Fundamental of Big Data Analytics

LAB FILE

BACHELOR OF TECHNOLOGY

(Computer Science and Engineering)

SEMESTER-7

Department of Computer Science & Engineering

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY, NOIDA

Niket Gandhir
A2305218311
7CSE - 5X

SUBMITTED TO: Dr. Juhi Singh

INDEX

9.	Write the script to get the structured dataset from RDBMS using sqoop.	04/10/2021	11/10/2021	
10.	WAP to Create database then create a table and insert data into table.	11/10/2021	18/10/2021	
11.	Create a database table and insert data into the HBase.	18/10/2021	25/10/2021	
12.	To run an application using Oozie.	25/10/2021	01/11/2021	

EXPERIMENT 1

AIM: INSTALL AND RUN HADOOP

Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.
Because Hadoop can process and store such a wide assortment of data, it enables organizations to set up data lakes as expansive reservoirs for incoming streams of information. In a Hadoop data lake, raw data is often stored as is so data scientists and other analysts can access the full data sets, if need be; the data is then filtered and prepared by analytics or IT teams, as needed, to support different applications.
Data lakes generally serve different purposes than traditional data warehouses that hold cleansed sets of transaction data. But, in some cases, companies view their Hadoop data lakes as modern-day data warehouses. Either way, the growing role of big data analytics in business decision-making has made effective data governance and data security processes a priority in data lake deployments and Hadoop systems in general.

ADVANTAGES OF HADOOP:

- Ability to store and process huge amounts of any kind of data, quickly. With data volumes and varieties constantly increasing, especially from social media and the Internet of Things (IoT), that's a key consideration.
- Computing power. Hadoop's distributed computing model processes big data fast. The more computing nodes you use, the more processing power you have.
- Fault tolerance. Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.
- Flexibility. Unlike traditional relational databases, you don't have to preprocess data before storing it. You can store as much data as you want and decide how to use it later. That includes unstructured data like text, images and videos.
- Low cost. The open-source framework is free and uses commodity hardware to store large quantities of data.
- Scalability. You can easily grow your system to handle more data simply by adding nodes. Little administration is required.

YARN greatly expanded the applications that Hadoop clusters can handle to include interactive querying, stream processing and real-time analytics. For example, manufacturers, utilities, oil and gas companies, and other businesses are using real-time data that's streaming into Hadoop systems from IoT devices in predictive maintenance applications to try to detect equipment failures before they occur. Fraud detection, website personalization and customer experience scoring are other real-time use cases.

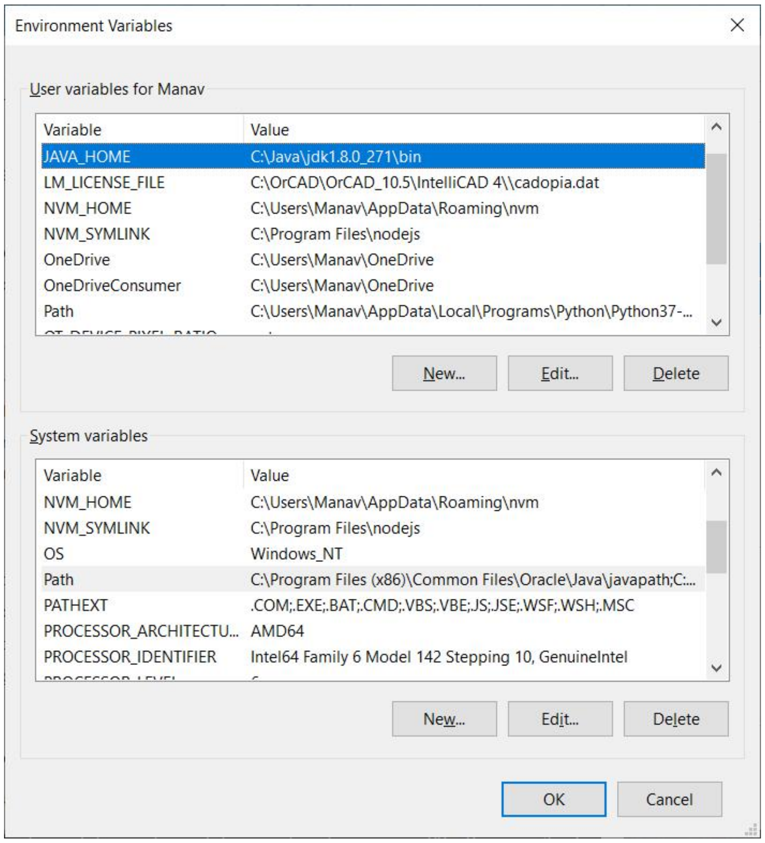
Some other common use cases for Hadoop include the following:

- Customer analytics. Examples include efforts to predict customer churn, analyze clickstream data to better target online ads to web users, and track customer sentiment based on comments about a company on social networks.
- Risk management. Financial services companies use Hadoop clusters to develop more accurate risk analysis models for use internally and by their customers. They also build investment models and develop trading algorithms in Hadoop-based big data systems.
- Operational intelligence. For example, Hadoop can help telecommunications companies better understand switching performance and network and frequency utilization for capacity planning and management. By analyzing how mobile services are consumed and the available bandwidth in geographic regions, telcos can also determine the best places to locate new cell towers and respond more quickly to network problems.
- Supply chain management. Manufacturers, retailers and trucking companies use Hadoop systems to track the movement of goods and vehicles so they can determine the costs of various transportation options. In addition, they can analyze large amounts of historical, time-stamped location data to map out potential delays and optimize delivery routes.

The technology has been deployed for many other uses, as well. For example, insurers use Hadoop for applications such as analyzing policy pricing and managing safe driver discount programs. Also, healthcare organizations look for ways to improve treatments and patient outcomes with Hadoop's aid.

INSTALLING HADOOP

1) Before installing Hadoop, we need to install Java 8
2) For this we have downloaded the JDK 8 version from Oracle and will install this on our device.
3) We now have to set the environment variables for Java Home with the necessary directory of the installed files.



Environment Variables

User variables for Manav

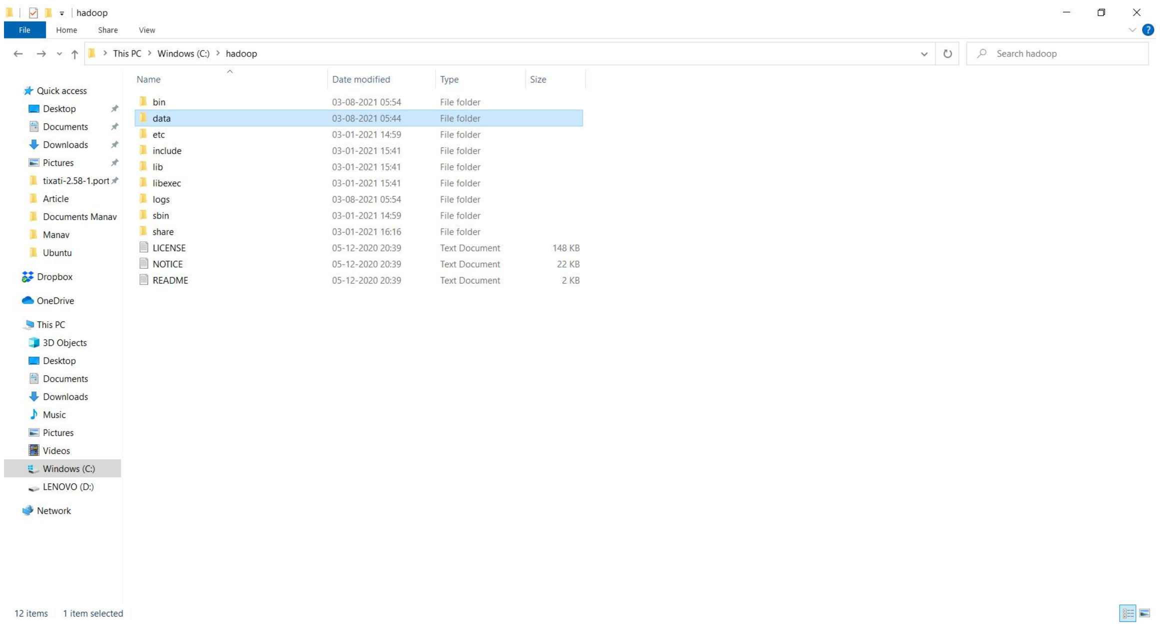
Variable	Value
JAVA_HOME	C:\Java\jdk1.8.0_271\bin
LM_LICENSE_FILE	C:\OrCAD\OrCAD_10.5\IntelliCAD 4\\cadopia.dat
NVM_HOME	C:\Users\Manav\AppData\Roaming\nvm
NVM_SYMLINK	C:\Program Files\nodejs
OneDrive	C:\Users\Manav\OneDrive
OneDriveConsumer	C:\Users\Manav\OneDrive
Path	C:\Users\Manav\AppData\Local\Programs\Python\Python37-...

System variables

Variable	Value
NVM_HOME	C:\Users\Manav\AppData\Roaming\nvm
NVM_SYMLINK	C:\Program Files\nodejs
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 142 Stepping 10, GenuineIntel

4) Now, we have to set the variable and path for the installed JDK.

5) Now, we have to proceed and download the binary files for Hadoop.



This PC > Windows (C:) > hadoop

Name	Date modified	Type	Size
bin	03-08-2021 05:54	File folder	
data	03-08-2021 05:44	File folder	
etc	03-01-2021 14:59	File folder	
include	03-01-2021 15:41	File folder	
lib	03-01-2021 15:41	File folder	
libexec	03-01-2021 15:41	File folder	
logs	03-08-2021 05:54	File folder	
sbin	03-01-2021 14:59	File folder	
share	03-01-2021 16:16	File folder	
LICENSE	05-12-2020 20:39	Text Document	148 KB
NOTICE	05-12-2020 20:39	Text Document	22 KB
README	05-12-2020 20:39	Text Document	2 KB

6) After downloading the files, we make some changes to a few .xml files using notepad.

core-site.xml

```
<configuration>
 <property>
 <name>fs.default.name</name>
 <value>hdfs://localhost:9000</value>
 </property>
</configuration>
```

hdfs-site.xml

```
<configuration>
 <property>
<name>dfs.replication</name>
 <value>1</value>
 </property>
 <property>
 <name>dfs.namenode.name.dir</name>
 <value>D:\hadoop\data</value>
 </property>
 <property>
 <name>dfs.datanode.data.dir</name>
 <value>D:\hadoop\data\namenode</value>
 </property>
</configuration>
```

mapred-site.xml

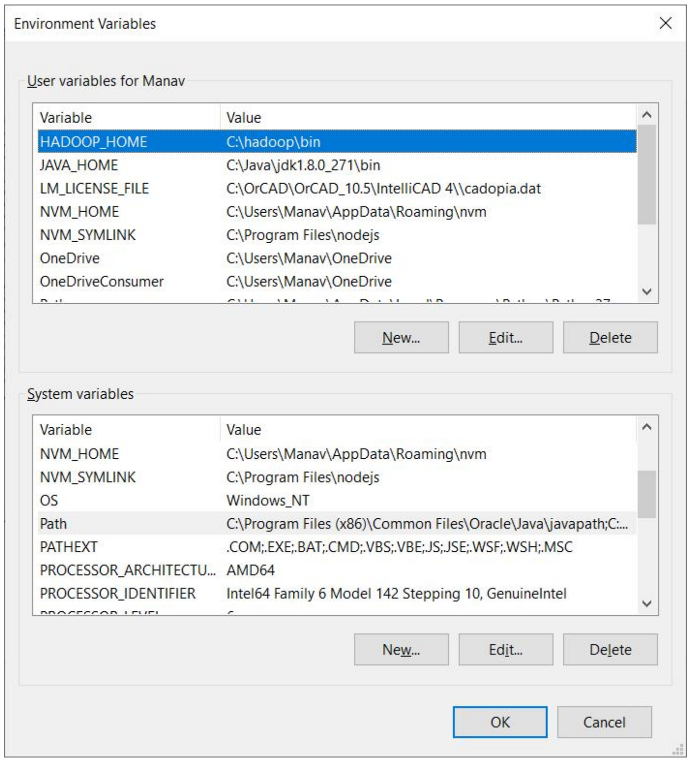
```
<configuration>
 <property>
 <name>mapreduce.framework.name</name>
 <value>yarn</value>
 </property>
</configuration>
```

yarn-site.xml

```
<configuration>
```

```
 <property>
 <name>yarn.nodemanager.aux-services</name>
 <value>mapreduce_shuffle</value>
 </property>
 <property>
 <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>
</configuration>
```

