

```
Dump Name;
```

```
tot = foreach Name generate group,SUM(Records.marks);
```

```
Dump tot;
```

```
per = foreach Name generate group,AVG(Records.marks);
```

```
Dump per;
```

```
count = foreach Name generate group,COUNT(Records.marks);
```

```
Dump count;
```

OUTPUT

Dump Records; // Displays the original records

(james, wp,78)

(smith, dsat,65)

(james, dsat,73)

(jones, unix,89)

(smith, unix,50)

(jones, unix,85)

Dump Name; // Displays student wise records

(james,{(james, dsat,73),(james, wp,78)})

(jones,{(jones, unix,85),(jones, unix,89)})

(smith,{(smith, unix,50),(smith, dsat,65)})

Dump tot ; (To display Total Marks obtained by each student)

(james,151)

(jones,174)

(smith,115)

Dump per ; (percentage of each student)

(james,75.5)

(jones,87.0)

(smith,57.5)

Dump count ; (Number of Subjects taken by each student)

(james,2)

(jones,2)

(smith,2)

Pig Code to find**1) Maximum Marks secured in Each subject and****2) Minimum Marks secured in Each Subject****USING Pig EVAL FUNTIONS****\$ vi eval2.pig**

```
Records = LOAD '/pigfunctions/student1' using PigStorage('\t')
as(name:charArray,subject:charArray,marks:int);
```

```
Dump Records;
```

```
sub = GROUP Records by subject;
```

```
Dump sub;
```

```
max = foreach sub generate group,MAX(Records.marks);
```

```
Dump max;
```

```
min = foreach sub generate group,MIN(Records.marks);
```

```
Dump min;
```

OUTPUT

Dump Records; // *Displays the original records*

(james, wp,78)

(smith, dsat,65)

(james, dsat,73)

(jones, unix,89)

(smith, unix,50)

(jones, unix,85)

Dump sub; // *Displays subject wise marks*

(wp, {(james, wp,78)})

(dsat, {(james, dsat,73),(smith, dsat,65)})

(unix, {(jones, unix,85),(smith, unix,50),(jones, unix,89)})

Dump min; // *Displays subject wise Minimum mark*

(wp,78)

(dsat,65)

(unix,50)

Dump max; // *Displays Subject wise Maximum markss*

(wp,78)

(dsat,73)

(unix,89)

To Illustrate the Pig String Functions

```
A = LOAD '/pigfunctions/student1' using PigStorage('\t') as(name:charArray,subject:charArray,marks:int);
```

```
Dump A;
```

```
B = foreach A generate name,LOWER(name);
```

```
Dump B;
```

```
C = foreach A generate name,UPPER(name);
```

```
Dump C;
```

```
D =foreach A generate name,SUBSTRING(name,1,3);
```

```
Dump D;
```

```
E = foreach A generate name,INDEXOF(name,'s');
```

```
Dump E;
```

```
F = foreach A generate name,ENDSWITH(name,'s'),STARTSWITH(name,'j');
```

```
Dump F;
```

OUTPUT**Dump A; // Displays Original Records**

(james, wp,78)

(smith, dsat,65)

(james, dsat,73)

(jones, unix,89)

(smith, unix,50)

(jones, unix,85)

Dump B; // Displays names in Lower Case

(james,james)

(smith,smith)

(james,james)

(jones,jones)

(smith,smith)

(jones,jones)

Dump C; // Displays Names in UpperCase

(james,JAMES)

(smith,SMITH)

(james,JAMES)

(jones,JONES)

(smith,SMITH)

(jones,JONES)

Dump D; Displays substring of name String

(james,am)

(smith,mi)

(jones,on)

Dump E; // Displays Index Of the letter 's'

(smith,0)

(james,4)

(king,-1)

(jones,4)

(rama,-1)

(sita,0)

Dump F; Displays the names that ends with 's' and the names that starts with 'j'

(smith,false,false)

(james,true,true)

(king,false,false)

(jones,true,true)

(rama,false,false)

(sita,false,false)

To Illustrate the Pig Math Functions

```
X = LOAD '/pigex/student4' using PigStorage ('\t') as (name:charArray,GPA:float);
```

```
Dump X;
```

```
Y = foreach X generate GPA,ROUND(GPA);
```

```
Dump Y;
```

```
Z = foreach X generate GPA,FLOOR(GPA);
```

```
Dump Z;
```

```
U = foreach X generate GPA,CEIL(GPA);
```

```
Dump U;
```

OUTPUT**Dump X; // Displays Original data**

(smith,7.8)

(james,8.7)

(king,6.9)

(jones,8.5)

(rama,7.7)

(sita,8.9)

Dump Y; // Displays the ROUND values of GPA

(7.8,8)

(8.7,9)

(6.9,7)

(8.5,9)

(7.7,8)

(8.9,9)

Dump Z; Displays the FLOOR values of GPA

(7.8,7.0)

(8.7,8.0)

(6.9,6.0)

(8.5,8.0)

(7.7,7.0)

(8.9,8.0)

Dump U; Displays the CEIL values of GPA

(7.8,8.0)

(8.7,9.0)

(6.9,7.0)

(8.5,9.0)

(7.7,8.0)

(8.9,9.0)

Pig Code to Analyse Weather dataset**\$ vi Weather.pig**

```
A = Load 'rp/WeatherDataset.txt' as (line:chararray);
B = foreach A generate SUBSTRING(line,15,19),SUBSTRING(line,88,92),SUBSTRING(line,92,93);
store B into 'rp/revised5';
E = load 'rp/revised5/part-m-00000' using PigStorage('\t') as (year:int,temp:int,quality:int);
filtered_records = FILTER E by temp!=9999 AND quality IN (0,1,4,5,9);
grouped_records = GROUP filtered_records by year;
max_temp = FOREACH grouped_records GENERATE group,MAX(filtered_records.temp);
dump max_temp;
```

```
$ pig -x mapreduce Weather.pig
```

OUTPUT

```
(1901,250)
(1902,72)
(1903,56)
(1904,44)
(1905,44)
```

To Illustrate Hive Tables

- 1) Create a file in Unix as “emp”

```
[root@sandbox ~]# cat emp
100,smith,Manager,50000
101,jones,Asst.Manager,40000
102,KIng,Clerk, 15000
103,ram,SE,25000
104,sita,SE,25000
105,Lakshman,Analyst,30000
```

- 2) Push the Unix file “emp” to hdfs directory “rpk”

```
[root@sandbox ~]# hadoop fs -put emp /rpk/
```

- 3) Check the contents of the file at HDFS

```
[root@sandbox ~]# hadoop fs -ls /rpk
Found 2 items
-rw-r--r--  3 root hdfs    138 2021-02-01 08:53 /rpk/emp
-rw-r--r--  3 root hdfs      0 2021-01-30 08:34 /rpk/empty.txt
[root@sandbox ~]# hadoop fs -cat /rpk/emp
100,smith,Manager,50000
101,jones,Asst.Manager,40000
102,KIng,Clerk, 15000
103,ram,SE,25000
104,sita,SE,25000
105,Lakshman,Analyst,30000
```

- 4) Create a Table in Hive

```
hive> create table employees(eno int,ename string,desg string,sal double);
OK
Time taken: 1.434 seconds
```

- 5) Check the schema of the Hive Table

```
hive> describe Employees;
OK
eno          int
ename        string
desg         string
sal          double
Time taken: 0.506 seconds, Fetched: 4 row(s)
```

- 6) Load the 'emp' file data into Employee Table

```
hive> load data inpath 'rpk/emp' into table employee;
Loading data to table default.employee
Table default.employee stats: [numFiles=1, totalSize=138]
OK
Time taken: 2.114 seconds
```

- 7) Retrieve the Records of Hive table "employee" table

```
hive> select * from employee;
OK
100 smith Manager 50000.0
101 jones Asst.Manager 40000.0
102 KIng Clerk 15000.0
103 ram SE 25000.0
104 sita SE 25000.0
105 Lakshman Analyst 30000.0
Time taken: 0.596 seconds, Fetched: 7 row(s)
```

- 8) Get employees who are working as SE

```
hive> select * from employee where desg == 'SE';
OK
103 ram SE 25000.0
104 sita SE 25000.0
Time taken: 1.639 seconds, Fetched: 2 row(s)
```

- 9) Display Employees who earn salary greater than 30000

```
hive> select * from employee where sal >30000;
OK
100 smith Manager 50000.0
101 jones Asst.Manager 40000.0
Time taken: 0.669 seconds, Fetched: 2 row(s)
```

