DHAANISH AHMED COLLEGE OF ENGINEERING

Dhaanish Nagar, Padappai, Chennai – 601301
Approved By AICTE, New Delhi,
Affiliated to Anna University, Chennai.
www.dhaanish.in

Department of Artificial Intelligence & Data Science

Lab Manual

CCS334 – Big Data Analytics Laboratory

Year/Sem : III/V

DHAANISH AHMED COLLEGE OF ENGINEERING

Vision

To establish a world-class institution that is recognized as a "Centre of Excellence" offering education and research in engineering, technology and management with a blend of social and moral values to serve the community with a futuristic perspective.

Mission

To produce eminent engineers and managers with academic excellence in their chosen fields, which would be able to take up the challenges in the modern era and fulfill the expectations of the organization they join, with moral values and social ethics.

Department of Artificial Intelligence and Data Science

Vision

To impart quality Education, Industry Collaboration, promote Research and produce Graduate Industry-ready Engineers in the field of Artificial Intelligence and Data Science to serve the society.

Mission

- To provide a conducive learning environment for quality education in the field of Artificial Intelligence and Data Science.
- To promote industry-institute interaction and collaborative research activities.
- To empower the students with ethical values and social responsibilities in their profession.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

- Show proficiency in the knowledge of basic sciences, mathematics, Artificial Intelligence, data science and statistics to build systems that require management and analysis of large volume of data.
- Demonstrate technical skills to pursue pioneering research in the field of AI and Data Science and create disruptive and sustainable solutions for the welfare of ecosystems.
- Exhibit effective communication skills, team work and lead their profession with ethics.

Program Specific Outcome (PSO)

PSO1: Evolve AI based efficient domain specific processes for Effective decision making in several domains such as business and governance domains.

PSO2: Create, select and apply the theoretical knowledge of AI and Analytics along with practical industrial tools and techniques to manage and solve societal problems.

INDEX

Ex.No:01
Date:

Install Apache Hadoop

Aim:

To Install Apache Hadoop.
Hadoop software can be installed in three modes of
Hadoop is a Java-based programming framework that supports the processing and storage of extremely large datasets on a cluster of inexpensive machines. It was the first major open source project in the big data playing field and is sponsored by the Apache Software Foundation.
Hadoop-2.7.3 is comprised of four main layers:
□□**Hadoop Common** is the collection of utilities and libraries that support other Hadoop modules.
□□**HDFS**, which stands for Hadoop Distributed File System, is responsible for persisting data to disk.
□□**YARN**, short for Yet Another Resource Negotiator, is the "operating system" for HDFS.
□□**MapReduce** is the original processing model for Hadoop clusters. It distributes work within the cluster or map, then organizes and reduces the results from the nodes into a response to a query. Many other processing models are available for the 2.x version of Hadoop.

Hadoop clusters are relatively complex to set up, so the project includes a stand-alone mode which is suitable for learning about Hadoop, performing simple operations, and debugging

Algorithm:

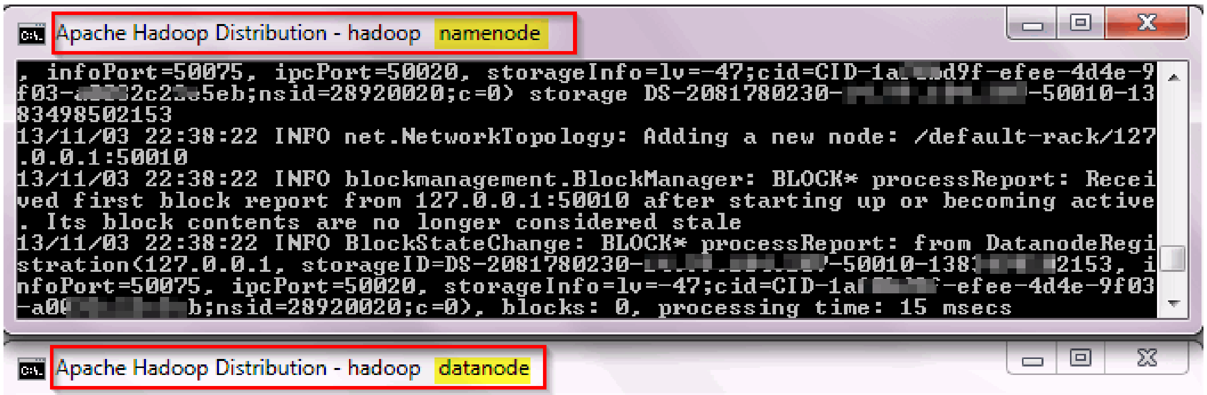
1. Install Apache Hadoop 2.2.0 in Microsoft Windows OS
If Apache Hadoop 2.2.0 is not already installed then follow the post Build, Install, Configure and Run Apache Hadoop 2.2.0 in Microsoft Windows OS.
2. Start HDFS (Namenode and Datanode) and YARN (Resource Manager and Node Manager) .

Program:

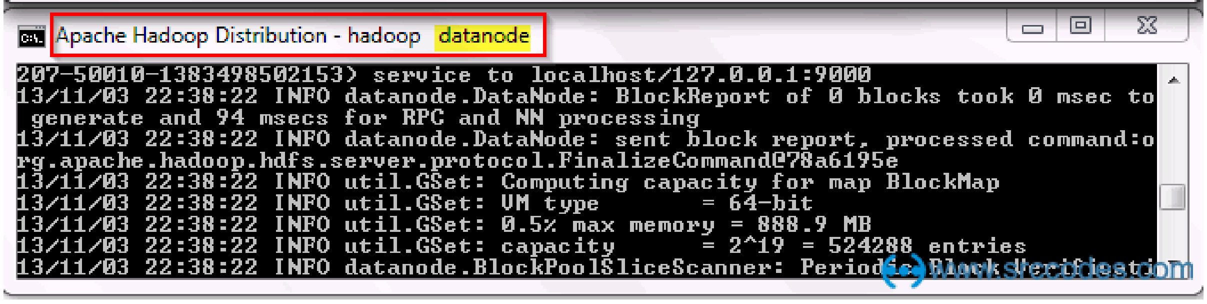
Run following commands. *Command Prompt*

```
C:\Users\abhijitg>cd c:\hadoop
c:\hadoop>sbin\start-dfs
c:\hadoop>sbin\start-yarn
starting yarn daemons
```

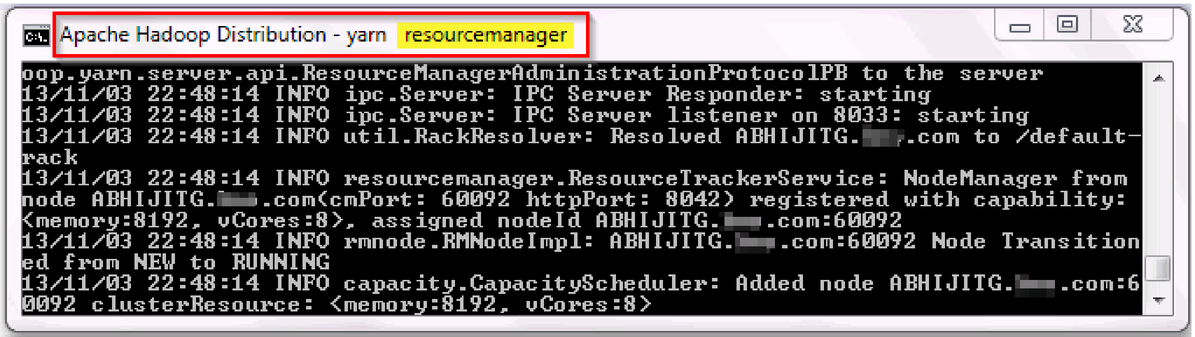
Namenode, **Datanode**, **Resource Manager** and **Node Manager** will be started in few minutes and ready to execute Hadoop **MapReduce** job in the Single Node (pseudo-distributed mode) cluster.



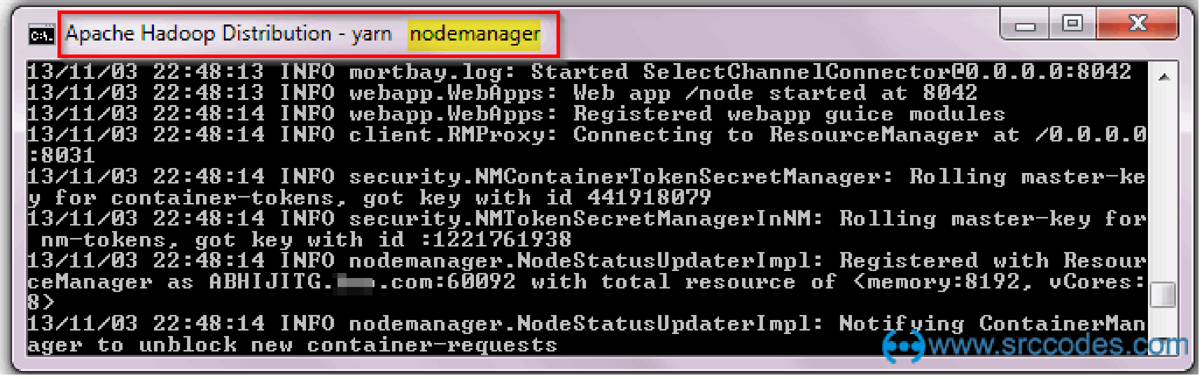
```
, infoPort=50075, ipcPort=50020, storageInfo=lv=-47;cid=CID-1a[redacted]d9f-efee-4d4e-9
f03-[redacted]2c2[redacted]5eb;nsid=28920020;c=0) storage DS-2081780230-[redacted]-50010-13
83498502153
13/11/03 22:38:22 INFO net.NetworkTopology: Adding a new node: /default-rack/127
.0.0.1:50010
13/11/03 22:38:22 INFO blockmanagement.BlockManager: BLOCK* processReport: Recei
ved first block report from 127.0.0.1:50010 after starting up or becoming active
. Its block contents are no longer considered stale
13/11/03 22:38:22 INFO BlockStateChange: BLOCK* processReport: from DatanodeRegi
stration(127.0.0.1, storageID=DS-2081780230-[redacted]-50010-138[redacted]2153, i
nfoPort=50075, ipcPort=50020, storageInfo=lv=-47;cid=CID-1a[redacted]-efee-4d4e-9f03
-a0[redacted]b;nsid=28920020;c=0), blocks: 0, processing time: 15 msecs
```



```
207-50010-1383498502153) service to localhost/127.0.0.1:9000
13/11/03 22:38:22 INFO datanode.DataNode: BlockReport of 0 blocks took 0 msec to
 generate and 94 msecs for RPC and NN processing
13/11/03 22:38:22 INFO datanode.DataNode: sent block report, processed command:o
rg.apache.hadoop.hdfs.server.protocol.FinalizeCommand@78a6195e
13/11/03 22:38:22 INFO util.GSet: Computing capacity for map BlockMap
13/11/03 22:38:22 INFO util.GSet: VM type       = 64-bit
13/11/03 22:38:22 INFO util.GSet: 0.5% max memory = 888.9 MB
13/11/03 22:38:22 INFO util.GSet: capacity      = 2^19 = 524288 entries
13/11/03 22:38:22 INFO datanode.BlockPoolSliceScanner: Period
```



```
oop.yarn.server.api.ResourceManagerAdministrationProtocolPB to the server
13/11/03 22:48:14 INFO ipc.Server: IPC Server Responder: starting
13/11/03 22:48:14 INFO ipc.Server: IPC Server listener on 8033: starting
13/11/03 22:48:14 INFO util.RackResolver: Resolved ABHIJITG.[redacted].com to /default-
rack
13/11/03 22:48:14 INFO resourcemanager.ResourceTrackerService: NodeManager from
node ABHIJITG.[redacted].com(cmPort: 60092 httpPort: 8042) registered with capability:
<memory:8192, vCores:8>, assigned nodeId ABHIJITG.[redacted].com:60092
13/11/03 22:48:14 INFO rmnode.RMNodeImpl: ABHIJITG.[redacted].com:60092 Node Transition
ed from NEW to RUNNING
13/11/03 22:48:14 INFO capacity.CapacityScheduler: Added node ABHIJITG.[redacted].com:6
0092 clusterResource: <memory:8192, vCores:8>
```



```
13/11/03 22:48:13 INFO mortbay.log: Started SelectChannelConnector@0.0.0.0:8042
13/11/03 22:48:13 INFO webapp.WebApps: Web app /node started at 8042
13/11/03 22:48:14 INFO webapp.WebApps: Registered webapp guice modules
13/11/03 22:48:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8031
13/11/03 22:48:14 INFO security.NMContainerTokenSecretManager: Rolling master-ke
y for container-tokens, got key with id 441918079
13/11/03 22:48:14 INFO security.NMTokenSecretManagerInNM: Rolling master-key for
 nm-tokens, got key with id :1221761938
13/11/03 22:48:14 INFO nodemanager.NodeStatusUpdaterImpl: Registered with Resour
ceManager as ABHIJITG.[redacted].com:60092 with total resource of <memory:8192, vCores:
8>
13/11/03 22:48:14 INFO nodemanager.NodeStatusUpdaterImpl: Notifying ContainerMan
ager to unblock new container-requests
```

Run wordcount MapReduce job

Now we'll run **wordcount** MapReduce job available in
%HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.2.0.jar
Create a text file with some content. We'll pass this file as input to the **wordcount** MapReduce job for counting words. *C:\file1.txt*

Install Hadoop

Run Hadoop Wordcount Mapreduce Example

Create a directory (say 'input') in HDFS to keep all the text files (say 'file1.txt') to be used for counting words.

C:\Users\abhijitg>cd c:\hadoop

C:\hadoop>bin\hdfs dfs -mkdir input

Copy the text file(say 'file1.txt') from local disk to the newly created 'input' directory in HDFS.

C:\hadoop>bin\hdfs dfs -copyFromLocal c:/file1.txt input

Check content of the copied file.