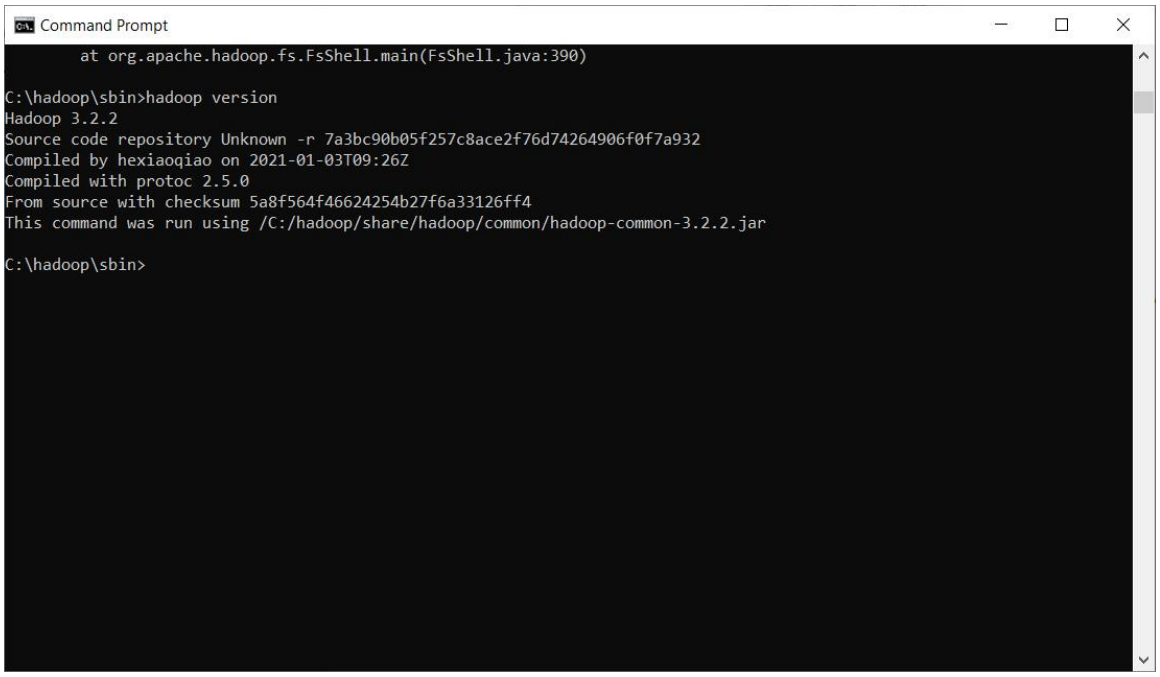
7) We will now configure the variables and path for the installed hadoop.



Variable	Value
HADOOP_HOME	C:\hadoop\bin
JAVA_HOME	C:\Java\jdk1.8.0_271\bin
LM_LICENSE_FILE	C:\OrCAD\OrCAD_10.5\IntelliCAD 4\\cadopia.dat
NVM_HOME	C:\Users\Manav\AppData\Roaming\nvm
NVM_SYMLINK	C:\Program Files\nodejs
OneDrive	C:\Users\Manav\OneDrive
OneDriveConsumer	C:\Users\Manav\OneDrive

Variable	Value
NVM_HOME	C:\Users\Manav\AppData\Roaming\nvm
NVM_SYMLINK	C:\Program Files\nodejs
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTU...	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 142 Stepping 10, GenuineIntel

8) After this, we will run the command prompt and check for the installed hadoop version.


```
        at org.apache.hadoop.fs.FsShell.main(FsShell.java:390)

C:\hadoop\sbin>hadoop version
Hadoop 3.2.2
Source code repository Unknown -r 7a3bc90b05f257c8ace2f76d74264906f0f7a932
Compiled by hexiaoqiao on 2021-01-03T09:26Z
Compiled with protoc 2.5.0
From source with checksum 5a8f564f46624254b27f6a33126ff4
This command was run using /C:/hadoop/share/hadoop/common/hadoop-common-3.2.2.jar

C:\hadoop\sbin>
```

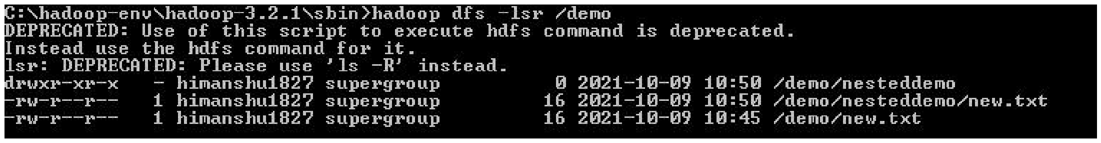
Experiment - 2

Aim : Explain some basic commands of Hadoop

- **-mkdir**: It is used to create a directory.

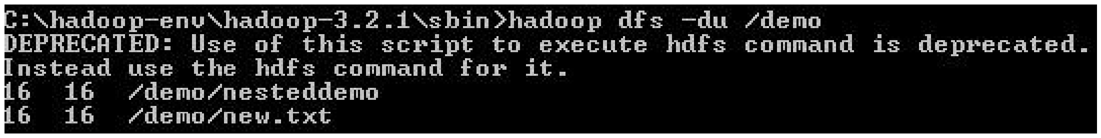
- **-ls:** This command is used to list all the files. It is useful when we want a hierarchy of a folder.

- -**lsr:** It recursively displays the directories, sub directories and files in the specified directory.



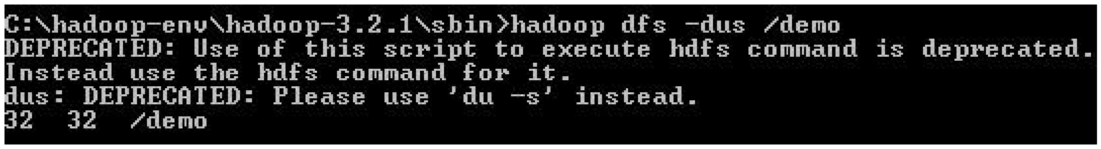
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -lsr /demo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
lsr: DEPRECATED: Please use 'ls -R' instead.
drwxr-xr-x   - himanshu1827 supergroup          0 2021-10-09 10:50 /demo/nesteddemo
-rw-r--r--   1 himanshu1827 supergroup         16 2021-10-09 10:50 /demo/nesteddemo/new.txt
-rw-r--r--   1 himanshu1827 supergroup         16 2021-10-09 10:45 /demo/new.txt
```

- **-du:** It displays aggregate length of files contained in the directory or the length of a file in case it’s just a file.



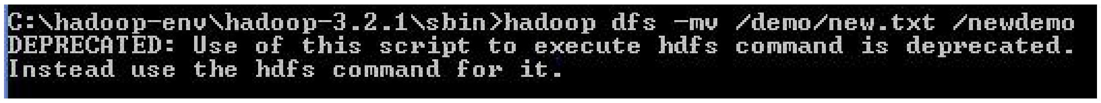
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -du /demo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
16  16  /demo/nesteddemo
16  16  /demo/new.txt
```

- **-dus:** It prints the summary of file lengths



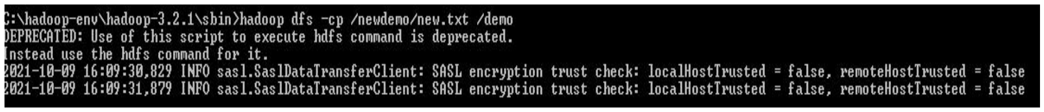
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -dus /demo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
dus: DEPRECATED: Please use 'du -s' instead.
32  32  /demo
```

- **-mv:** It moves the files from source hdfs to destination hdfs. Hadoop mv command can also be used to move multiple source files into the target directory. In this case the target should be a directory.



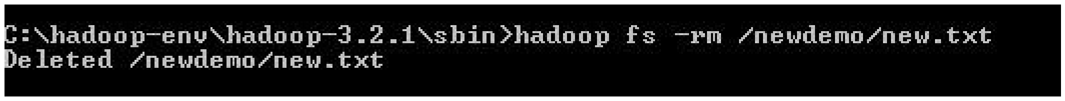
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -mv /demo/new.txt /newdemo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
```

- **-cp:** This command is used for copying the source into the target. The cp command can also be used to copy multiple files into the target. In this case the target should be a directory.



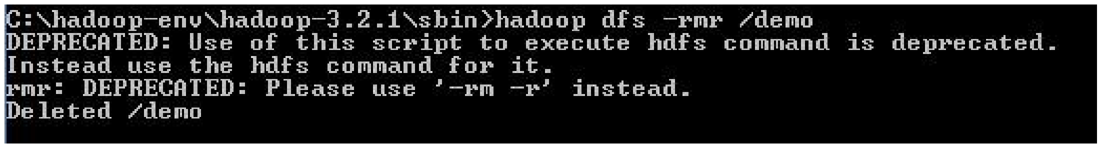
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -cp /newdemo/new.txt /demo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
2021-10-09 16:09:30,829 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
2021-10-09 16:09:31,879 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

- **-rm:** It removes the specified list of files and empty directories.



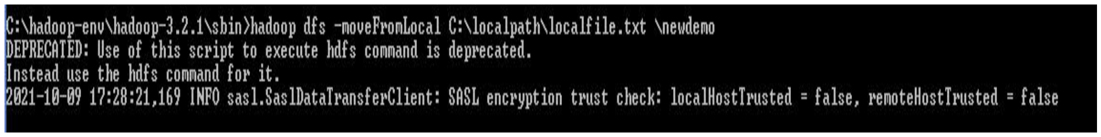
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop fs -rm /newdemo/new.txt
Deleted /newdemo/new.txt
```

- **-rmr:** It recursively deletes the files and sub directories.



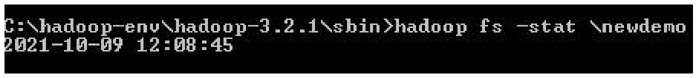
```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -rmr /demo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
rmr: DEPRECATED: Please use '-rm -r' instead.
Deleted /demo
```

- **-moveFromLocal:** It moves a file from local file system to the hdfs directory. It removes the original source file.



