ENGINEERING COLLEGE, AJMER

NAME	: Vishvesh Parashar
COLLEGE ID	: 18CS93
BATCH	: C-4
CLASS	: 8th sem CS
SUBJECT	: BDA Lab
SUBMITTED TO	: RAVINDER SINGH SIR

ASSIGNMENT - 1

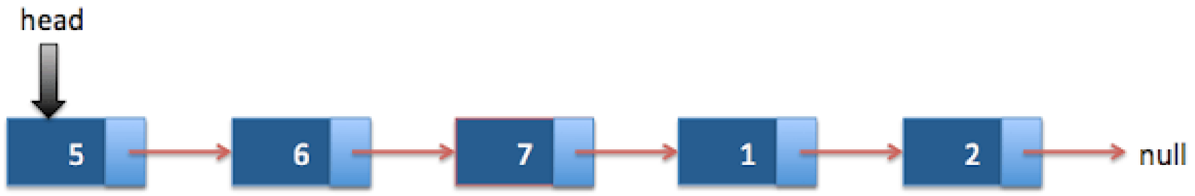
Q1. Implement the following Data structures in Java
i) Linked Lists ii) Stacks iii) Queues iv) Set v) Map

Ans.

1) LINKED LISTS :-

- **Singly LinkedList**

In singly linked list, Node has data and pointer to next node. It does not have pointer to the previous node. Last node 's next points to null, so you can iterate over linked list by using this condition.



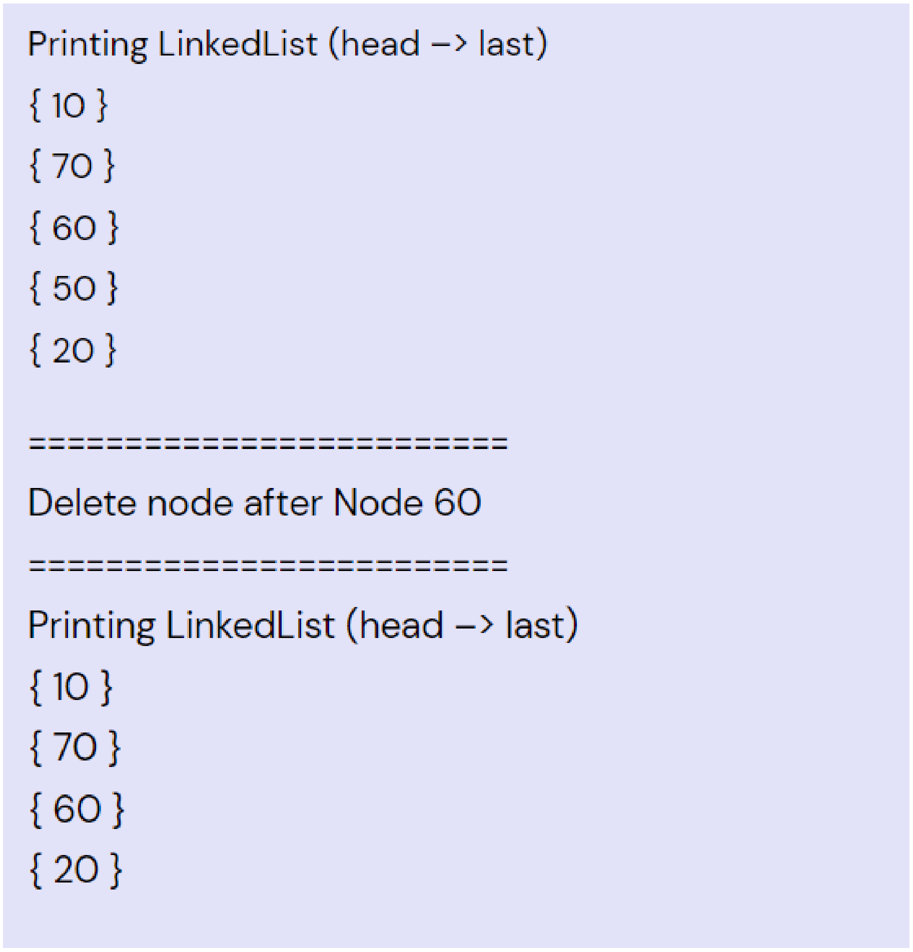
CODE -

```
package com.example.yash_java_linkedlist;

class Node {
    public int data;
    public Node next;

    public void displayNodeData() {
        System.out.println("{ " + data + " } ");
    }
}

public class MyLinkedList {
    private Node head;

    public boolean isEmpty() {
        return (head == null);
    }

    // Method for inserting node at the start of linked list
    public void insertFirst(int data) {
        Node newHead = new Node();
        newHead.data = data;
        newHead.next = head;
        head = newHead;
    }

    // Method for deleting node from start of linked list
    public Node deleteFirst() {
        Node temp = head;
        head = head.next;
        return temp;
    }

    // Method used to delete node after provided node
    public void deleteAfter(Node after) {
        Node temp = head;
        while (temp.next != null && temp.data != after.data) {
```

```
        temp = temp.next;
    }
    if (temp.next != null)
        temp.next = temp.next.next;
}

// Method used to insert at end of LinkedList
public void insertLast(int data) {
    Node current = head;
    while (current.next != null) {
        current = current.next; // we'll loop until current.next is null
    }
    Node newNode = new Node();
    newNode.data = data;
    current.next = newNode;
}

// Method for printing Linked List
public void printLinkedList() {
    System.out.println("Printing LinkedList (head --> last) ");
    Node current = head;
    while (current != null) {
        current.displayNodeData();
        current = current.next;
    }
    System.out.println();
}

public static void main(String args[])
{
    MyLinkedList myLinkedlist = new MyLinkedList();
    myLinkedlist.insertFirst(50);
    myLinkedlist.insertFirst(60);
    myLinkedlist.insertFirst(70);
    myLinkedlist.insertFirst(10);
    myLinkedlist.insertLast(20);
    myLinkedlist.printLinkedList();
    // Linked list will be
    // 10 -> 70 ->  60 -> 50 -> 20
```

```
        System.out.println("=========================");
        System.out.println("Delete node after Node 60");
        Node node=new Node();
        node.data=60;
        myLinkedlist.deleteAfter(node);
        // After deleting node after 1,Linked list will be
        // 10 -> 70 -> 60 -> 20

        System.out.println("=========================");
        myLinkedlist.printLinkedList();
    }
}
```

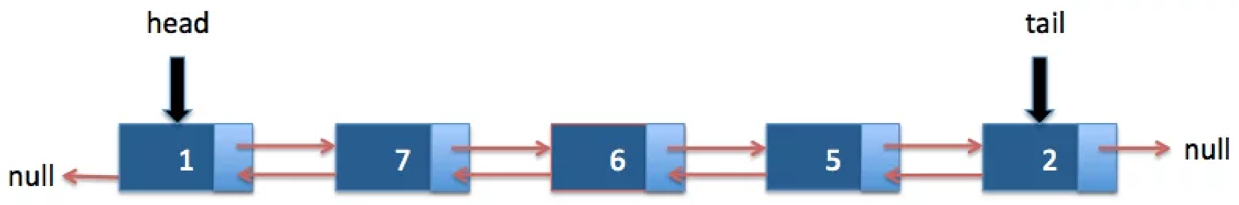
OUTPUT -



```
Printing LinkedList (head --> last)
{ 10 }
{ 70 }
{ 60 }
{ 50 }
{ 20 }

=========================
Delete node after Node 60
=========================
Printing LinkedList (head --> last)
{ 10 }
{ 70 }
{ 60 }
{ 20 }
```

- **Doubly LinkedList**

In doubly linked list, Node has data and pointers to next node and previous node.You can iterate over linkedlist either in forward or backward direction as it has pointers to prev node and next node. This is one of most used data structures in java.



Doubly Linked List

CODE -

```
package com.example.yash_java_linkedlist;

class Node {
    public int data;
    public Node next;
    public Node prev;

    public void displayNodeData() {
        System.out.println("{ " + data + " } ");
    }
}

public class MyDoublyLinkedList {

    private Node head;
    private Node tail;
    int size;

    public boolean isEmpty() {
        return (head == null);
    }

    // Method to insert a node at the start of linked list
    public void insertFirst(int data) {
        Node newNode = new Node();
        newNode.data = data;
        newNode.next = head;
        newNode.prev=null;
        if(head!=null)
            head.prev=newNode;
        head = newNode;
        if(tail==null)
            tail=newNode;
        size++;
    }

    // Method to insert a node at the start of linked list
```

```
public void insertLast(int data) {
    Node newNode = new Node();
    newNode.data = data;
    newNode.next = null;
    newNode.prev=tail;
    if(tail!=null)
        tail.next=newNode;
    tail = newNode;
    if(head==null)
        head=newNode;
    size++;
}
// used to delete node from start of Doubly linked list
public Node deleteFirst() {

    if (size == 0)
        throw new RuntimeException("Doubly linked list is already empty");
    Node temp = head;
    head = head.next;
    head.prev = null;
    size--;
    return temp;
}

// Method to delete node from last of Doubly linked list
public Node deleteLast() {

    Node temp = tail;
    tail = tail.prev;
    tail.next=null;
    size--;
    return temp;
}

// Method to delete node after particular node
public void deleteAfter(Node after) {
    Node temp = head;
    while (temp.next != null && temp.data != after.data) {
        temp = temp.next;
    }
    if (temp.next != null)
        temp.next.next.prev=temp;
    temp.next = temp.next.next;

}

// Method to print Doubly Linked List forward
public void printLinkedListForward() {
    System.out.println("Printing Doubly LinkedList (head --> tail) ");
    Node current = head;
    while (current != null) {
        current.displayNodeData();
        current = current.next;
    }
    System.out.println();
}

// Method to print Doubly Linked List forward
public void printLinkedListBackward() {
    System.out.println("Printing Doubly LinkedList (tail --> head) ");
    Node current = tail;
    while (current != null) {
        current.displayNodeData();
        current = current.prev;
    }
    System.out.println();
}

public static void main(String args[])
{
    MyDoublyLinkedList mdll = new MyDoublyLinkedList();
    mdll.insertFirst(50);
    mdll.insertFirst(60);
    mdll.insertFirst(70);
    mdll.insertFirst(10);
    mdll.insertLast(20);
    mdll.printLinkedListForward();
    mdll.printLinkedListBackward();

    System.out.println("================");
```

```
        // Doubly Linked list will be
        // 10 ->  70 -> 60 -> 50 -> 20

        Node node=new Node();
        node.data=10;
        mdll.deleteAfter(node);
        mdll.printLinkedListForward();
        mdll.printLinkedListBackward();
        // After deleting node after 1,doubly Linked list will be
        // 20 -> 10 -> 60-> 50
        System.out.println("================");
        mdll.deleteFirst();
        mdll.deleteLast();

        // After performing above operation, doubly Linked list will be
        //  60 -> 50
        mdll.printLinkedListForward();
        mdll.printLinkedListBackward();
    }
}
```

OUTPUT -

```
Printing Doubly LinkedList (head –> tail)
{ 10 }
{ 70 }
{ 60 }
{ 50 }
{ 20 }

Printing Doubly LinkedList (tail –> head)
{ 20 }
{ 50 }
{ 60 }
{ 70 }
{ 10 }

================
```

```
Printing Doubly LinkedList (head –> tail)
{ 10 }
{ 60 }
{ 50 }
{ 20 }

Printing Doubly LinkedList (tail –> head)
{ 20 }
{ 50 }
{ 60 }
{ 10 }

================
```

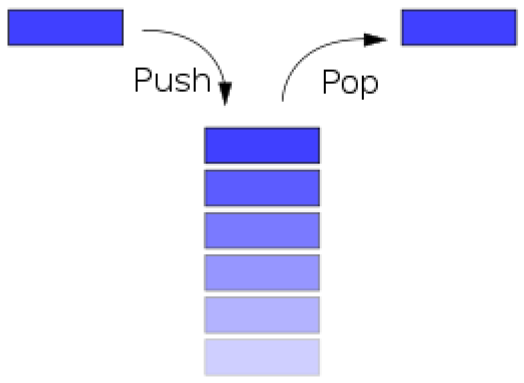
```
Printing Doubly LinkedList (head –> tail)
{ 60 }
{ 50 }

Printing Doubly LinkedList (tail –> head)
{ 50 }
{ 60 }
```

2) STACK :-

A stack is a linear data structure that follows the LIFO (Last–In, First–Out) principle. That means the objects can be inserted or removed only at one end of it, also called a top.

The stack supports the following operations:

- push inserts an item at the top of the stack (i.e., above its current top element).
- pop removes the object at the top of the stack and returns that object from the function. The stack size will be decremented by one.



- isEmpty tests if the stack is empty or not.
- isFull tests if the stack is full or not.
- peek returns the object at the top of the stack without removing it from the stack or modifying the stack in any way.
- size returns the total number of elements present in the stack.

Stack Implementation using an array

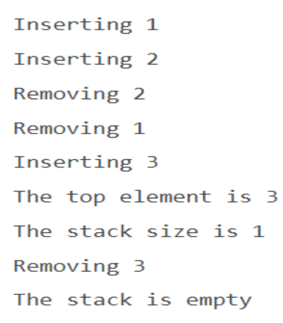
A stack can easily be implemented as an array. Following is the stack implementation in Java using an array:

CODE -

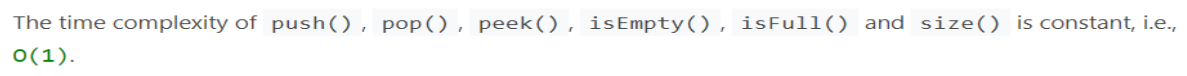
```
class Stack
{
   private int arr[];
   private int top;
   private int capacity;

   // Constructor to initialize the stack
   Stack(int size)
   {
      arr = new int[size];
      capacity = size;
      top = -1;
   }

   // Utility function to add an element `x` to the stack
   public void push(int x)
   {
      if (isFull())
      {
         System.out.println("Overflow\nProgram Terminated\n");
         System.exit(-1);
      }

      System.out.println("Inserting " + x);
      arr[++top] = x;
   }

   // Utility function to pop a top element from the stack
   public int pop()
   {
      // check for stack underflow
      if (isEmpty())
      {
         System.out.println("Underflow\nProgram Terminated");
         System.exit(-1);
      }

      System.out.println("Removing " + peek());
```

```
        // decrease stack size by 1 and (optionally) return the popped element
        return arr[top--];
    }

    // Utility function to return the top element of the stack
    public int peek()
    {
        if (!isEmpty()) {
            return arr[top];
        }
        else {
            System.exit(-1);
        }

        return -1;
    }

    // Utility function to return the size of the stack
    public int size() {
        return top + 1;
    }

    // Utility function to check if the stack is empty or not
    public boolean isEmpty() {
        return top == -1;               // or return size() == 0;
    }

    // Utility function to check if the stack is full or not
    public boolean isFull() {
        return top == capacity - 1;     // or return size() == capacity;
    }
}

class Main
{
    public static void main (String[] args)
    {
        Stack stack = new Stack(3);
        stack.push(1);      // inserting 1 in the stack
        stack.push(2);      // inserting 2 in the stack

        stack.pop();        // removing the top element (2)
        stack.pop();        // removing the top element (1)

        stack.push(3);      // inserting 3 in the stack

        System.out.println("The top element is " + stack.peek());
        System.out.println("The stack size is " + stack.size());

        stack.pop();        // removing the top element (3)

        // check if the stack is empty
        if (stack.isEmpty()) {
            System.out.println("The stack is empty");
        }
        else {
            System.out.println("The stack is not empty");
        }
    }
}
```

OUTPUT -



```
Inserting 1
Inserting 2
Removing 2
Removing 1
Inserting 3
The top element is 3
The stack size is 1
Removing 3
The stack is empty
```



The time complexity of `push()`, `pop()`, `peek()`, `isEmpty()`, `isFull()` and `size()` is constant, i.e., `O(1)`.

Using Java Collections

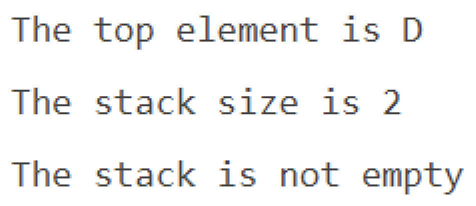
The stack is also included in Java’s collection framework.