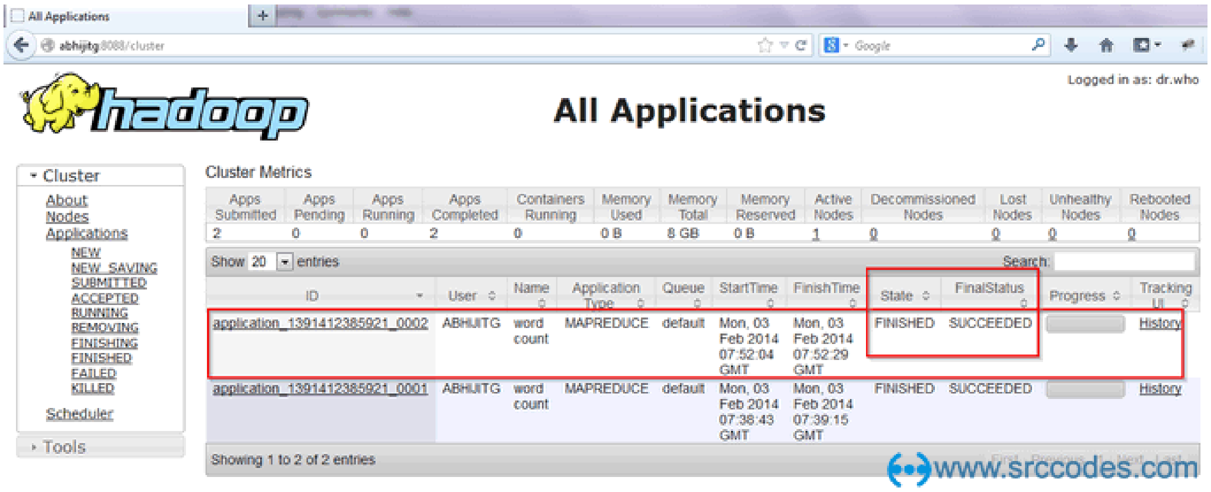
```
C:\hadoop>hdfs dfs -ls input
Found 1 items
-rw-r--r-- 1 ABHIJITG supergroup 55 2014-02-03 13:19 input/file1.txt
C:\hadoop>bin\hdfs dfs -cat input/file1.txt
Install Hadoop
Run Hadoop Wordcount Mapreduce Example
```

Run the wordcount MapReduce job provided in
%HADOOP_HOME%\share\hadoop\mapreduce\hadoop-mapreduce-examples-2.2.0.jar

```
C:\hadoop>bin\yarn jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar
wordcount input output
14/02/03 13:22:02 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
14/02/03 13:22:03 INFO input.FileInputFormat: Total input paths to process : 1
14/02/03 13:22:03 INFO mapreduce.JobSubmitter: number of splits:1
:
:
14/02/03 13:22:04 INFO mapreduce.JobSubmitter: Submitting tokens for job:
job_1391412385921_0002
14/02/03 13:22:04 INFO impl.YarnClientImpl: Submitted application
application_1391412385921_0002 to ResourceManager at /0.0.0.0:8032
14/02/03 13:22:04 INFO mapreduce.Job: The url to track the job:
http://ABHIJITG:8088/proxy/application_1391412385921_0002/
14/02/03 13:22:04 INFO mapreduce.Job: Running job: job_1391412385921_0002
14/02/03 13:22:14 INFO mapreduce.Job: Job job_1391412385921_0002 running in uber mode :
false
14/02/03 13:22:14 INFO mapreduce.Job: map 0% reduce 0%
14/02/03 13:22:22 INFO mapreduce.Job: map 100% reduce 0%
14/02/03 13:22:30 INFO mapreduce.Job: map 100% reduce 100%
14/02/03 13:22:30 INFO mapreduce.Job: Job job_1391412385921_0002 completed successfully
14/02/03 13:22:31 INFO mapreduce.Job: Counters: 43
File System Counters
FILE: Number of bytes read=89
FILE: Number of bytes written=160142
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0

HDFS: Number of bytes read=171
HDFS: Number of bytes written=59
HDFS: Number of read operations=6
HDFS: Number of large read operations=0
HDFS: Number of write operations=2
Job Counters
Launched map tasks=1
Launched reduce tasks=1
```

```
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=5657
Total time spent by all reduces in occupied slots (ms)=6128
Map-Reduce Framework
Map input records=2
Map output records=7
Map output bytes=82
Map output materialized bytes=89
Input split bytes=116
Combine input records=7
Combine output records=6
Reduce input groups=6
Reduce shuffle bytes=89
Reduce input records=6
Reduce output records=6
Spilled Records=12
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=145
CPU time spent (ms)=1418
Physical memory (bytes) snapshot=368246784
Virtual memory (bytes) snapshot=513716224
Total committed heap usage (bytes)=307757056
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=55
File Output Format Counters

Bytes Written=59
```

Output:



All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2	0	0	2	0	0 B	8 GB	0 B	1	0	0	0	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1391412385921_0002	ABHIJITG	word count	MAPREDUCE	default	Mon, 03 Feb 2014 07:52:04 GMT	Mon, 03 Feb 2014 07:52:29 GMT	FINISHED	SUCCEEDED		History
application_1391412385921_0001	ABHIJITG	word count	MAPREDUCE	default	Mon, 03 Feb 2014 07:38:43 GMT	Mon, 03 Feb 2014 07:39:15 GMT	FINISHED	SUCCEEDED		History

Showing 1 to 2 of 2 entries

Result:

We've installed Hadoop in stand-alone mode and verified it by running an example program it provided.

EXP.NO:02

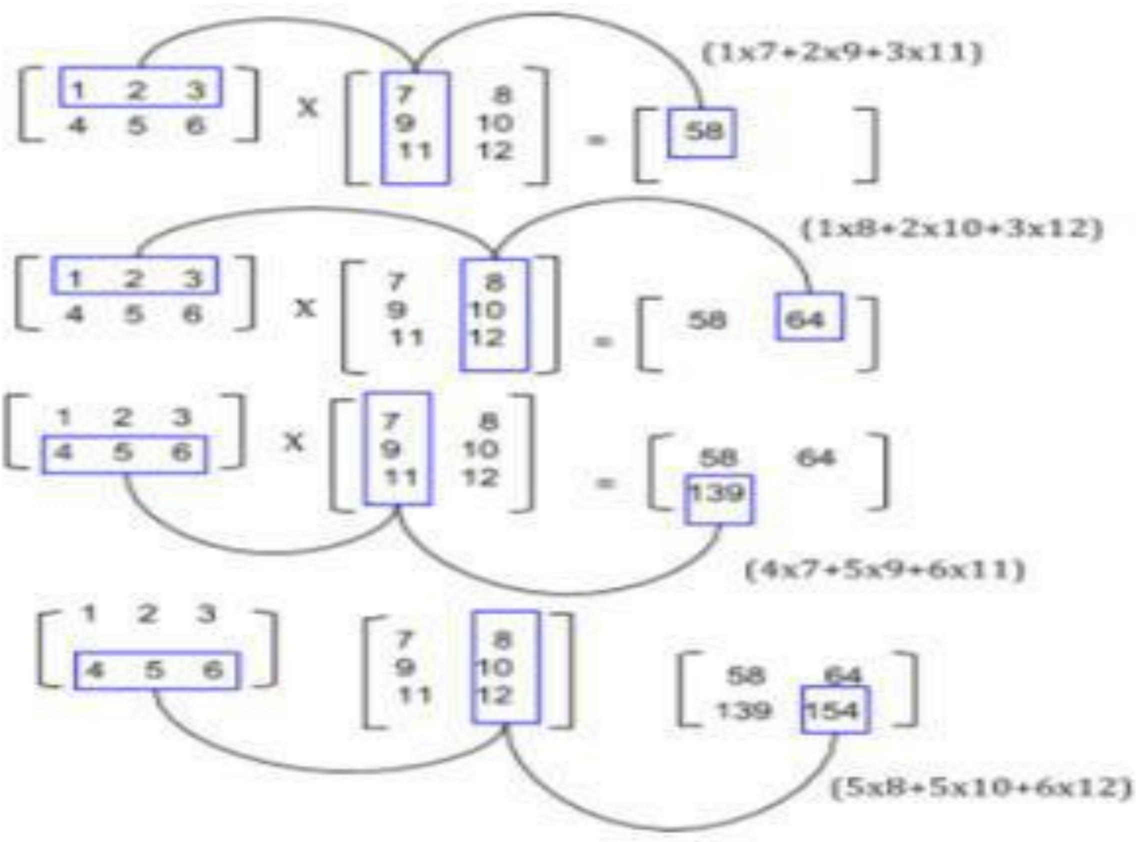
DATE:

Implement of Matrix Multiplication with Hadoop Map Reduce

AIM:

To Develop a MapReduce program to implement Matrix Multiplication.

In **mathematics**, **matrix multiplication** or the **matrix product** is a binary operation that produces a matrix from two matrices. The definition is motivated by linear equations and linear transformations on vectors, which have numerous applications in applied mathematics, physics, and engineering. In more detail, if **A** is an *n* × *m* matrix and **B** is an *m* × *p* matrix, their matrix product **AB** is an *n* × *p* matrix, in which the m entries across a row of **A** are multiplied with the m entries down a column of **B** and summed to produce an entry of **AB**. When two linear transformations are represented by matrices, then the matrix product represents the composition of the two transformations.



Algorithm for Map Function:

a. for each element mij of M do produce (key,value) pairs as ((i,k), (M,j,mij), for k=1,2,3,.. upto the number of columns of N
b. for each element njk of N do produce (key,value) pairs as ((i,k),(N,j,Njk), for i = 1,2,3,.. Upto the number of rows of M.
c. return Set of (key,value) pairs that each key (i,k), has list with values (M,j,mij) and (N, j,njk) for all possible values of j.

Algorithm for Reduce Function:

d. for each key (i,k) do
e. sort values begin with M by j in listM sort values begin with N by j in listN multiply mij and njk for jth value of each list
f. sum up mij x njk return (i,k), Σj=1 mij x njk

Step 1. Download the hadoop jar files with these links.

Download Hadoop Common Jar files: https://goo.gl/G4MyHp

```
$ wget https://goo.gl/G4MyHp -O hadoop-common-2.2.0.jar
```

Download Hadoop Mapreduce Jar File: https://goo.gl/KT8yfB

```
$ wget https://goo.gl/KT8yfB -O hadoop-mapreduce-client-core-2.7.1.jar
```

Step 2. Creating Mapper file for Matrix Multiplication.

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.ReflectionUtils;
class Element implements Writable {
int tag;
int index;
double value;
Element() {
tag = 0;
```

```
index = 0;
value = 0.0;
}
Element(int tag, int index, double value) {
this.tag = tag;
this.index = index;
this.value = value;
}
@Override
public void readFields(DataInput input) throws IOException {
tag = input.readInt();
index = input.readInt();
value = input.readDouble();
}
@Override
public void write(DataOutput output) throws IOException {
output.writeInt(tag);
output.writeInt(index);
output.writeDouble(value);
}
}
class Pair implements WritableComparable<Pair> {
int i;
int j;
Pair() {
i = 0;

j = 0;
}
Pair(int i, int j) {
this.i = i;
this.j = j;
}
@Override
public void readFields(DataInput input) throws IOException {
i = input.readInt();
j = input.readInt();
}
@Override
public void write(DataOutput output) throws IOException {
output.writeInt(i);
output.writeInt(j);
}
@Override
public int compareTo(Pair compare) {
if (i > compare.i) {
```

```
return 1;
} else if ( i < compare.i) {
return -1;
} else {
if(j > compare.j) {
return 1;
} else if (j < compare.j) {
return -1;
}
}
return 0;
}
public String toString() {
return i + " " + j + " ";
}
}
public class Multiply
{
public static class MatriceMapperM extends Mapper<Object,Text,IntWritable,Element>
{ 24 Department of CSE
@Override
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
String readLine = value.toString();
String[] stringTokens = readLine.split(",");
int index = Integer.parseInt(stringTokens[0]);
double elementValue = Double.parseDouble(stringTokens[2]);
Element e = new Element(0, index, elementValue);
IntWritable keyValue = new IntWritable(Integer.parseInt(stringTokens[1]));
context.write(keyValue, e);
}
}
public static class MatriceMapperN extends Mapper<Object,Text,IntWritable,Element> {
@Override
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
String readLine = value.toString();
String[] stringTokens = readLine.split(",");
int index = Integer.parseInt(stringTokens[1]);
double elementValue = Double.parseDouble(stringTokens[2]);
Element e = new Element(1,index, elementValue);
IntWritable keyValue = new IntWritable(Integer.parseInt(stringTokens[0]));
context.write(keyValue, e);
}
}
```

```
public static class ReducerMxN extends Reducer<IntWritable,Element, Pair, DoubleW
@Override
public void reduce(IntWritable key, Iterable<Element> values, Context context) throws
IOException, InterruptedException {
ArrayList<Element> M = new ArrayList<Element>();
ArrayList<Element> N = new ArrayList<Element>();
Configuration conf = context.getConfiguration();
for(Element element : values) {
Element tempElement = ReflectionUtils.newInstance(Element.class, conf);
ReflectionUtils.copy(conf, element, tempElement);
if (tempElement.tag == 0) {
M.add(tempElement);
} else if(tempElement.tag == 1) {
N.add(tempElement);
}
}
for(int i=0;i<M.size();i++) {
for(int j=0;j<N.size();j++) {
Pair p = new Pair(M.get(i).index,N.get(j).index);
double multiplyOutput = M.get(i).value * N.get(j).value;
context.write(p, new DoubleWritable(multiplyOutput));
}
}
}
}
public static class MapMxN extends Mapper<Object, Text, Pair, DoubleWritable> {
@Override
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
String readLine = value.toString();
String[] pairValue = readLine.split(" ");
Pair p = new Pair(Integer.parseInt(pairValue[0]),Integer.parseInt(pairValue[1]));
DoubleWritable val = new DoubleWritable(Double.parseDouble(pairValue[2]));
context.write(p, val);
}
}
public static class ReduceMxN extends Reducer<Pair, DoubleWritable, Pair,
DoubleWritable> {
@Override
public void reduce(Pair key, Iterable<DoubleWritable> values, Context context) throws
IOException, InterruptedException {
double sum = 0.0;
for(DoubleWritable value : values) { sum += value.get();
}
context.write(key, new DoubleWritable(sum));
}
```