```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop dfs -moveFromLocal C:\localpath\localfile.txt \newdemo
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
2021-10-09 17:28:21,169 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

- **-stat:** It returns the stats information on a path.



```
C:\hadoop-env\hadoop-3.2.1\sbin>hadoop fs -stat \newdemo
2021-10-09 12:08:45
```

- **-touchz**: It creates an empty file.

- **-lsr:** It recursively displays the directories, sub directories and files in the specified directory.
- **-copyFromLocal :** To copy files/folders from local file system to hdfs store.
- **cat:** To print file contents
- **copyToLocal (or) get:** To copy files/folders from hdfs store to local file system.

EXPERIMENT-3

Aim: Write a java program to count occurrence of each word in a file using Split function and hash map.

Theory:

MapReduce is a Hadoop framework used for writing applications that can process vast amounts of data on large clusters. It can also be called a programming model in which we can process large datasets across computer clusters. This application allows data to be stored in a distributed form. It simplifies enormous volumes of data and large scale computing.

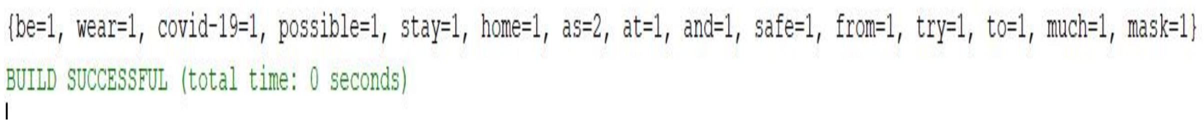
There are two primary tasks in MapReduce: map and reduce. We perform the former task before the latter. In the map job, we split the input dataset into chunks. Map task processes these chunks in parallel. The map we use outputs as inputs for the reduce tasks. Reducers process the intermediate data from the maps into smaller tuples, that reduces the tasks, leading to the final output of the framework.

The MapReduce framework enhances the scheduling and monitoring of tasks. The failed tasks are re-executed by the framework. This framework can be used easily, even by programmers with little expertise in distributed processing. MapReduce can be implemented using various programming languages such as Java, Hive, Pig, Scala, and Python.

Code:

```
package wordcount;

import java.util.HashMap;

import java.util.Map;

import java.io.BufferedReader;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.Map;

import java.io.FileReader;

public class WordCount {

    public static void main(String[] args) {

        String content="",line;

        try
        {
        FileReader file=new FileReader("D:/localpath/new.txt");
        BufferedReader br=new BufferedReader(file);
        while((line=br.readLine())!=null)
        {
          content=content+line;
        }
        Map<String,Integer> words=new HashMap<String,Integer>();
        for(String word: content.split(" "))
        {
          if(words.containsKey(word))
          {
            words.put(word,words.get(word)+1);
          }
          else
          {
            words.put(word,1);
          }
        }
        System.out.println(words);
    }
  catch(Exception e)
  {
        System.out.println(e);

    }}}
```

Output:



```
{be=1, wear=1, covid-19=1, possible=1, stay=1, home=1, as=2, at=1, and=1, safe=1, from=1, try=1, to=1, much=1, mask=1}
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXPERIMENT-4

Aim: Write a MapReduce script to count the occurrence of each word in a file.

Theory:
MapReduce is a Hadoop framework used for writing applications that can process vast amounts of data on large clusters. It can also be called a programming model in which we can process large datasets across computer clusters. This application allows data to be stored in a distributed form. It simplifies enormous volumes of data and large scale computing.

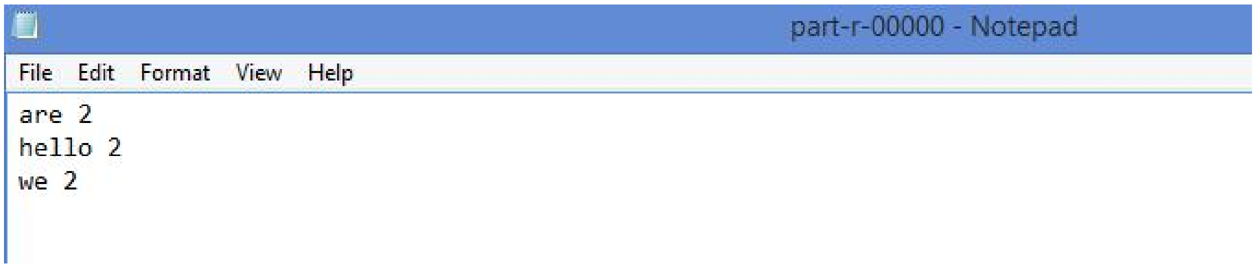
There are two primary tasks in MapReduce: map and reduce. We perform the former task before the latter. In the map job, we split the input dataset into chunks. Map task processes these chunks in parallel. The map we use outputs as inputs for the reduce tasks. Reducers process the intermediate data from the maps into smaller tuples, that reduces the tasks, leading to the final output of the framework.

The MapReduce framework enhances the scheduling and monitoring of tasks. The failed tasks are re-executed by the framework. This framework can be used easily, even by programmers with little expertise in distributed processing. MapReduce can be implemented using various programming languages such as Java, Hive, Pig, Scala, and Python.

Code:

```
import java.io.IOException;

import java.util.StringTokenizer;

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 WordCount {

public static class TokenizerMapper

 extends Mapper<Object, Text, Text, IntWritable>{

 private final static IntWritable one = new IntWritable(1);

 private Text word = new Text();

 public void map(Object key, Text value, Context context
 ) throws IOException, InterruptedException {
 StringTokenizer itr = new StringTokenizer(value.toString());
 while (itr.hasMoreTokens()) {
 word.set(itr.nextToken());
 context.write(word, one);
 }
 }
 }
 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, "word count");

 job.setJarByClass(WordCount.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); }}
```

Output:



```
are 2
hello 2
we 2
```

EXPERIMENT-5

Aim: Write a MapReduce script to find the maximum and minimum temperature from a record set stored in a text file.

Theory:

MapReduce is a Hadoop framework used for writing applications that can process vast amounts of data on large clusters. It can also be called a programming model in which we can process large datasets across computer clusters. This application allows data to be stored in a distributed form. It simplifies enormous volumes of data and large scale computing.

There are two primary tasks in MapReduce: map and reduce. We perform the former task before the latter. In the map job, we split the input dataset into chunks. Map task processes these chunks in parallel. The map we use outputs as inputs for the reduce tasks. Reducers process the intermediate data from the maps into smaller tuples, that reduces the tasks, leading to the final output of the framework.

The MapReduce framework enhances the scheduling and monitoring of tasks. The failed tasks are re-executed by the framework. This framework can be used easily, even by programmers with little expertise in distributed processing. MapReduce can be implemented using various programming languages such as Java, Hive, Pig, Scala, and Python.

Steps:

- Write CalculateMaxAndMinTemperatureWithTime.java with tokeniser mapper and other functions
- Go to Run Configurations and pass input file path and output directory path.
- Execute the code

Program:

```
import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class CalculateMaxAndMinTemperatureWithTime {

public static String calOutputName = "California";

public static String nyOutputName = "Newyork";

public static String njOutputName = "Newjersy";

public static String ausOutputName = "Austin";

public static String bosOutputName = "Boston";

public static String balOutputName = "Baltimore";



public static class WhetherForcastMapper extends

  Mapper<Object, Text, Text, Text> {



 public void map(Object keyOffset, Text dayReport, Context con)

   throws IOException, InterruptedException {
```

```
StringTokenizer strTokens = new StringTokenizer(
  dayReport.toString(), "\t");
int counter = 0;
Float currnetTemp = null;
Float minTemp = Float.MAX_VALUE;
Float maxTemp = Float.MIN_VALUE;
String date = null;
String currentTime = null;
String minTempANDTime = null;
String maxTempANDTime = null;
while (strTokens.hasMoreElements()) {
 if (counter == 0) {
  date = strTokens.nextToken();
 } else {
  if (counter % 2 == 1) {
   currentTime = strTokens.nextToken();
  } else {
   currnetTemp = Float.parseFloat(strTokens.nextToken());
   if (minTemp > currnetTemp) {
    minTemp = currnetTemp;
    minTempANDTime = minTemp + "AND" + currentTime;
   }
   if (maxTemp < currnetTemp) {
    maxTemp = currnetTemp;
    maxTempANDTime = maxTemp + "AND" + currentTime;
   }
  }
 }
 counter++;
}
Text temp = new Text();
temp.set(maxTempANDTime);
Text dateText = new Text();
dateText.set(date);
try {
 con.write(dateText, temp);
```

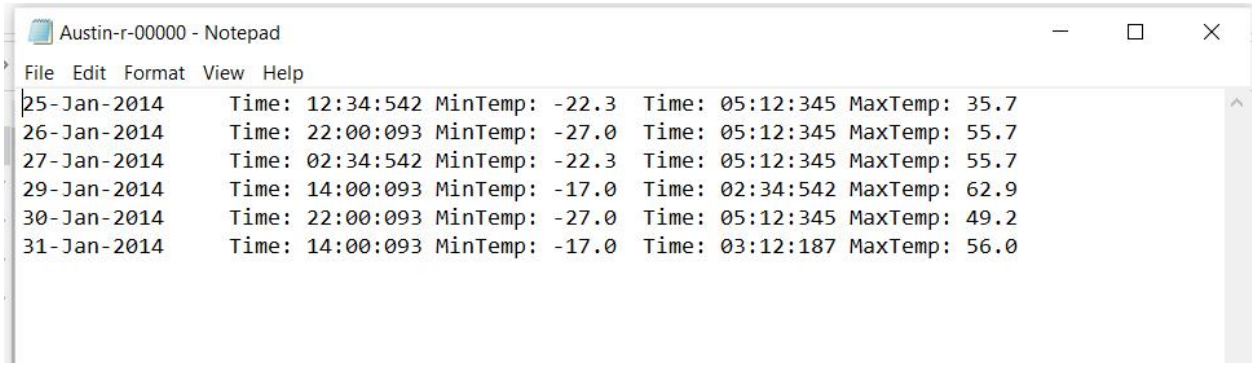
```
  } catch (Exception e) {

   e.printStackTrace();

  }

  temp.set(minTempANDTime);

  dateText.set(date);

  con.write(dateText, temp);



 }

}



public static class WhetherForcastReducer extends

  Reducer<Text, Text, Text, Text> {

 MultipleOutputs<Text, Text> mos;

 public void setup(Context context) {

  mos = new MultipleOutputs<Text, Text>(context);

 }

 public void reduce(Text key, Iterable<Text> values, Context context)

   throws IOException, InterruptedException {
```

```
int counter = 0;

String reducerInputStr[] = null;

String f1Time = "";

String f2Time = "";

String f1 = "", f2 = "";

Text result = new Text();

for (Text value : values) {

 if (counter == 0) {

  reducerInputStr = value.toString().split("AND");

  f1 = reducerInputStr[0];

  f1Time = reducerInputStr[1];

 }

 else {

  reducerInputStr = value.toString().split("AND");

  f2 = reducerInputStr[0];

  f2Time = reducerInputStr[1];

 }

 counter = counter + 1;
```

```
}

if (Float.parseFloat(f1) > Float.parseFloat(f2)) {

 result = new Text("Time: " + f2Time + " MinTemp: " + f2 + "\t"

   + "Time: " + f1Time + " MaxTemp: " + f1);