- 10) Change the name of the Hive Table

```
hive> alter table employee rename to EMP;
OK
Time taken: 1.301 seconds
hive> DESCRIBE EMP;
OK
```

```
eno          int
ename       string
desg        string
sal         double
dept        string
```

Time taken: 0.555 seconds, Fetched: 5 row(s)

11) Change the Column name of Employee Table

```
hive> alter table employee change ename EmployeeName string;
```

OK

Time taken: 0.642 seconds

```
hive> describe employee;
```

OK

```
eno          int
employeename string
desg        string
sal         double
dept        string
```

Time taken: 0.576 seconds, Fetched: 5 row(s)

12) Add new Column to the Employee Table

```
hive> alter table employee add columns (dept string);
```

OK

Time taken: 0.502 seconds

```
hive> describe employee;
```

OK

```
eno          int
ename       string
desg        string
sal         double
dept        string
```

Time taken: 0.468 seconds, Fetched: 5 row(s)

13) Drop the hive table "employee"

```
hive> drop table employee;
```

OK

Time taken: 0.723 seconds

```
hive> describe employee;
```

FAILED: SemanticException [Error 10001]: Table not found employee

To Create Partitions in Hive

- 1) Create a Table called "AllStates" in hive

```
hive> create table AllStates(eno int, ename string, state string) row format delimited fields terminated by ',';
OK
Time taken: 1.298 seconds
```

- 2) Check the input file exists in HDFS

```
[root@sandbox ~]# hadoop fs -ls /rpk
Found 2 items
-rw-r--r--  3 root hdfs      0 2021-01-30 08:34 /rpk/empty.txt
-rw-r--r--  3 root hdfs     95 2021-01-18 09:42 /rpk/states.txt
```

- 3) Check the contents of input file "Sates.txt"

```
[root@sandbox ~]# hadoop fs -cat /rpk/states.txt
100,smith,AP
101,jones,TN
102,KIng,AP
103,ram,TN
104,sita,K
105,Lakshman,Kerala
106,aaa,Kerala
```

- 3) Load the Input file from HDFS to hive

```
hive> load data inpath '/rpk/states.txt' into table AllStates;
Loading data to table default.allstates
Table default.allstates stats: [numFiles=1, totalSize=95]
OK
Time taken: 2.613 seconds
```

- 4) Check the contents of Input File in Hive

```
hive> select * from AllStates;
OK
100  smith  AP
101  jones  TN
102  KIng  AP
103  ram    TN
104  sita   K
105  Lakshman  Kerala
```

106 aaa Kerala

Time taken: 0.726 seconds, Fetched: 7 row(s)

5) Create a Partitions Table

```
hive> create table state_part(eno int,ename string) partitioned by (state string);
```

OK

Time taken: 1.686 seconds

6) Insert data into Partition Tables

```
hive> insert overwrite table state_part partition(state) select eno,ename,state from AllStates;
```

Query ID = root_20210201083126_fbc2931c-12ea-4f6b-b561-fede08805943

Total jobs = 1

Launching Job 1 out of 1

Tez session was closed. Reopening...

Session re-established.

Status: Running (Executing on YARN cluster with App id application_1612167624046_0002)

```
-----
      VERTICES   STATUS TOTAL COMPLETED RUNNING PENDING FAILED KILLED
-----
Map 1 .....  SUCCEEDED    1     1     0     0     0     0
-----
```

VERTICES: 01/01 [=====>>] 100% ELAPSED TIME: 5.91 s

Loading data to table default.state_part partition (state=null)

Time taken for load dynamic partitions : 1769

Loading partition {state=TN}

Loading partition {state=Kerala}

Loading partition {state=AP}

Loading partition {state=K}

Time taken for adding to write entity : 2

Partition default.state_part{state=AP} stats: [numFiles=1, numRows=2, totalSize=19, rawDataSize=17]

Partition default.state_part{state=K} stats: [numFiles=1, numRows=1, totalSize=9, rawDataSize=8]

Partition default.state_part{state=Kerala} stats: [numFiles=1, numRows=2, totalSize=21, rawDataSize=19]

Partition default.state_part{state=TN} stats: [numFiles=1, numRows=2, totalSize=18, rawDataSize=16]

7) To View the Partitions

```
hive> show partitions state_part;
```

OK

state=AP

state=K

state=Kerala

state=TN

Time taken: 0.798 seconds, Fetched: 4 row(s)