```
}
public static void main(String[] args) throws Exception {
Job job = Job.getInstance();
job.setJobName("MapIntermediate");
job.setJarByClass(Project1.class);
MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class,
MatriceMapperM.class);
MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class,
MatriceMapperN.class);
job.setReducerClass(ReducerMxN.class);
job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(Element.class);
job.setOutputKeyClass(Pair.class);
job.setOutputValueClass(DoubleWritable.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileOutputFormat.setOutputPath(job, new Path(args[2]));
job.waitForCompletion(true);
Job job2 = Job.getInstance();
job2.setJobName("MapFinalOutput");
job2.setJarByClass(Project1.class);
job2.setMapperClass(MapMxN.class);
job2.setReducerClass(ReduceMxN.class);
job2.setMapOutputKeyClass(Pair.class);
job2.setMapOutputValueClass(DoubleWritable.class);
job2.setOutputKeyClass(Pair.class);
job2.setOutputValueClass(DoubleWritable.class);
job2.setInputFormatClass(TextInputFormat.class);
job2.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.setInputPaths(job2, new Path(args[2]));
FileOutputFormat.setOutputPath(job2, new Path(args[3])); job2.waitForCompletion(true);
}
}
```

Step 5. Compiling the program in particular folder named as operation

```
#!/bin/bash
rm -rf multiply.jar classes
module load hadoop/2.6.0
mkdir -p classes
javac -d classes -cp classes:`$HADOOP_HOME/bin/hadoop classpath` Multiply.java
jar cf multiply.jar -C classes .
echo "end"
```

Step 6. Running the program in particular folder named as operation

```
export HADOOP_CONF_DIR=/home/$USER/cometcluster
module load hadoop/2.6.0
myhadoop-configure.sh
start-dfs.sh
```

```
start-yarn.sh
hdfs dfs -mkdir -p /user/$USER
hdfs dfs -put M-matrix-large.txt /user/$USER/M-matrix-large.txt
hdfs dfs -put N-matrix-large.txt /user/$USER/N-matrix-large.txt
hadoop jar multiply.jar edu.uta.cse6331.Multiply /user/$USER/M-matrix-larg
/user/$USER/N-matrix-large.txt /user/$USER/intermediate /user/$USER/out
rm -rf output-distr
mkdir output-distr
hdfs dfs -get /user/$USER/output/part* output-distr
stop-yarn.sh
stop-dfs.sh
myhadoop-cleanup.sh
```

Output:

```
module load hadoop/2.6.0
rm -rf output intermediate
hadoop --config $HOME jar multiply.jar edu.uta.cse6331.Multiply M-matrix-small.txt N-matrix-small.txt intermediate output.
```

Result:

EXP.NO:03

DATE:

MapReduce program to find the grades of student’s

AIM:

To Develop a MapReduce program to find the grades of student’s.

PROGRAME:

```
import java.util.Scanner;
public class JavaExample
{
public static void main(String args[])
{
int marks[] = new int[6];
int i;
float total=0, avg;
Scanner scanner = new Scanner(System.in);
for(i=0; i<6; i++) {
System.out.print("Enter Marks of Subject"+(i+1)+":");
marks[i] = scanner.nextInt();
total = total + marks[i];
}
scanner.close();
//Calculating average here avg = total/6;
System.out.print("The student Grade is: ");
if(avg>=80)
{
System.out.print("A");
}
else if(avg>=60 && avg<80)
{
System.out.print("B");
}
else if(avg>=40 && avg<60)
{
System.out.print("C");
}
else
{
System.out.print("D");
}
}
}
```

OUTPUT:

```
Enter Marks of Subject1:40
Enter Marks of Subject2:80
Enter Marks of Subject3:80
Enter Marks of Subject4:40
Enter Marks of Subject5:60
Enter Marks of Subject6:60
The student Grade is: B
```

Result:

EXP.NO:04

DATE:

MapReduce program to calculate the frequency

AIM:

To Develop a MapReduce program to calculate the frequency of a given word in agiven file

Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).

Example – (Map function in Word Count)

Input

Set of data

Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN

Output

Convert into another set of data

(Key,Value)

(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.

Example – (Reduce function in Word Count)

Input Set of Tuples

(output of Map function)

(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1),

(buS,1),(caR,1),(CAR,1), (car,1), (BUS,1), (TRAIN,1)

Output Converts into smaller set of tuples

(BUS,7), (CAR,7), (TRAIN,4)

Work Flow of Program

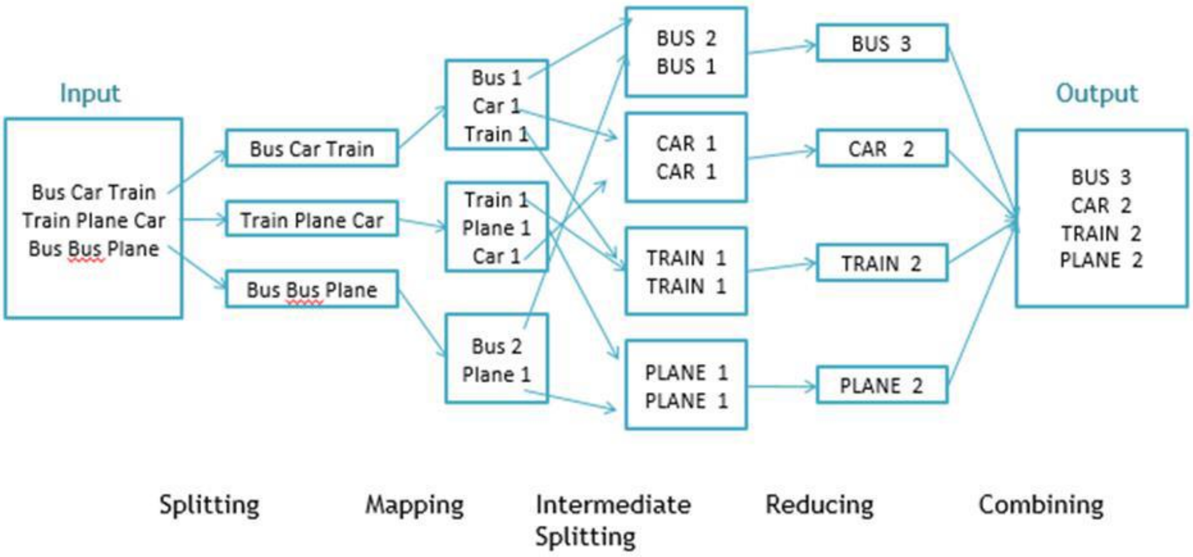


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps

1. Splitting – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line (‘\n’).
2. **Mapping –** as explained above
3. Intermediate splitting – the entire process in parallel on different clusters. In order to group them in “Reduce Phase” the similar KEY data should be on same cluster.
4. **Reduce –** it is nothing but mostly group by phase
5. **Combining –** The last phase where all the data (individual result set from each cluster) is combine together to form a Result

Now Let’s See the Word Count Program in Java
Make sure that Hadoop is installed on your system with java idk
Steps to follow
Step 1. Open Eclipse> File > New > Java Project > (Name it – MRProgramsDemo) > Finish
Step 2. Right Click > New > Package (Name it - PackageDemo) > Finish
Step 3. Right Click on Package > New > Class (Name it - WordCount)
Stp 4. Add Following Reference Libraries

Right Click on Project > Build Path> Add External Archivals

□ /usr/lib/hadoop-0.20/hadoop-core.jar

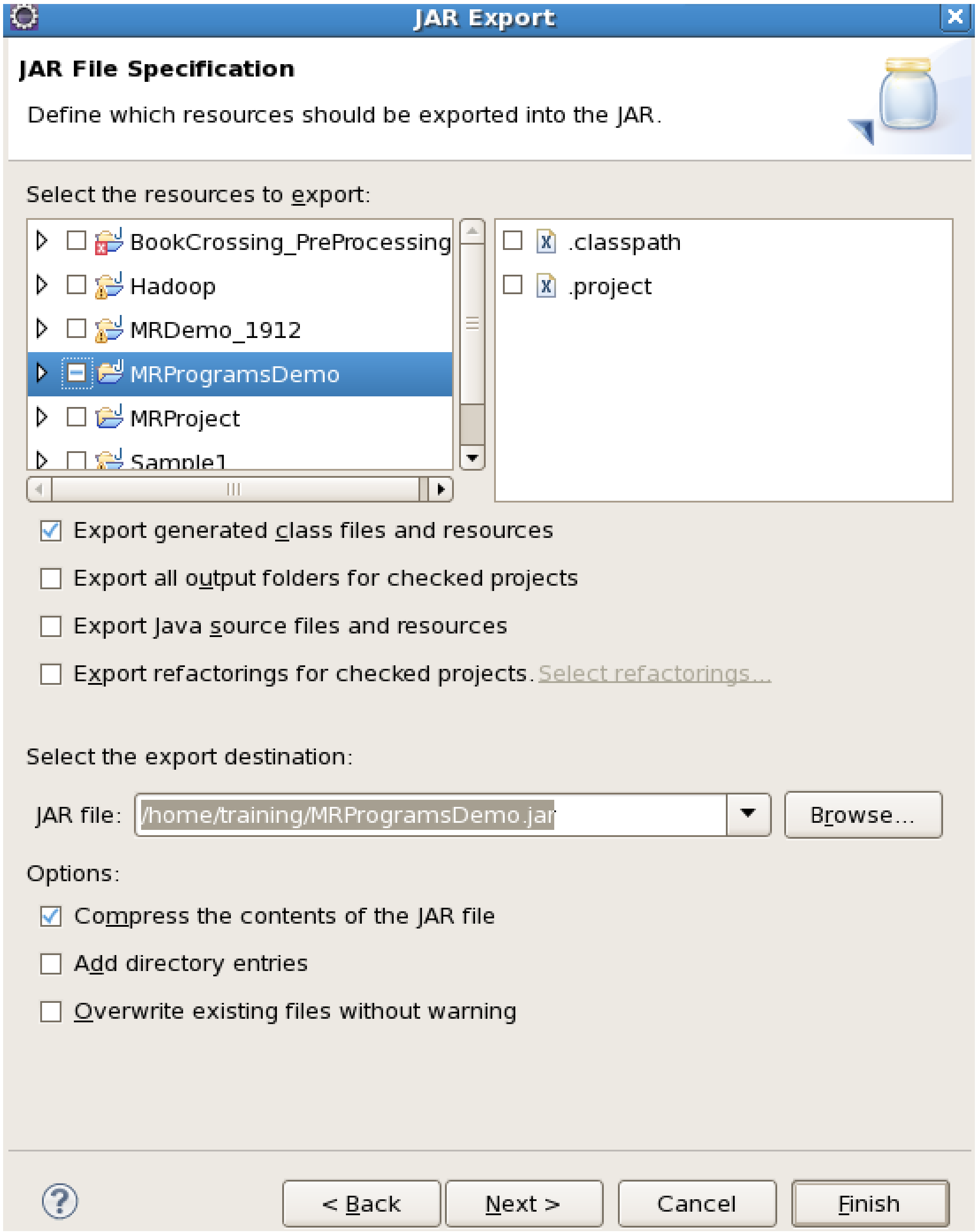
□ Usr/lib/hadoop-0.20/lib/Commons-cli-1.2.jar

Program: Step 5. Type following Program :

```
package PackageDemo;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text, Text,
IntWritable>{
public void map(LongWritable key, Text value, Context con) throws IOException,
InterruptedException
{
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
Text outputKey = new Text(word.toUpperCase().trim());
IntWritable outputValue = new IntWritable(1);
con.write(outputKey, outputValue);
}
}
}
public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text,
IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context con) throws
IOException,
InterruptedException
{
int sum = 0;
for(IntWritable value : values)
{
sum += value.get();
}
con.write(word, new IntWritable(sum));
}
}
}
```

Make Jar File

Right Click on Project> Export> Select export destination as Jar File > next> Finish



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- □ BookCrossing_PreProcessing
- □ Hadoop
- □ MRDemo_1912
- ⊟ MRProgramsDemo
- □ MRProject
- □ Sample1

- □ .classpath
- □ .project

☑ Export generated class files and resources

□ Export all output folders for checked projects

□ Export Java source files and resources

□ Export refactorings for checked projects. Select refactorings...

Select the export destination:

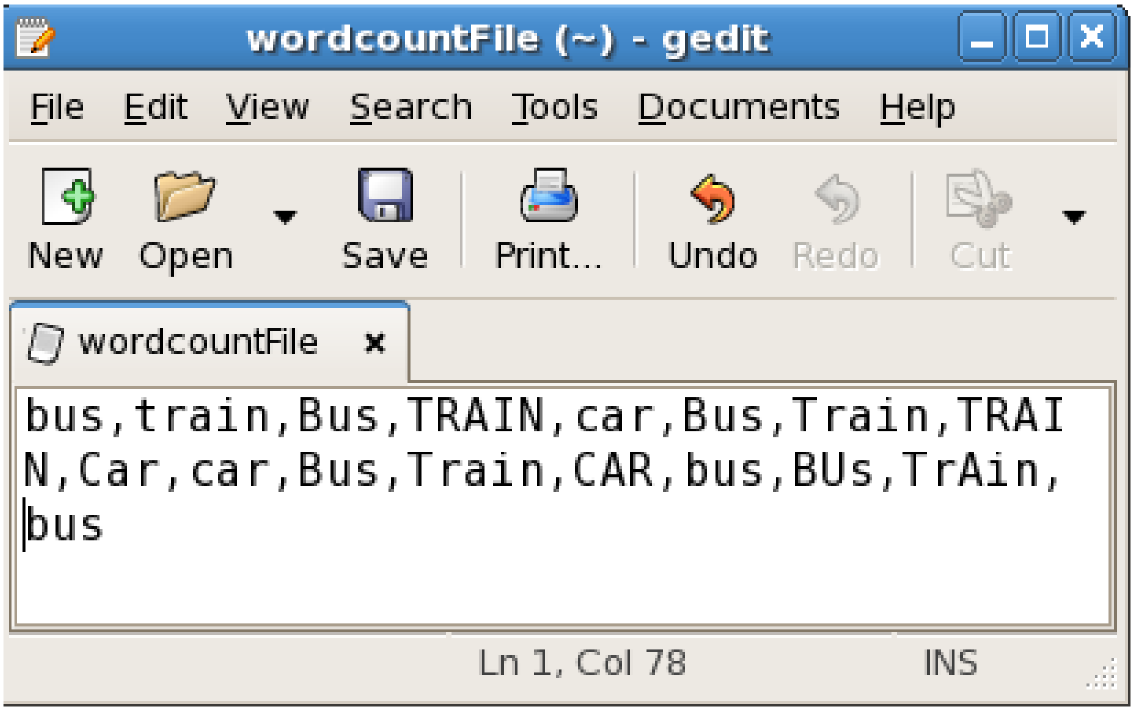
JAR file: /home/training/MRProgramsDemo.jar

Options:

☑ Compress the contents of the JAR file

□ Add directory entries

□ Overwrite existing files without warning