} else {

 result = new Text("Time: " + f1Time + " MinTemp: " + f1 + "\t"

   + "Time: " + f2Time + " MaxTemp: " + f2);

}

String fileName = "";

if (key.toString().substring(0, 2).equals("CA")) {

 fileName = CalculateMaxAndMinTemperatureWithTime.calOutputName;

} else if (key.toString().substring(0, 2).equals("NY")) {

 fileName = CalculateMaxAndMinTemperatureWithTime.nyOutputName;

} else if (key.toString().substring(0, 2).equals("NJ")) {

 fileName = CalculateMaxAndMinTemperatureWithTime.njOutputName;

} else if (key.toString().substring(0, 3).equals("AUS")) {

 fileName = CalculateMaxAndMinTemperatureWithTime.ausOutputName;

} else if (key.toString().substring(0, 3).equals("BOS")) {
   fileName = CalculateMaxAndMinTemperatureWithTime.bosOutputName;
  } else if (key.toString().substring(0, 3).equals("BAL")) {
   fileName = CalculateMaxAndMinTemperatureWithTime.balOutputName;
  }
  String strArr[] = key.toString().split("_");
  key.set(strArr[1]);
  mos.write(fileName, key, result);
 }
 @Override
 public void cleanup(Context context) throws IOException,
   InterruptedException {
  mos.close();
 }
}
public static void main(String[] args) throws IOException,
  ClassNotFoundException, InterruptedException {
 Configuration conf = new Configuration();
 Job job = Job.getInstance(conf, "Wheather Statistics of USA");
```

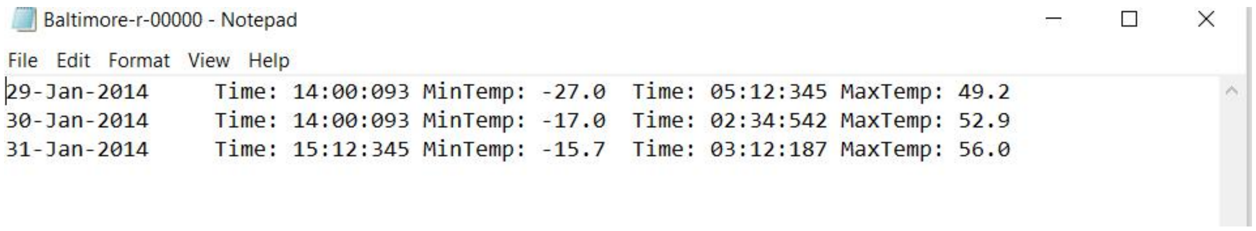
```
job.setJarByClass(CalculateMaxAndMinTemperatureWithTime.class);

job.setMapperClass(WhetherForcastMapper.class);

job.setReducerClass(WhetherForcastReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

MultipleOutputs.addNamedOutput(job, calOutputName,

  TextOutputFormat.class, Text.class, Text.class);

MultipleOutputs.addNamedOutput(job, nyOutputName,

  TextOutputFormat.class, Text.class, Text.class);

MultipleOutputs.addNamedOutput(job, njOutputName,

  TextOutputFormat.class, Text.class, Text.class);

MultipleOutputs.addNamedOutput(job, bosOutputName,

  TextOutputFormat.class, Text.class, Text.class);

MultipleOutputs.addNamedOutput(job, ausOutputName,

  TextOutputFormat.class, Text.class, Text.class);

MultipleOutputs.addNamedOutput(job, balOutputName,
  TextOutputFormat.class, Text.class, Text.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

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

try {

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

} catch (Exception e) {

   e.printStackTrace();

}

}

}
```

Output:



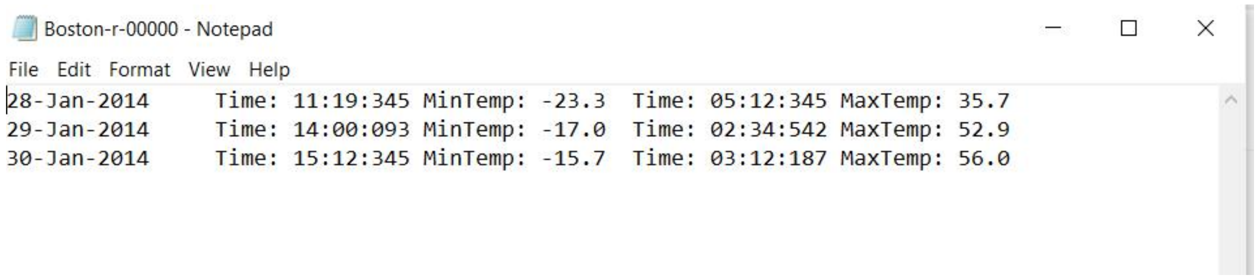
Austin-r-00000 - Notepad

```
25-Jan-2014     Time: 12:34:542 MinTemp: -22.3  Time: 05:12:345 MaxTemp: 35.7
26-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 55.7
27-Jan-2014     Time: 02:34:542 MinTemp: -22.3  Time: 05:12:345 MaxTemp: 55.7
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 62.9
30-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
31-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 03:12:187 MaxTemp: 56.0
```


Baltimore-r-00000 - Notepad

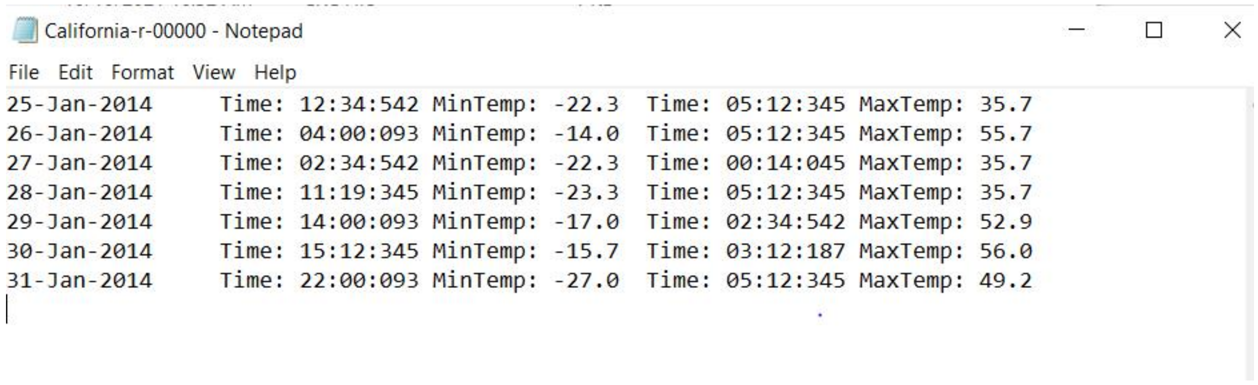
File Edit Format View Help

```
29-Jan-2014      Time: 14:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
30-Jan-2014      Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
31-Jan-2014      Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
```


Boston-r-00000 - Notepad

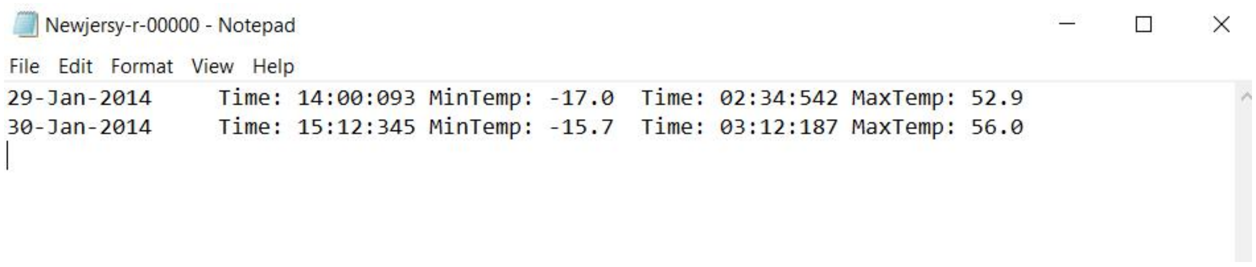
File Edit Format View Help

```
28-Jan-2014      Time: 11:19:345 MinTemp: -23.3  Time: 05:12:345 MaxTemp: 35.7
29-Jan-2014      Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014      Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
```


California-r-00000 - Notepad

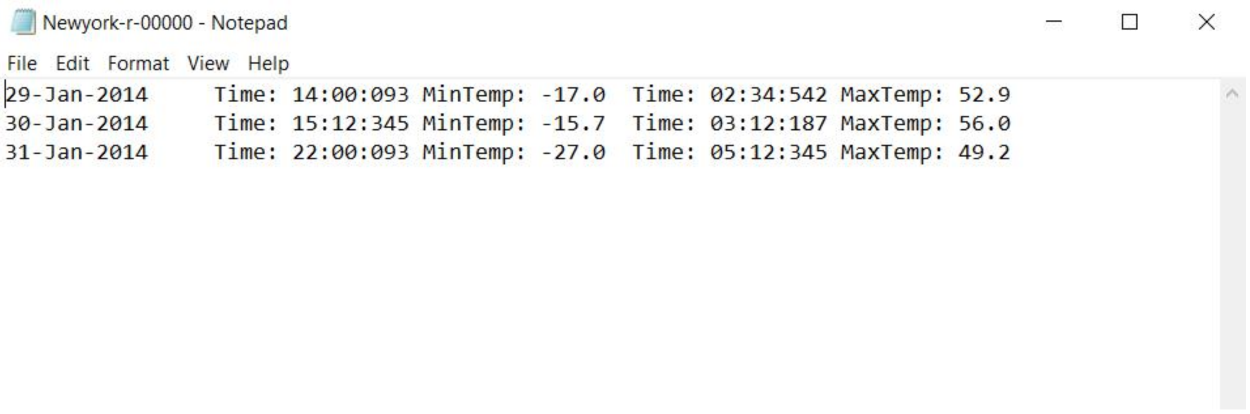
File Edit Format View Help

```
25-Jan-2014      Time: 12:34:542 MinTemp: -22.3  Time: 05:12:345 MaxTemp: 35.7
26-Jan-2014      Time: 04:00:093 MinTemp: -14.0  Time: 05:12:345 MaxTemp: 55.7
27-Jan-2014      Time: 02:34:542 MinTemp: -22.3  Time: 00:14:045 MaxTemp: 35.7
28-Jan-2014      Time: 11:19:345 MinTemp: -23.3  Time: 05:12:345 MaxTemp: 35.7
29-Jan-2014      Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014      Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
31-Jan-2014      Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
```


Newjersy-r-00000 - Notepad

File Edit Format View Help

```
29-Jan-2014      Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014      Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
```



```
29-Jan-2014     Time: 14:00:093 MinTemp: -17.0  Time: 02:34:542 MaxTemp: 52.9
30-Jan-2014     Time: 15:12:345 MinTemp: -15.7  Time: 03:12:187 MaxTemp: 56.0
31-Jan-2014     Time: 22:00:093 MinTemp: -27.0  Time: 05:12:345 MaxTemp: 49.2
```

EXPERIMENT-6

Aim: Write a MapReduce script to implement map side and reduce side joins.

Theory:
Just like SQL join, we can also perform join operations in MapReduce on different data sets. There are two types of join operations in MapReduce:

Map Side Join: As the name implies, the join operation is performed in the map phase itself. Therefore, in the map side join, the mapper performs the join and it is mandatory that the input to each map is partitioned and sorted according to the keys.

Reduce Side Join: As the name suggests, in the reduce side join, the reducer is responsible for performing the join operation. It is comparatively simple and easier to implement than the map side join as the sorting and shuffling phase sends the values having identical keys to the same reducer and therefore, by default, the data is organized for us.

Code:

DeptNameMapper.java

```
import java.io.IOException;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapred.*;

public class DeptNameMapper extends MapReduceBase implements Mapper<LongWritable,
Text, TextPair, Text>  {

        @Override

        public void map(LongWritable key, Text value, OutputCollector<TextPair, Text> output,
Reporter reporter)

                                throws IOException

        {

                String valueString = value.toString();

                String[] SingleNodeData = valueString.split("\t");

                output.collect(new TextPair(SingleNodeData[0], "0"), new
Text(SingleNodeData[1]));

        }
}
```

<u>DeptEmpStrengthMapper.java</u>

```
import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.*;

import org.apache.hadoop.io.IntWritable;

public class DeptEmpStrengthMapper extends MapReduceBase implements
Mapper<LongWritable, Text, TextPair, Text> {

        @Override

        public void map(LongWritable key, Text value, OutputCollector<TextPair, Text> output,
Reporter reporter)
```

```
                    throws IOException

        {



                String valueString = value.toString();

                String[] SingleNodeData = valueString.split("\t");

                output.collect(new TextPair(SingleNodeData[0], "1"), new
Text(SingleNodeData[1]));

        }

}
```

<u>JoinDriver.java</u>

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.MultipleInputs;
import org.apache.hadoop.util.*;


public class JoinDriver extends Configured implements Tool {

        public static class KeyPartitioner implements Partitioner<TextPair, Text> {
                @Override
                public void configure(JobConf job) {}

                @Override
                public int getPartition(TextPair key, Text value, int numPartitions) {
                        return (key.getFirst().hashCode() & Integer.MAX_VALUE) %
numPartitions;
```

```
				}
		}

		@Override
		public int run(String[] args) throws Exception {

				if (args.length != 3) {
						System.out.println("Usage: <Department Emp Strength input>
<Department Name input> <output>");
						return -1;
				}

				JobConf conf = new JobConf(getConf(), getClass());
				conf.setJobName("Join 'Department Emp Strength input' with 'Department Name
input'");

				Path AInputPath = new Path(args[0]);
				Path BInputPath = new Path(args[1]);
				Path outputPath = new Path(args[2]);

				MultipleInputs.addInputPath(conf, AInputPath, TextInputFormat.class,
DeptNameMapper.class);
				MultipleInputs.addInputPath(conf, BInputPath, TextInputFormat.class,
DeptEmpStrengthMapper.class);

				FileOutputFormat.setOutputPath(conf, outputPath);

				conf.setPartitionerClass(KeyPartitioner.class);
				conf.setOutputValueGroupingComparator(TextPair.FirstComparator.class);

				conf.setMapOutputKeyClass(TextPair.class);

				conf.setReducerClass(JoinReducer.class);

				conf.setOutputKeyClass(Text.class);

				JobClient.runJob(conf);

				return 0;
		}
```

```
        public static void main(String[] args) throws Exception {

                int exitCode = ToolRunner.run(new JoinDriver(), args);
                System.exit(exitCode);
        }
}
```

<u>JoinReducer.java</u>

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class JoinReducer extends MapReduceBase implements Reducer<TextPair, Text, Text,
Text> {

        @Override
        public void reduce (TextPair key, Iterator<Text> values, OutputCollector<Text, Text>
output, Reporter reporter)
                    throws IOException
        {

                Text nodeId = new Text(values.next());
                while (values.hasNext()) {
                        Text node = values.next();
                        Text outValue = new Text(nodeId.toString() + "\t\t" + node.toString());
                        output.collect(key.getFirst(), outValue);
                }
        }
}
```

<u>TextPair.java</u>

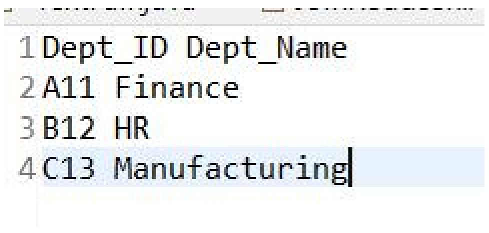
```
import java.io.*;

import org.apache.hadoop.io.*;

public class TextPair implements WritableComparable<TextPair> {
```

```
private Text first;
private Text second;

public TextPair() {
  set(new Text(), new Text());
}

public TextPair(String first, String second) {
  set(new Text(first), new Text(second));
}

public TextPair(Text first, Text second) {
  set(first, second);
}

public void set(Text first, Text second) {
  this.first = first;
  this.second = second;
}

public Text getFirst() {
  return first;
}

public Text getSecond() {
  return second;
}

@Override
public void write(DataOutput out) throws IOException {
  first.write(out);
  second.write(out);
}

@Override
public void readFields(DataInput in) throws IOException {
  first.readFields(in);
  second.readFields(in);
}
```

```
@Override
public int hashCode() {
  return first.hashCode() * 163 + second.hashCode();
}

@Override
public boolean equals(Object o) {
  if (o instanceof TextPair) {
    TextPair tp = (TextPair) o;
    return first.equals(tp.first) && second.equals(tp.second);
  }
  return false;
}

@Override
public String toString() {
  return first + "\t" + second;
}

@Override
public int compareTo(TextPair tp) {
  int cmp = first.compareTo(tp.first);
  if (cmp != 0) {
    return cmp;
  }
  return second.compareTo(tp.second);
}
// ^^ TextPair

// vv TextPairComparator
public static class Comparator extends WritableComparator {

  private static final Text.Comparator TEXT_COMPARATOR = new Text.Comparator();

  public Comparator() {
    super(TextPair.class);
  }

  @Override
  public int compare(byte[] b1, int s1, int l1,
                     byte[] b2, int s2, int l2) {

    try {
      int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);
      int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);
      int cmp = TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);
      if (cmp != 0) {
        return cmp;
      }
      return TEXT_COMPARATOR.compare(b1, s1 + firstL1, l1 - firstL1,
                                     b2, s2 + firstL2, l2 - firstL2);
    } catch (IOException e) {
      throw new IllegalArgumentException(e);
    }
  }
}

static {
  WritableComparator.define(TextPair.class, new Comparator());
}
// ^^ TextPairComparator

// vv TextPairFirstComparator
public static class FirstComparator extends WritableComparator {

  private static final Text.Comparator TEXT_COMPARATOR = new Text.Comparator();

  public FirstComparator() {
    super(TextPair.class);
  }

  @Override
  public int compare(byte[] b1, int s1, int l1,
                     byte[] b2, int s2, int l2) {

    try {
      int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);
      int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);
      return TEXT_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);
      } catch (IOException e) {
        throw new IllegalArgumentException(e);
      }
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
      if (a instanceof TextPair && b instanceof TextPair) {
        return ((TextPair) a).first.compareTo(((TextPair) b).first);
      }
      return super.compare(a, b);
    }
  }

}
```

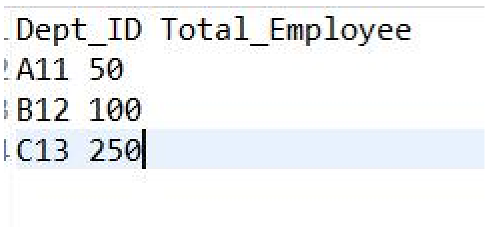
Input Files:

DeptName



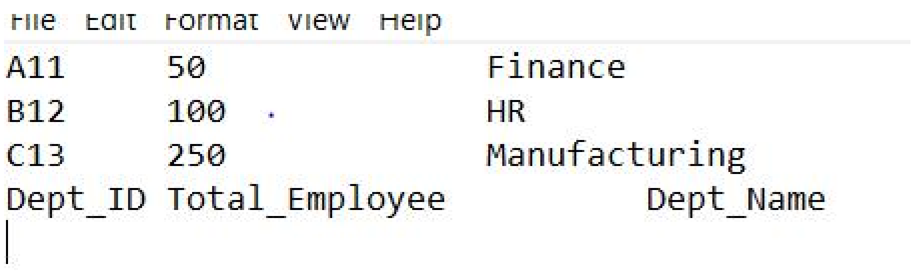
```
Dept_ID Dept_Name
A11 Finance
B12 HR
C13 Manufacturing
```

DeptStrength



```
Dept_ID Total_Employee
A11 50
B12 100
C13 250
```

Output:



```
A11     50              Finance
B12     100             HR
C13     250             Manufacturing
Dept_ID Total_Employee          Dept_Name
```

Result: MapReduce script was executed successfully.

Experiment-7

Aim: Write a pig script to analyze twitter data.

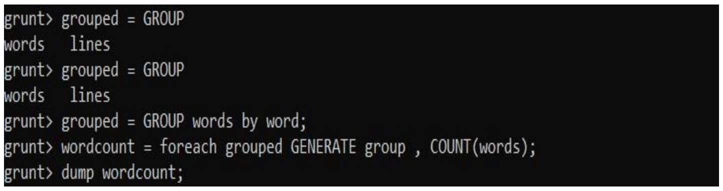
Theory:

Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Apache Pig.
To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.
To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Steps:

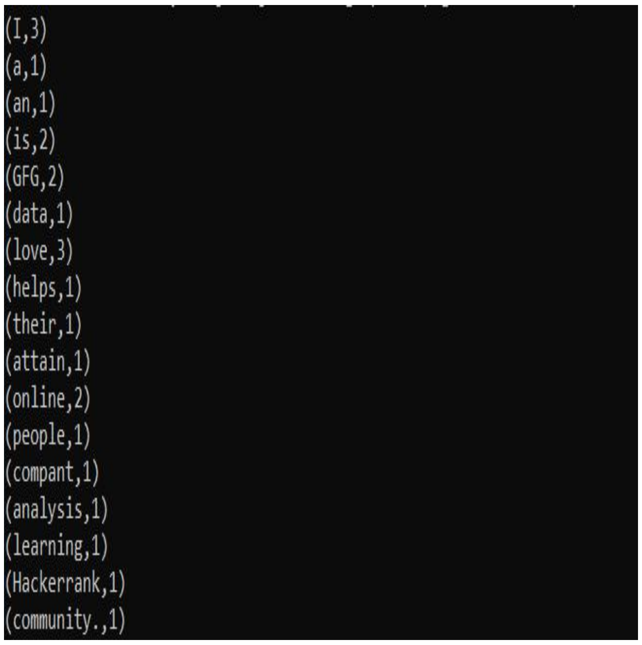
1. First of all, a text file having tweets is loaded from local system
2. Then Apache Pig loads (LOAD) the tables into Apache Pig framework.
3. Tweets are analysed as per the desired problem statements using Pig Latin commands.

Script:



```
grunt> grouped = GROUP
words   lines
grunt> grouped = GROUP
words   lines
grunt> grouped = GROUP words by word;
grunt> wordcount = foreach grouped GENERATE group , COUNT(words);
grunt> dump wordcount;
```

Output:



```
(I,3)
(a,1)
(an,1)
(is,2)
(GFG,2)
(data,1)
(love,3)
(helps,1)
(their,1)
(attain,1)
(online,2)
(people,1)
(compant,1)
(analysis,1)
(learning,1)
(Hackerrank,1)
(community.,1)
```

Experiment-8

Aim: Write a hive script to analyze last 10 years of crime data.

Steps:

1. First of all, load crime data into the HDFS.
2. Create a table using HIVE

3. Execute HIVE commands to analyze data

Theory:

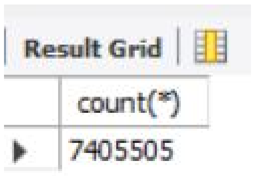
Hive is a data warehouse system which is used to analyze structured data. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

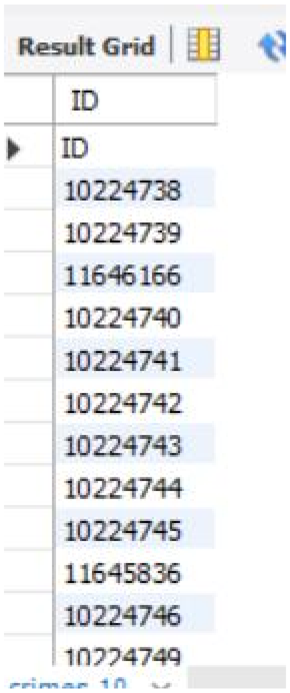
Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

Program & Output:

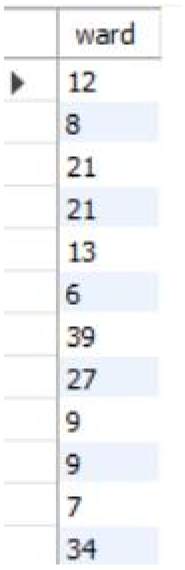
```
CREATE TABLE crime_record.crimes (

 id varchar(255),

 casenumber varchar(255),

 caldate varchar(255), block
 varchar(255),

 iucr varchar(255),

 primarytype varchar(255),

 description varchar(255),

 locationdescription varchar(255),arrest boolean,

 domestic boolean,

 beat varchar(255), district varchar(255), ward varchar(255),

 ommunityarea varchar(255),

 fbicode varchar(255),

 xcoordinate varchar(255),

 ycoordinate varchar(255), year varchar(255), updatedon varchar(255),

latitude decimal(10, 0),

 longitude decimal(10, 0),

 location varchar(255));

load data local infile 'D:/crimedata/Crime.csv' into table crime_record.crimes fields terminated
by ',';

select count(*) from crime_record.crimes;
```


```
Result Grid
  count(*)
▶ 7405505
```

select ID from crime_record.crimes where arrest=0;


```
Result Grid
  ID
▶ ID
  10224738
  10224739
  11646166
  10224740
  10224741
  10224742
  10224743
  10224744
  10224745
  11645836
  10224746
  10224749
```

select ward from crime_record.crimes where locationdescription='RESIDENCE';



ward
12
8
21
21
13
6
39
27
9
9
7
34

Result: Script was executed successfully.

Experiment-9

Aim : Write the script how can you get the structured dataset from rdbms using sqoop.

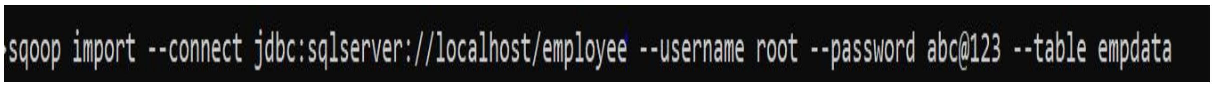
Theory:

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. It is provided by the Apache Software Foundation.

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

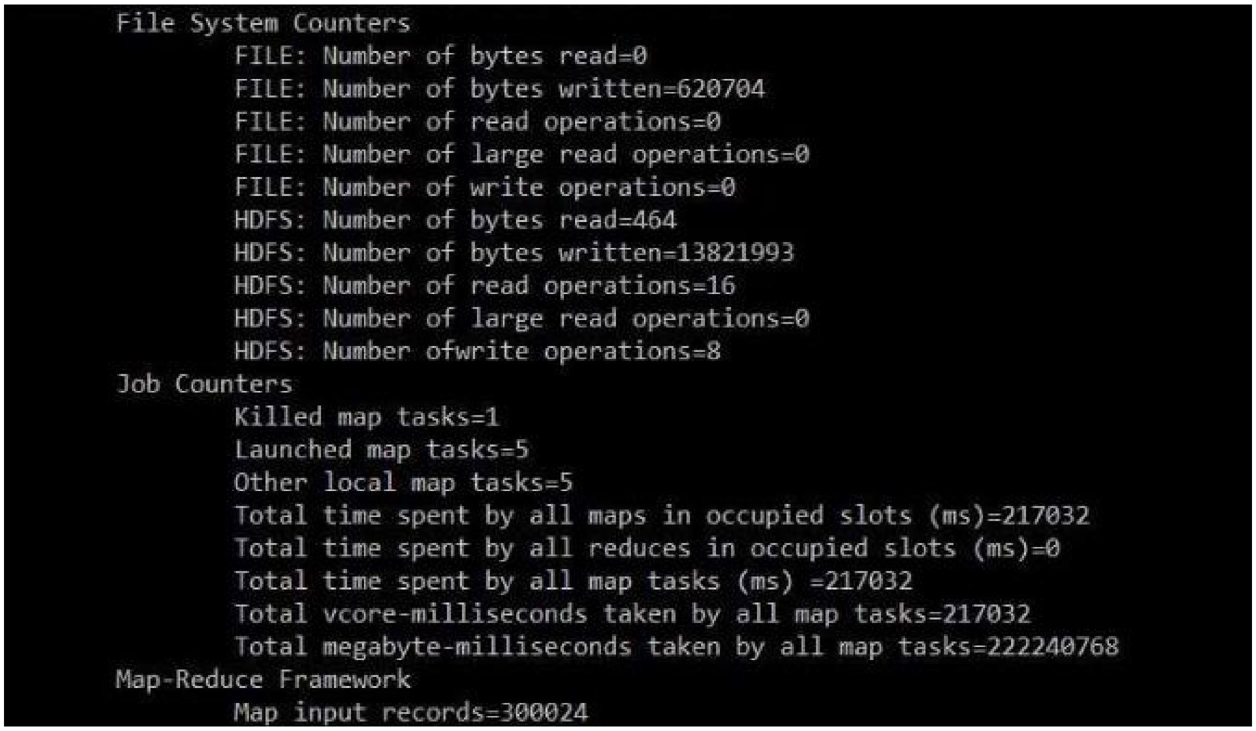
The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

Command:



```
sqoop import --connect jdbc:sqlserver://localhost/employee --username root --password abc@123 --table empdata
```

Output:



```
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=620704
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=464
                HDFS: Number of bytes written=13821993
                HDFS: Number of read operations=16
                HDFS: Number of large read operations=0
                HDFS: Number ofwrite operations=8
        Job Counters
                Killed map tasks=1
                Launched map tasks=5
                Other local map tasks=5
                Total time spent by all maps in occupied slots (ms)=217032
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms) =217032
                Total vcore-milliseconds taken by all map tasks=217032
                Total megabyte-milliseconds taken by all map tasks=222240768
        Map-Reduce Framework
                Map input records=300024
```

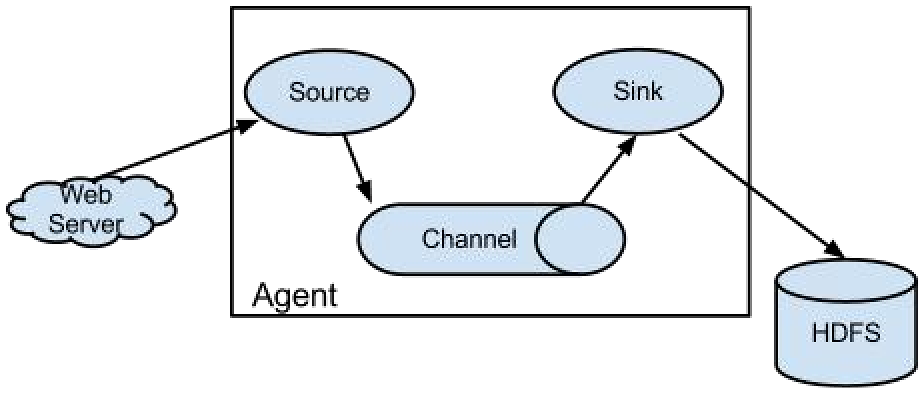
Result: Script was executed successfully.

Experiment-10

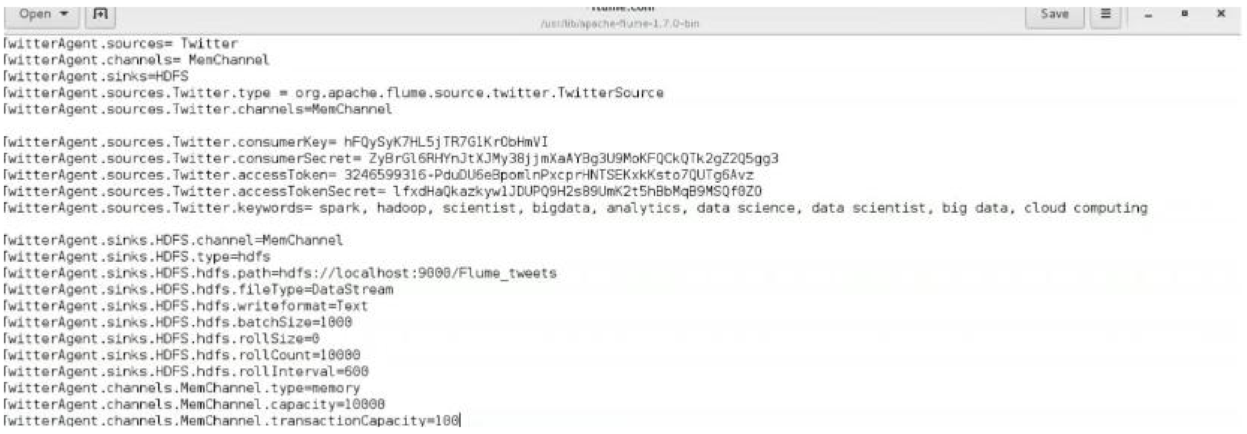
Aim: Write the script how can you get the unstructured dataset from different sources using flume.

Theory:

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.

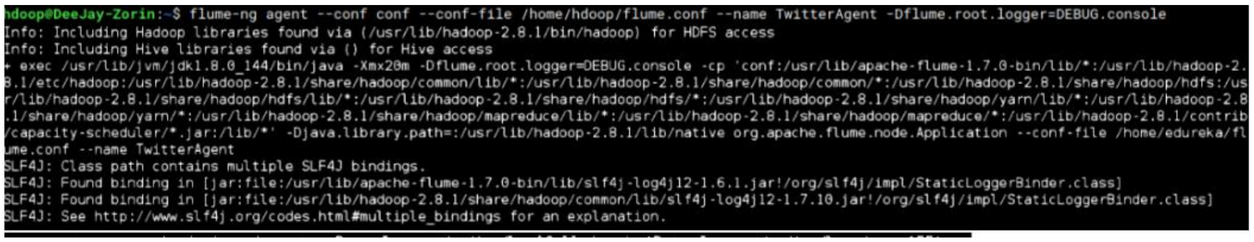


Commands:

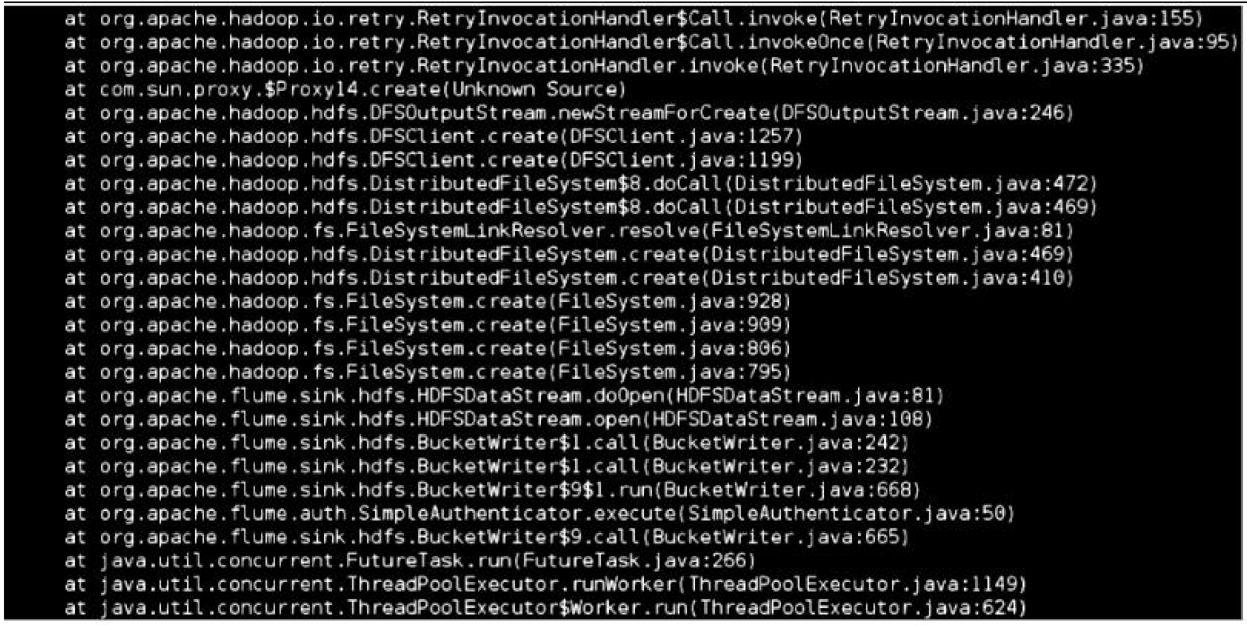