CODE -

```
import java.util.Stack;

class Main
{
   public static void main(String[] args)
   {
      Stack<String> stack = new Stack<String>();

      stack.push("A");    // Insert `A` into the stack
      stack.push("B");    // Insert `B` into the stack
      stack.push("C");    // Insert `C` into the stack
      stack.push("D");    // Insert `D` into the stack

      // prints the top of the stack (`D`)
      System.out.println("The top element is " + stack.peek());

      stack.pop();        // removing the top element (`D`)
      stack.pop();        // removing the next top (`C)

      // returns the total number of elements present in the stack
      System.out.println("The stack size is " + stack.size());

      // check if the stack is empty
      if (stack.empty()) {
         System.out.println("The stack is empty");
      }
      else {
         System.out.println("The stack is not empty");
      }
```

```
    }
}
```

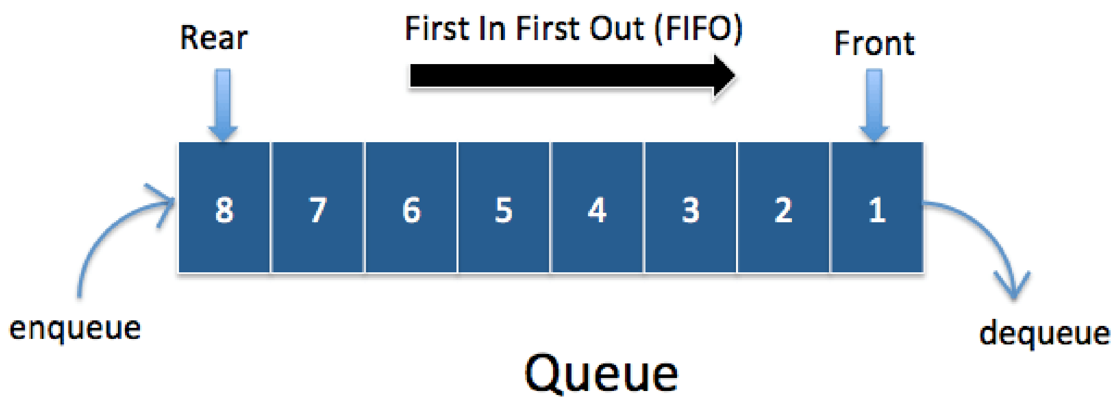
OUTPUT -



```
The top element is D
The stack size is 2
The stack is not empty
```

3) QUEUES :-

Queue is abstract data type which depicts First in first out (FIFO) behavior.



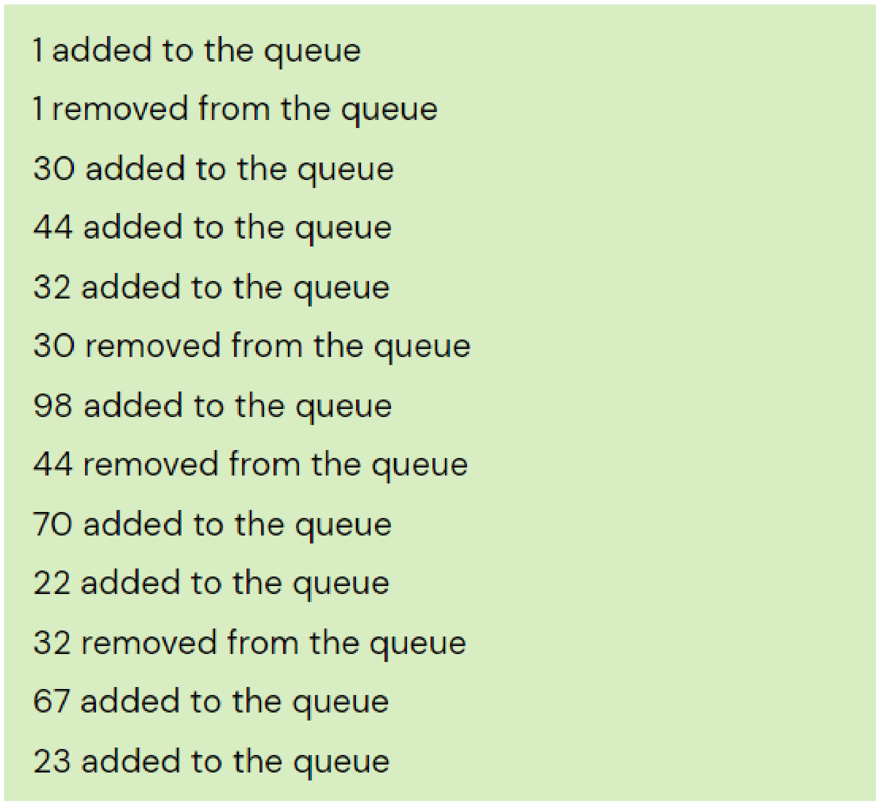
Queue implementation using array

CODE -

```
package com.example.yash_java_linkedlist;

public class MyQueue {

    private int capacity;
    int queueArr[];
    int front;
    int rear;
    int currentSize = 0;

    public MyQueue(int size) {
        this.capacity = size;
        front = 0;
        rear = -1;
    queueArr = new int[this.capacity];
}

/**
     * Method for adding element in the queue
     *
     * @param data
     */
public void enqueue(int data) {
    if (isFull()) {
        System.out.println("Queue is full!! Can not add more elements");
    } else {
        rear++;
        if (rear == capacity - 1) {
            rear = 0;
        }
        queueArr[rear] = data;
        currentSize++;
        System.out.println(data + " added to the queue");
    }
}

/**
     * This method removes an element from the front of the queue
     */
public void dequeue() {
    if (isEmpty()) {
        System.out.println("Queue is empty!! Can not dequeue element");
    } else {
        front++;
        if (front == capacity - 1) {
            System.out.println(queueArr[front - 1] + " removed from the queue");
            front = 0;
        } else {
            System.out.println(queueArr[front - 1] + " removed from the queue");
        }
        currentSize--;
    }
}

/**
     * Method for checking if Queue is full
     *
     * @return
     */
public boolean isFull() {
    if (currentSize == capacity) {
        return true;
    }
    return false;
}

/**
     * Method for checking if Queue is empty
     *
     * @return
     */
public boolean isEmpty() {

    if (currentSize == 0) {
        return true;
    }
    return false;
}

public static void main(String a[]) {

    MyQueue myQueue = new MyQueue(6);
    myQueue.enqueue(1);
    myQueue.dequeue();
```

```
        myQueue.enqueue(30);
        myQueue.enqueue(44);
        myQueue.enqueue(32);
        myQueue.dequeue();
        myQueue.enqueue(98);
        myQueue.dequeue();
        myQueue.enqueue(70);
        myQueue.enqueue(22);
        myQueue.dequeue();
        myQueue.enqueue(67);
        myQueue.enqueue(23);
    }
}
```

OUTPUT -



```
1 added to the queue
1 removed from the queue
30 added to the queue
44 added to the queue
32 added to the queue
30 removed from the queue
98 added to the queue
44 removed from the queue
70 added to the queue
22 added to the queue
32 removed from the queue
67 added to the queue
23 added to the queue
```

4) SET :-

The set interface is present in java.util package and extends the Collection interface is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the mathematical set. This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements.

HashSet

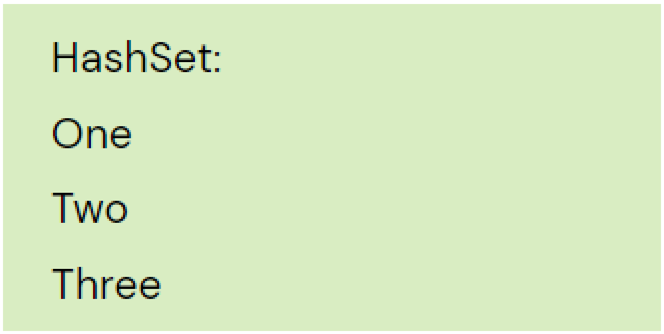
1. https://java2blog.com/how-hashset-works-in-java/ implements Set interface and it does not allow duplicate elements
2. HashSet does not maintain insertion order
3. It is not synchronized and not thread safe

CODE -

```
package com.example.yash_java_linkedlist;

import java.util.HashSet;
import java.util.Iterator;

public class HashSetMain {
   /*
        * @author : Yash Jain
        */
   public static void main(String[] args) {
      HashSet<String> set = new HashSet<>();
      set.add("One");
      set.add("Two");
      set.add("One");
```

```
        set.add("Three");
        set.add("Two");

        System.out.println("HashSet:");

        Iterator<String> it=set.iterator();
        while(it.hasNext()){
           System.out.println(it.next());
        }
    }
}
```

OUTPUT -



```
HashSet:
One
Two
Three
```

5) MAP :-

The map interface is present in java.util package represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit differently from the rest of the collection types. A map contains unique keys.

Maps are perfect to use for key-value association mapping such as dictionaries. The maps are used to perform lookups by keys or when someone wants to retrieve and update elements by keys.

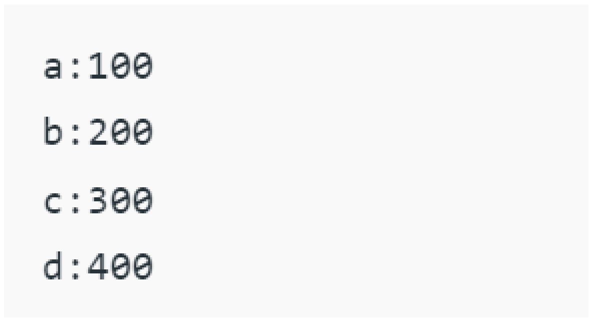
Some common scenarios are as follows:

- A map of error codes and their descriptions.
- A map of zip codes and cities.
- A map of managers and employees. Each manager (key) is associated with a list of employees (value) he manages.
- A map of classes and students. Each class (key) is associated with a list of students (value).