```
bus,train,Bus,TRAIN,car,Bus,Train,TRAIN,Car,car,Bus,Train,CAR,bus,BUs,TrAin,bus
```

To Move this into Hadoop directly, open the terminal and enter the following commands:

```
[training@localhost ~]$ hadoop fs -put wordcountFile wordCountFile
```

Run Jar file

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)

[training@localhost ~]$ Hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile MRDir1

Output:

```
[training@localhost ~]$ hadoop fs -ls MRDir1
Found 3 items
-rw-r--r-- 1 training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_SUCCESS
drwxr-xr-x - training supergroup
0 2016-02-23 03:36 /user/training/MRDir1/_logs
-rw-r--r-- 1 training supergroup
20 2016-02-23 03:36 /user/training/MRDir1/part-r-00000
[training@localhost ~]$ hadoop fs -cat MRDir1/part-r-00000
BUS 7
CAR 4
TRAIN 6
```

Result:

EXP.NO:05

DATE:

MapReduce to find the maximum electrical consumption in each year

AIM:

To Develop a MapReduce to find the maximum electrical consumption in each year given electrical consumption for each month in each year.

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

If the above data is given as input, we have to write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on. This is a walkover for the programmers with finite number of records. They will simply write the logic to produce the required output, and pass the data to the application written.

But, think of the data representing the electrical consumption of all the largescale industries of a particular state, since its formation.

When we write applications to process such bulk data,

- They will take a lot of time to execute.
- There will be a heavy network traffic when we move data from source to network server and so on.

To solve these problems, we have the MapReduce framework

Input Data

The above data is saved as sample.txt and given as input. The input file looks as shown below.

```
1979 23 23 2 43 24 25 26 26 26 26 25 26 25
1980 26 27 28 28 28 30 31 31 31 30 30 30 29
1981 31 32 32 32 33 34 35 36 36 34 34 34 34
1984 39 38 39 39 39 41 42 43 40 39 38 38 40
1985 38 39 39 39 39 41 41 41 00 40 39 39 45
```

Source code:

```
import java.util.*;
import java.io.IOException;
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
public class ProcessUnits
{
//Mapper class
public static class E_EMapper extends MapReduceBase implements
Mapper&lt;LongWritable ,/*Input key Type */ Text, /*Input value Type*/
```

```
Text, /*Output key Type*/ IntWritable&gt; /*Output value Type*/
{
//Map function
public void map(LongWritable key, Text value, OutputCollector&lt;Text, IntWritable&gt; output,
Reporter reporter) throws IOException
{
String line = value.toString(); String lasttoken = null;
StringTokenizer s = new StringTokenizer(line,"\t");
String year = s.nextToken();
while(s.hasMoreTokens())
{
lasttoken=s.nextToken();
}
int avgprice = Integer.parseInt(lasttoken);
output.collect(new Text(year), new IntWritable(avgprice));
}
}
//Reducer class
public static class E_EReduce extends MapReduceBase implements
Reducer&lt; Text, IntWritable, Text, IntWritable &gt;
{
//Reduce function
public void reduce( Text key, Iterator &lt;IntWritable&gt; values, OutputCollector&lt;Text,
IntWritable&gt; output, Reporter reporter) throws
IOException
{
int maxavg=30;
int val=Integer.MIN_VALUE;
while (values.hasNext())
{
if((val=values.next().get())&gt;maxavg)
{
output.collect(key, new IntWritable(val));
}
}
} }
//Main function
public static void main(String args[])throws Exception
{
JobConf conf = new JobConf(ProcessUnits.class);
conf.setJobName("max_eletricityunits");
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(E_EMapper.class);
conf.setCombinerClass(E_EReduce.class);
conf.setReducerClass(E_EReduce.class);
conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(conf, new Path(args[0])); FileOutputFormat.setOutputPath(conf,
new Path(args[1]));
JobClient.runJob(conf);
}
```

Output:

Input:

```
Kolkata,56
Jaipur,45
Delhi,43
Mumbai,34
Goa,45
Kolkata,35
Jaipur,34
Delhi,32
```

Output:

```
Kolkata 56
Jaipur 45
Delhi 43
Mumbai 34
```

Result:

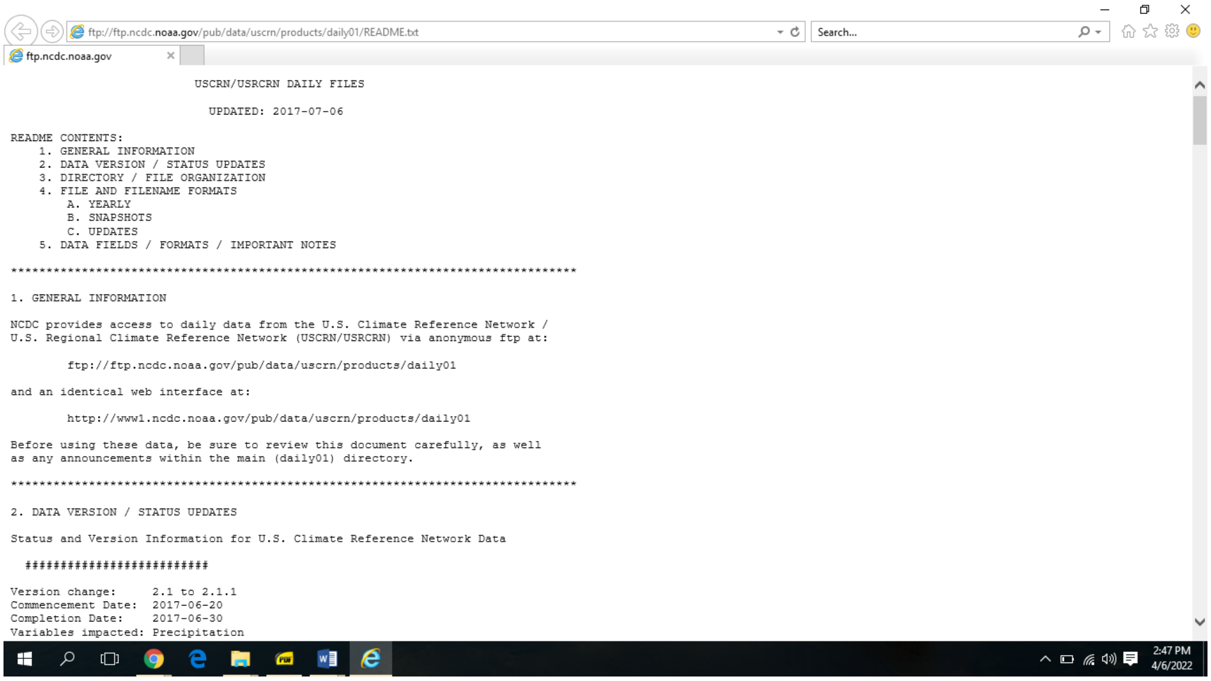
EXP.NO:06

DATE:

MapReduce to analyze weather data set and print whether the day is shinny or cool

AIM:

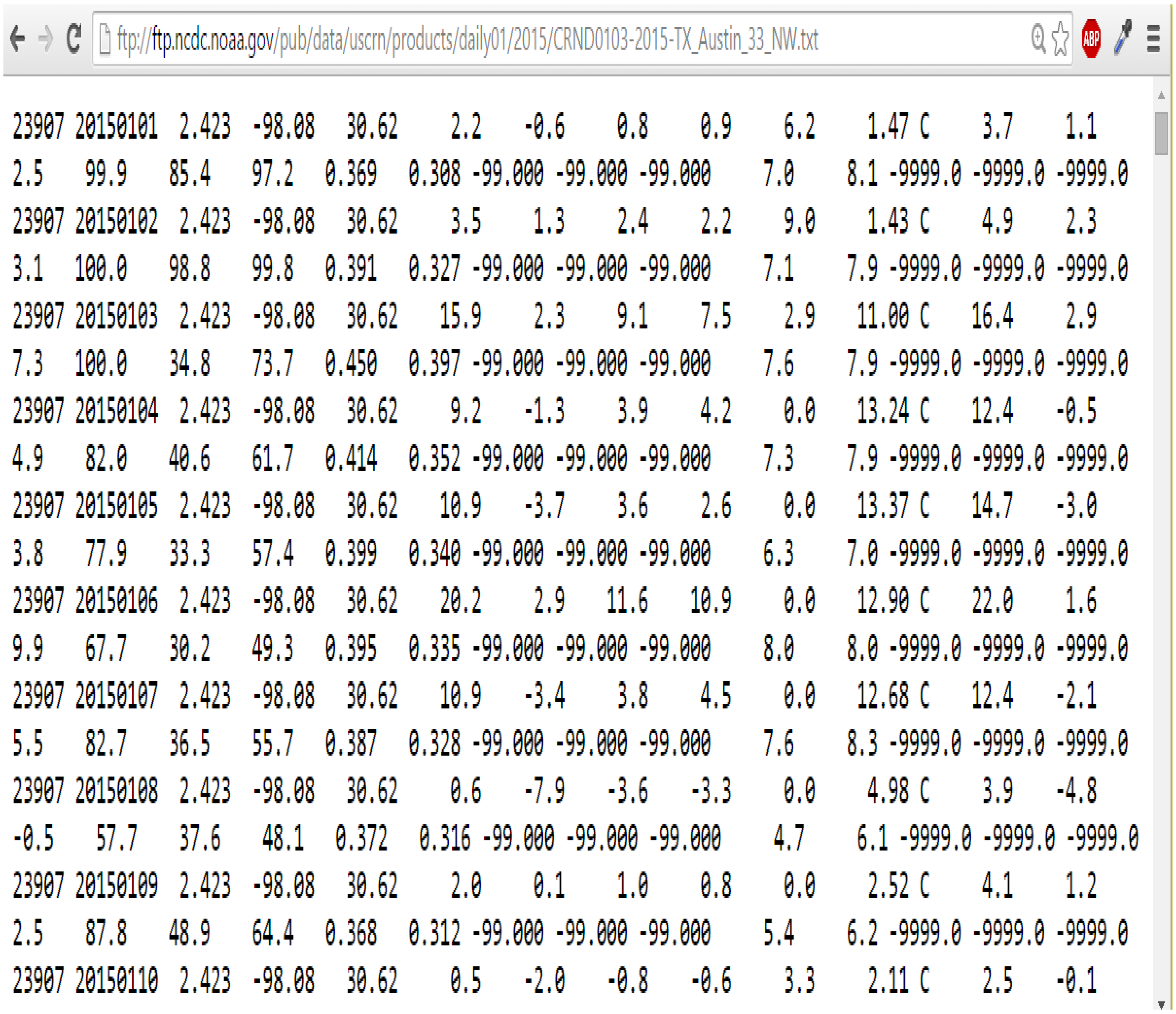
To Develop a MapReduce to analyze weather data set and print whether the day is shinny or cool day.
NOAA's National Climatic Data Center (***NCDC***) is responsible for preserving, monitoring, assessing, and providing public access to weather data.
NCDC provides access to daily data from the U.S. Climate Reference Network / U.S. Regional Climate Reference Network (USCRN/USRCRN) via anonymous ftp at:


ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/README.txt
Search...
ftp.ncdc.noaa.gov
2:47 PM
4/6/2022

```
                          USCRN/USRCRN DAILY FILES

                            UPDATED: 2017-07-06

README CONTENTS:
    1. GENERAL INFORMATION
    2. DATA VERSION / STATUS UPDATES
    3. DIRECTORY / FILE ORGANIZATION
    4. FILE AND FILENAME FORMATS
        A. YEARLY
        B. SNAPSHOTS
        C. UPDATES
    5. DATA FIELDS / FORMATS / IMPORTANT NOTES

********************************************************************************

1. GENERAL INFORMATION

NCDC provides access to daily data from the U.S. Climate Reference Network /
U.S. Regional Climate Reference Network (USCRN/USRCRN) via anonymous ftp at:

        ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01

and an identical web interface at:

        http://www1.ncdc.noaa.gov/pub/data/uscrn/products/daily01

Before using these data, be sure to review this document carefully, as well
as any announcements within the main (daily01) directory.