```
/usr/lib/apache-flume-1.7.0-bin
```

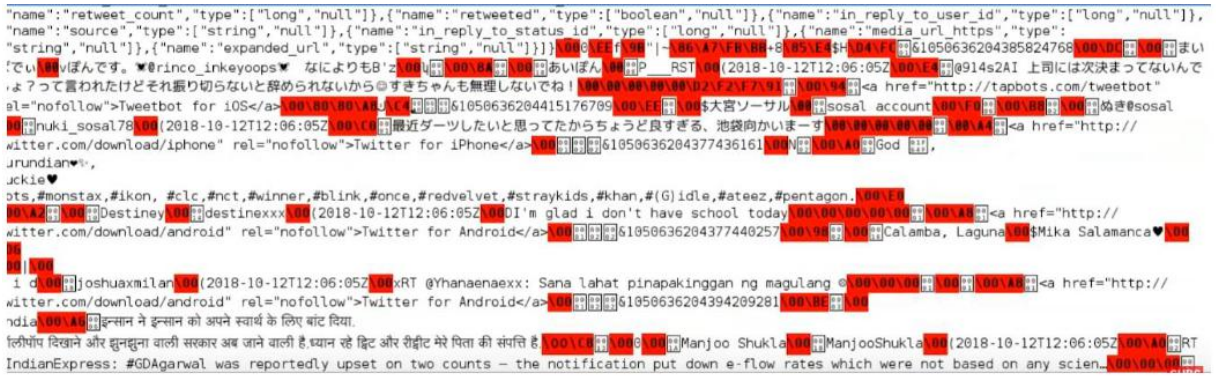
```
[witterAgent.sources= Twitter
[witterAgent.channels= MemChannel
[witterAgent.sinks=HDFS
[witterAgent.sources.Twitter.type = org.apache.flume.source.twitter.TwitterSource
[witterAgent.sources.Twitter.channels=MemChannel

[witterAgent.sources.Twitter.consumerKey= hFQySyK7HL5jTR7GlKrObHmVI
[witterAgent.sources.Twitter.consumerSecret= ZyBrGl6RHYnJtXJMy38jjmXaAYBg3U9MoKFQCkQTk2gZ2Q5gg3
[witterAgent.sources.Twitter.accessToken= 3246599316-PduDU6eBpomlnPxcprHNTSEKxkKsto7QUTg6Avz
[witterAgent.sources.Twitter.accessTokenSecret= lfxdHaQkazkyw1JDUPQ9H2s89UmK2t5hBbMqB9MSQf0ZO
[witterAgent.sources.Twitter.keywords= spark, hadoop, scientist, bigdata, analytics, data science, data scientist, big data, cloud computing

[witterAgent.sinks.HDFS.channel=MemChannel
[witterAgent.sinks.HDFS.type=hdfs
[witterAgent.sinks.HDFS.hdfs.path=hdfs://localhost:9000/Flume_tweets
[witterAgent.sinks.HDFS.hdfs.fileType=DataStream
[witterAgent.sinks.HDFS.hdfs.writeformat=Text
[witterAgent.sinks.HDFS.hdfs.batchSize=1000
[witterAgent.sinks.HDFS.hdfs.rollSize=0
[witterAgent.sinks.HDFS.hdfs.rollCount=10000
[witterAgent.sinks.HDFS.hdfs.rollInterval=600
[witterAgent.channels.MemChannel.type=memory
[witterAgent.channels.MemChannel.capacity=10000
[witterAgent.channels.MemChannel.transactionCapacity=100
```



```
hdoop@DeeJay-Zorin:~$ flume-ng agent --conf conf --conf-file /home/hdoop/flume.conf --name TwitterAgent -Dflume.root.logger=DEBUG.console
Info: Including Hadoop libraries found via (/usr/lib/hadoop-2.8.1/bin/hadoop) for HDFS access
Info: Including Hive libraries found via () for Hive access
+ exec /usr/lib/jvm/jdk1.8.0_144/bin/java -Xmx20m -Dflume.root.logger=DEBUG.console -cp 'conf:/usr/lib/apache-flume-1.7.0-bin/lib/*:/usr/lib/hadoop-2.
8.1/etc/hadoop:/usr/lib/hadoop-2.8.1/share/hadoop/common/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/common/*:/usr/lib/hadoop-2.8.1/share/hadoop/hdfs:/us
r/lib/hadoop-2.8.1/share/hadoop/hdfs/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/hdfs/*:/usr/lib/hadoop-2.8.1/share/hadoop/yarn/lib/*:/usr/lib/hadoop-2.8
.1/share/hadoop/yarn/*:/usr/lib/hadoop-2.8.1/share/hadoop/mapreduce/lib/*:/usr/lib/hadoop-2.8.1/share/hadoop/mapreduce/*:/usr/lib/hadoop-2.8.1/contrib
/capacity-scheduler/*.jar:/lib/*' -Djava.library.path=:/usr/lib/hadoop-2.8.1/lib/native org.apache.flume.node.Application --conf-file /home/edureka/fl
ume.conf --name TwitterAgent
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/apache-flume-1.7.0-bin/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/hadoop-2.8.1/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
```



```
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invoke(RetryInvocationHandler.java:155)
        at org.apache.hadoop.io.retry.RetryInvocationHandler$Call.invokeOnce(RetryInvocationHandler.java:95)
        at org.apache.hadoop.io.retry.RetryInvocationHandler.invoke(RetryInvocationHandler.java:335)
        at com.sun.proxy.$Proxy14.create(Unknown Source)
        at org.apache.hadoop.hdfs.DFSOutputStream.newStreamForCreate(DFSOutputStream.java:246)
        at org.apache.hadoop.hdfs.DFSClient.create(DFSClient.java:1257)
        at org.apache.hadoop.hdfs.DFSClient.create(DFSClient.java:1199)
        at org.apache.hadoop.hdfs.DistributedFileSystem$8.doCall(DistributedFileSystem.java:472)
        at org.apache.hadoop.hdfs.DistributedFileSystem$8.doCall(DistributedFileSystem.java:469)
        at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
        at org.apache.hadoop.hdfs.DistributedFileSystem.create(DistributedFileSystem.java:469)
        at org.apache.hadoop.hdfs.DistributedFileSystem.create(DistributedFileSystem.java:410)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:928)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:909)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:806)
        at org.apache.hadoop.fs.FileSystem.create(FileSystem.java:795)
        at org.apache.flume.sink.hdfs.HDFSDataStream.doOpen(HDFSDataStream.java:81)
        at org.apache.flume.sink.hdfs.HDFSDataStream.open(HDFSDataStream.java:108)
        at org.apache.flume.sink.hdfs.BucketWriter$1.call(BucketWriter.java:242)
        at org.apache.flume.sink.hdfs.BucketWriter$1.call(BucketWriter.java:232)
        at org.apache.flume.sink.hdfs.BucketWriter$9$1.run(BucketWriter.java:668)
        at org.apache.flume.auth.SimpleAuthenticator.execute(SimpleAuthenticator.java:50)
        at org.apache.flume.sink.hdfs.BucketWriter$9.call(BucketWriter.java:665)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
```