CODE -

```
// Java Program to Demonstrate
// Working of Map interface

// Importing required classes
import java.util.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {
        // Creating an empty HashMap
        Map<String, Integer> hm
            = new HashMap<String, Integer>();

        // Inserting pairs in above Map
        // using put() method
        hm.put("a", new Integer(100));
        hm.put("b", new Integer(200));
        hm.put("c", new Integer(300));
        hm.put("d", new Integer(400));

        // Traversing through Map using for-each loop
        for (Map.Entry<String, Integer> me :
             hm.entrySet()) {

            // Printing keys
            System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

OUTPUT -



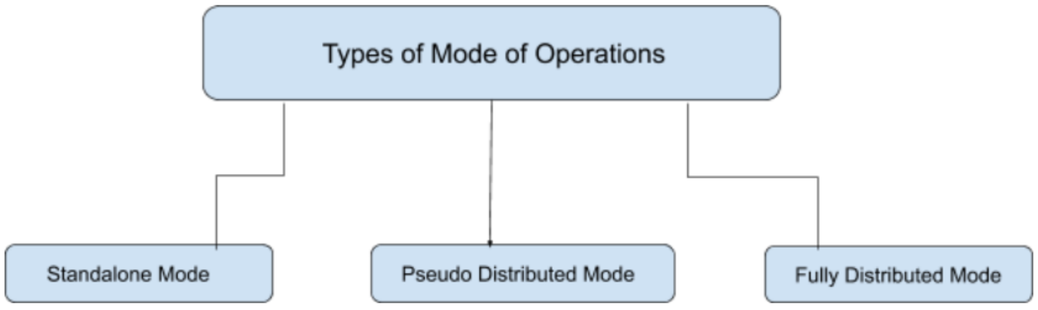
```
a:100
b:200
c:300
d:400
```

ASSIGNMENT - 2

Q1. Perform setting up and Installing Hadoop in its three operating modes: Standalone, Pseudo-distributed, Fully distributed.

Ans.

As we all know Hadoop is an open-source framework which is mainly used for storage purpose and maintaining and analyzing a large amount of data or datasets on the clusters of commodity hardware, which means it is actually a data management tool. Hadoop also posses a scale-out storage property, which means that we can scale up or scale down the number of nodes as per are a requirement in the future which is really a cool feature.



Hadoop Mainly works on 3 different Modes:

1. Standalone Mode
2. Pseudo-distributed Mode
3. Fully-Distributed Mode

1. Standalone Mode :-

In *Standalone Mode* none of the Daemon will run i.e. Namenode, Datanode, Secondary Name node, Job Tracker, and Task Tracker. We use job-tracker and task-tracker for processing purposes in Hadoop1. For Hadoop2 we use Resource Manager and Node Manager. Standalone Mode also means that we are installing Hadoop only in a single system. By default, Hadoop is made to run in this Standalone Mode or we can also call it as the *Local mode*. We mainly use Hadoop in this Mode for the Purpose of Learning, testing, and debugging.

Hadoop works very much Fastest in this mode among all of these 3 modes. As we all know HDFS (Hadoop distributed file system) is one of the major components for Hadoop which utilized for storage Permission is not utilized in this mode. You can think of HDFS as similar to the file system's available for windows i.e. NTFS (New Technology File System) and FAT32(File Allocation Table which stores the data in the blocks of 32 bits). when your Hadoop works in this mode there is no need to configure the files – *hdfs-site.xml*, *mapred-site.xml*, *core-site.xml* for Hadoop environment. In this Mode, all of your Processes will run on a single JVM(Java Virtual Machine) and this mode can only be used for small development purposes.

- Link To Installation Process -

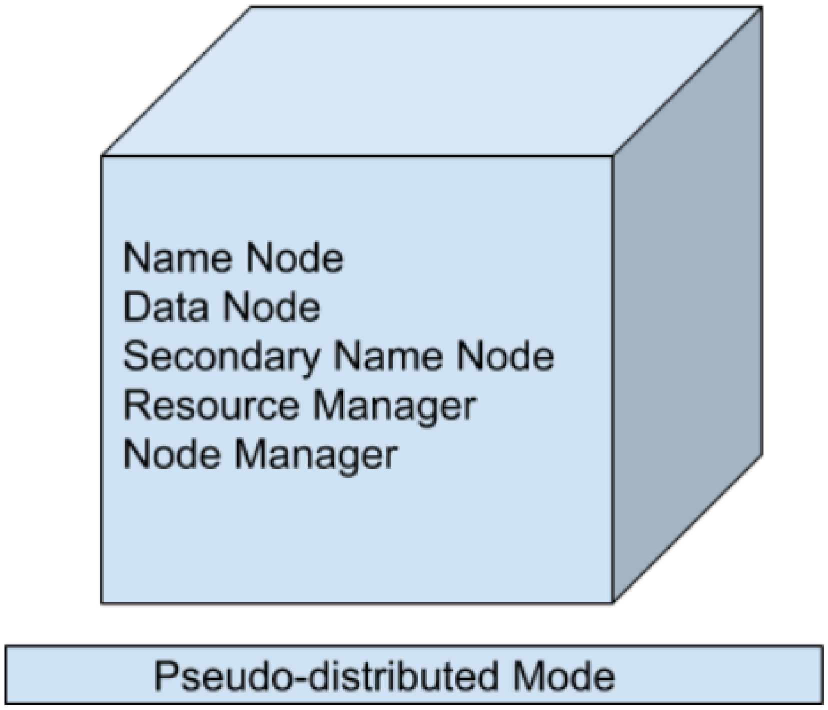
https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html

2. Pseudo Distributed Mode (Single Node Cluster):-

In Pseudo-distributed Mode we also use only a single node, but the main thing is that the cluster is simulated, which means that all the processes inside the cluster will run independently to each other. All the daemons that are Namenode, Datanode, Secondary Name node, Resource Manager, Node Manager, etc. will be running as a separate process on separate JVM(Java Virtual Machine) or we can say run on different java processes that is why it is called a Pseudo-distributed.

One thing we should remember that as we are using only the single node set up so all the Master and Slave processes are handled by the single system. Namenode and Resource Manager are used as Master and Datanode and Node Manager is used as a slave. A secondary name node is also used as a Master. The purpose of the Secondary Name node is to just keep the hourly based backup of the Name node. In this Mode,

- Hadoop is used for development and for debugging purposes both.
- Our HDFS(Hadoop Distributed File System) is utilized for managing the Input and Output processes.
- We need to change the configuration files *mapred-site.xml*, *core-site.xml*, *hdfs-site.xml* for setting up the environment.



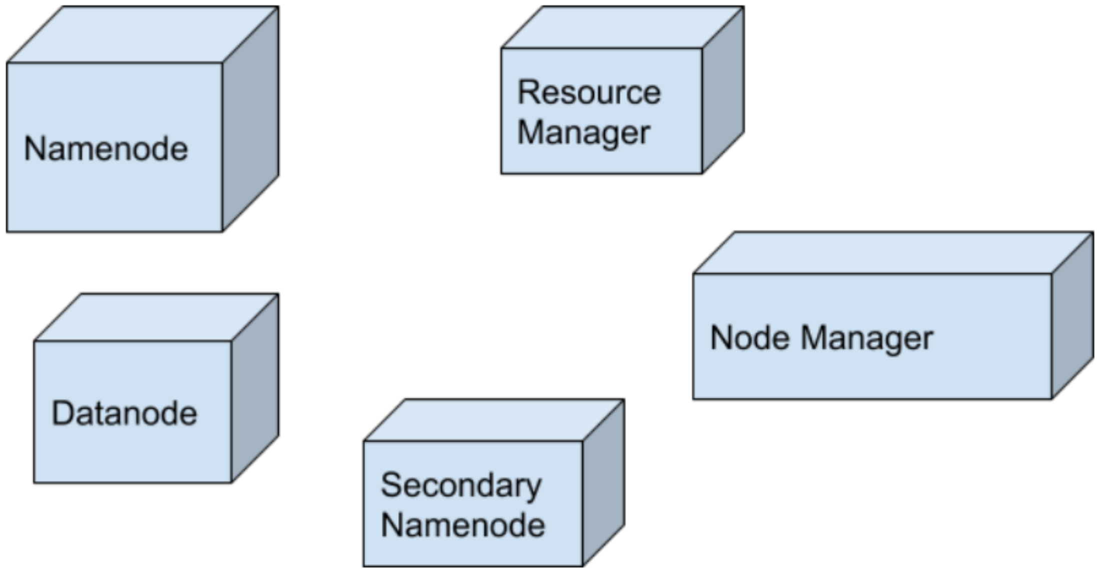
- Link To Installation Process -

https://www.geeksforgeeks.org/installing-and-setting-up-hadoop-in-pseudo-distributed-mode-in-windows-10/

3. Fully Distributed Mode (Multi-Node Cluster) :-

This is the most important one in which multiple nodes are used few of them run the Master Daemon's that are Namenode and Resource Manager and the rest of them run the Slave Daemon's that are DataNode and Node Manager. Here Hadoop will run on the clusters of Machine or nodes. Here the data that is used is distributed across different nodes. This is actually the *Production Mode* of Hadoop let's clarify or understand this Mode in a better way in Physical Terminology.

Once you download the Hadoop in a tar file format or zip file format then you install it in your system and you run all the processes in a single system but here in the fully distributed mode we are extracting this tar or zip file to each of the nodes in the Hadoop cluster and then we are using a particular node for a particular process. Once you distribute the process among the nodes then you'll define which nodes are working as a master or which one of them is working as a slave.



- Link To Installation Process -

https://mindmajix.com/installation-and-configuration-in-hadoop

ASSIGNMENT - 3

Q1. Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

Ans.

Prerequisite

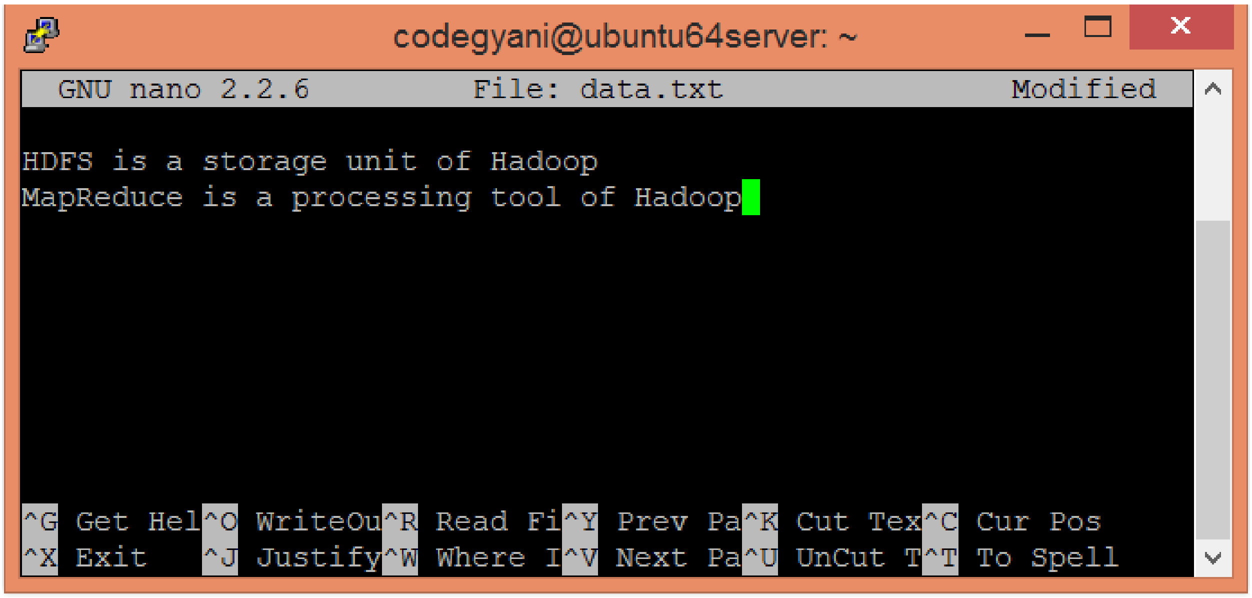
- **Java Installation -** Check whether the Java is installed or not using the following command.
  java -version
- **Hadoop Installation -** Check whether the Hadoop is installed or not using the following command.
  hadoop version

If any of them is not installed in your system, follow the below link to install it.

www.javatpoint.com/hadoop-installation

Steps to execute MapReduce word count example

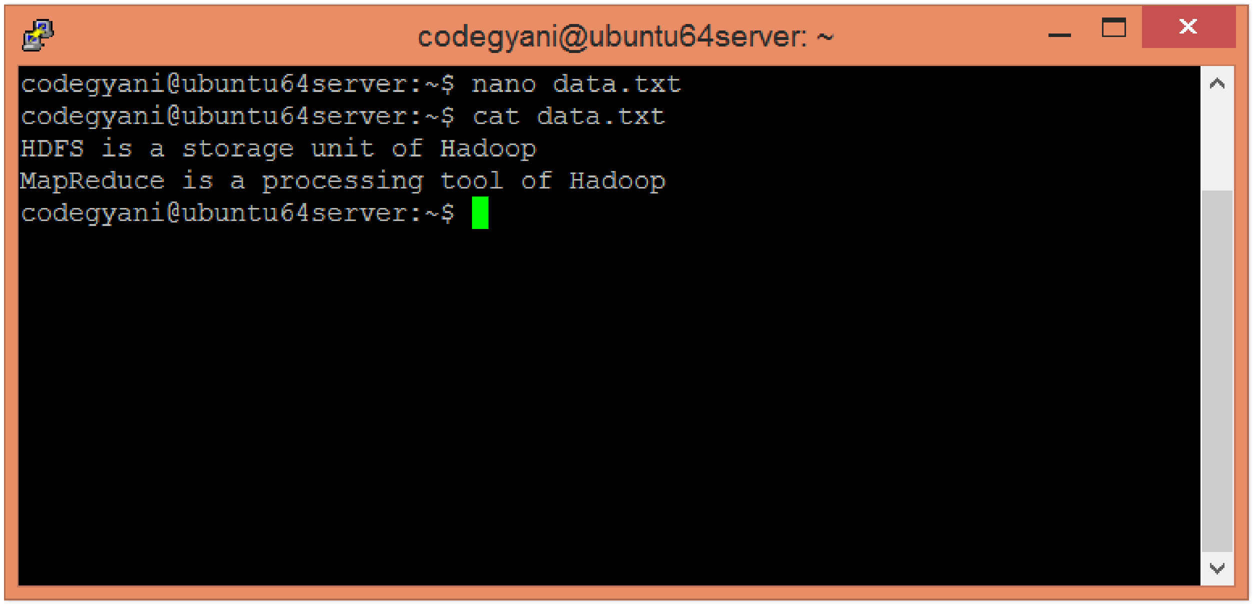
- Create a text file in your local machine and write some text into it.
  $ nano data.txt



```
 GNU nano 2.2.6             File: data.txt                       Modified

HDFS is a storage unit of Hadoop
MapReduce is a processing tool of Hadoop






^G Get Hel^O WriteOu^R Read Fi^Y Prev Pa^K Cut Tex^C Cur Pos
^X Exit   ^J Justify^W Where I^V Next Pa^U UnCut T^T To Spell
```

- Check the text written in the data.txt file.

  $ cat data.txt



```
codegyani@ubuntu64server:~$ nano data.txt
codegyani@ubuntu64server:~$ cat data.txt
HDFS is a storage unit of Hadoop
MapReduce is a processing tool of Hadoop
codegyani@ubuntu64server:~$
```

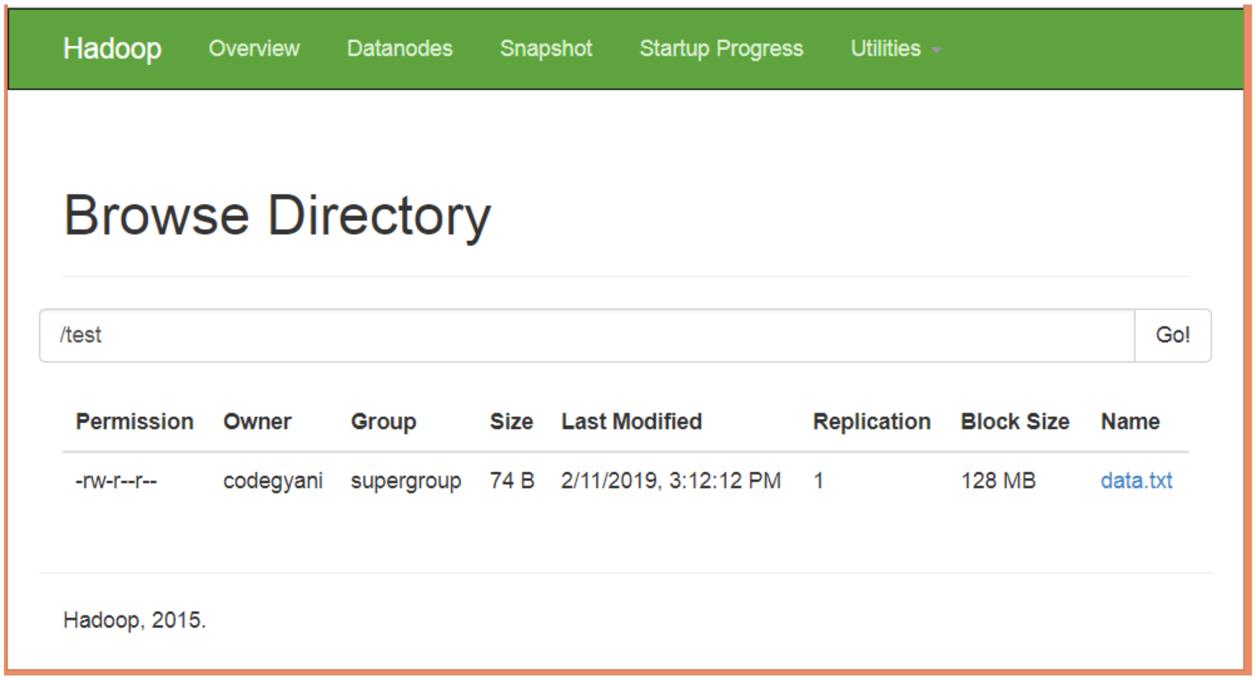
In this example, we find out the frequency of each word exists in this text file.

- Create a directory in HDFS, where to kept text file.

  $ hdfs dfs -mkdir /test

- Upload the data.txt file on HDFS in the specific directory.

```
$ hdfs dfs -put /home/codegyani/data.txt /test
```



Browse Directory

/test

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	codegyani	supergroup	74 B	2/11/2019, 3:12:12 PM	1	128 MB	data.txt

Hadoop, 2015.

- Write the MapReduce program using eclipse.

File: WC_Mapper.java

```java
package com.javatpoint;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class WC_Mapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text
value,OutputCollector<Text,IntWritable> output,
         Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer  tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }

}
```

File: WC_Reducer.java

```
package com.javatpoint;
    import java.io.IOException;
    import java.util.Iterator;
    import org.apache.hadoop.io.IntWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapred.MapReduceBase;
    import org.apache.hadoop.mapred.OutputCollector;
    import org.apache.hadoop.mapred.Reducer;
    import org.apache.hadoop.mapred.Reporter;
```

```

public class WC_Reducer  extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output,
 Reporter reporter) throws IOException {
int sum=0;
while (values.hasNext()) {
sum+=values.next().get();
}
output.collect(key,new IntWritable(sum));
}
}
```

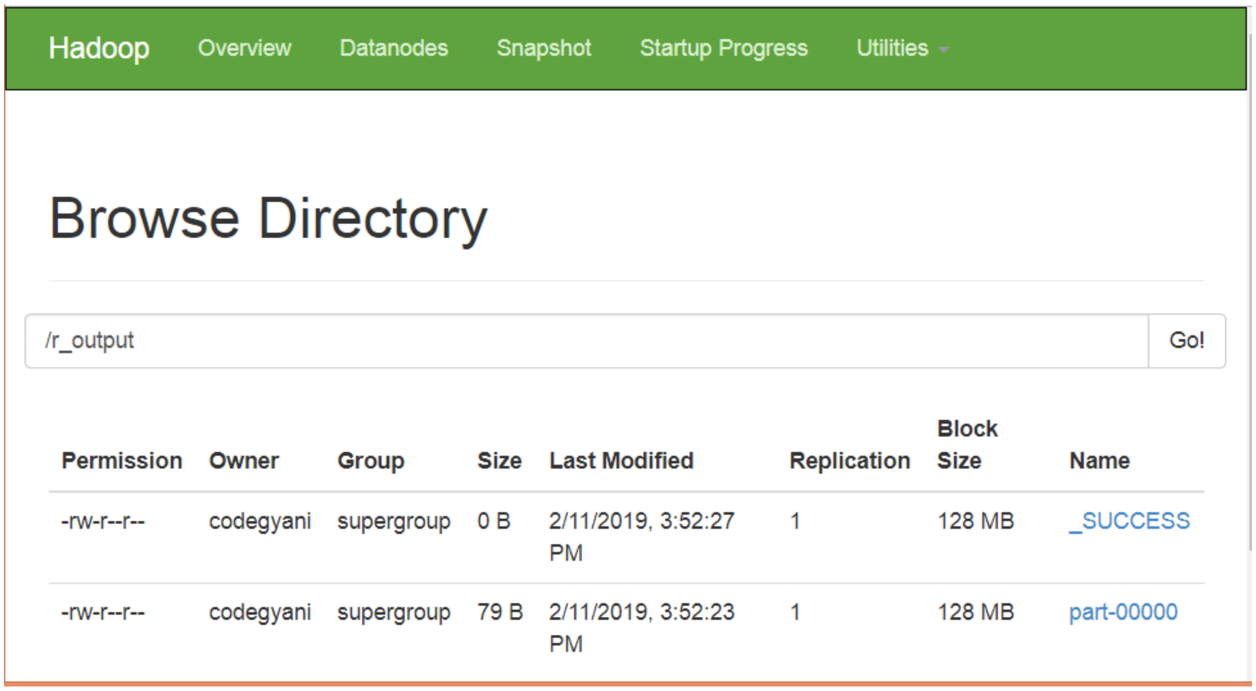
File: WC_Runner.java

```
package com.javatpoint;

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class WC_Runner {
    public static void main(String[] args) throws IOException{
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.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);
    }
}
```

Download the source code.

- Create the jar file of this program and name it **countworddemo.jar.**
- Run the jar file

  hadoop jar /home/codegyani/wordcountdemo.jar com.javatpoint.WC_Runner /test/data.txt /r_output
- The output is stored in /r_output/part-00000



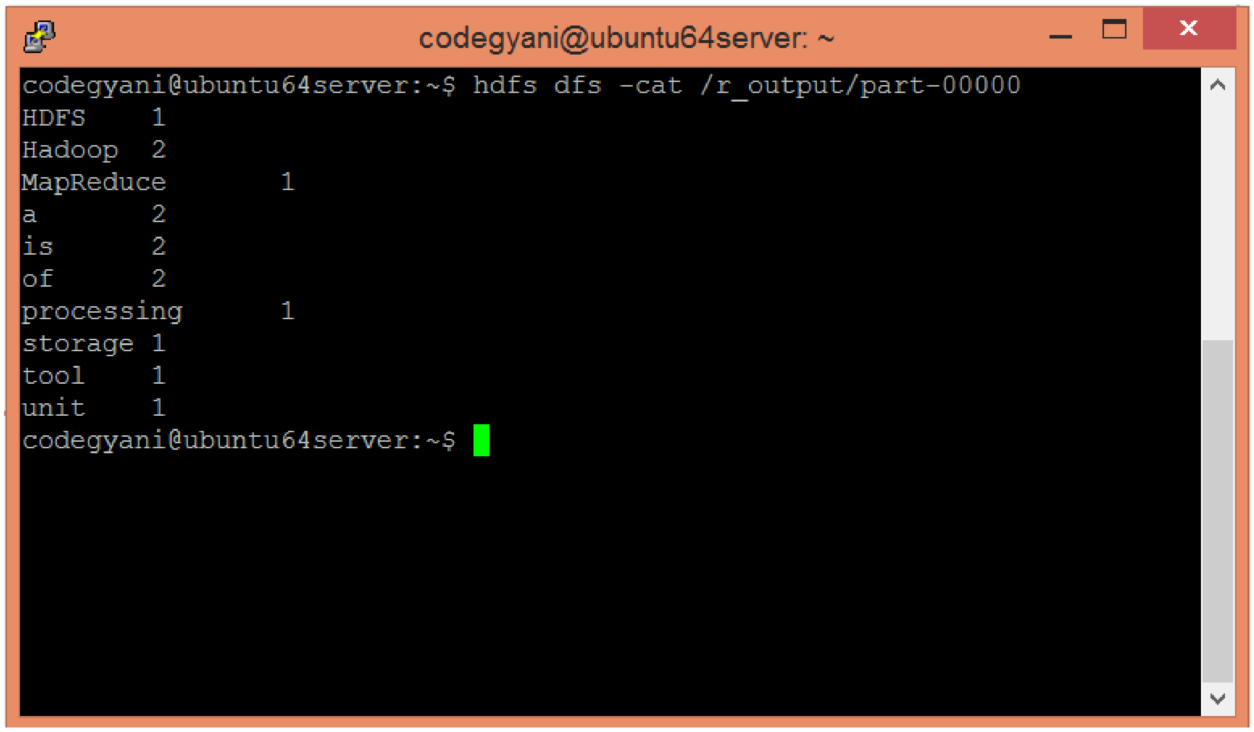
Browse Directory

/r_output

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	codegyani	supergroup	0 B	2/11/2019, 3:52:27 PM	1	128 MB	_SUCCESS
-rw-r--r--	codegyani	supergroup	79 B	2/11/2019, 3:52:23 PM	1	128 MB	part-00000

- Now execute the command to see the output.

  hdfs dfs -cat /r_output/part-00000



codegyani@ubuntu64server: ~

```
codegyani@ubuntu64server:~$ hdfs dfs -cat /r_output/part-00000
HDFS    1
Hadoop  2
MapReduce       1
a       2
is      2
of      2
processing      1
storage 1
tool    1
unit    1
codegyani@ubuntu64server:~$
```

ASSIGNMENT - 4

Q1. Write a Map Reduce program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record oriented.

Ans.

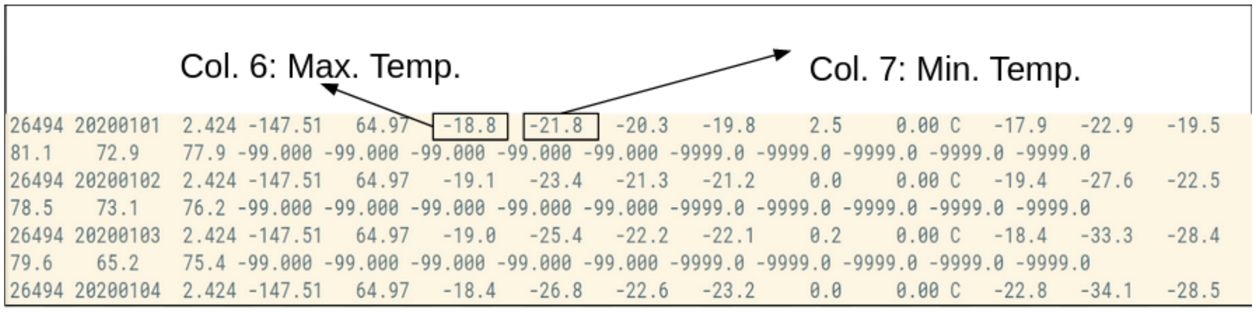
Step 1:

We can download the dataset from this Link, For various cities in different years. choose the year of your choice and select any one of the data text-file for analyzing. In my case, I have selected *CRND0103-2020-AK_Fairbanks_11_NE.txt* dataset for analysis of hot and cold days in Fairbanks, Alaska.

We can get information about data from *README.txt* file available on the NCEI website.

Step 2:

Below is the example of our dataset where column 6 and column 7 is showing Maximum and Minimum temperature, respectively.

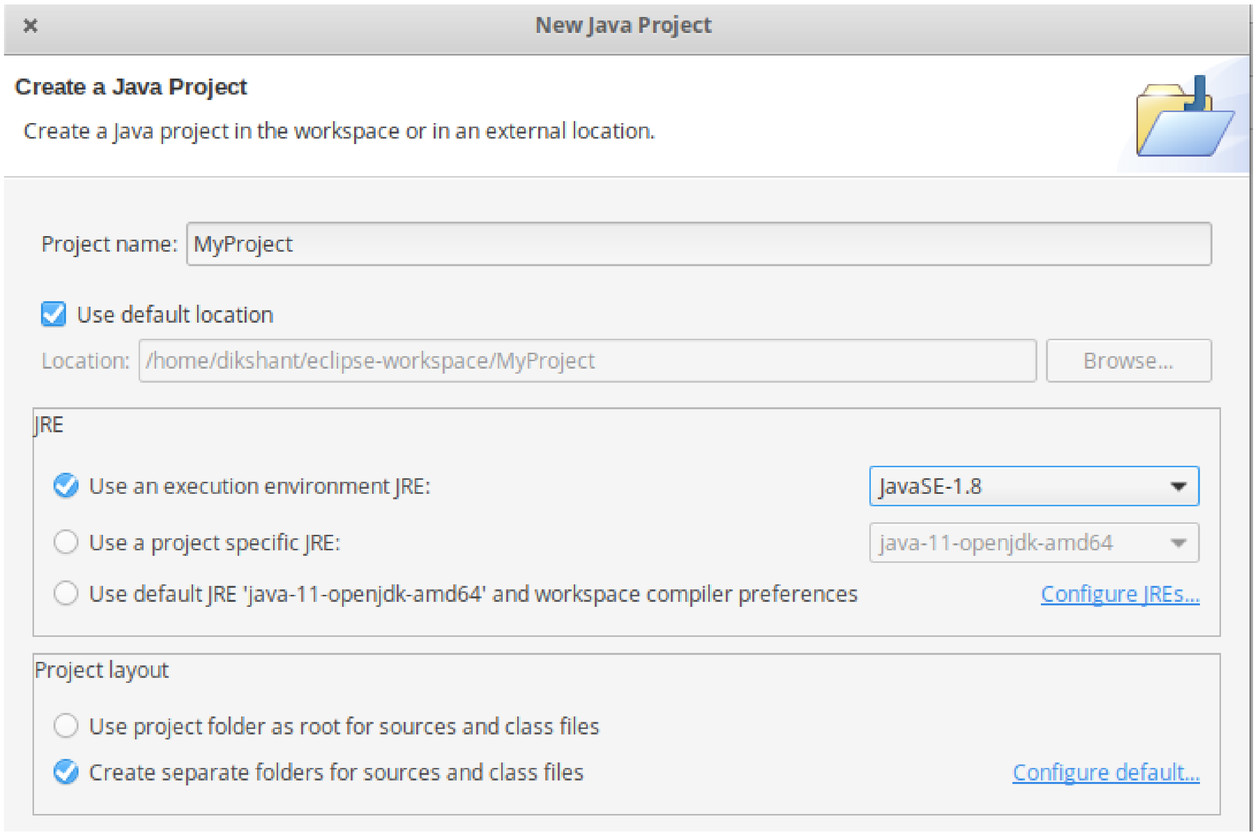


```
26494 20200101  2.424 -147.51   64.97   -18.8    -21.8    -20.3    -19.8     2.5     0.00 C   -17.9    -22.9    -19.5
81.1    72.9    77.9 -99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
26494 20200102  2.424 -147.51   64.97   -19.1    -23.4    -21.3    -21.2     0.0     0.00 C   -19.4    -27.6    -22.5
78.5    73.1    76.2 -99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
26494 20200103  2.424 -147.51   64.97   -19.0    -25.4    -22.2    -22.1     0.2     0.00 C   -18.4    -33.3    -28.4
79.6    65.2    75.4 -99.000 -99.000 -99.000 -99.000 -99.000 -9999.0 -9999.0 -9999.0 -9999.0 -9999.0
26494 20200104  2.424 -147.51   64.97   -18.4    -26.8    -22.6    -23.2     0.0     0.00 C   -22.8    -34.1    -28.5
```

<u>Step 3:</u>

Make a project in Eclipse with below steps:

- First Open **Eclipse** -> then select **File -> New -> Java Project** ->Name it **MyProject** -> then select **use an execution environment** -> choose **JavaSE-1.8** then **next** -> **Finish**.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: MyProject

[✓] Use default location

Location: /home/dikshant/eclipse-workspace/MyProject

JRE

[✓] Use an execution environment JRE: JavaSE-1.8

Use a project specific JRE: java-11-openjdk-amd64

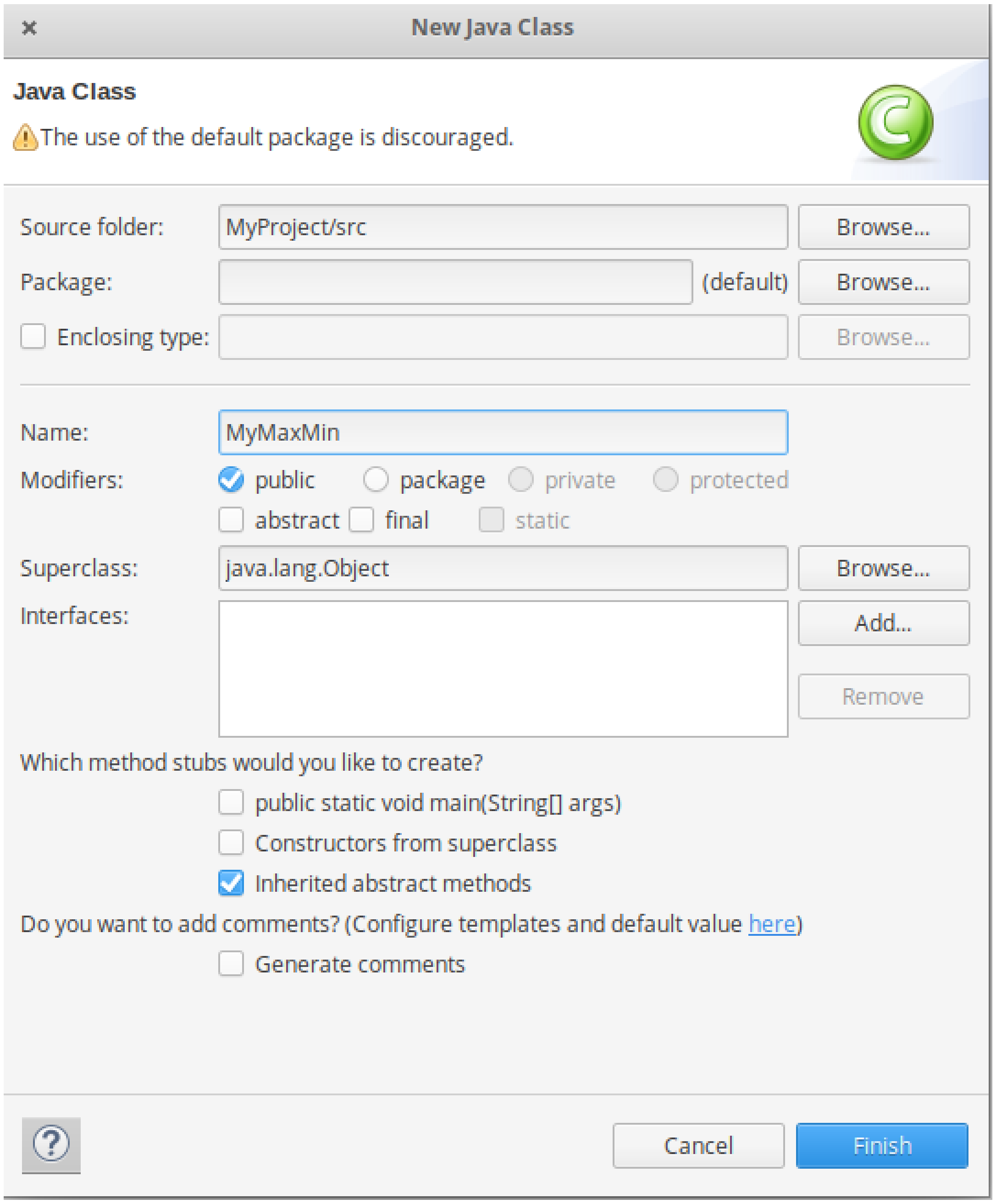
Use default JRE 'java-11-openjdk-amd64' and workspace compiler preferences

Project layout

Use project folder as root for sources and class files

[✓] Create separate folders for sources and class files

- In this Project Create Java class with name **MyMaxMin** -> then click **Finish**



New Java Class

Java Class

⚠The use of the default package is discouraged.

Source folder: MyProject/src

Package: (default)

[ ] Enclosing type:

Name: MyMaxMin

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

- Copy the below source code to this **MyMaxMin** java class

- JAVA

```
// importing Libraries
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 {


    // Mapper

    /*MaxTemperatureMapper class is static
     * and extends Mapper abstract class
     * having four Hadoop generics type
     * LongWritable, Text, Text, Text.
    */


    public static class MaxTemperatureMapper extends
            Mapper<LongWritable, Text, Text, Text> {

        /**
        * @method map
        * This method takes the input as a 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 > 30 and temp_min < 15 are
    * passed to the reducer.
    */

// the data in our data set with
// this value is inconsistent data
public static final int MISSING = 9999;

@Override
    public void map(LongWritable arg0, Text Value, Context context)
            throws IOException, InterruptedException {

    // Convert the single row(Record) to
    // String and store it in String
    // variable name line

    String line = Value.toString();

        // Check for the empty line
        if (!(line.length() == 0)) {

            // from character 6 to 14 we have
            // the date in our dataset
            String date = line.substring(6, 14);

            // similarly we have taken the maximum
            // temperature from 39 to 45 characters
            float temp_Max = Float.parseFloat(line.substring(39, 45).trim());

            // similarly we have taken the minimum
            // temperature from 47 to 53 characters

            float temp_Min = Float.parseFloat(line.substring(47, 53).trim());

            // if maximum temperature is
            // greater than 30, it is a hot day
            if (temp_Max > 30.0) {

                // Hot day
                context.write(new Text("The Day is Hot Day :" + date),
                                     new Text(String.valueOf(temp_Max)));
            }

            // if the minimum temperature is
            // less than 15, it is a cold day
            if (temp_Min < 15) {

                // Cold day
                context.write(new Text("The Day is Cold Day :" + date),
                        new Text(String.valueOf(temp_Min)));
            }
        }
    }

}

// Reducer

    /*MaxTemperatureReducer class is static
      and extends Reducer abstract class
      having four Hadoop generics type
      Text, Text, Text, Text.
    */
```

```
public static class MaxTemperatureReducer extends
        Reducer<Text, Text, Text, Text> {

    /**
    * @method reduce
    * This method takes the input as key and
    * list of values pair from the mapper,
    * it does aggregation based on keys and
    * produces the final context.
    */

    public void reduce(Text Key, Iterator<Text> Values, Context context)
            throws IOException, InterruptedException {


        // putting all the values in
        // temperature variable of type String
        String temperature = Values.next().toString();
        context.write(Key, new Text(temperature));
    }

}



/**
* @method main
* This method is used for setting
* all the configuration properties.
* It acts as a driver for map-reduce
* code.
*/
```

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

    // reads the default configuration of the
    // cluster from the configuration XML files
    Configuration conf = new Configuration();

    // Initializing the job with the
    // default configuration of the cluster
    Job job = new Job(conf, "weather example");

    // Assigning the driver class name
    job.setJarByClass(MyMaxMin.class);

    // Key type coming out of mapper
    job.setMapOutputKeyClass(Text.class);

    // value type coming out of mapper
    job.setMapOutputValueClass(Text.class);

    // Defining the mapper class name
    job.setMapperClass(MaxTemperatureMapper.class);

    // Defining the reducer class name
    job.setReducerClass(MaxTemperatureReducer.class);

    // Defining input Format class which is
    // responsible to parse the dataset
    // into a key value pair
    job.setInputFormatClass(TextInputFormat.class);

    // Defining output Format class which is
    // responsible to parse the dataset
```

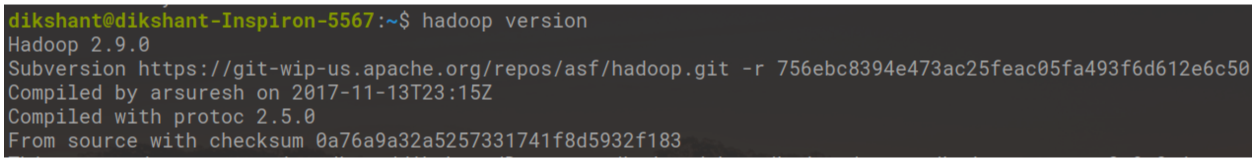
```
        // into a key value pair
        job.setOutputFormatClass(TextOutputFormat.class);

        // setting the second argument
        // as a path in a path variable
        Path OutputPath = new Path(args[1]);

        // Configuring the input path
        // from the filesystem into the job
        FileInputFormat.addInputPath(job, new Path(args[0]));

        // Configuring the output path from
        // the filesystem into the job
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // deleting the context path automatically
        // from hdfs so that we don't have
        // to delete it explicitly
        OutputPath.getFileSystem(conf).delete(OutputPath);

        // exiting the job only if the
        // flag value becomes false
        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
}
```

- Now we need to add external jar for the packages that we have import.

  Download the jar package Hadoop Common and Hadoop MapReduce Core according to your Hadoop version.

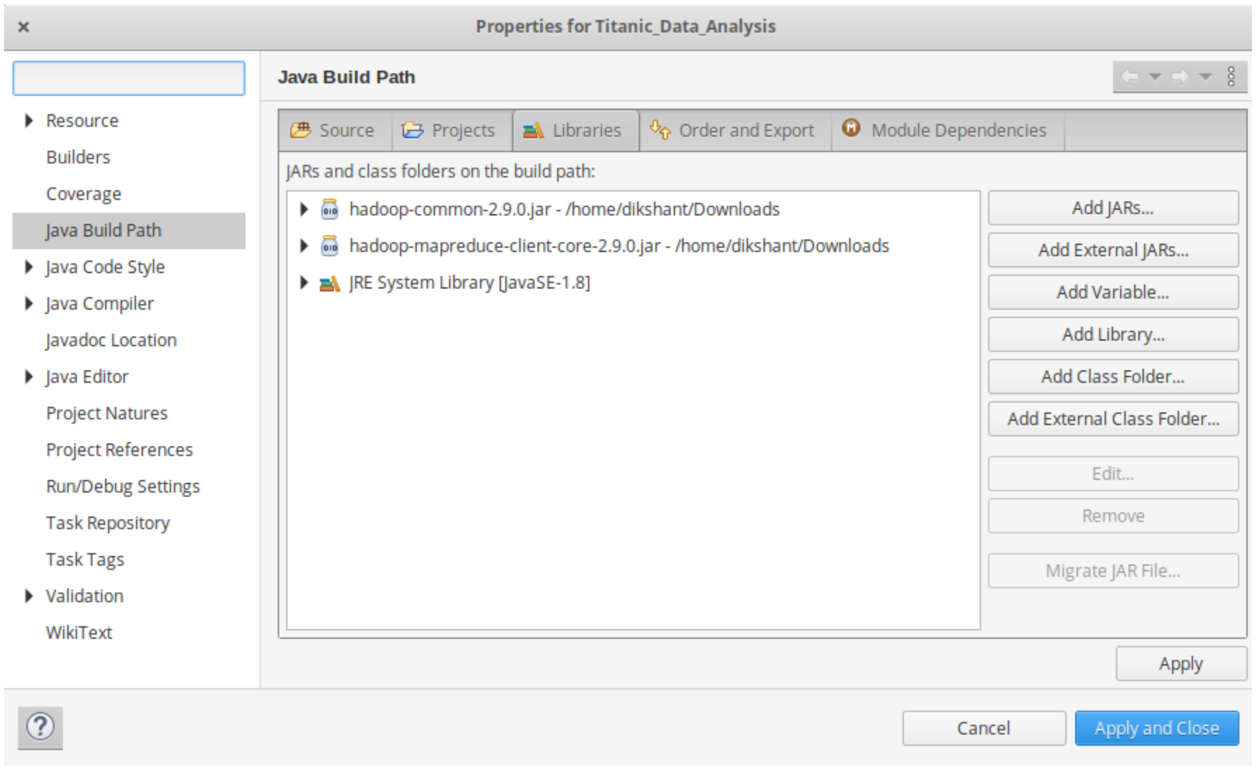
  You can check Hadoop Version:

hadoop version



```
dikshant@dikshant-Inspiron-5567:~$ hadoop version
Hadoop 2.9.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 756ebc8394e473ac25feac05fa493f6d612e6c50
Compiled by arsuresh on 2017-11-13T23:15Z
Compiled with protoc 2.5.0
From source with checksum 0a76a9a32a5257331741f8d5932f183
```

- Now we add these external jars to our **MyProject**. Right Click on **MyProject** -> then select **Build Path**-> Click on **Configure Build Path** and select **Add External jars….** and add jars from it's download location then click -> **Apply and Close**.

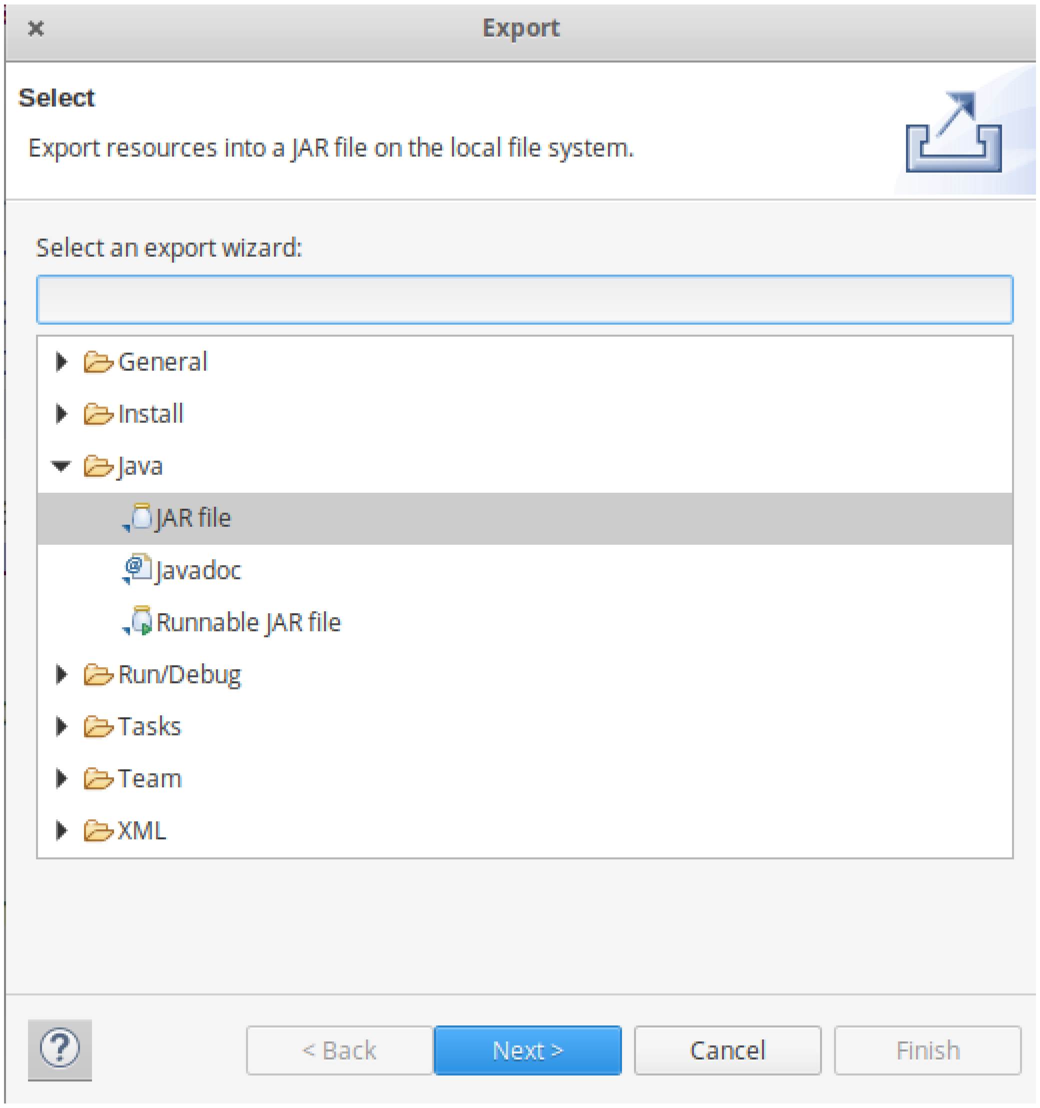


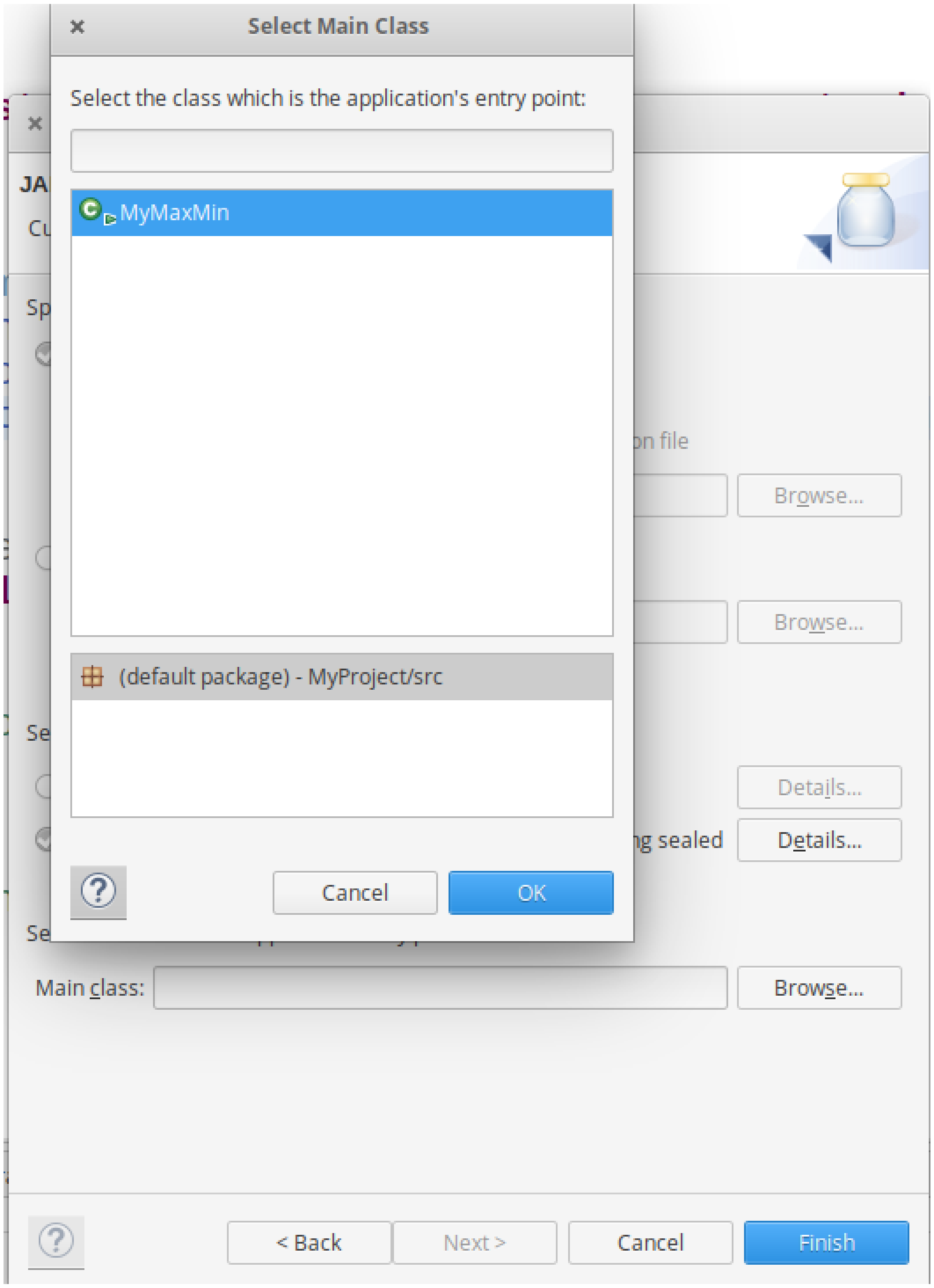
JARs and class folders on the build path:

hadoop-common-2.9.0.jar - /home/dikshant/Downloads
hadoop-mapreduce-client-core-2.9.0.jar - /home/dikshant/Downloads
JRE System Library [JavaSE-1.8]

- Now export the project as jar file. Right-click on **MyProject** choose **Export..** and go to **Java -> JAR file** click -> **Next** and choose your export destination then click -> **Next**.

choose Main Class as **MyMaxMin** by clicking -> **Browse** and then click -> **Finish** -> **Ok**.





Select Main Class

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

MyMaxMin

(default package) - MyProject/src

Step 4:

Start our Hadoop Daemons

start-dfs.sh

Start-yarn.sh

Step 5:

Move your dataset to the Hadoop HDFS.

Syntax:

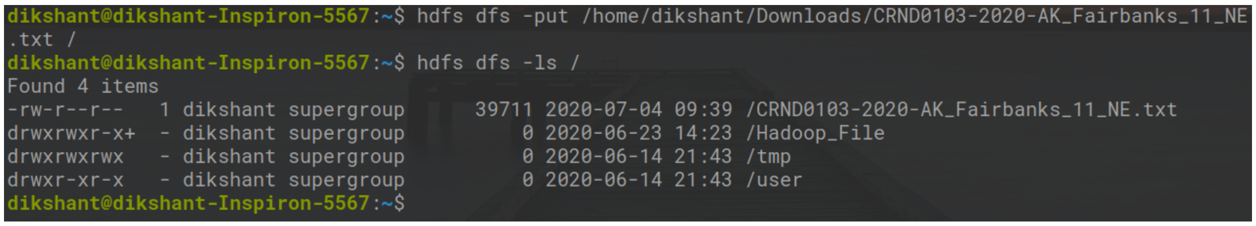
hdfs dfs -put /file_path /destination

In below command / shows the root directory of our HDFS.

hdfs dfs -put
/home/dikshant/Downloads/CRND0103-2020-AK_Fairbanks_11_NE.txt /

Check the file sent to our HDFS.

hdfs dfs -ls /



```
dikshant@dikshant-Inspiron-5567:~$ hdfs dfs -put /home/dikshant/Downloads/CRND0103-2020-AK_Fairbanks_11_NE
.txt /
dikshant@dikshant-Inspiron-5567:~$ hdfs dfs -ls /
Found 4 items
-rw-r--r--   1 dikshant supergroup      39711 2020-07-04 09:39 /CRND0103-2020-AK_Fairbanks_11_NE.txt
drwxrwxr-x+  - dikshant supergroup          0 2020-06-23 14:23 /Hadoop_File
drwxrwxrwx   - dikshant supergroup          0 2020-06-14 21:43 /tmp
drwxr-xr-x   - dikshant supergroup          0 2020-06-14 21:43 /user
dikshant@dikshant-Inspiron-5567:~$
```

Step 6:

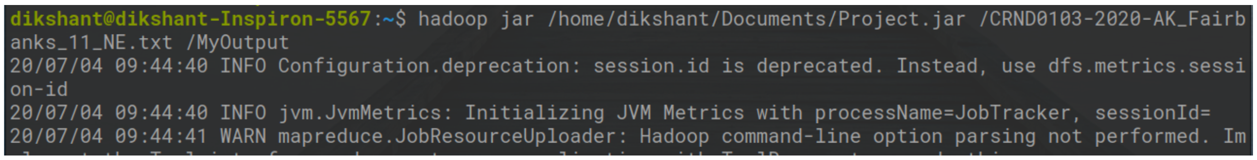
Now Run your Jar File with below command and produce the output in **MyOutput** File.

Syntax:

hadoop jar /jar_file_location /dataset_location_in_HDFS /output-file_name

Command:

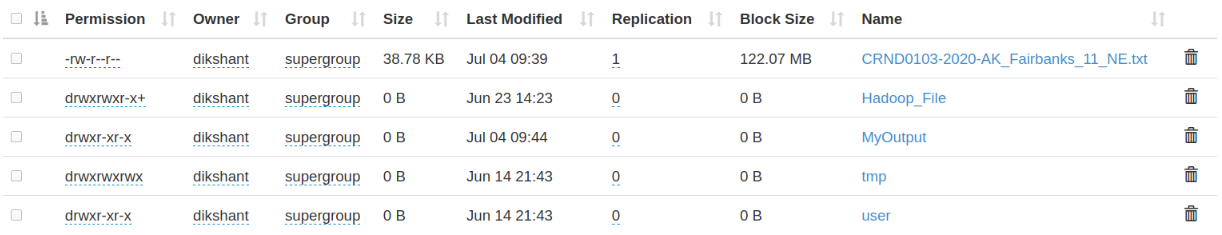
hadoop jar /home/dikshant/Documents/Project.jar /CRND0103-2020-AK_Fairbanks_11_NE.txt /MyOutput



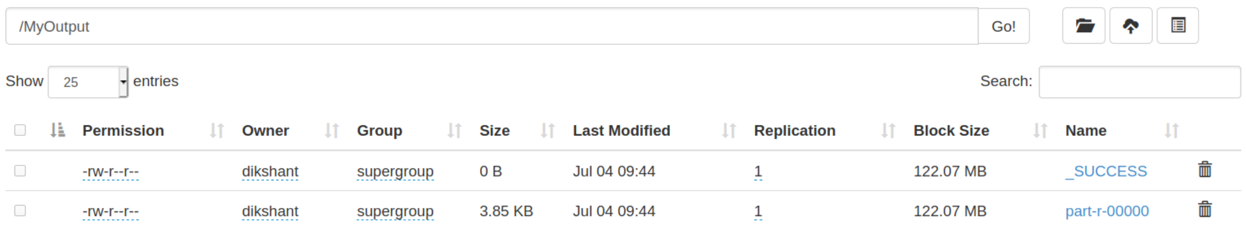
```
dikshant@dikshant-Inspiron-5567:~$ hadoop jar /home/dikshant/Documents/Project.jar /CRND0103-2020-AK_Fairb
anks_11_NE.txt /MyOutput
20/07/04 09:44:40 INFO Configuration.deprecation: session.id is deprecated. Instead, use dfs.metrics.sessi
on-id
20/07/04 09:44:40 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
20/07/04 09:44:41 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Im
```

Step 7:

Now Move to *localhost:50070/*, under utilities select *Browse the file system* and download **part-r-00000** in **/MyOutput** directory to see result.



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	dikshant	supergroup	38.78 KB	Jul 04 09:39	1	122.07 MB	CRND0103-2020-AK_Fairbanks_11_NE.txt
drwxrwxr-x+	dikshant	supergroup	0 B	Jun 23 14:23	0	0 B	Hadoop_File
drwxr-xr-x	dikshant	supergroup	0 B	Jul 04 09:44	0	0 B	MyOutput
drwxrwxrwx	dikshant	supergroup	0 B	Jun 14 21:43	0	0 B	tmp
drwxr-xr-x	dikshant	supergroup	0 B	Jun 14 21:43	0	0 B	user



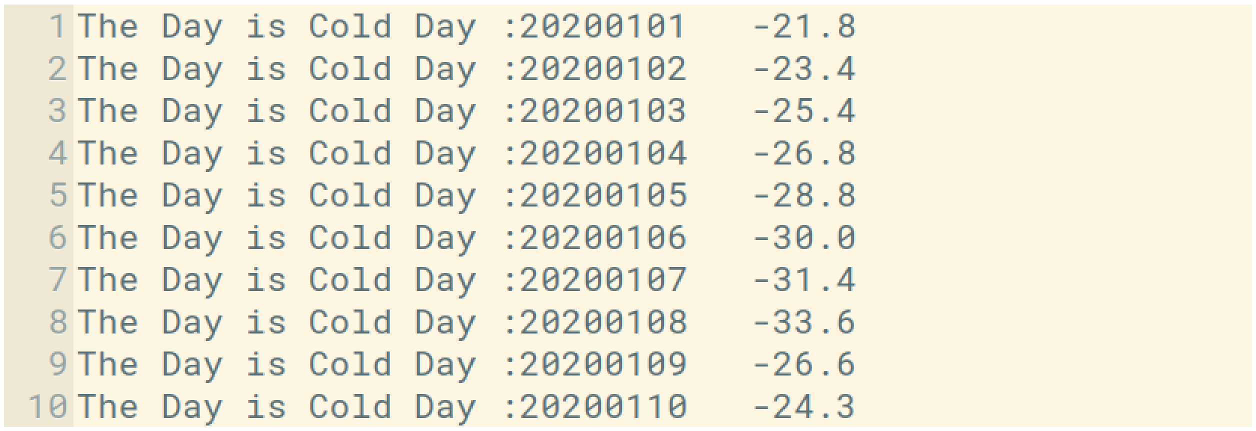
/MyOutput Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	dikshant	supergroup	0 B	Jul 04 09:44	1	122.07 MB	_SUCCESS
-rw-r--r--	dikshant	supergroup	3.85 KB	Jul 04 09:44	1	122.07 MB	part-r-00000

Step 8:

See the result in the Downloaded File.



```
The Day is Cold Day :20200101   -21.8
The Day is Cold Day :20200102   -23.4
The Day is Cold Day :20200103   -25.4
The Day is Cold Day :20200104   -26.8
The Day is Cold Day :20200105   -28.8
The Day is Cold Day :20200106   -30.0
The Day is Cold Day :20200107   -31.4
The Day is Cold Day :20200108   -33.6
The Day is Cold Day :20200109   -26.6
The Day is Cold Day :20200110   -24.3
```

In the above image, you can see the top 10 results showing the cold days. The second column is a day in yyyy/mm/dd format. For Example, **20200101** means

year = 2020
month = 01
Date = 01

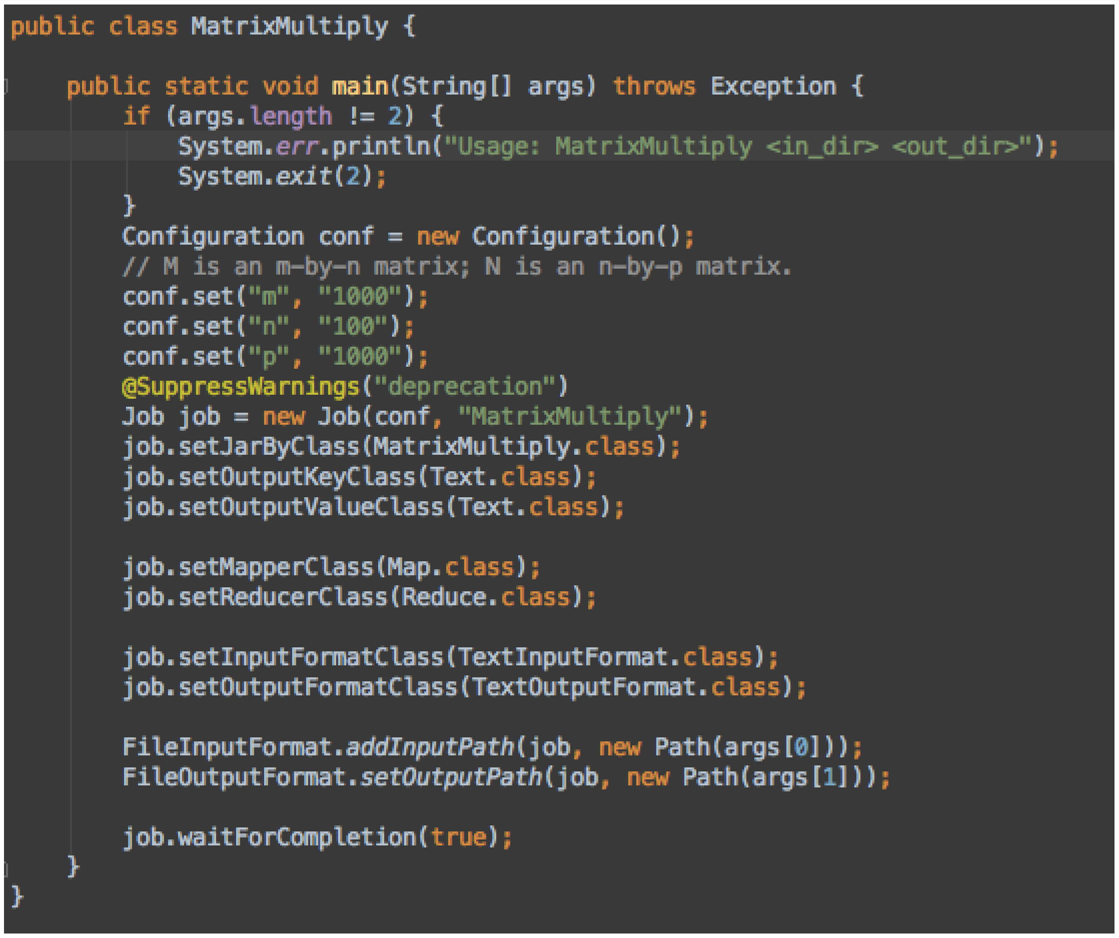
ASSIGNMENT - 5

Q1. Implement Matrix Multiplication with Hadoop Map Reduce

Ans.

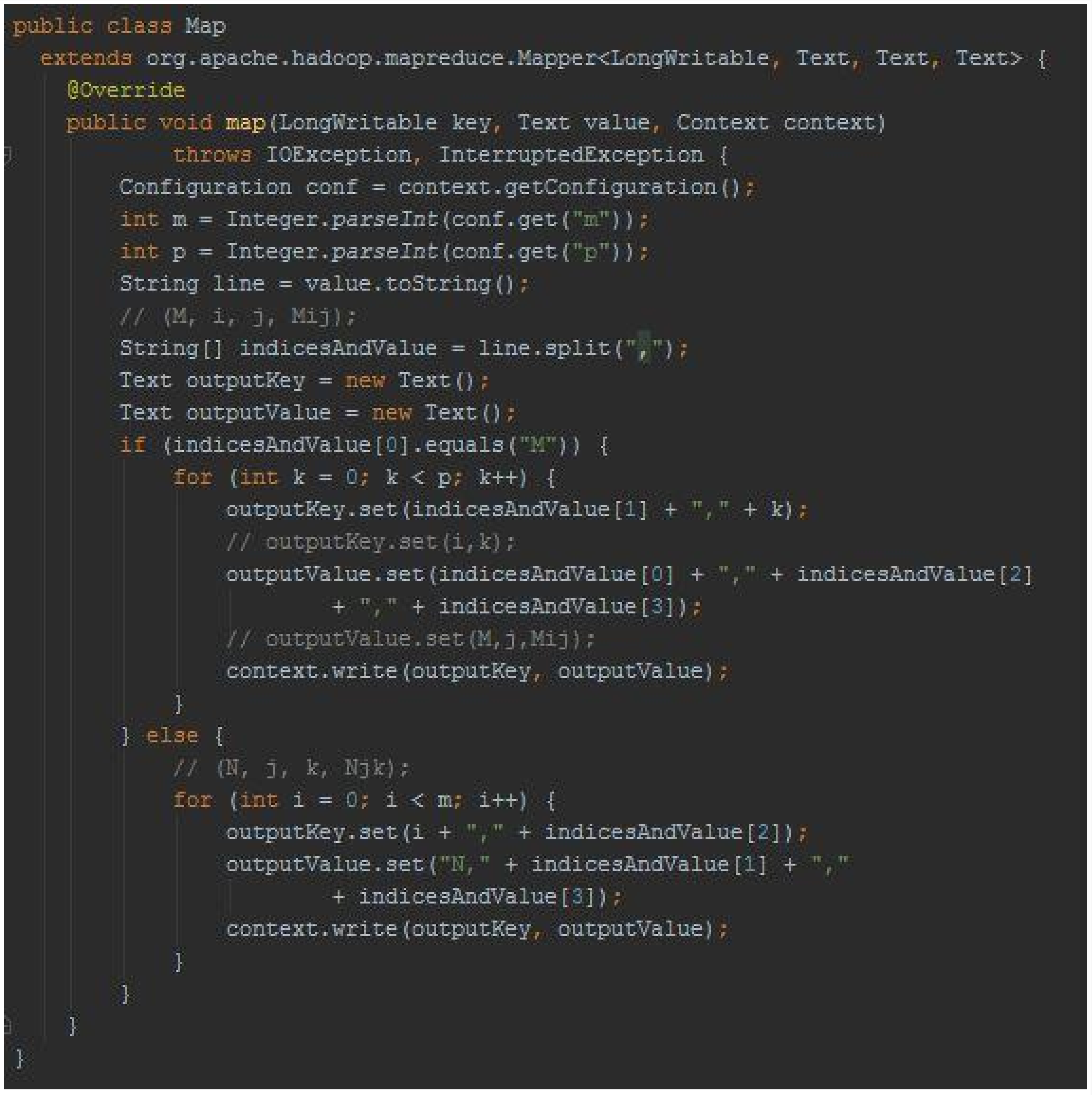
Source Code

I developed mapper and reducer classes as Map.java and Reduce.java as well as the main application class called MatrixMultiply.java. As it seen in MatrixMultiply.java code below, in main method the configuration parameters are being set as well as the input/output directories of MapReduce job [2].



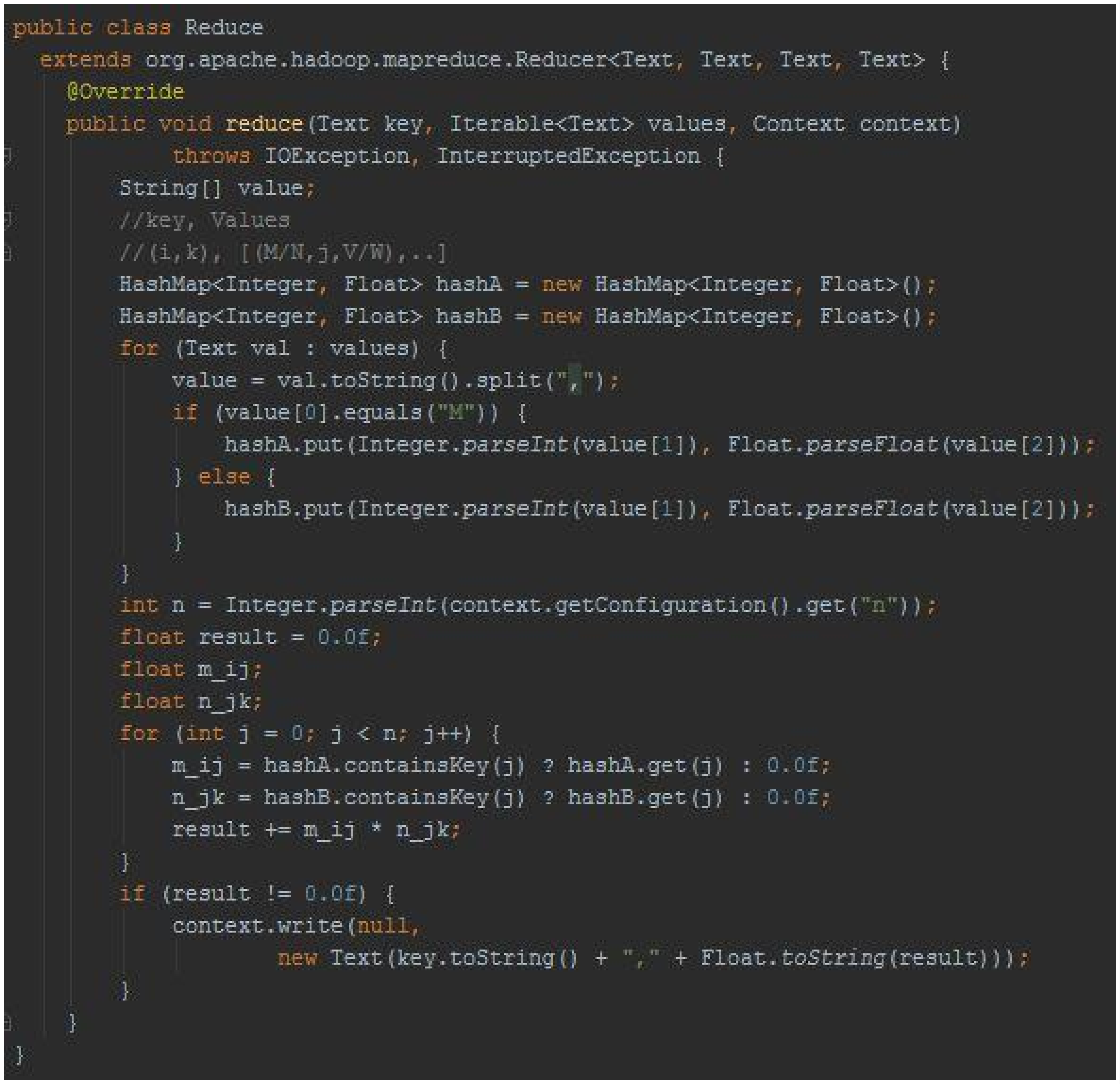
```java
public class MatrixMultiply {

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");
            System.exit(2);
        }
        Configuration conf = new Configuration();
        // M is an m-by-n matrix; N is an n-by-p matrix.
        conf.set("m", "1000");
        conf.set("n", "100");
        conf.set("p", "1000");
        @SuppressWarnings("deprecation")
        Job job = new Job(conf, "MatrixMultiply");
        job.setJarByClass(MatrixMultiply.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

Mapper class extends org.apache.hadoop.mapreduce.Mapper class and implements the map task described in Algorithm 1 and creates the $key, value$ pairs from the input files as it shown in the code below:



```java
public class Map
  extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        int m = Integer.parseInt(conf.get("m"));
        int p = Integer.parseInt(conf.get("p"));
        String line = value.toString();
        // (M, i, j, Mij);
        String[] indicesAndValue = line.split(",");
        Text outputKey = new Text();
        Text outputValue = new Text();
        if (indicesAndValue[0].equals("M")) {
            for (int k = 0; k < p; k++) {
                outputKey.set(indicesAndValue[1] + "," + k);
                // outputKey.set(i,k);
                outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]
                        + "," + indicesAndValue[3]);
                // outputValue.set(M,j,Mij);
                context.write(outputKey, outputValue);
            }
        } else {
            // (N, j, k, Njk);
            for (int i = 0; i < m; i++) {
                outputKey.set(i + "," + indicesAndValue[2]);
                outputValue.set("N," + indicesAndValue[1] + ","
                        + indicesAndValue[3]);
                context.write(outputKey, outputValue);
            }
        }
    }
}
```

Reducer class, extends org.apache.hadoop.mapreduce.Reducer class and implements the reduce task described in Algorithm 2 and creates the $key, value$ pairs for the product matrix then writes its output on HDFS as it shown in the code below:



```java
public class Reduce
  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        String[] value;
        //key, Values
        //(i,k), [(M/N,j,V/W),..]
        HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
        HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
        for (Text val : values) {
            value = val.toString().split(",");
            if (value[0].equals("M")) {
                hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            } else {
                hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            }
        }
        int n = Integer.parseInt(context.getConfiguration().get("n"));
        float result = 0.0f;
        float m_ij;
        float n_jk;
        for (int j = 0; j < n; j++) {
            m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
            n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
            result += m_ij * n_jk;
        }
        if (result != 0.0f) {
            context.write(null,
                    new Text(key.toString() + "," + Float.toString(result)));
        }
    }
}
```

ASSIGNMENT - 6

Q1. Install and Run Pig then write Pig Latin scripts to sort, group, join, project, and filter your data.

Ans.

1) SORT :-

The ORDER BY operator is used to display the contents of a relation in a sorted order based on one or more fields.

Syntax

Given below is the syntax of the ORDER BY operator.

```
grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray,
city:chararray);
```

Let us now sort the relation in a descending order based on the age of the student and store it into another relation named order_by_data using the ORDER BY operator as shown below.

```
grunt> order_by_data = ORDER student_details BY age DESC;
```

Verification

Verify the relation order_by_data using the DUMP operator as shown below.

```
grunt> Dump order_by_data;
```

Output

It will produce the following output, displaying the contents of the relation order_by_data.

```
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
(7,Komal,Nayak,24,9848022334,trivendram)
(6,Archana,Mishra,23,9848022335,Chennai)
(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)
(3,Rajesh,Khanna,22,9848022339,Delhi)
(2,siddarth,Battacharya,22,9848022338,Kolkata)
(4,Preethi,Agarwal,21,9848022330,Pune)
(1,Rajiv,Reddy,21,9848022337,Hyderabad)
```

2) GROUP :-

The GROUP operator is used to group the data in one or more relations. It collects the data having the same key.

Syntax

Given below is the syntax of the group operator.

```
grunt> Group_data = GROUP Relation_name BY age;
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Apache Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
```

Now, let us group the records/tuples in the relation by age as shown below.

```
grunt> group_data = GROUP student_details by age;
```

Verification

Verify the relation group_data using the DUMP operator as shown below.

```
grunt> Dump group_data;
```

Output

Then you will get output displaying the contents of the relation named group_data as shown below. Here you can observe that the resulting schema has two columns −

- One is age, by which we have grouped the relation.
- The other is a bag, which contains the group of tuples, student records with the respective age.

```
(21,{(4,Preethi,Agarwal,21,9848022330,Pune),(1,Rajiv,Reddy,21,9848022337,Hydera bad)})
(22,{(3,Rajesh,Khanna,22,9848022339,Delhi),(2,siddarth,Battacharya,22,984802233
8,Kolkata)})
(23,{(6,Archana,Mishra,23,9848022335,Chennai),(5,Trupthi,Mohanthy,23,9848022336
,Bhuwaneshwar)})
(24,{(8,Bharathi,Nambiayar,24,9848022333,Chennai),(7,Komal,Nayak,24,9848022334,
trivendram)})
```

You can see the schema of the table after grouping the data using the describe command as shown below.

```
grunt> Describe group_data;
```

```
group_data: {group: int,student_details: {(id: int,firstname: chararray,
lastname: chararray,age: int,phone: chararray,city: chararray)}}
```

In the same way, you can get the sample illustration of the schema using the illustrate command as shown below.

```
$ Illustrate group_data;
```

It will produce the following output −

```
-----------------------------------------------------------------------------------------------
|group_data| group:int |
student_details:bag{:tuple(id:int,firstname:chararray,lastname:chararray,age:int,phone:chararray,
city:chararray)}|
-----------------------------------------------------------------------------------------------
| | 21 | { 4, Preethi, Agarwal, 21, 9848022330, Pune), (1, Rajiv, Reddy, 21, 9848022337,
Hyderabad)}|
| | 2 |
{(2,siddarth,Battacharya,22,9848022338,Kolkata),(003,Rajesh,Khanna,22,9848022339,Delhi)}|
-----------------------------------------------------------------------------------------------
```

3) FILTER :-

The FILTER operator is used to select the required tuples from a relation based on a condition.

Syntax

Given below is the syntax of the FILTER operator.

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai
```

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
```

Let us now use the Filter operator to get the details of the students who belong to the city Chennai.

```
filter_data = FILTER student_details BY city == 'Chennai';
```

Verification

Verify the relation filter_data using the DUMP operator as shown below.

```
grunt> Dump filter_data;
```

Output

It will produce the following output, displaying the contents of the relation filter_data as follows.

```
(6,Archana,Mishra,23,9848022335,Chennai)
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
```

4) JOIN :-

The right outer join operation returns all rows from the right table, even if there are no matches in the left table.

Syntax

Given below is the syntax of performing right outer join operation using the JOIN operator.

```
grunt> outer_right = JOIN customers BY id RIGHT, orders BY customer_id;
```

Example

Let us perform right outer join operation on the two relations customers and orders as shown below.

```
grunt> outer_right = JOIN customers BY id RIGHT, orders BY customer_id;
```

Verification

Verify the relation outer_right using the DUMP operator as shown below.

```
grunt> Dump outer_right
```

Output

It will produce the following output, displaying the contents of the relation outer_right.

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
```

ASSIGNMENT - 7

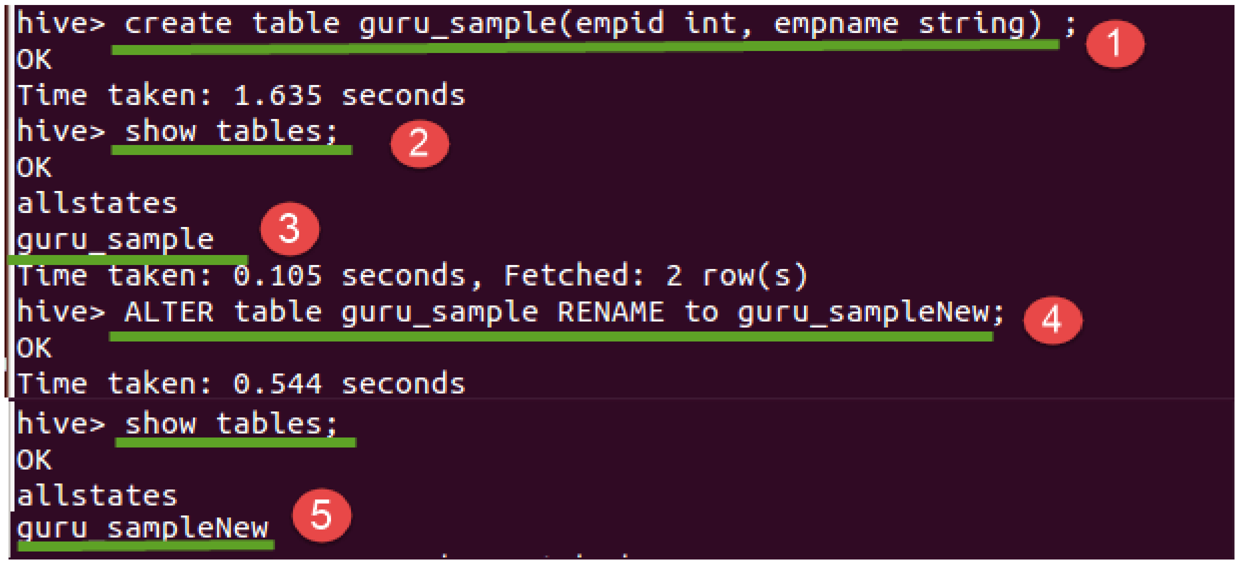
Q1. Install and Run Hive then use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

Ans.

Hive Create Table: Internal Table, Alter, Drop with Examples

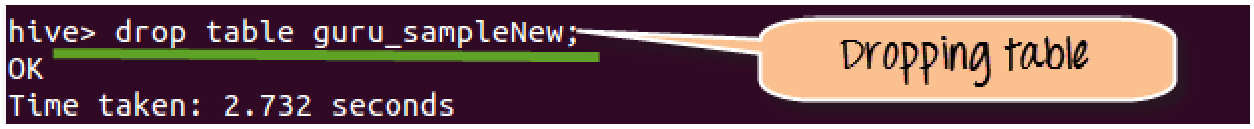
In the below screenshot, we are creating a table with columns and altering the table name.

1. Creating table guru_sample with two column names such as “empid” and “empname”
2. Displaying tables present in guru99 database
3. Guru_sample displaying under tables
4. Altering table “guru_sample” as “guru_sampleNew”
5. Again, when you execute “show” command, it will display the new name Guru_sampleNew



```
hive> create table guru_sample(empid int, empname string) ;
OK
Time taken: 1.635 seconds
hive> show tables;
OK
allstates
guru_sample
Time taken: 0.105 seconds, Fetched: 2 row(s)
hive> ALTER table guru_sample RENAME to guru_sampleNew;
OK
Time taken: 0.544 seconds
hive> show tables;
OK
allstates
guru_sampleNew
```

Dropping table guru_sampleNew:



```
hive> drop table guru_sampleNew;
OK
Time taken: 2.732 seconds
```

Table types and its Usage:

Coming to Tables it's just like the way that we create in traditional Relational Databases. The functionalities such as filtering, joins can be performed on the tables.
Hive deals with two types of table structures like Internal and External tables depending on the loading and design of schema in Hive.

Internal tables

- Internal Table is tightly coupled in nature. In this type of table, first we have to create table and load the data.
- We can call this one as data on schema.
- By dropping this table, both data and schema will be removed.
- The stored location of this table will be at /user/hive/warehouse.

When to Choose Internal Table?

- If the processing data available in local file system
- If we want Hive to manage the complete lifecycle of data including the deletion

Sample code Snippet for Internal Table

1. To create the internal table

```
Hive>CREATE TABLE guruhive_internaltable (id INT,Name STRING);
         Row format delimited
         Fields terminated by '\t';
```

2. Load the data into internal table

```
  Hive>LOAD DATA INPATH '/user/guru99hive/data.txt' INTO table guruhive_internaltable;
```

3. Display the content of the table

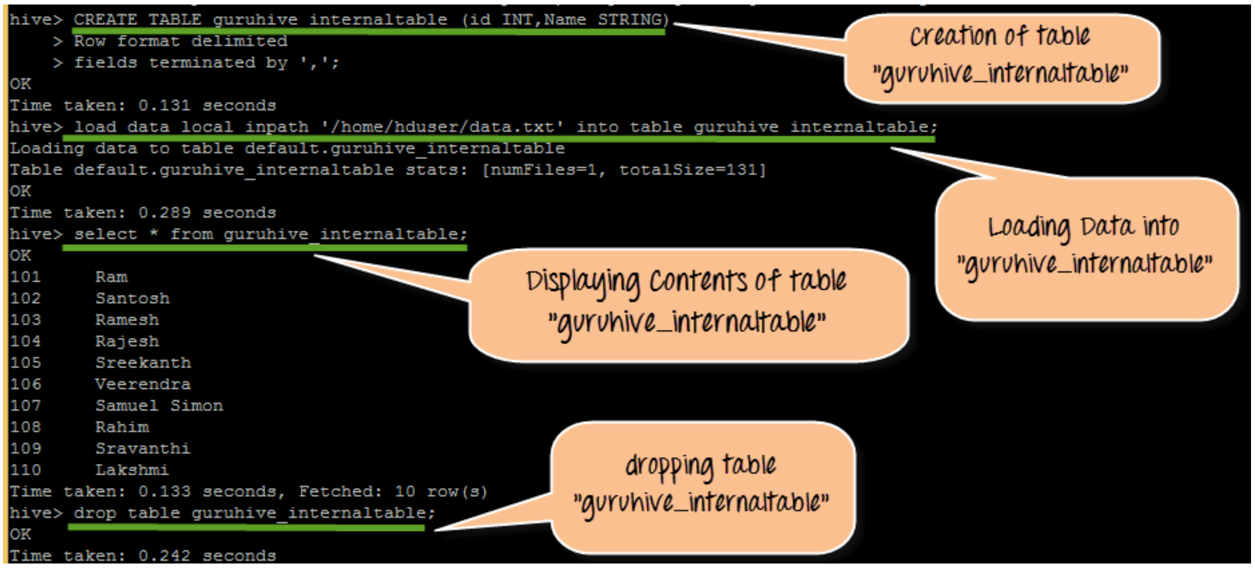
```
  Hive>select * from guruhive_internaltable;
```

4. To drop the internal table

```
  Hive>DROP TABLE guruhive_internaltable;
```

If you dropped the guruhive_internaltable, including its metadata and its data will be deleted from Hive.

From the following screenshot, we can observe the output



```
hive> CREATE TABLE guruhive internaltable (id INT,Name STRING)
    > Row format delimited
    > fields terminated by ',';
OK
Time taken: 0.131 seconds
hive> load data local inpath '/home/hduser/data.txt' into table guruhive internaltable;
Loading data to table default.guruhive_internaltable
Table default.guruhive_internaltable stats: [numFiles=1, totalSize=131]
OK
Time taken: 0.289 seconds
hive> select * from guruhive_internaltable;
OK
101     Ram
102     Santosh
103     Ramesh
104     Rajesh
105     Sreekanth
106     Veerendra
107     Samuel Simon
108     Rahim
109     Sravanthi
110     Lakshmi
Time taken: 0.133 seconds, Fetched: 10 row(s)
hive> drop table guruhive_internaltable;
OK
Time taken: 0.242 seconds
```

creation of table "guruhive_internaltable"
Loading Data into "guruhive_internaltable"
Displaying Contents of table "guruhive_internaltable"
dropping table "guruhive_internaltable"

In above code and from screenshot we do following things,

- Create the internal table
- Load the data into internal table
- Display the content of the table
- To drop the internal table

External tables

- External Table is loosely coupled in nature. Data will be available in HDFS.The table is going to create on HDFS data.
- In other way, we can say like its creating schema on data.
- At the time of dropping the table it drops only schema, the data will be still available in HDFS as before.
- External tables provide an option to create multiple schemas for the data stored in HDFS instead of deleting the data every time whenever schema updates

When to Choose External Table?

- If processing data available in HDFS
- Useful when the files are being used outside of Hive

Sample code Snippet for External Table

1. Create External table

```
Hive>CREATE EXTERNAL TABLE guruhive_external(id INT,Name STRING)
        Row format delimited
        Fields terminated by '\t'
        LOCATION '/user/guru99hive/guruhive_external;
```

2. If we are not specifying the location at the time of table creation, we can load the data manually

```
  Hive>LOAD DATA INPATH '/user/guru99hive/data.txt' INTO TABLE guruhive_external;
```

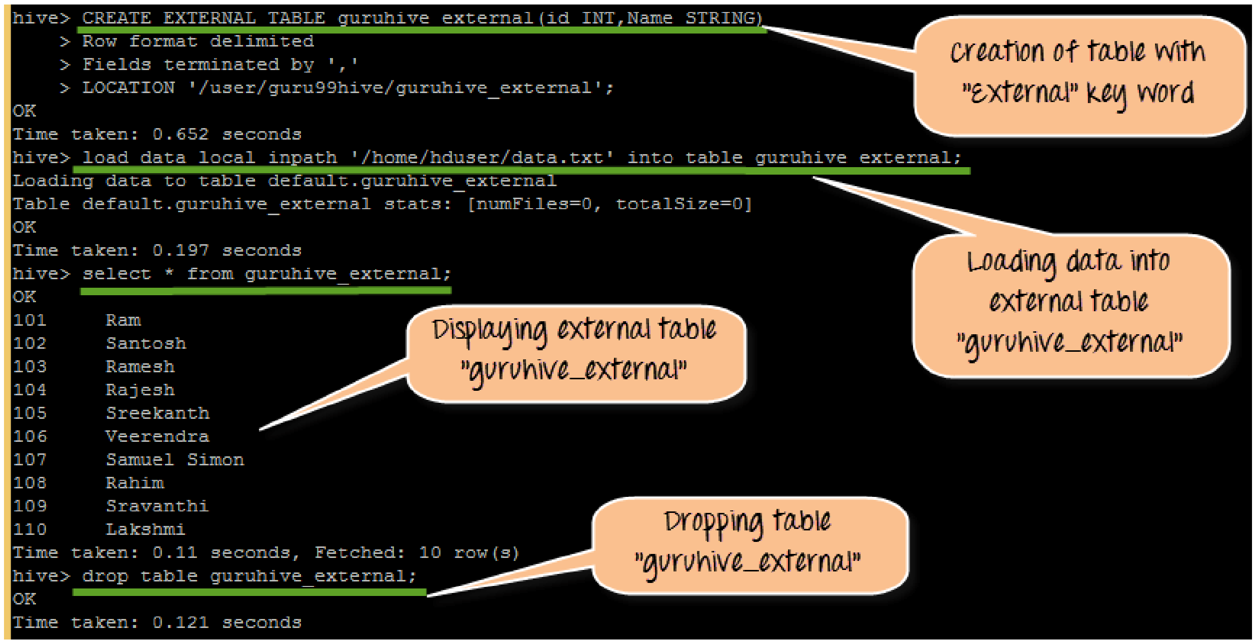
3. Display the content of the table

```
 Hive>select * from guruhive_external;
```

4. To drop the internal table

```
 Hive>DROP TABLE guruhive_external;
```

From the following screenshot, we can observe the output



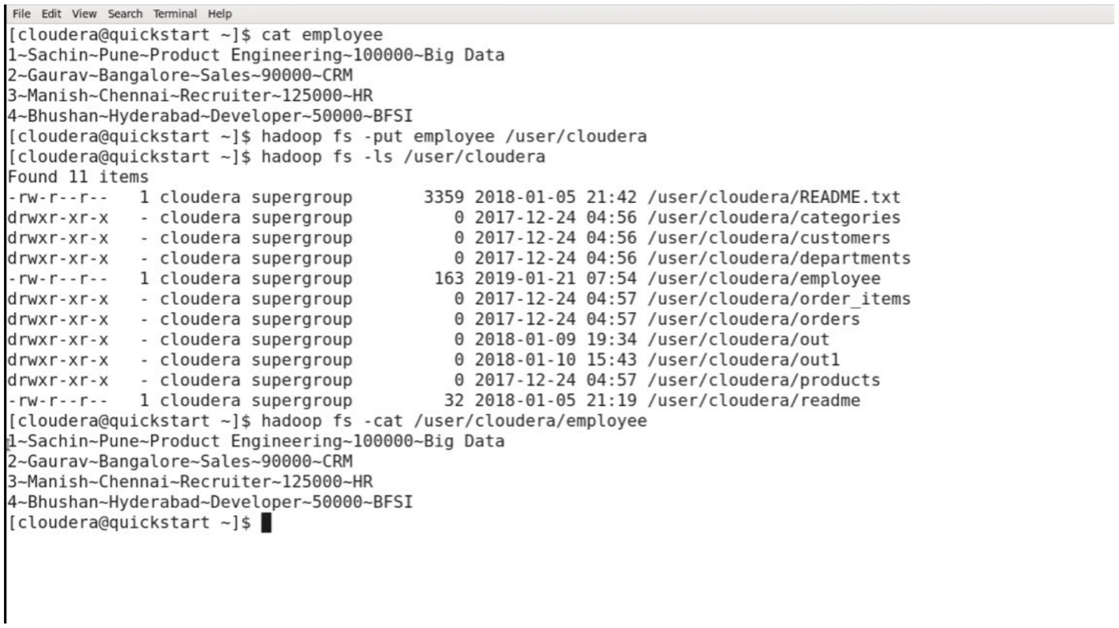
```
hive> CREATE EXTERNAL TABLE guruhive_external(id INT,Name STRING)
    > Row format delimited
    > Fields terminated by ','
    > LOCATION '/user/guru99hive/guruhive_external';
OK
Time taken: 0.652 seconds
hive> load data local inpath '/home/hduser/data.txt' into table guruhive_external;
Loading data to table default.guruhive_external
Table default.guruhive_external stats: [numFiles=0, totalSize=0]
OK
Time taken: 0.197 seconds
hive> select * from guruhive_external;
OK
101     Ram
102     Santosh
103     Ramesh
104     Rajesh
105     Sreekanth
106     Veerendra
107     Samuel Simon
108     Rahim
109     Sravanthi
110     Lakshmi
Time taken: 0.11 seconds, Fetched: 10 row(s)
hive> drop table guruhive_external;
OK
Time taken: 0.121 seconds
```

creation of table with "External" key word
Loading data into external table "guruhive_external"
Displaying external table "guruhive_external"
Dropping table "guruhive_external"

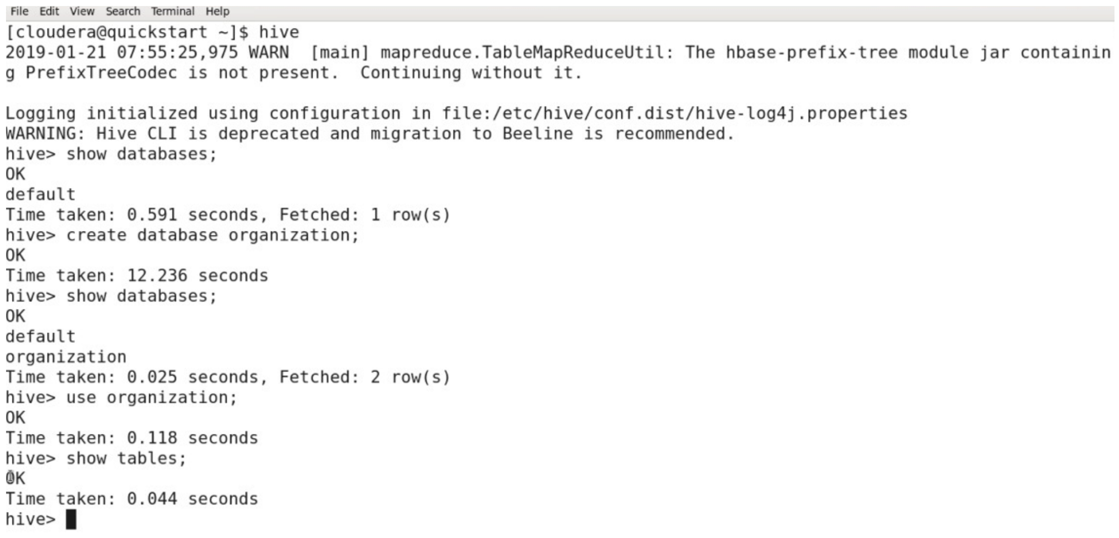
In above code, we do following things

- Create the External table
- Load the data into External table
- Display the content of the table
- Dropping external table

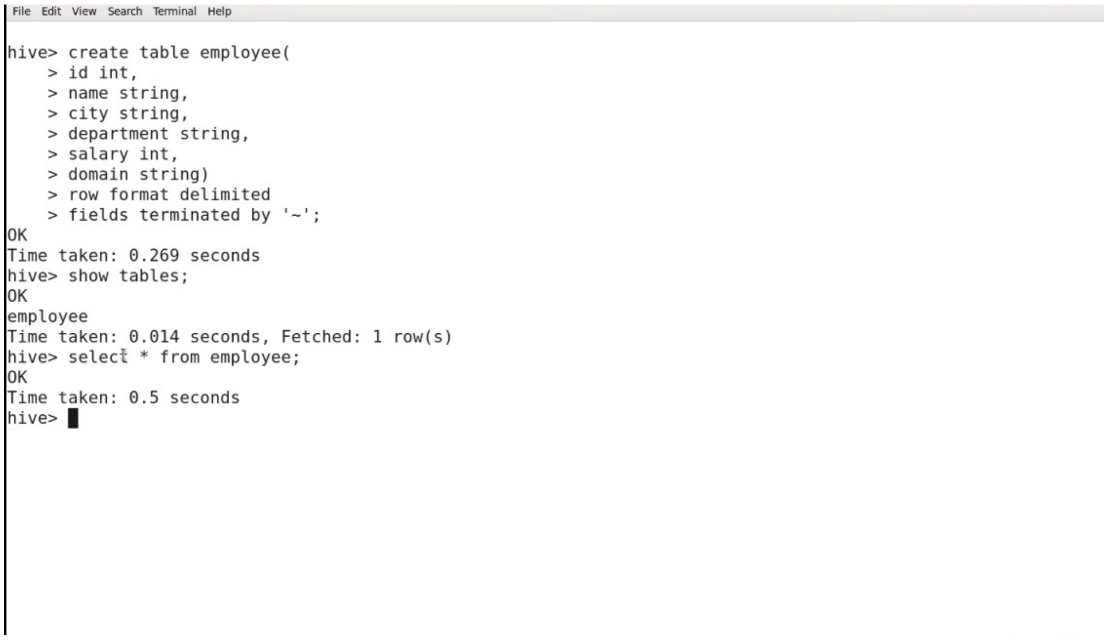
Creating Database :



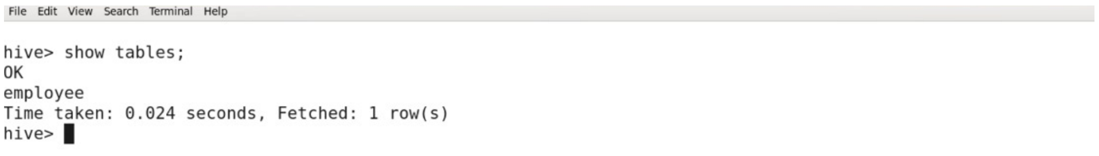
```
[cloudera@quickstart ~]$ cat employee
1~Sachin~Pune~Product Engineering~100000~Big Data
2~Gaurav~Bangalore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
4~Bhushan~Hyderabad~Developer~50000~BFSI
[cloudera@quickstart ~]$ hadoop fs -put employee /user/cloudera
[cloudera@quickstart ~]$ hadoop fs -ls /user/cloudera
Found 11 items
-rw-r--r--   1 cloudera supergroup       3359 2018-01-05 21:42 /user/cloudera/README.txt
drwxr-xr-x   - cloudera supergroup          0 2017-12-24 04:56 /user/cloudera/categories
drwxr-xr-x   - cloudera supergroup          0 2017-12-24 04:56 /user/cloudera/customers
drwxr-xr-x   - cloudera supergroup          0 2017-12-24 04:56 /user/cloudera/departments
-rw-r--r--   1 cloudera supergroup        163 2019-01-21 07:54 /user/cloudera/employee
drwxr-xr-x   - cloudera supergroup          0 2017-12-24 04:57 /user/cloudera/order_items
drwxr-xr-x   - cloudera supergroup          0 2017-12-24 04:57 /user/cloudera/orders
drwxr-xr-x   - cloudera supergroup          0 2018-01-09 19:34 /user/cloudera/out
drwxr-xr-x   - cloudera supergroup          0 2018-01-10 15:43 /user/cloudera/out1
drwxr-xr-x   - cloudera supergroup          0 2017-12-24 04:57 /user/cloudera/products
-rw-r--r--   1 cloudera supergroup         32 2018-01-05 21:19 /user/cloudera/readme
[cloudera@quickstart ~]$ hadoop fs -cat /user/cloudera/employee
1~Sachin~Pune~Product Engineering~100000~Big Data
2~Gaurav~Bangalore~Sales~90000~CRM
3~Manish~Chennai~Recruiter~125000~HR
4~Bhushan~Hyderabad~Developer~50000~BFSI
[cloudera@quickstart ~]$ 
```



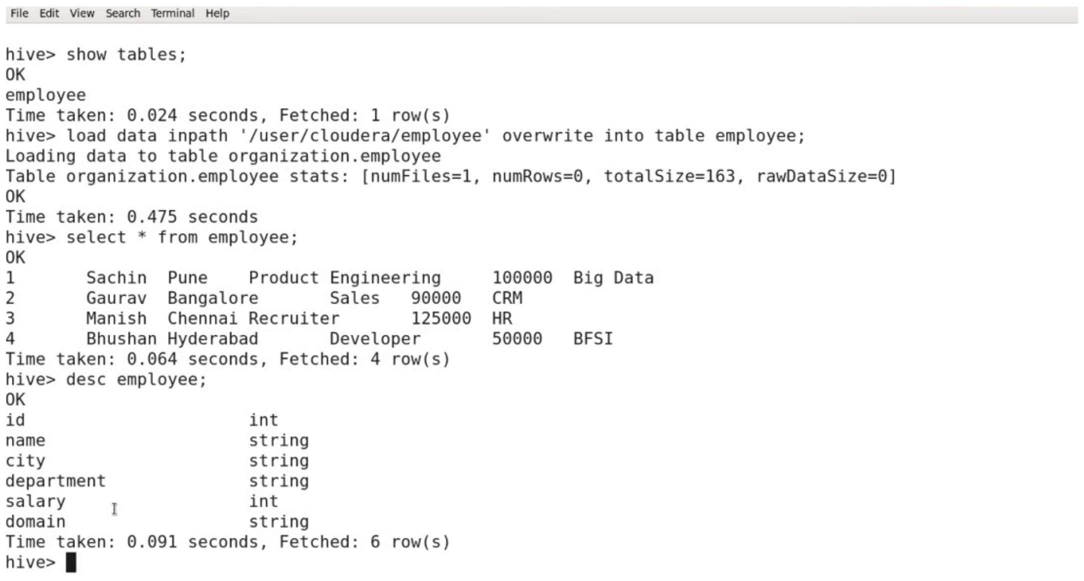
```
[cloudera@quickstart ~]$ hive
2019-01-21 07:55:25,975 WARN  [main] mapreduce.TableMapReduceUtil: The hbase-prefix-tree module jar containin
g PrefixTreeCodec is not present.  Continuing without it.

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> show databases;
OK
default
Time taken: 0.591 seconds, Fetched: 1 row(s)
hive> create database organization;
OK
Time taken: 12.236 seconds
hive> show databases;
OK
default
organization
Time taken: 0.025 seconds, Fetched: 2 row(s)
hive> use organization;
OK
Time taken: 0.118 seconds
hive> show tables;
OK
Time taken: 0.044 seconds
hive> 
```



```
hive> create table employee(
    > id int,
    > name string,
    > city string,
    > department string,
    > salary int,
    > domain string)
    > row format delimited
    > fields terminated by '~';
OK
Time taken: 0.269 seconds
hive> show tables;
OK
employee
Time taken: 0.014 seconds, Fetched: 1 row(s)
hive> select * from employee;
OK
Time taken: 0.5 seconds
hive> █
```



```
hive> show tables;
OK
employee
Time taken: 0.024 seconds, Fetched: 1 row(s)
hive> █
```



```
hive> show tables;
OK
employee
Time taken: 0.024 seconds, Fetched: 1 row(s)
hive> load data inpath '/user/cloudera/employee' overwrite into table employee;
Loading data to table organization.employee
Table organization.employee stats: [numFiles=1, numRows=0, totalSize=163, rawDataSize=0]
OK
Time taken: 0.475 seconds
hive> select * from employee;
OK
1       Sachin  Pune    Product Engineering     100000  Big Data
2       Gaurav  Bangalore       Sales   90000   CRM
3       Manish  Chennai Recruiter       125000  HR
4       Bhushan Hyderabad       Developer       50000   BFSI
Time taken: 0.064 seconds, Fetched: 4 row(s)
hive> desc employee;
OK
id                      int
name                    string
city                    string
department              string
salary                  int
domain                  string
Time taken: 0.091 seconds, Fetched: 6 row(s)
hive> 
```