********************************************************************************

2. DATA VERSION / STATUS UPDATES

Status and Version Information for U.S. Climate Reference Network Data

  ##########################

Version change:     2.1 to 2.1.1
Commencement Date:  2017-06-20
Completion Date:    2017-06-30
Variables impacted: Precipitation
```

Dataset ftp:/

After going through wordcount mapreduce guide, you now have the basic idea of how a mapreduce program works. So, let us see a complex mapreduce program on weather dataset. Here I am using one of the dataset of year 2015 of Austin, Texas . We will do analytics on the dataset and classify whether it was a hot day or a cold day NCDC gives us all the weather data we need for this mapreduce project. The dataset which we will be using looks like below snapshot.
ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/2015/CRND0103-2015-TX_Austin_33_NW.txt depending on the temperature recorded by NCDC.



ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/daily01/2015/CRND0103-2015-TX_Austin_33_NW.txt

```
23907 20150101  2.423  -98.08   30.62     2.2    -0.6     0.8     0.9     6.2     1.47 C     3.7     1.1
2.5    99.9    85.4    97.2   0.369   0.308 -99.000 -99.000 -99.000     7.0     8.1 -9999.0 -9999.0 -9999.0
23907 20150102  2.423  -98.08   30.62     3.5     1.3     2.4     2.2     9.0     1.43 C     4.9     2.3
3.1   100.0    98.8    99.8   0.391   0.327 -99.000 -99.000 -99.000     7.1     7.9 -9999.0 -9999.0 -9999.0
23907 20150103  2.423  -98.08   30.62    15.9     2.3     9.1     7.5     2.9    11.00 C    16.4     2.9
7.3   100.0    34.8    73.7   0.450   0.397 -99.000 -99.000 -99.000     7.6     7.9 -9999.0 -9999.0 -9999.0
23907 20150104  2.423  -98.08   30.62     9.2    -1.3     3.9     4.2     0.0    13.24 C    12.4    -0.5
4.9    82.0    40.6    61.7   0.414   0.352 -99.000 -99.000 -99.000     7.3     7.9 -9999.0 -9999.0 -9999.0
23907 20150105  2.423  -98.08   30.62    10.9    -3.7     3.6     2.6     0.0    13.37 C    14.7    -3.0
3.8    77.9    33.3    57.4   0.399   0.340 -99.000 -99.000 -99.000     6.3     7.0 -9999.0 -9999.0 -9999.0
23907 20150106  2.423  -98.08   30.62    20.2     2.9    11.6    10.9     0.0    12.90 C    22.0     1.6
9.9    67.7    30.2    49.3   0.395   0.335 -99.000 -99.000 -99.000     8.0     8.0 -9999.0 -9999.0 -9999.0
23907 20150107  2.423  -98.08   30.62    10.9    -3.4     3.8     4.5     0.0    12.68 C    12.4    -2.1
5.5    82.7    36.5    55.7   0.387   0.328 -99.000 -99.000 -99.000     7.6     8.3 -9999.0 -9999.0 -9999.0
23907 20150108  2.423  -98.08   30.62     0.6    -7.9    -3.6    -3.3     0.0     4.98 C     3.9    -4.8
-0.5    57.7    37.6    48.1   0.372   0.316 -99.000 -99.000 -99.000     4.7     6.1 -9999.0 -9999.0 -9999.0
23907 20150109  2.423  -98.08   30.62     2.0     0.1     1.0     0.8     0.0     2.52 C     4.1     1.2
2.5    87.8    48.9    64.4   0.368   0.312 -99.000 -99.000 -99.000     5.4     6.2 -9999.0 -9999.0 -9999.0
23907 20150110  2.423  -98.08   30.62     0.5    -2.0    -0.8    -0.6     3.3     2.11 C     2.5    -0.1
```

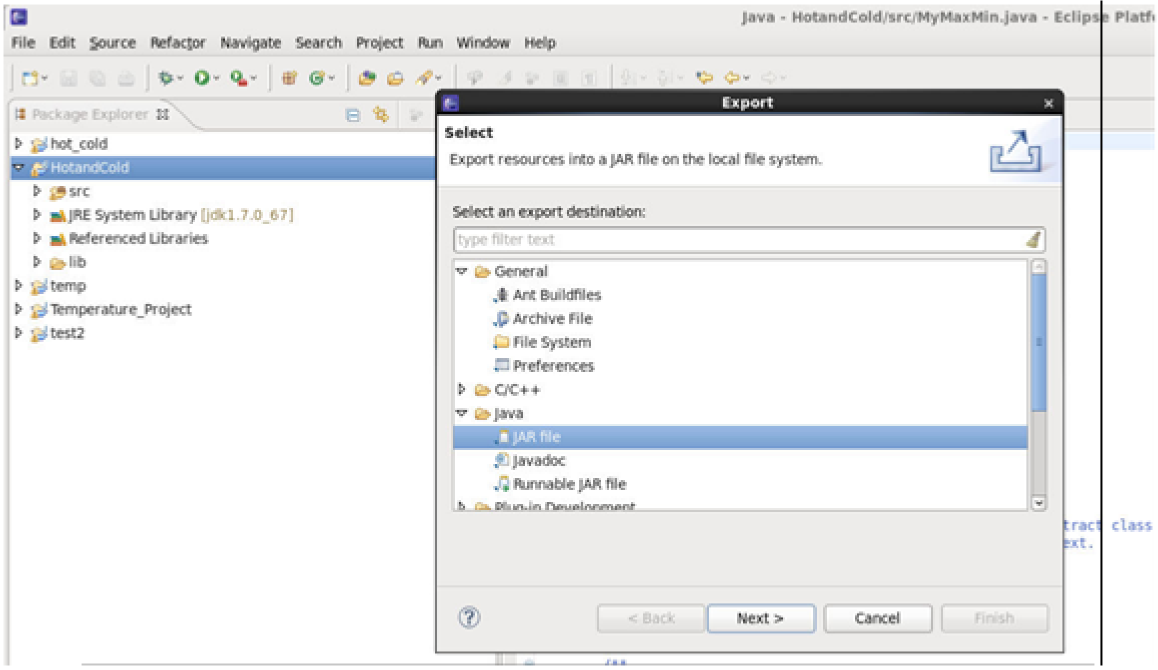
PROGRAM:

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
public class MyMaxMin {
public static class MaxTemperatureMapper extends
Mapper<LongWritable, Text, Text, Text> {
/**
* @method map
* This method takes the input as text data type
* Now leaving the first five tokens,it takes 6th token is taken as temp_max and
* 7th token is taken as temp_min. Now temp_max > 35 and
temp_min < 10 are passed to the reducer.
*/ @Override
public void map(LongWritable arg0, Text Value, Context 2 context) throws IOException,
InterruptedException {
//Converting the record (single line) to String and storing it in a String variable line
String line = Value.toString();
//Checking if the line is not empty
if (!(line.length() == 0)) {
//date
String date = line.substring(6, 14);
//maximum temperature
float temp_Max = Float
parseFloat(line.substring(39, 45).trim());
//minimum temperature
float temp_Min = Float
parseFloat(line.substring(47, 53).trim());
//if maximum temperature is greater than 35 , its a hot day
if (temp_Max > 35.0) {
// Hot day
context.write(new Text("Hot Day " + date),
new Text(String.valueOf(temp_Max)));
}
//if minimum temperature is less than 10, it’s a cold day
if (temp_Min < 10) {
// Cold day
context.write(new Text("Cold Day " + date),
new Text(String.valueOf(temp_Min)));
}
}
}
}
//Reducer
*MaxTemperatureReducer class is static and extends Reducer abstract
having four hadoop generics type Text, Text, Text, Text.
*/
public static class MaxTemperatureReducer extends Reducer<Text, Text, Text, Text> {
public void reduce (Text Key, Iterator<Text> Values, Context context) throws IOException,
Interrupted Exception {
String temperature = Values.next().toString();
context.write(Key, new Text(temperature));
}
}
public static void main(String[] args) throws Exception {
Confguration conf = new Configuration();
```

```
Job job = new Job(conf, "weather example");
job.setJarByClass(MyMaxMin.class);
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
job.setMapperClass(MaxTemperatureMapper.class);
job.setReducerClass(MaxTemperatureReducer.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path OutputPath = new Path(args[1]);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
OutputPath.getFileSystem(conf).delete(OutputPath);
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

Import the project in eclipse IDE in the same way it was told in earlier guide and change the jar paths with the jar files present in the lib directory of this project.

When the project is not having any error, we will export it as a jar file, same as we did in wordcount mapreduce guide. Right Click on the Project file and click on Export. Select jar file. Give the path where you want to save to file.



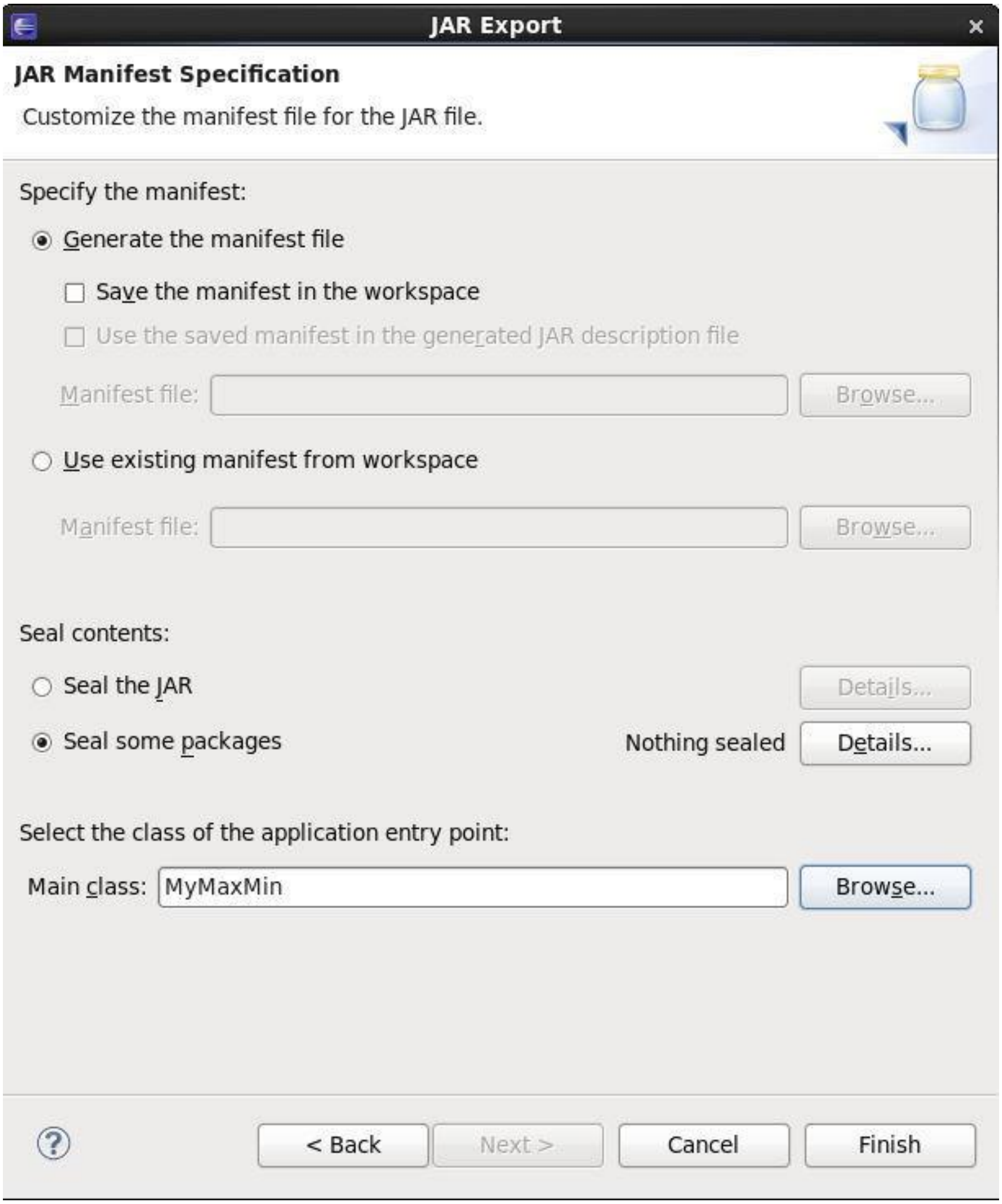
Java - HotandCold/src/MyMaxMin.java - Eclipse Platf

Export

Select

Export resources into a JAR file on the local file system.

Select an export destination:



JAR Export

JAR Manifest Specification

Customize the manifest file for the JAR file.

Specify the manifest:

● Generate the manifest file

□ Save the manifest in the workspace

□ Use the saved manifest in the generated JAR description file

Manifest file:

○ Use existing manifest from workspace

Manifest file:

Seal contents:

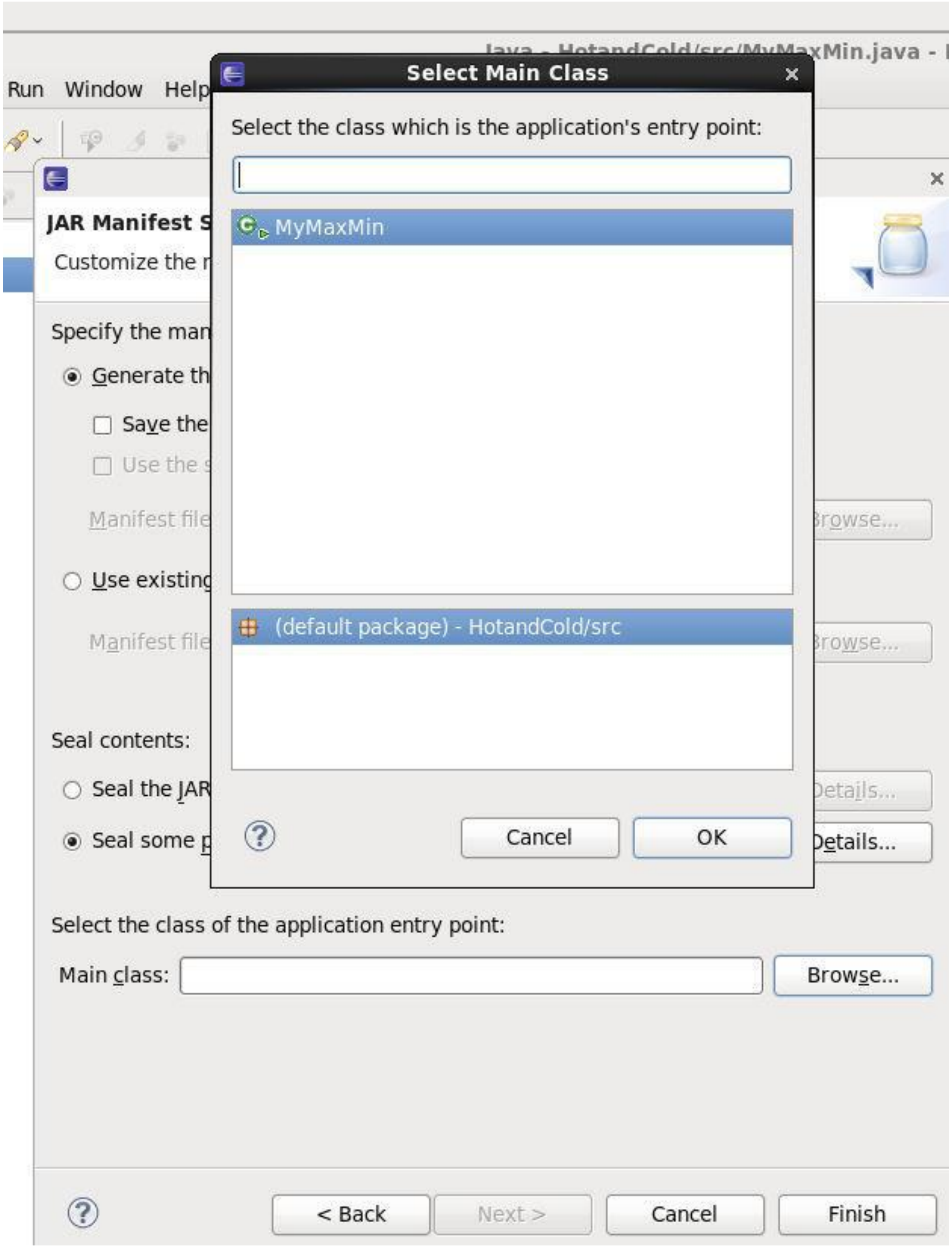
○ Seal the JAR

● Seal some packages

Nothing sealed

Select the class of the application entry point:

Main class: MyMaxMin



Select Main Class

Select the class which is the application's entry point:

MyMaxMin

(default package) - HotandCold/src

JAR Manifest S

Customize the r

Specify the man

● Generate th

□ Save the

□ Use the s

Manifest file

○ Use existing

Manifest file

Seal contents:

○ Seal the JAR

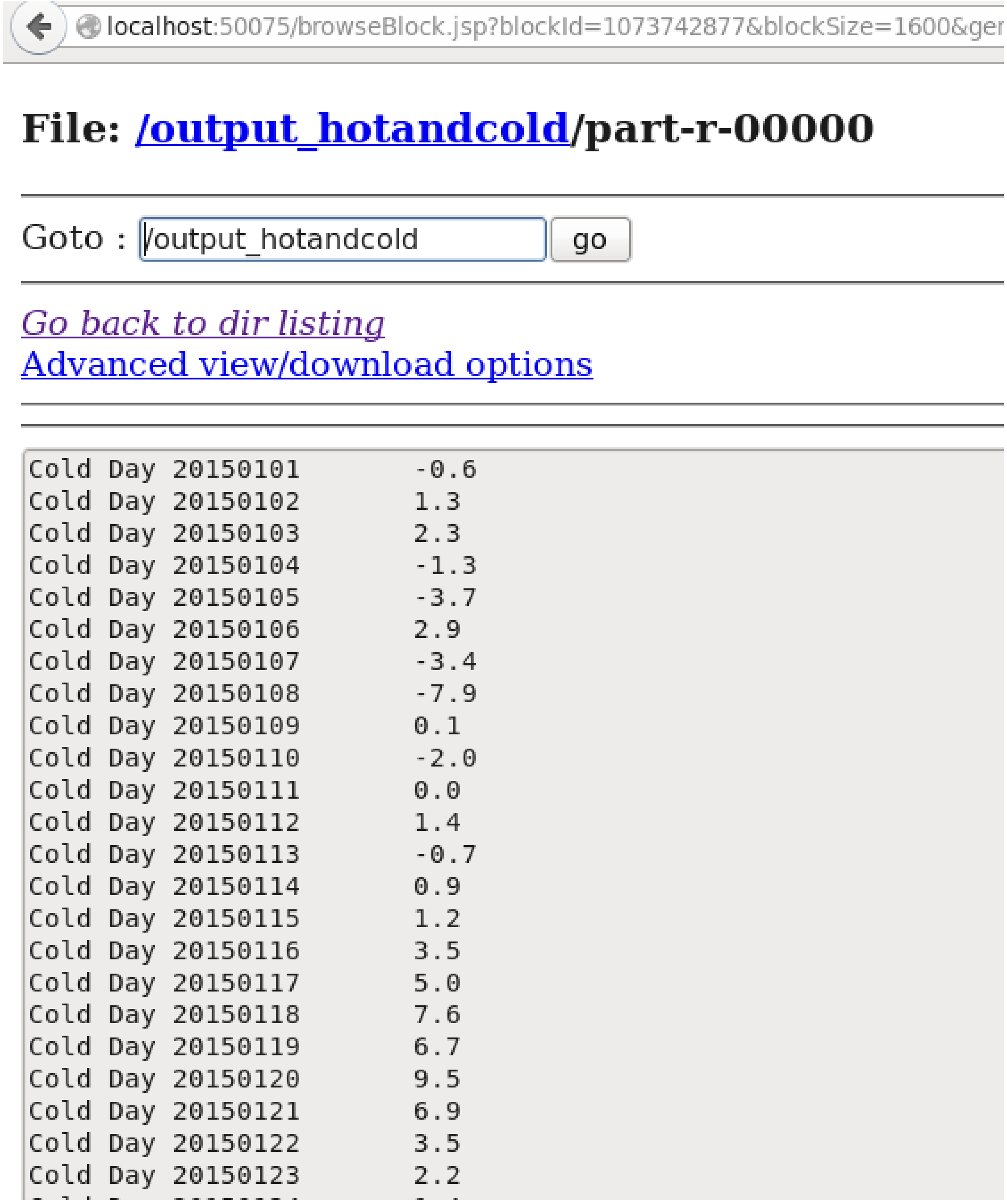
● Seal some p

Select the class of the application entry point:

Main class:

temperature.jar
https://drive.google.com/file/d/0B2SFMPvhXPQ5RUlZZDZSR3FYVDA/view?usp=sharing

Download Dataset used by me using below link weather_data.txt
https://drive.google.com/file/d/0B2SFMPvhXPQ5aFVILXAxbFh6ejA/view?usp=sharing.



localhost:50075/browseBlock.jsp?blockId=1073742877&blockSize=1600&ger

File: /output_hotandcold/part-r-00000

Goto : /output_hotandcold go

Go back to dir listing
Advanced view/download options