```
"name":"retweet_count","type":["long","null"]},{"name":"retweeted","type":["boolean","null"]},{"name":"in_reply_to_user_id","type":["long","null"]},
"name":"source","type":["string","null"]},{"name":"in_reply_to_status_id","type":["long","null"]},{"name":"media_url_https","type":
"string","null"]},{"name":"expanded_url","type":["string","null"]}]}\00@\EEf\98"|~\86\A7\FB\B8+8\85\E4$H\D4\FC□&1050636204385824768\00\DC□\00□まい
でい\00vぽんです。♥@rinco_inkeyoops♥  なによりもB'z\00Ц□\00\8A□\00□あいぽん\00□P___RST\00(2018-10-12T12:06:05Z\00\E4□@914s2AI 上司には次決まってないんで
ょ？って言われたけどそれ振り切らないと辞められないから☺すきちゃんも無理しないでね！\00\00\00\00\00\D2\F2\F7\91□\00\94□<a href="http://tapbots.com/tweetbot"
el="nofollow">Tweetbot for iOS</a>\00\80\80\A8J\C4□□□&1050636204415176709\00\EE□\00$大宮ソーサル\00□sosal account\00\F0□\00\B8□\00□ぬき@sosal
00□nuki_sosal78\00(2018-10-12T12:06:05Z\00\C0□最近ダーツしたいと思ってたからちょうど良すぎる、池袋向かいまーす\00\00\00\00\00□\00\A4□<a href="http://
witter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>\00□□□&1050636204377436161\00N□\00\A0□God □,
urundian♥,
uckie♥
ots,#monstax,#ikon, #clc,#nct,#winner,#blink,#once,#redvelvet,#straykids,#khan,#(G)idle,#ateez,#pentagon.\00\E0
00\A2□\00□Destiney\00□destinexxx\00(2018-10-12T12:06:05Z\00DI'm glad i don't have school today\00\00\00\00\00□\00\A8□<a href="http://
witter.com/download/android" rel="nofollow">Twitter for Android</a>\00□□□&1050636204377440257\00\98□\00□Calamba, Laguna\00$Mika Salamanca♥\00
06
00|\00
 i d\00□joshuaxmilan\00(2018-10-12T12:06:05Z\00xRT @Yhanaenaexx: Sana lahat pinapakinggan ng magulang ©\00\00\00□\00□\00\A8□<a href="http://
witter.com/download/android" rel="nofollow">Twitter for Android</a>\00□□□&1050636204394209281\00\BE□\00
ndia\00\A6□इन्सान ने इन्सान को अपने स्वार्थ के लिए बांट दिया.
लीपॉप दिखाने और झुनझुना वाली सरकार अब जाने वाली है.ध्यान रहे ट्विट और रीट्वीट मेरे पिता की संपत्ति है.\00\C8□\00@\00□Manjoo Shukla\00□ManjooShukla\00(2018-10-12T12:06:05Z\00\A0□RT
IndianExpress: #GDAgarwal was reportedly upset on two counts – the notification put down e-flow rates which were not based on any scien…\00\00\00□
```

Result: Script was executed successfully

Experiment-11

Aim: WAP to Create database then create a table and insert data into HBase.

Theory:

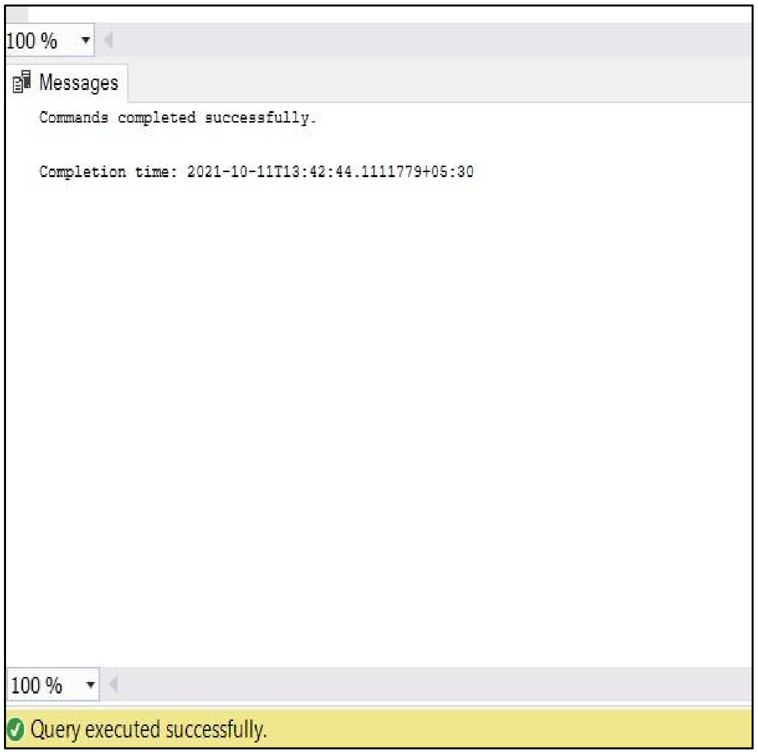
HBase is an open-source non-relational distributed database modeled after Google's Bigtable and written in Java. It is developed as part of Apache Software

Foundation's Apache Hadoop project and runs on top of HDFS or Alluxio, providing Bigtable-like capabilities for Hadoop

Code:

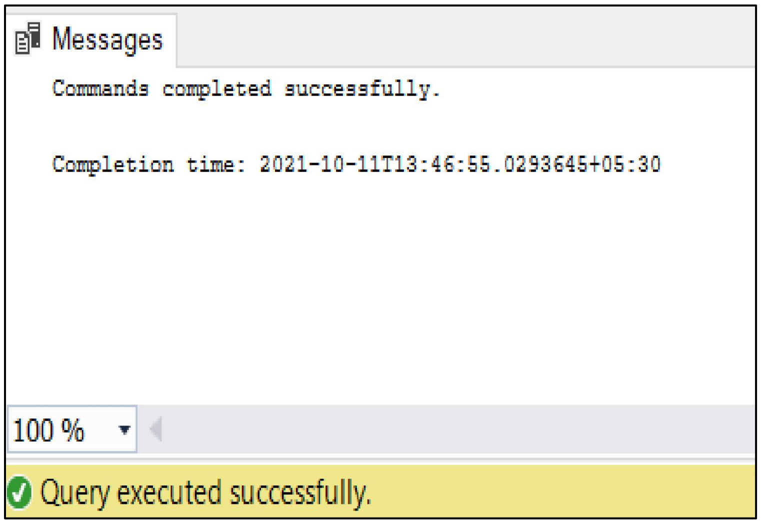
```
create database db1; create table details (

                    ID int primary key,

                    LastName varchar,

                    FirstName varchar,

                    Address varchar,

                    City varchar

);

INSERT INTO details VALUES (010, ‘H’, ‘S’, ‘2’, ‘L’);

SELECT * FROM details;
```

Output:



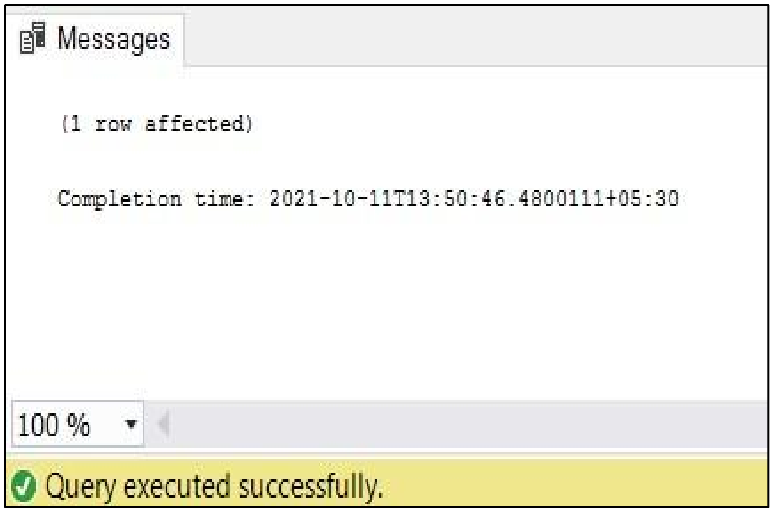
```
Commands completed successfully.

Completion time: 2021-10-11T13:42:44.1111779+05:30
```

Query executed successfully.

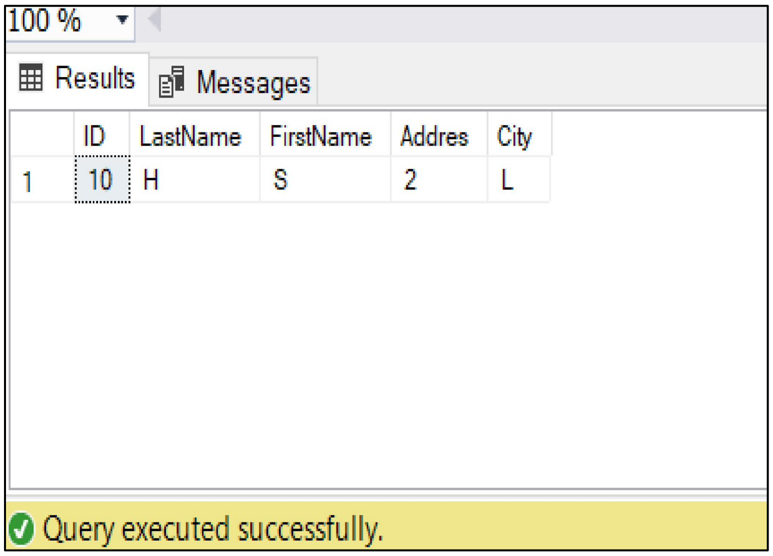


```
Commands completed successfully.

Completion time: 2021-10-11T13:46:55.0293645+05:30
```

Query executed successfully.



```
(1 row affected)

Completion time: 2021-10-11T13:50:46.4800111+05:30
```



	ID	LastName	FirstName	Addres	City
1	10	H	S	2	L

Result:

Database successfully created and data has been inserted into the table.

Experiment-12

Aim: To run an application using Oozie.

Theory:

- Apache Oozie is a workflow scheduler system to manage Apache Hadoop jobs.
- Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions.
- Oozie Coordinator jobs are recurrent Oozie Workflow jobs triggered by time (frequency) and data availability.
- Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java map-reduce, Streaming map-reduce, Pig, Hive, Sqoop and Distcp) as well as system specific jobs (such as Java programs and shell scripts).
- Oozie is a scalable, reliable and extensible system.

Source Code:

Oozie examples are bundled within the Oozie distribution in the oozie-examples.tar.gz file. Expanding this file will create an examples/ directory in the local file system.

The examples/ directory must be copied to the user HOME directory in HDFS:

```
$ hadoop fs -put examples examples
```

For the Streaming and Pig example, the Oozie Share Library must be installed in HDFS. Add Oozie bin/ to the environment PATH.

The examples assume the JobTracker is localhost:8021 and the NameNode is hdfs://localhost:8020 . If the actual values are different, the job properties files in the examples directory must be edited tothe correct values.

The example applications are under the examples/app directory, one directory per example. The directory contains the application XML file (workflow, or worklfow and coordinator), the job.properties file to submit the job and any JAR files the example may need.

The inputs for all examples are in the examples/input-data/ directory.

The examples create output under the examples/output-data/${EXAMPLE_NAME} directory.

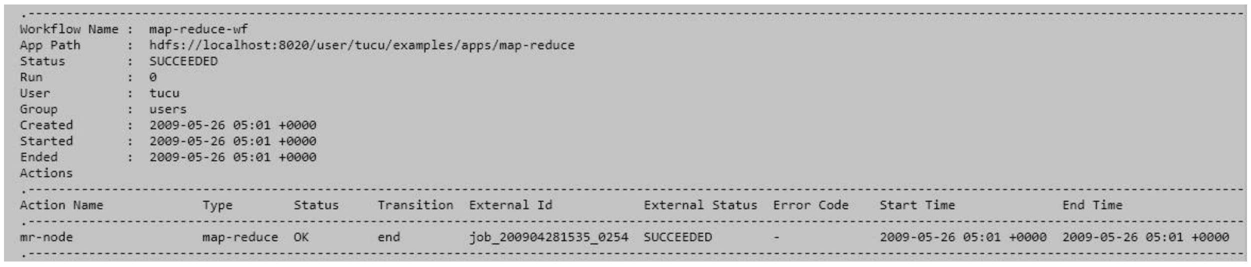
$ oozie job -oozie http://localhost:11000/oozie -config examples/apps/map-reduce/job.properties -run.

job: 14-20090525161321-oozie-tucu

Check the workflow job status:

$ oozie job -oozie http://localhost:11000/oozie -info 14-20090525161321- oozie-tucu

Output:



```
.----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Workflow Name :  map-reduce-wf
App Path      :  hdfs://localhost:8020/user/tucu/examples/apps/map-reduce
Status        :  SUCCEEDED
Run           :  0
User          :  tucu
Group         :  users
Created       :  2009-05-26 05:01 +0000
Started       :  2009-05-26 05:01 +0000
Ended         :  2009-05-26 05:01 +0000
Actions
.----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Action Name             Type        Status     Transition  External Id            External Status  Error Code    Start Time              End Time
.----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
mr-node                 map-reduce  OK         end         job_200904281535_0254  SUCCEEDED        -             2009-05-26 05:01 +0000  2009-05-26 05:01 +0000
.----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Result: Application successfully run in Oozie.