```
Cold Day 20150101       -0.6
Cold Day 20150102       1.3
Cold Day 20150103       2.3
Cold Day 20150104       -1.3
Cold Day 20150105       -3.7
Cold Day 20150106       2.9
Cold Day 20150107       -3.4
Cold Day 20150108       -7.9
Cold Day 20150109       0.1
Cold Day 20150110       -2.0
Cold Day 20150111       0.0
Cold Day 20150112       1.4
Cold Day 20150113       -0.7
Cold Day 20150114       0.9
Cold Day 20150115       1.2
Cold Day 20150116       3.5
Cold Day 20150117       5.0
Cold Day 20150118       7.6
Cold Day 20150119       6.7
Cold Day 20150120       9.5
Cold Day 20150121       6.9
Cold Day 20150122       3.5
Cold Day 20150123       2.2
```

Result:

EXP.NO:07

DATE

XYZ.com is an online music website where users listen to various tracks

AIM:

XYZ.com is an online music website where users listen to various tracks, the data gets collected which is given below.

Write a MapReduce program to get the following

□ Number of unique listeners
□ Number of times the track was shared with others
□ Number of times the track was listened to on the radio
□ Number of times the track was listened to in total
□ Number of times the track was skipped on the radio

Solution

XYZ.com is an online music website where users listen to various tracks, the data gets collected like shown below. Write a map reduce program to get following stats

• Number of unique listeners
• Number of times the track was shared with others
• Number of times the track was listened to on the radio
• Number of times the track was listened to in total
• Number of times the track was skipped on the radio

The data is coming in log files and looks like as shown below.

```
UserId|TrackId|Shared|Radio|Skip
111115|222|0|1|0
111113|225|1|0|0
```

```
111117|223|0|1|1
111115|225|1|0|0
```

In this tutorial we are going to solve the first problem, that is finding out unique listeners per track.

First of all we need to understand the data, here the first column is UserId and the second one is Track Id. So we need to write a mapper class which would emit trackId and userIds and intermediate key value pairs. To make it simple to remember the data sequence, let's create a constants class as shown below

```
public class LastFMConstants {
public static final int USER_ID = 0; public static final int TRACK_ID = 1; public static final
int IS_SHARED = 2; public static final int RADIO = 3;
public static final int IS_SKIPPED = 4;
}
```

Now, lets create the mapper class which would emit intermediate key value pairs as (TrackId, UserId) as shwon below

```
public static class UniqueListenersMapper extends
Mapper< Object , Text, IntWritable, IntWritable > { IntWritable trackId = new IntWritable();
IntWritable userId = new IntWritable();
public void map(Object key, Text value,
Mapper< Object , Text, IntWritable, IntWritable > .Context context)
throws IOException, InterruptedException {
String[] parts = value.toString().split("[|]");
trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
userId.set(Integer.parseInt(parts[LastFMConstants.USER_ID]));
if (parts.length == 5) {
context.write(trackId, userId);
```

```
} else {
context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L)
```

```
}
}
}
public static class UniqueListenersReducer extends
Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {
public void reduce( IntWritable trackId,
Iterable< IntWritable > userIds,
Reducer< IntWritable , IntWritable, IntWritable, IntWritable>.Context
context)
throws IOException, InterruptedException {
Set< Integer > userIdSet = new HashSet< Integer >();
for (IntWritable userId : userIds) {
userIdSet.add(userId.get());
}
IntWritable size = new IntWritable(userIdSet.size());
context.write(trackId, size);
}
}
```

Here we are using Set to eliminate duplicate userIds. Now we can take look at the Driver class

```
public static void main(String[] args) throws Exception { Configuration
conf = new Configuration(); if (args.length != 2) {
System.err.println("Usage: uniquelisteners < in > < out >");
System.exit(2);
}
Job job = new Job(conf, "Unique listeners per track");
job.setJarByClass(UniqueListeners.class);
job.setMapperClass(UniqueListenersMapper.class);
job.setReducerClass(UniqueListenersReducer.class);
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
System.out.println("No. of Invalid Records :"
+ counters.findCounter(COUNTERS.INVALID_RECORD_COUNT)
.getValue());
}
```

Output:

```
UserId |TrackId |Shared |Radio | Skip
111115 | 222 | 0 | 1 | 0
111113 | 225 | 1 | 0 | 0
111117 | 223 | 0 | 1 | 1
111115 | 225 | 1 | 0 | 0
```

Result:

EXP.NO:08

DATE

MapReduce program to analyze Uber data set

AIM:

To Develop a MapReduce program to analyze Uber data set to find the days on which each basement has more trips using the following dataset.

Problem Statement 1: In this problem statement, we will find the days on which each basement has more trips.

Source Code

Mapper Class:

```
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
java.text.SimpleDateFormat format = new
java.text.SimpleDateFormat("MM/dd/yyyy");
String[] days ={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};
private Text basement = new Text();
Date date = null;
private int trips;
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
String line = value.toString();
String[] splits = line.split(",");
basement.set(splits[0]);
try {
date = format.parse(splits[1]);
} catch (ParseException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
trips = new Integer(splits[3]);
String keys = basement.toString()+ " "+days[date.getDay()];
context.write(new Text(keys), new IntWritable(trips));
}
}
```

Reducer Class:

```
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
Context context
) throws IOException, InterruptedException {
int sum = 0
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
```

Whole Source Code:

```
import java.io.IOException;
import java.text.ParseException;
import java.util.Date;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Uber1 {
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
java.text.SimpleDateFormat format = new
java.text.SimpleDateFormat("MM/dd/yyyy");
String[] days ={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};
private Text basement = new Text();
Date date = null;
private int trips;
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
String line = value.toString();
String[] splits = line.split(",");
basement.set(splits[0]);
try {
date = format.parse(splits[1]);
} catch (ParseException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
trips = new Integer(splits[3]);
String keys = basement.toString()+ " "+days[date.getDay()];
context.write(new Text(keys), new IntWritable(trips));
}
}
public static class IntSumReducer
extends Reducer<Text,IntWritable,Text,IntWritable>
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values
Context context
) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = Job.getInstance(conf, "Uber1");
job.setJarByClass(Uber1.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

```
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

Running the Program:

First, we need to build a jar file for the above program and we need to run it as a normal Hadoop program by passing the input dataset and the output file path as shown below.

```
hadoop jar uber1.jar /uber /user/output1
```

In the output file directory, a part of the file is created and contains the below

Output:

```
B02512 Sat 15026
B02512 Sun 10487
B02512 Thu 15809
B02512 Tue 12041
B02512 Wed 12691
B02598 Fri 93126
B02598 Mon 60882
B02598 Sat 94588
B02598 Sun 66477
B02598 Thu 90333
B02598 Tue 63429
B02598 Wed 71956
B02617 Fri 125067
B02617 Mon 80591
B02617 Sat 127902
B02617 Sun 91722
B02617 Thu 118254
B02617 Tue 86602
B02617 Wed 94887
B02682 Fri 114662
B02682 Mon 74939
B02682 Sat 120283
B02682 Sun 82825
B02682 Thu 106643
B02682 Tue 76905
B02682 Wed 86252
```

```
B02764 Fri 326968
B02764 Mon 214116
B02764 Sat 356789
B02764 Sun 249896
B02764 Thu 304200
B02764 Tue 221343
B02764 Wed 241137
B02765 Fri 34934
B02765 Mon 21974
B02765 Sat 36737
```

Result:

EXP.NO:9
DATE

HIVE OPERATIONS

AIM:

To Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

RESOURCES:

VMWare, XAMPP Server, Web Browser, 1GB RAM, Hard Disk 80 GB.

PROGRAM LOGIC:

SYNTAX for HIVE Database Operations
DATABASE Creation
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
Drop Database Statement
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS]
database_name [RESTRICT|CASCADE];
Creating and Dropping Table in HIVE
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]
table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment] [ROW FORMAT row_format] [STORED AS file_format]
Loading Data into table log_data
Syntax:
LOAD DATA LOCAL INPATH '<path>/u.data' OVERWRITE INTO TABLE
u_data;
Alter Table in HIVE
Syntax
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name DROP [COLUMN] column_name
ALTER TABLE name CHANGE column_name new_name new_type
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
Creating and Dropping View
CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT

```
column_comment], ...) ] [COMMENT table_comment] AS SELECT ..
```

Dropping View

Syntax:

```
DROP VIEW view_name
```

Functions in HIVE

String Functions:- round(), ceil(), substr(), upper(), reg_exp() etc Date and Time Functions:- year(), month(), day(), to_date() etc

Aggregate Functions :- sum(), min(), max(), count(), avg() etc

INDEXES

```
CREATE INDEX index_name ON TABLE base_table_name (col_name,
...)
AS 'index.handler.class.name'
[WITH DEFERRED REBUILD]
[IDXPROPERTIES (property_name=property_value, ...)]
[IN TABLE index_table_name]
[PARTITIONED BY (col_name, ...)]
[
[ ROW FORMAT ...] STORED AS ...
| STORED BY ...
]
[LOCATION hdfs_path]
[TBLPROPERTIES (...)]
```

Creating Index

```
CREATE INDEX index_ip ON TABLE log_data(ip_address) AS
'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH
DEFERRED
REBUILD;
```

Altering and Inserting Index

```
ALTER INDEX index_ip_address ON log_data REBUILD;
```

Storing Index Data in Metastore

```
SET
hive.index.compact.file=/home/administrator/Desktop/big/metastore_db/t
mp/index_ipadd
ress_result;
SET
hive.input.format=org.apache.hadoop.hive.ql.index.compact.HiveCompa
ctIndexInputFor
mat;
```

Dropping Index

```
DROP INDEX INDEX_NAME on TABLE_NAME;
```

Output:

Result:

EXP.NO:10

DATE

Queries to sort and aggregate the data in a table using HiveQL

AIM:

To Write queries to sort and aggregate the data in a table using HiveQL.

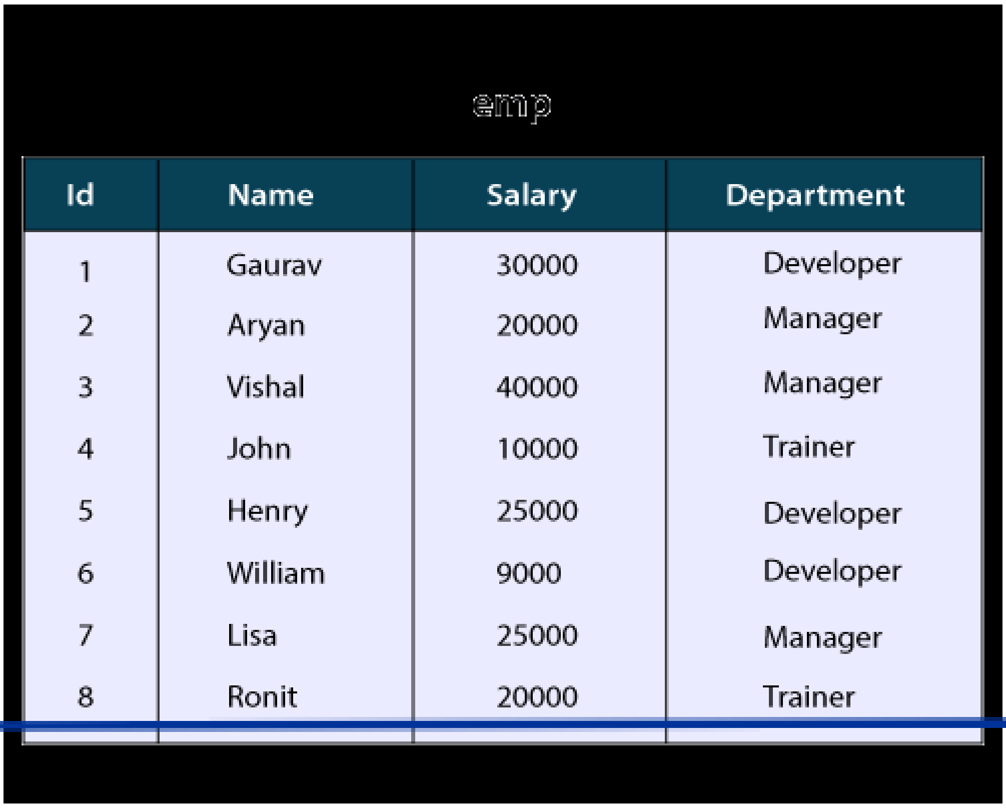
Description:

Hive is an open-source data warehousing solution built on top of Hadoop. It supports an SQL-like query language called HiveQL. These queries are compiled into MapReduce jobs that are executed on Hadoop. While Hive uses Hadoop for execution of queries, it reduces the effort that goes into writing and maintaining MapReduce jobs.

Hive supports database concepts like tables, columns, rows and partitions. Both primitive (integer, float, string) and complex data-types(map, list, struct) are supported. Moreover, these types can be composed to support structures of arbitrary complexity. The tables are serialized/deserialized using default serializers/deserializer. Any new data format and type can be supported by implementing SerDe and ObjectInspector java interface.

HiveQL - ORDER BY and SORT BY Clause

By using HiveQL ORDER BY and SORT BY clause, we can apply sort on the column. It returns the result set either in ascending or descending order. Here, we are going to execute these clauses on the records of the below table:



emp

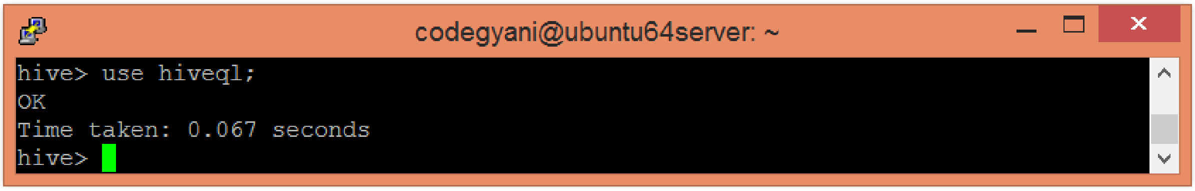
Id	Name	Salary	Department
1	Gaurav	30000	Developer
2	Aryan	20000	Manager
3	Vishal	40000	Manager
4	John	10000	Trainer
5	Henry	25000	Developer
6	William	9000	Developer
7	Lisa	25000	Manager
8	Ronit	20000	Trainer

In HiveQL, ORDER BY clause performs a complete ordering of the query result set. Hence, the complete data is passed through a single reducer. This may take much time in the execution of large datasets. However, we can use LIMIT to minimize the sorting time.

Example:

Select the database in which we want to create a table.

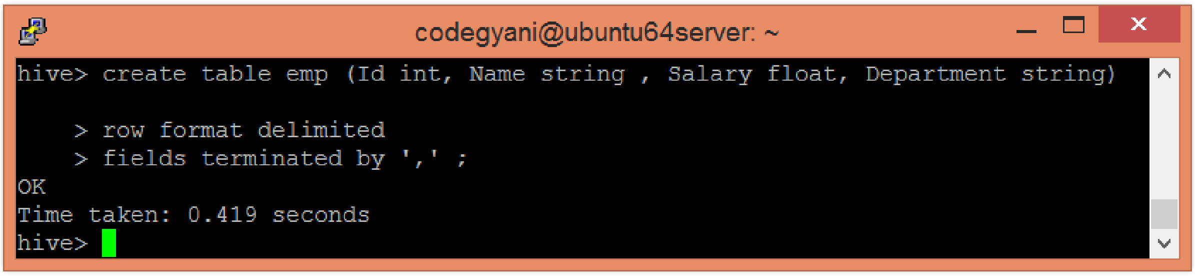
hive**>** use hiveql;



```
hive> use hiveql;
OK
Time taken: 0.067 seconds
hive>
```

Now, create a table by using the following command:

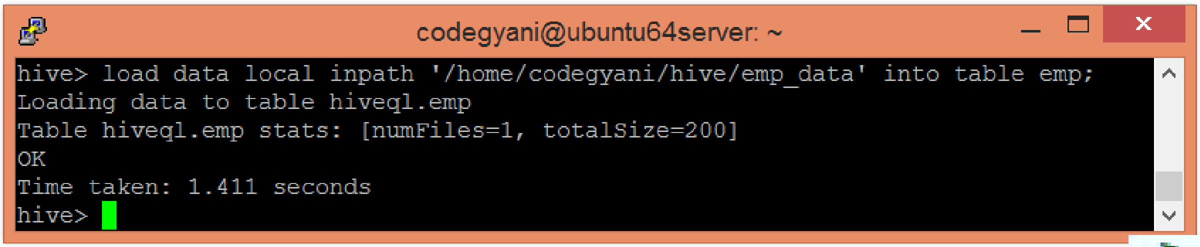
hive> create table emp (Id int, Name string , Salary float, Department string)
row format delimited
fields terminated by ',' ;



```
hive> create table emp (Id int, Name string , Salary float, Department string)

    > row format delimited
    > fields terminated by ',' ;
OK
Time taken: 0.419 seconds
hive>
```

Load the data into the table

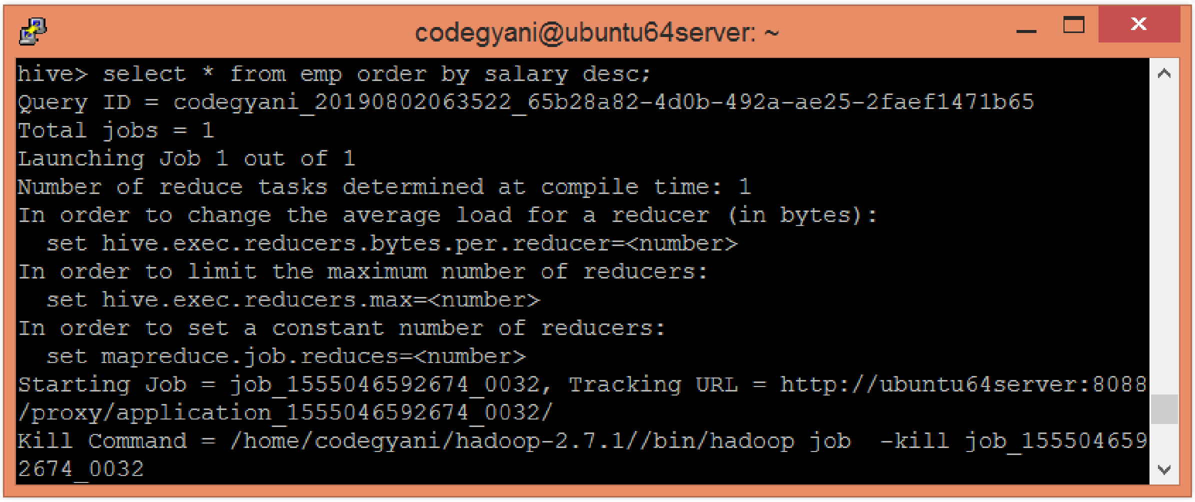
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;



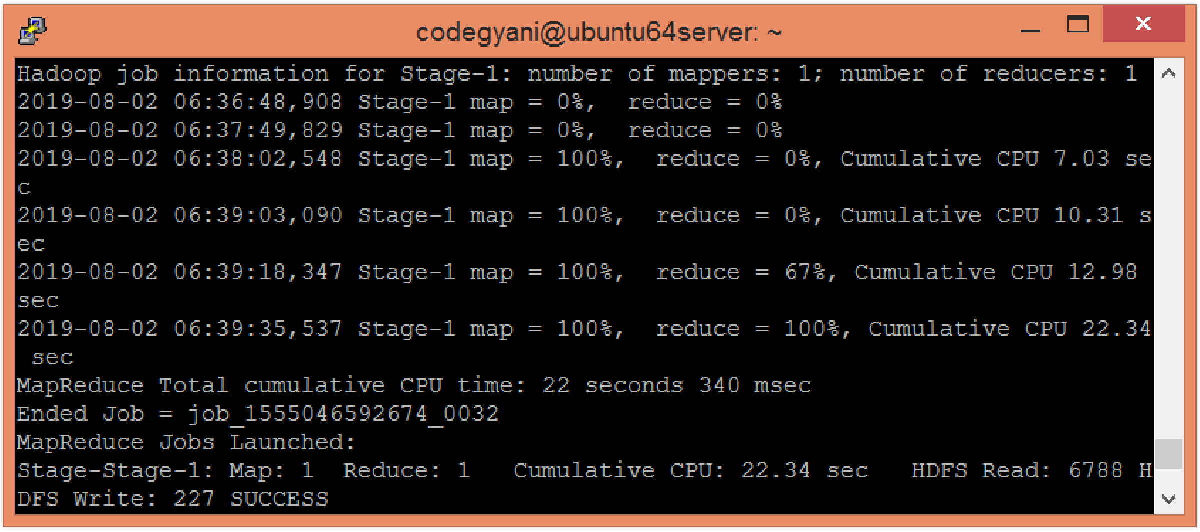
```
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;
Loading data to table hiveql.emp
Table hiveql.emp stats: [numFiles=1, totalSize=200]
OK
Time taken: 1.411 seconds
hive>
```

Now, fetch the data in the descending order by using the following command

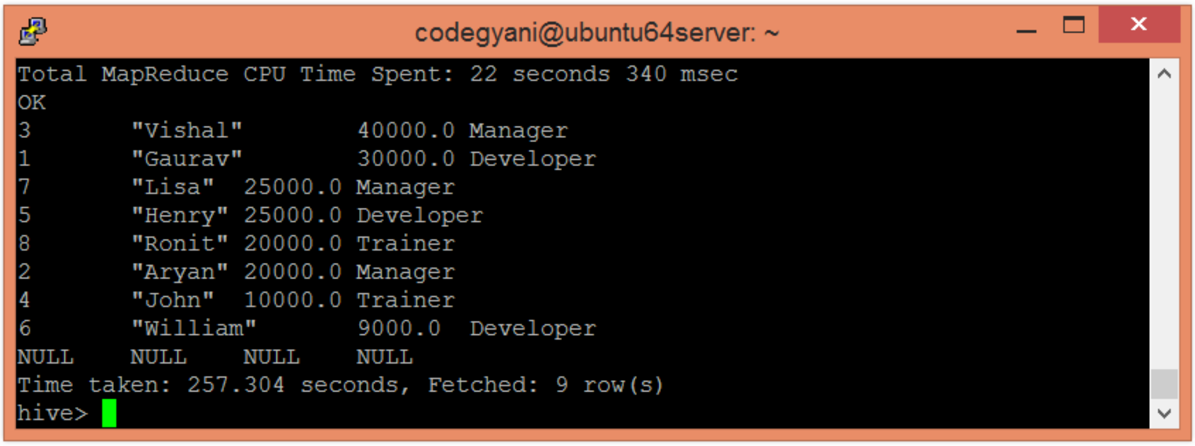
hive> select * from emp order by salary desc;



```
hive> select * from emp order by salary desc;
Query ID = codegyani_20190802063522_65b28a82-4d0b-492a-ae25-2faef1471b65
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1555046592674_0032, Tracking URL = http://ubuntu64server:8088
/proxy/application_1555046592674_0032/
Kill Command = /home/codegyani/hadoop-2.7.1//bin/hadoop job  -kill job_155504659
2674_0032
```



```
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-08-02 06:36:48,908 Stage-1 map = 0%,  reduce = 0%
2019-08-02 06:37:49,829 Stage-1 map = 0%,  reduce = 0%
2019-08-02 06:38:02,548 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.03 se
c
2019-08-02 06:39:03,090 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 10.31 s
ec
2019-08-02 06:39:18,347 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 12.98
sec
2019-08-02 06:39:35,537 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 22.34
 sec
MapReduce Total cumulative CPU time: 22 seconds 340 msec
Ended Job = job_1555046592674_0032
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 22.34 sec   HDFS Read: 6788 H
DFS Write: 227 SUCCESS
```



```
Total MapReduce CPU Time Spent: 22 seconds 340 msec
OK
3       "Vishal"        40000.0 Manager
1       "Gaurav"        30000.0 Developer
7       "Lisa"  25000.0 Manager
5       "Henry" 25000.0 Developer
8       "Ronit" 20000.0 Trainer
2       "Aryan" 20000.0 Manager
4       "John"  10000.0 Trainer
6       "William"       9000.0  Developer
NULL    NULL    NULL    NULL
Time taken: 257.304 seconds, Fetched: 9 row(s)
hive>
```

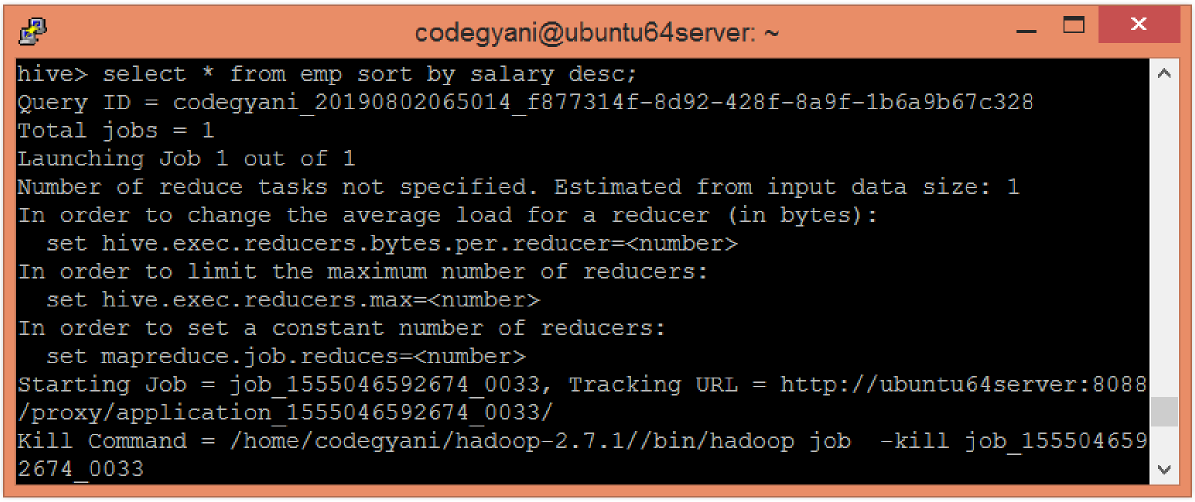
HiveQL - SORT BY Clause

The HiveQL SORT BY clause is an alternative of ORDER BY clause. It orders the data within each reducer. Hence, it performs the local ordering, where each reducer's output is sorted separately. It may also give a partially ordered result.

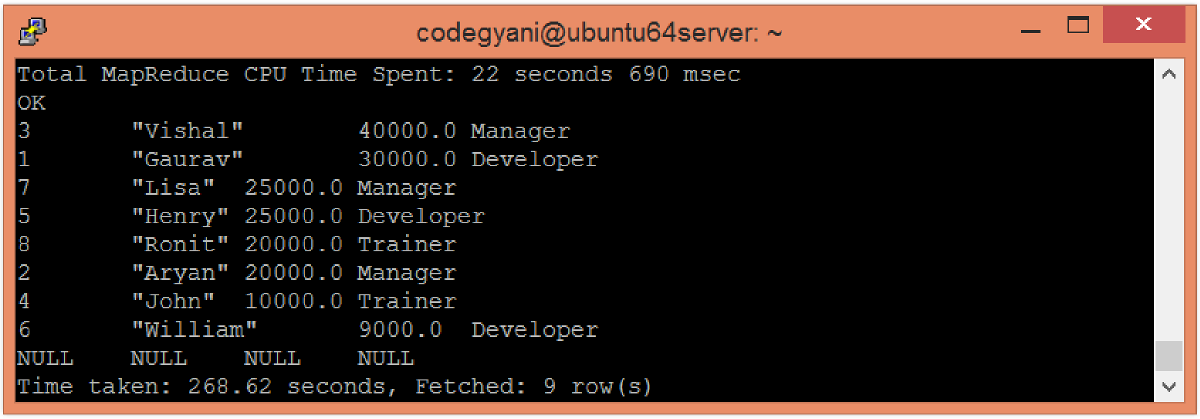
Example:

Let's fetch the data in the descending order by using the following command

hive> select * from emp sort by salary desc;



```
hive> select * from emp sort by salary desc;
Query ID = codegyani_20190802065014_f877314f-8d92-428f-8a9f-1b6a9b67c328
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1555046592674_0033, Tracking URL = http://ubuntu64server:8088
/proxy/application_1555046592674_0033/
Kill Command = /home/codegyani/hadoop-2.7.1//bin/hadoop job  -kill job_155504659
2674_0033
```



```
Total MapReduce CPU Time Spent: 22 seconds 690 msec
OK
3       "Vishal"        40000.0 Manager
1       "Gaurav"        30000.0 Developer
7       "Lisa"  25000.0 Manager
5       "Henry" 25000.0 Developer
8       "Ronit" 20000.0 Trainer
2       "Aryan" 20000.0 Manager
4       "John"  10000.0 Trainer
6       "William"       9000.0  Developer
NULL    NULL    NULL    NULL
Time taken: 268.62 seconds, Fetched: 9 row(s)
```

Cluster By:

Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL.

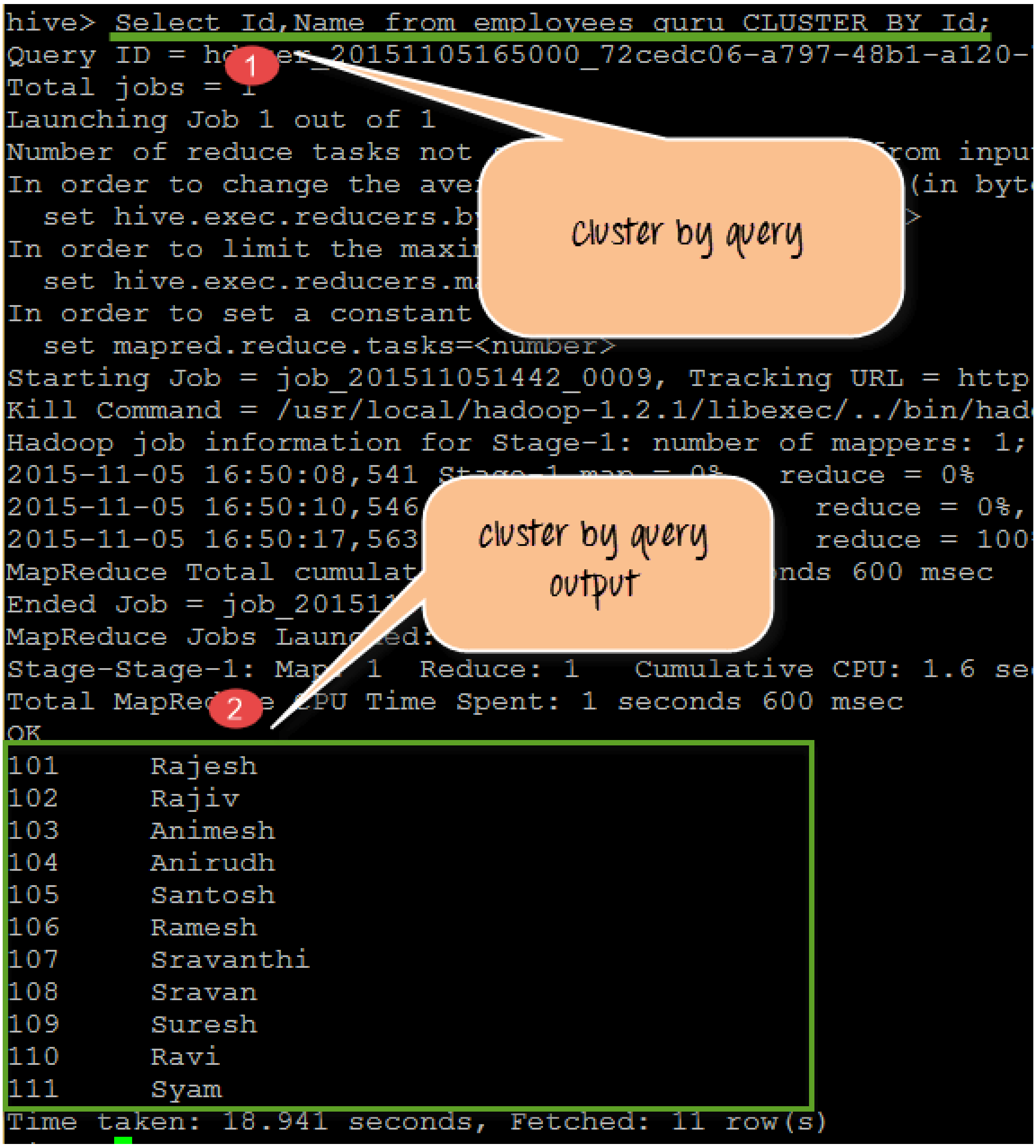
Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by to distribute the rows among reducers. Cluster BY columns will go to the multiple reducers.

□ It ensures sorting orders of values present in multiple reducers

For example, Cluster By clause mentioned on the Id column name of the table employees_guru table. The output when executing this query will give results to multiple reducers at the back end. But as front end it is an alternative clause for both Sort By and Distribute By.

Example:

SELECT Id, Name from employees_guru CLUSTER BY Id;



```
hive> Select Id,Name from employees_guru CLUSTER BY Id;
Query ID = hd    er_20151105165000_72cedc06-a797-48b1-a120-
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not                         from inpu
In order to change the ave                         (in byt
  set hive.exec.reducers.b                         >
In order to limit the maxi
  set hive.exec.reducers.m
In order to set a constant
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0009, Tracking URL = http
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/had
Hadoop job information for Stage-1: number of mappers: 1;
2015-11-05 16:50:08,541 Stage-1 map = 0%,  reduce = 0%
2015-11-05 16:50:10,546                     reduce = 0%,
2015-11-05 16:50:17,563                     reduce = 100
MapReduce Total cumulat                  nds 600 msec
Ended Job = job_20151
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.6 se
Total MapRe    e CPU Time Spent: 1 seconds 600 msec
OK
101     Rajesh
102     Rajiv
103     Animesh
104     Anirudh
105     Santosh
106     Ramesh
107     Sravanthi
108     Sravan
109     Suresh
110     Ravi
111     Syam
Time taken: 18.941 seconds, Fetched: 11 row(s)
```

Result: