**NAME: SHRADDHA YADAV**

**ROLL NUMBER: 516**

**COURSE: MSc CS**

**SUBJECT: BUSINESS INTELLIGENCE & BIG DATA ANALYSIS**

**PRACTICAL: 1-9**

|  |  |  |  |
| --- | --- | --- | --- |
| SR No | Name | Date | Sign |
| 1 | To Install Cloudera QuickStart VM on VMware |  |  |
| 2 | Map-Reduce Program for WordCount Problem |  |  |
| 3 | PIG Script for Solving Counting Problems |  |  |
| 4 | Install HBase and Use HBase Data Model to Store and Retrieve Databases |  |  |
| 5 | Install Hive and Use Hive to Create and Store Structured Databases |  |  |
| 6 | Construct Different Types of K-Shingles for Given Document |  |  |
| 7 | Measuring Similarity Among Documents and Detecting Passages Which Have Been Reused |  |  |
| 8 | Computing n-moment |  |  |
| 9 | Alon-Matias-Szegedy Algorithm |  |  |

**Practical 1: To Install Cloudera QuickStart VM on VMware**

Cloudera is software that provides a platform for data analytics, data warehousing, and machine learning. Initially, Cloudera started as an open-source Apache Hadoop distribution project, commonly known as Cloudera Distribution for Hadoop or CDH. It contains Apache Hadoop and other related projects where all the components are 100% open-source under Apache License.

Cloudera provides virtual machine images of complete Apache Hadoop clusters, making it easy to get started with Cloudera CDH. This assignment covers the following topics related to Cloudera QuickStart VM Installation:

1. What is Cloudera QuickStart VM?
2. Prerequisites for Cloudera QuickStart VM Installation
3. Downloading the Cloudera QuickStart VM
4. Cloudera QuickStart VM Installation on Windows

What is Cloudera QuickStart VM?

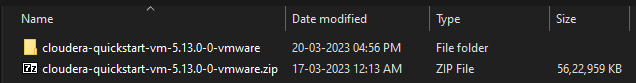
Cloudera QuickStart VM includes everything you need for using CDH, Impala, Cloudera Search, and Cloudera Manager. The Cloudera QuickStart VM uses a package-based install that allows you to work with or without the Cloudera Manager. It provides a sample of Cloudera's platform for "Big Data."

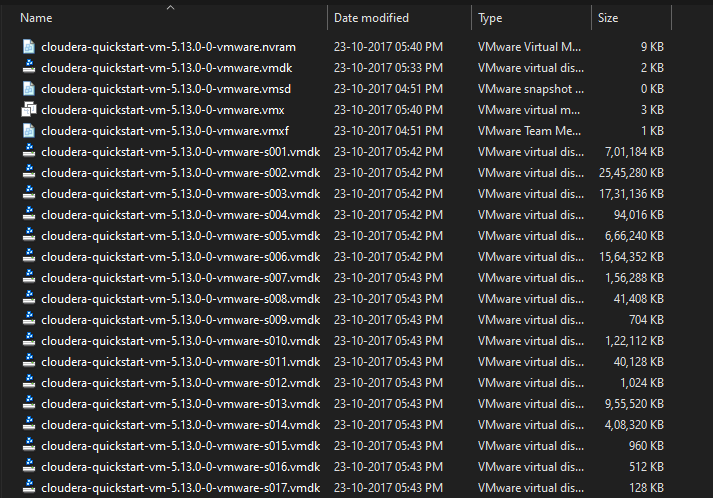
Prerequisites for Cloudera QuickStart VM Installation

* A virtual machine such as Oracle VirtualBox or VMware
* RAM of 12+ GB, that is 4+ GB for the operating system and 8+ GB for Cloudera
* 80 GB hard disk
* Oracle VirtualBox downloaded from https://www.virtualbox.org/wiki/Downloads and installed on your system

Downloading the Cloudera QuickStart VM

* Cloudera QuickStart VMs are available as Zip archives in VirtualBox, VMware, and KVM formats. To download the VM, go to https://www.cloudera.com/downloads.html and select the appropriate version of CDH that you require.
* Click on the "GET IT NOW" button, and it will prompt you to fill in your details.
* Once the file is downloaded, go to the download folder and unzip the files. They can then be used to set up a single-node Cloudera cluster.

The two virtual images of Cloudera QuickStart VM are shown below:



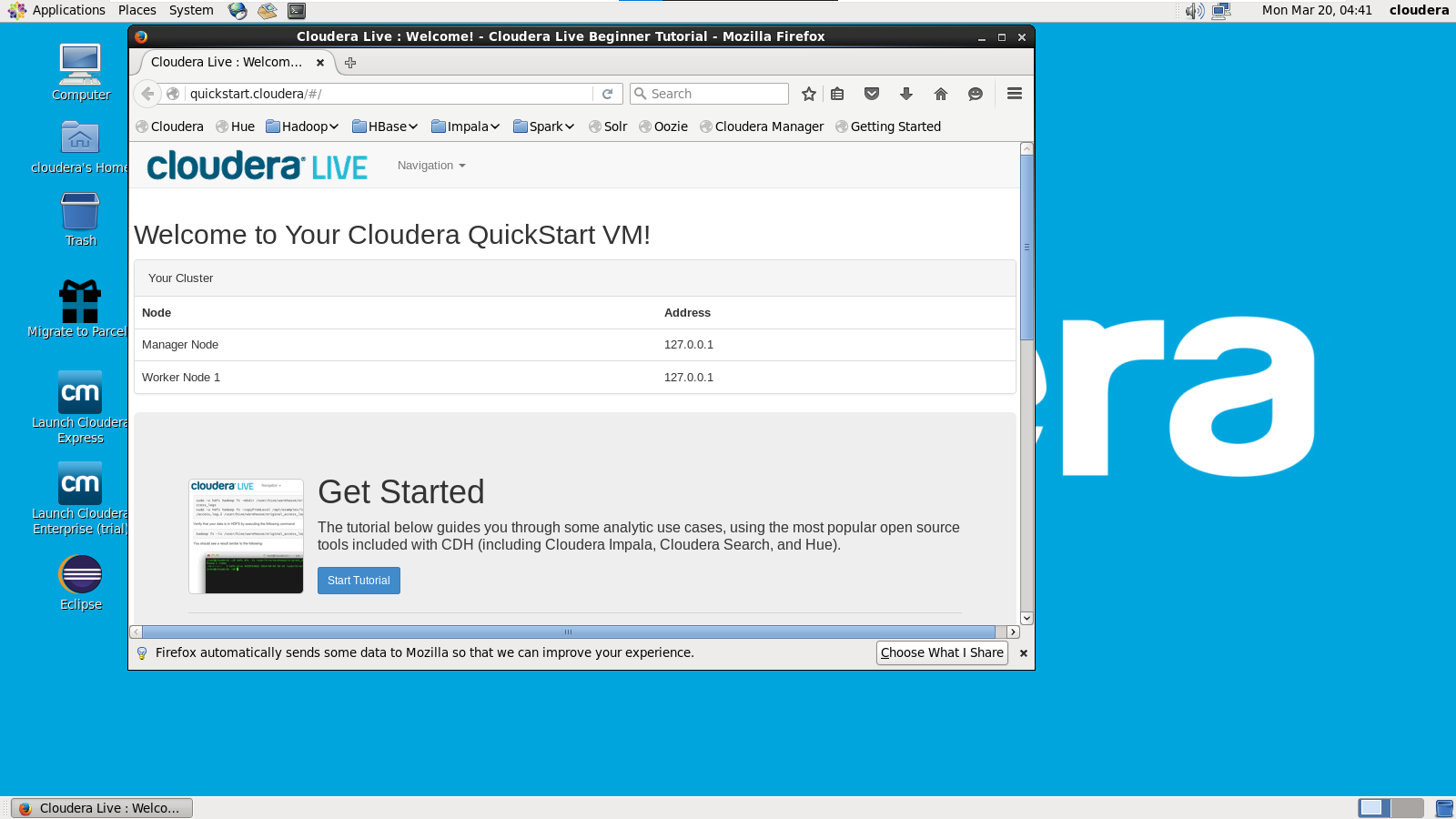
* Now that the downloading process is done with, let's move forward with this Cloudera QuickStart VM Installation guide and see the actual process.

Cloudera QuickStart VM Installation

* Before setting up the Cloudera Virtual Machine, you would need to have a virtual machine such as VMware or Oracle VirtualBox on your system.
* In this case, we are using Oracle VirtualBox to set up the Cloudera QuickStart VM.
* In order to download and install the Oracle VirtualBox on your operating system, click on the following link: Oracle VirtualBox(https://www.virtualbox.org/wiki/Downloads).
* To set up the Cloudera QuickStart VM in your Oracle VirtualBox Manager, click on file with extension as “.VMX “ and then it will automatically open in VMware Workstation.
* Once complete you can see the Cloudera QuickStart VM on the left side panel.



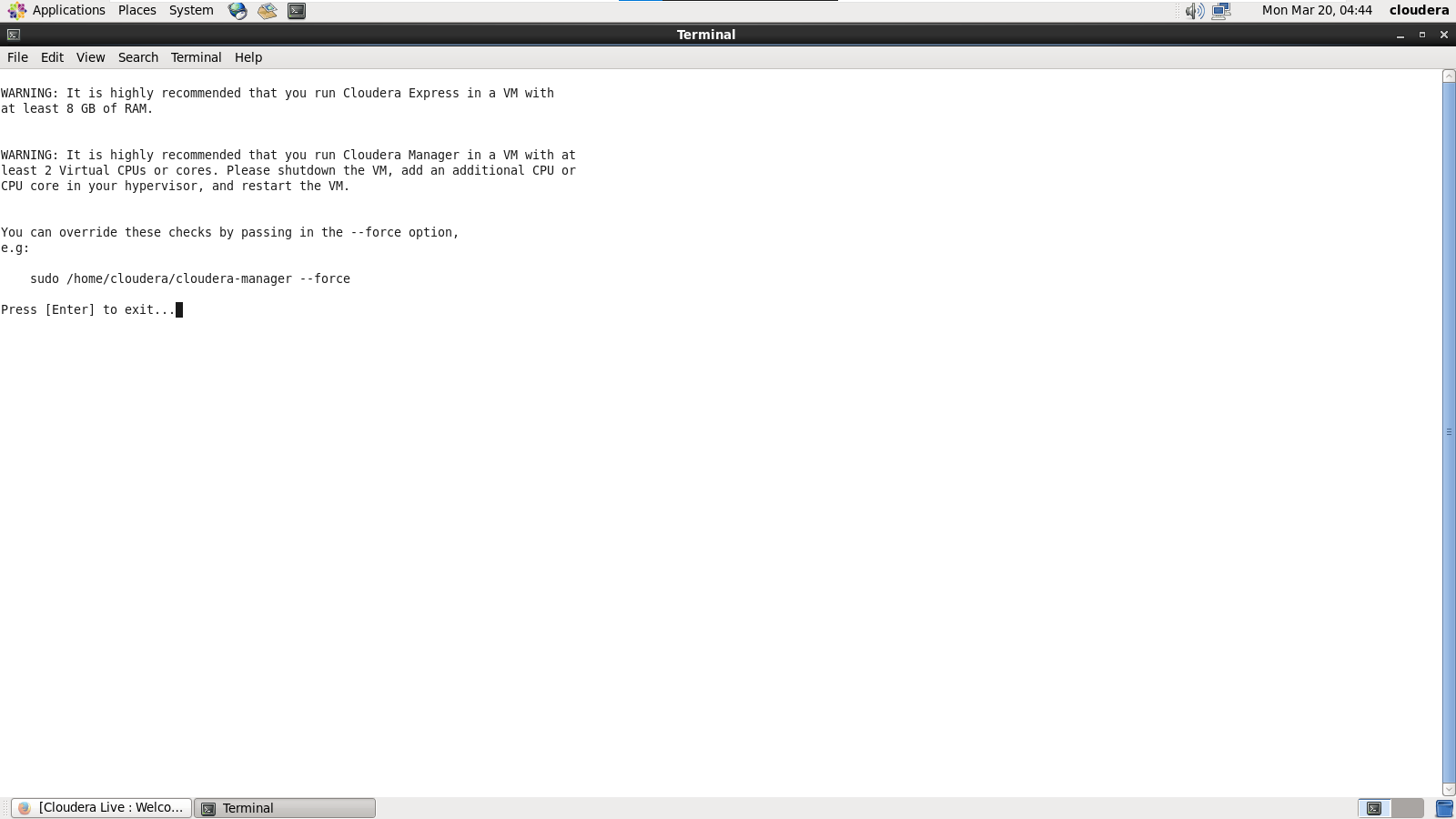
* The next step will be going ahead and starting the machine by clicking the ‘Start’ symbol on top.
* Once your machine comes on, it will look like this:



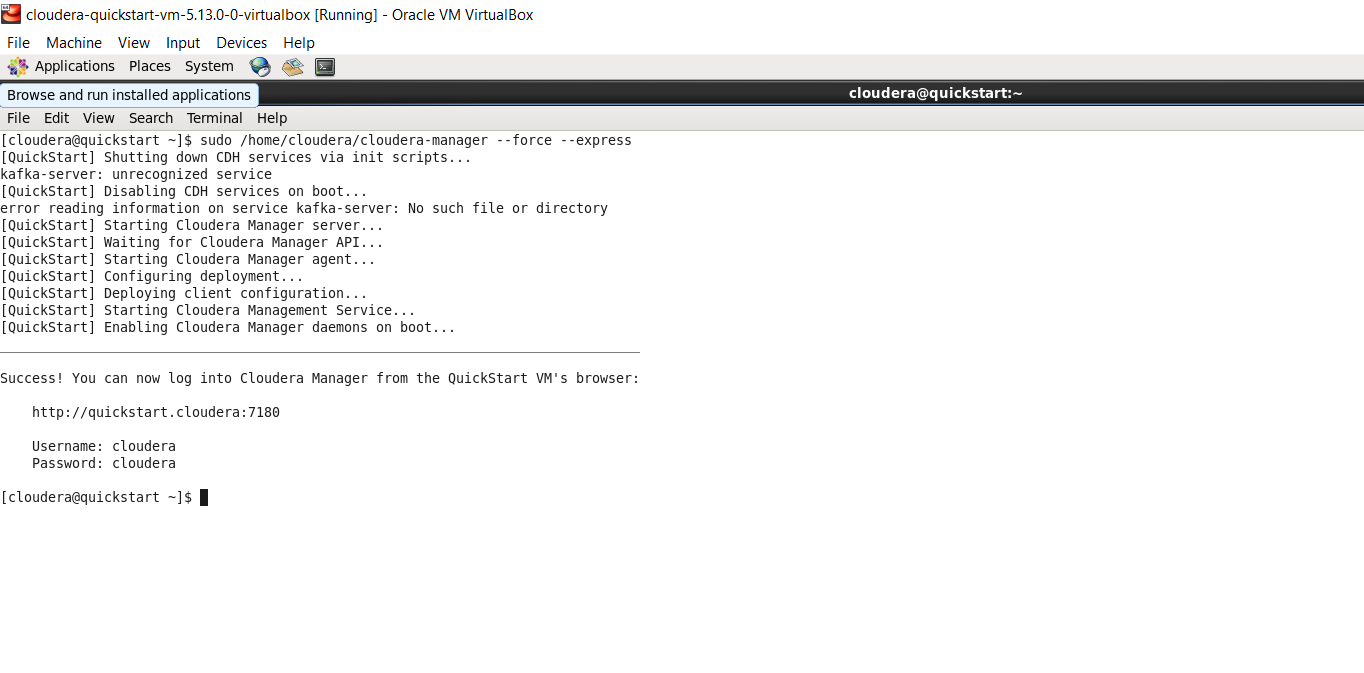
•Next, we have to follow a few steps to gain admin console access. You need to click on the terminal present on top of the desktop screen, and type in the following:

1. hostname # This shows the hostname which will be quickstart.cloudera
2. hdfs dfs -ls / # Checks if you have access and if your cluster is working. It displays what exists on your HDFS location by default
3. service cloudera-scm-server status # Tells what command you have to type to use cloudera express free
4. su - #Login as root
5. service cloudera-scm-server status # The password for root is cloudera

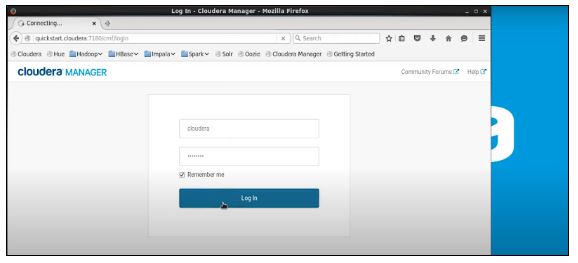
* Once you see that your HDFS access is working fine, you can close the terminal. Then, you have to click on the following icon that says ‘Launch Cloudera Express’.



* You are required to copy the command, and run it on a separate terminal. Hence, open a new terminal, and use the below command to close the Cloudera based services. It will restart the services, after which you can access your admin console.



* Now that our deployment has been configured, client configurations have also been deployed. Additionally, it has restarted the Cloudera Management Service, which gives access to the Cloudera QuickStart admin console with the help of a username and password.
* Go on and open up the browser and change the port number to 7180.
* You can log in to the Cloudera Manager by providing your username and password.



* You can go ahead and restart the services now. It will ensure that the cluster becomes accessible either by Hue as a web interface or Cloudera QuickStart Terminal, where you can write your commands.

**Practical 2: Map-Reduce Program for WordCount Problem**

Commands:

[cloudera@quickstart ~]$ hdfs dfs -ls /

[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /inputdirectory

[cloudera@quickstart ~]$ hdfs dfs -ls /

[cloudera@quickstart ~]$ cat>/home/cloudera/processfile.txt

[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -put /home/cloudera/processfile.txt /inputdirectory

[cloudera@quickstart ~]$ hdfs dfs -ls /inputdirectory

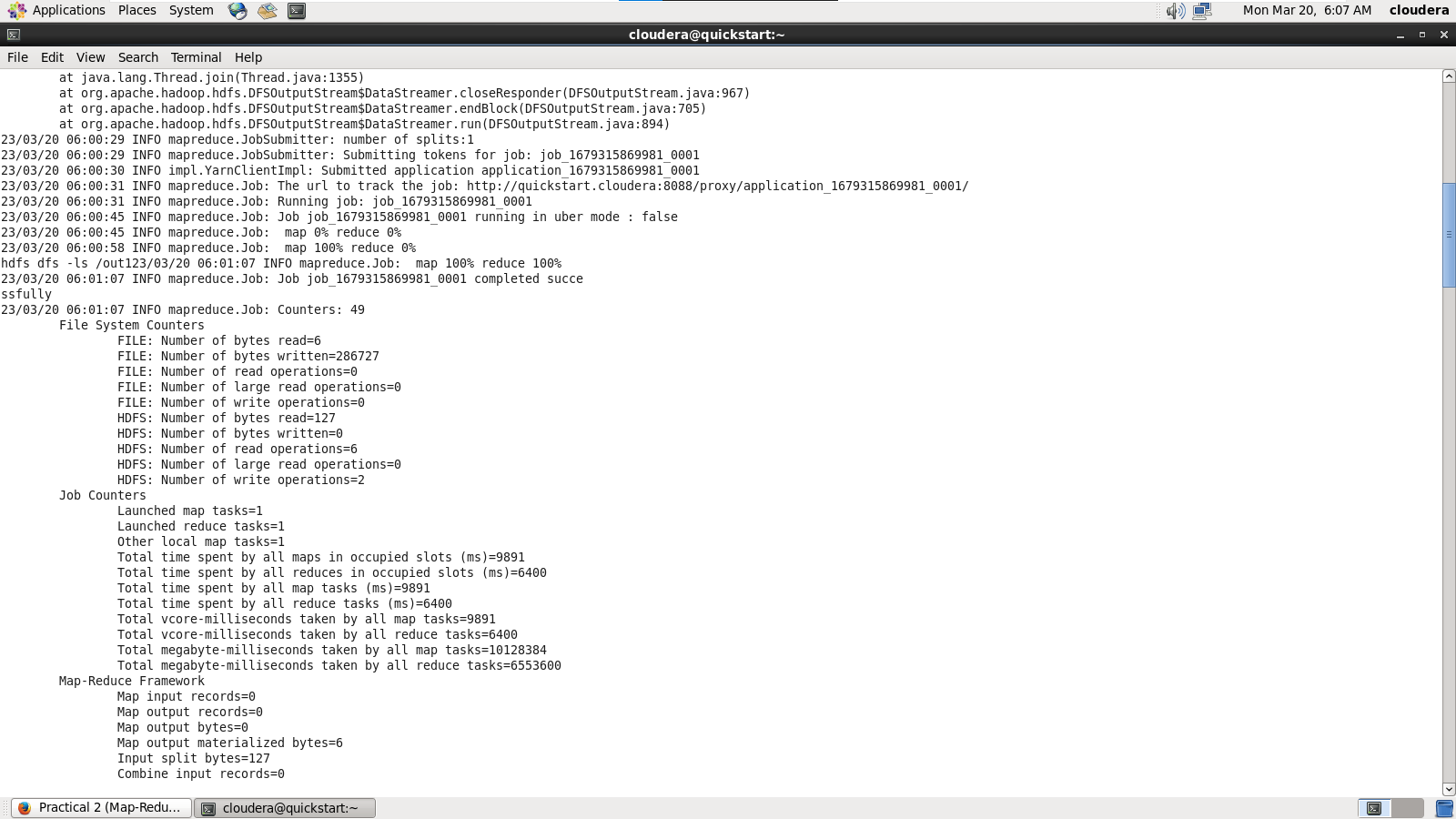
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/WordCount.jar WordCount /inputdirectory/processfile.txt /out1

[cloudera@quickstart ~]$ hdfs dfs -ls /out1

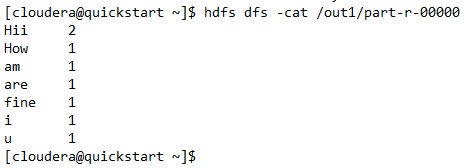
[cloudera@quickstart ~]$ hdfs dfs -cat /out1/part-r-00000

**OUTPUT**









**Practical 3: PIG Script for Solving Counting Problems**

Commands:

cat> /home/cloudera/input.csv

cat  /home/cloudera/input.csv

pig -x local

lines = load '/home/cloudera/input.csv' as (line:chararray);

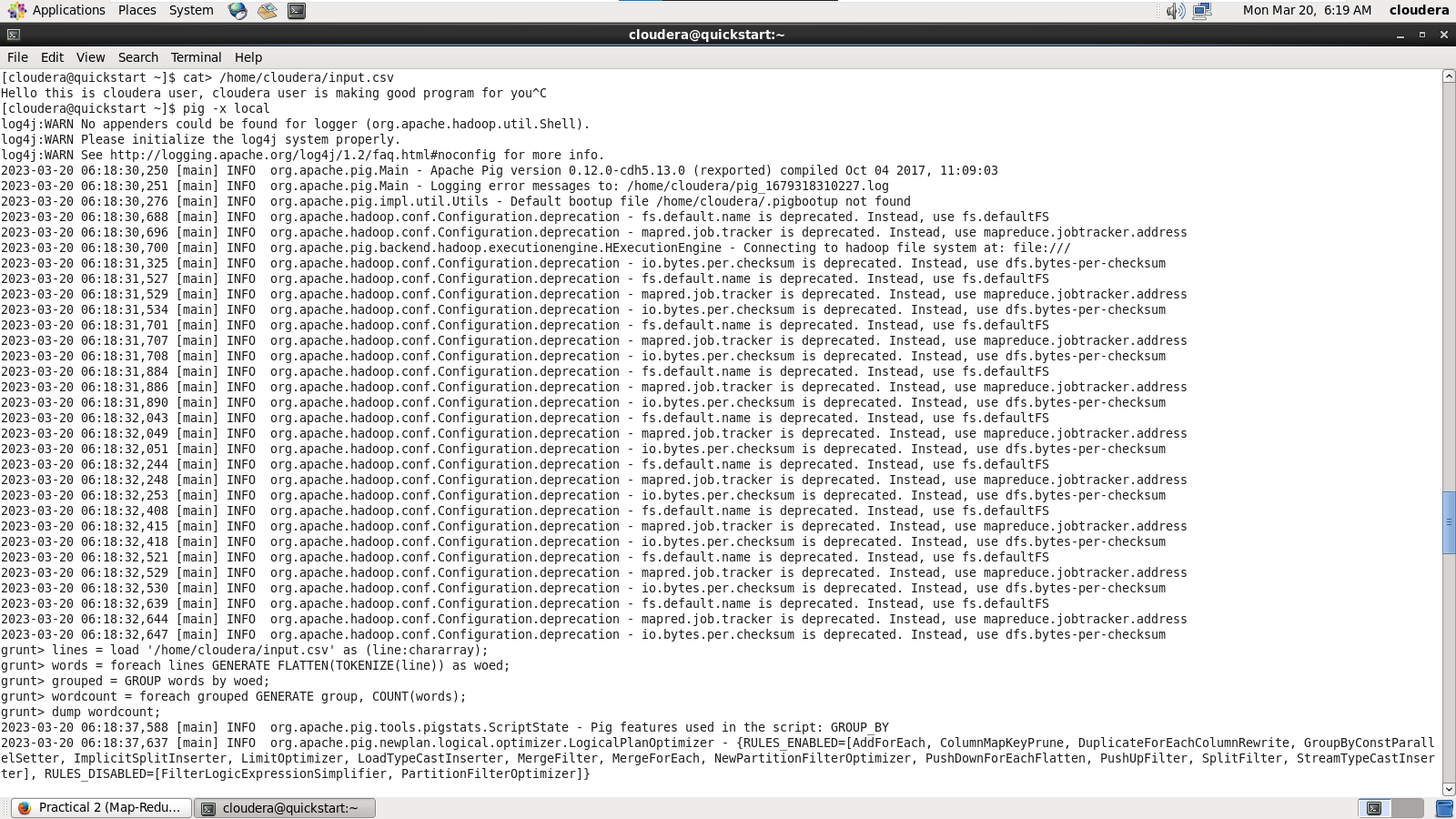
words = foreach lines GENERATE FLATTEN(TOKENIZE(line)) as woed;

grouped = GROUP words by woed;

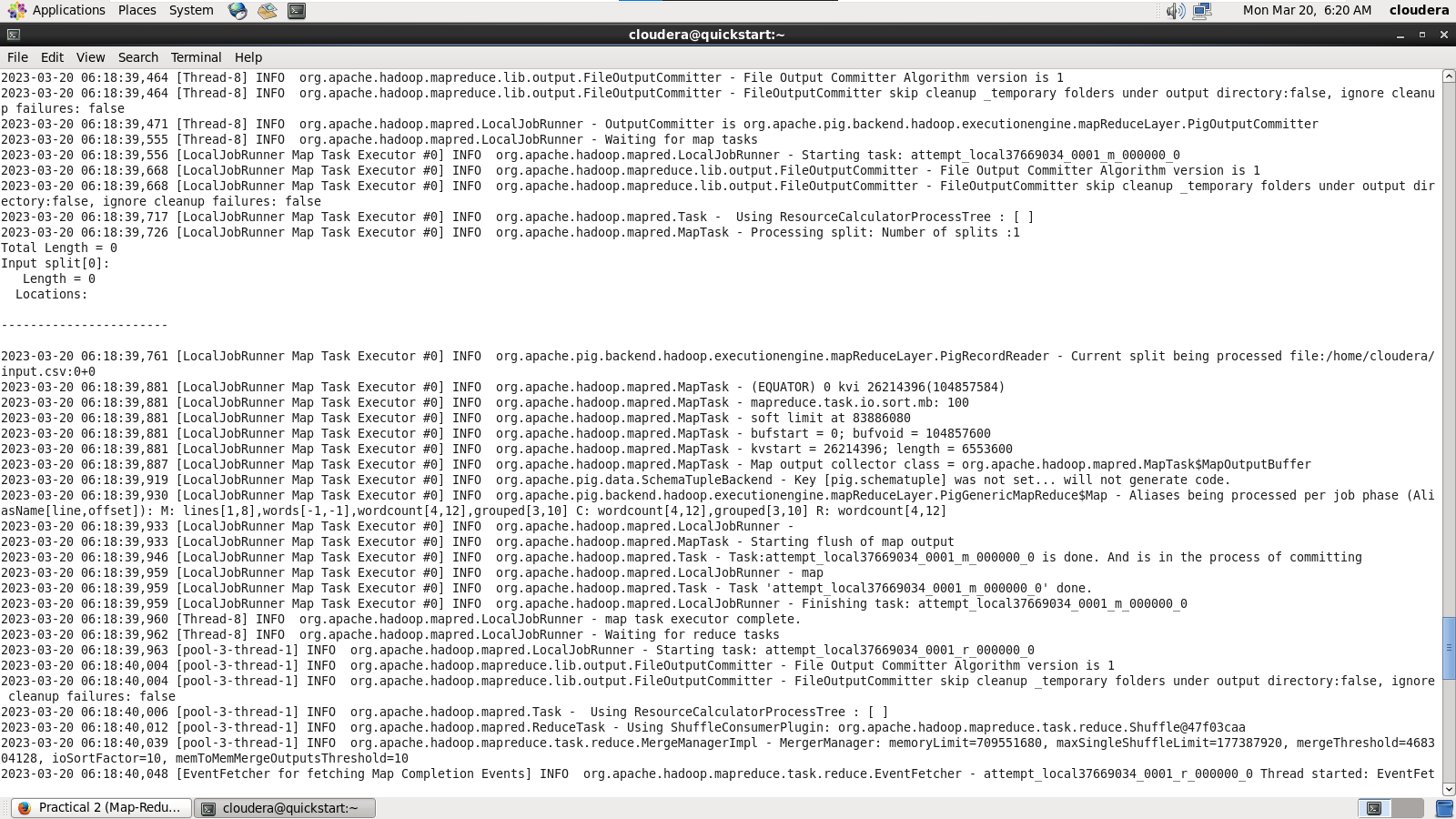
wordcount = foreach grouped GENERATE group, COUNT(words);

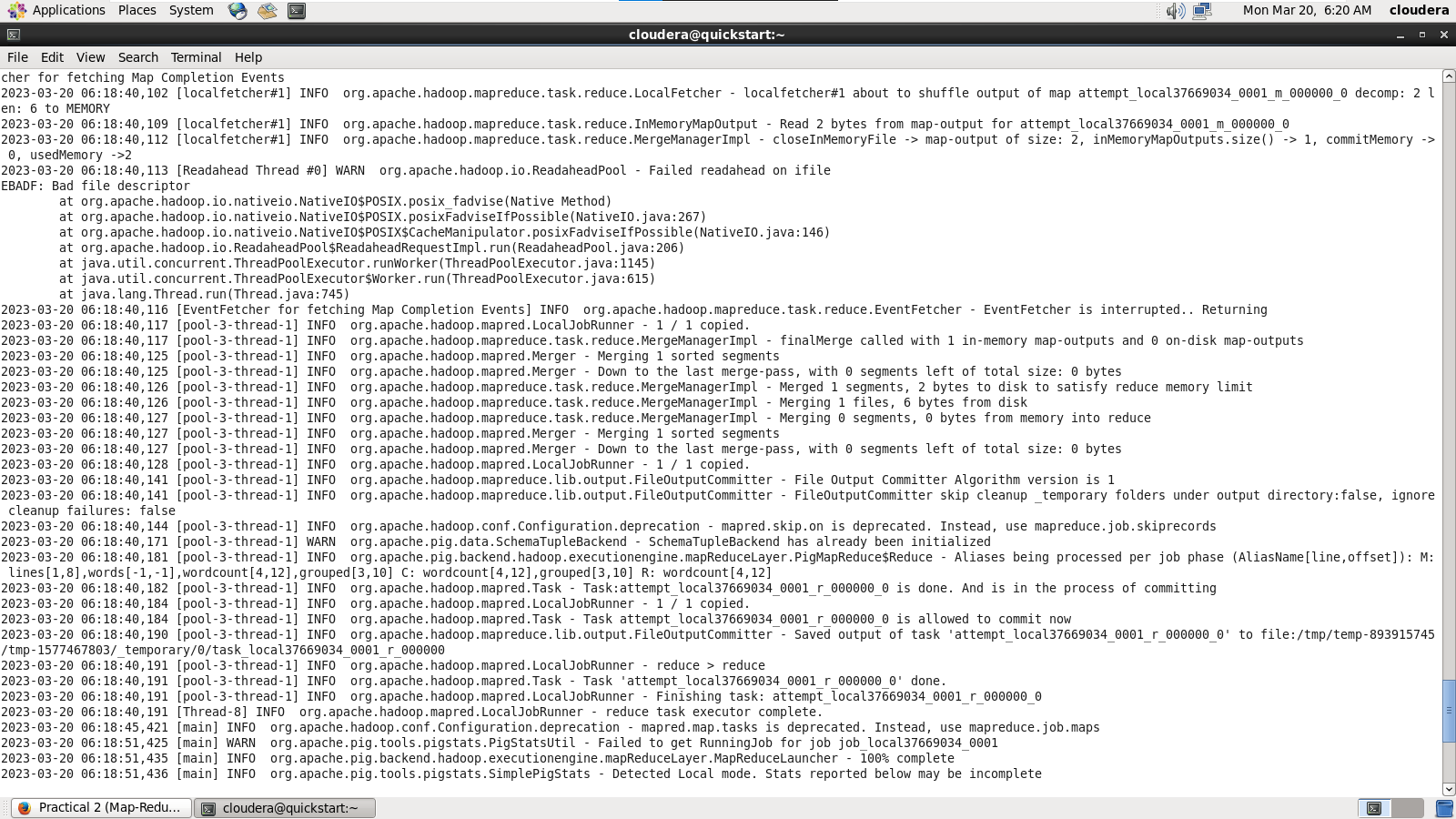
dump wordcount;

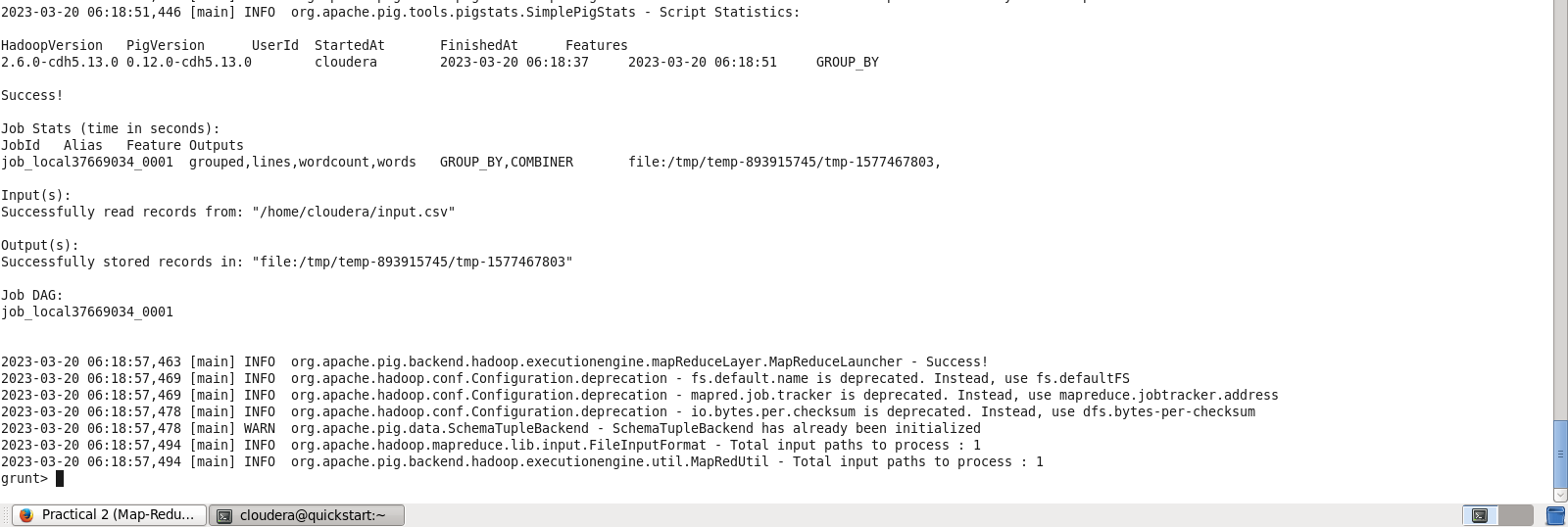
**OUTPUT**













**Practical 4: Install HBase and Use HBase Data Model to Store and Retrieve Databases**

Commands:

hbase shell

status

version,

table\_help

whoami

create ‘employee’, ’Name’, ’ID’, ’Designation’, ’Salary’, ’Department’

list

disable ‘employee’ (or is\_disable ‘employee’)

scan ‘employee’

disable\_all ‘e.\*’

enable‘employee’ (or scan ‘is\_enabled'employee')

//create new table

create‘student’, ‘name’, ‘age’, ‘course’

put ‘student’, ‘sharath’, ‘name:fullname’, ‘sharathkumar’

put ‘student’, ‘sharath’, ‘age:presentage’, ‘24’

put ‘student’, ‘sharath’, ‘course:pursuing’, ‘Hadoop’

put ‘student’, ‘shashank’, ‘name:fullname’, ‘shashank R

put ‘student’, ‘shashank’, ‘age:presentage’, ‘23’

put ‘student’, ‘shashank’, ‘course:pursuing’, ‘Java’

//Get Information

get ‘student’, ‘shashank’

get ‘student’, ‘sharath’

get ‘student’, ‘sharath’, ‘course’

get ‘student’, ‘shashank’, ‘course’

get ‘student’, ‘sharath’, ‘name’

scan ‘student’

Count ‘student’

//Alter

alter ‘student’, NAME=>’name’, VERSIONS=>5

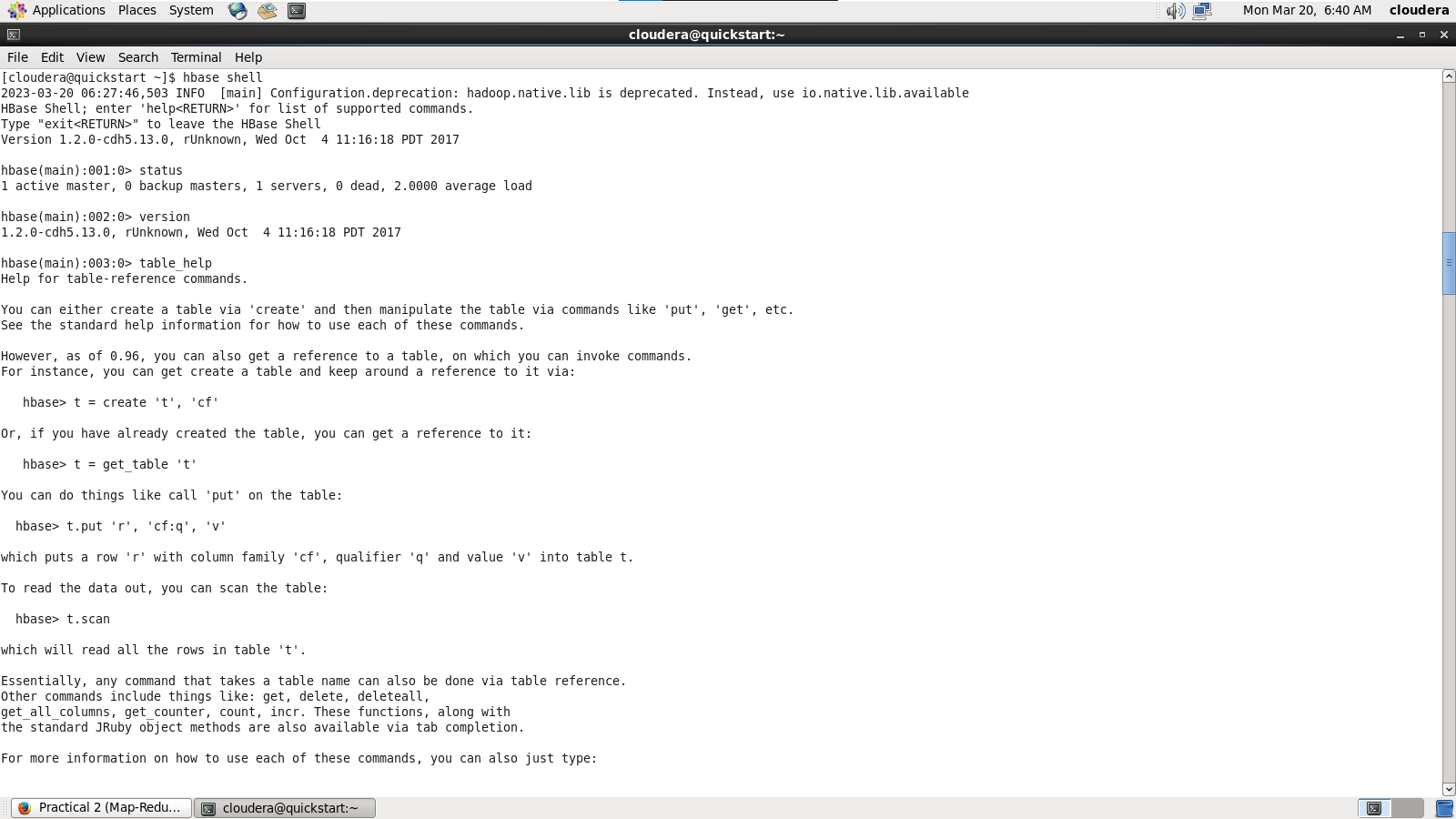
put ‘student’, ‘shashank’, ‘name:fullname’, ‘shashank Rao’

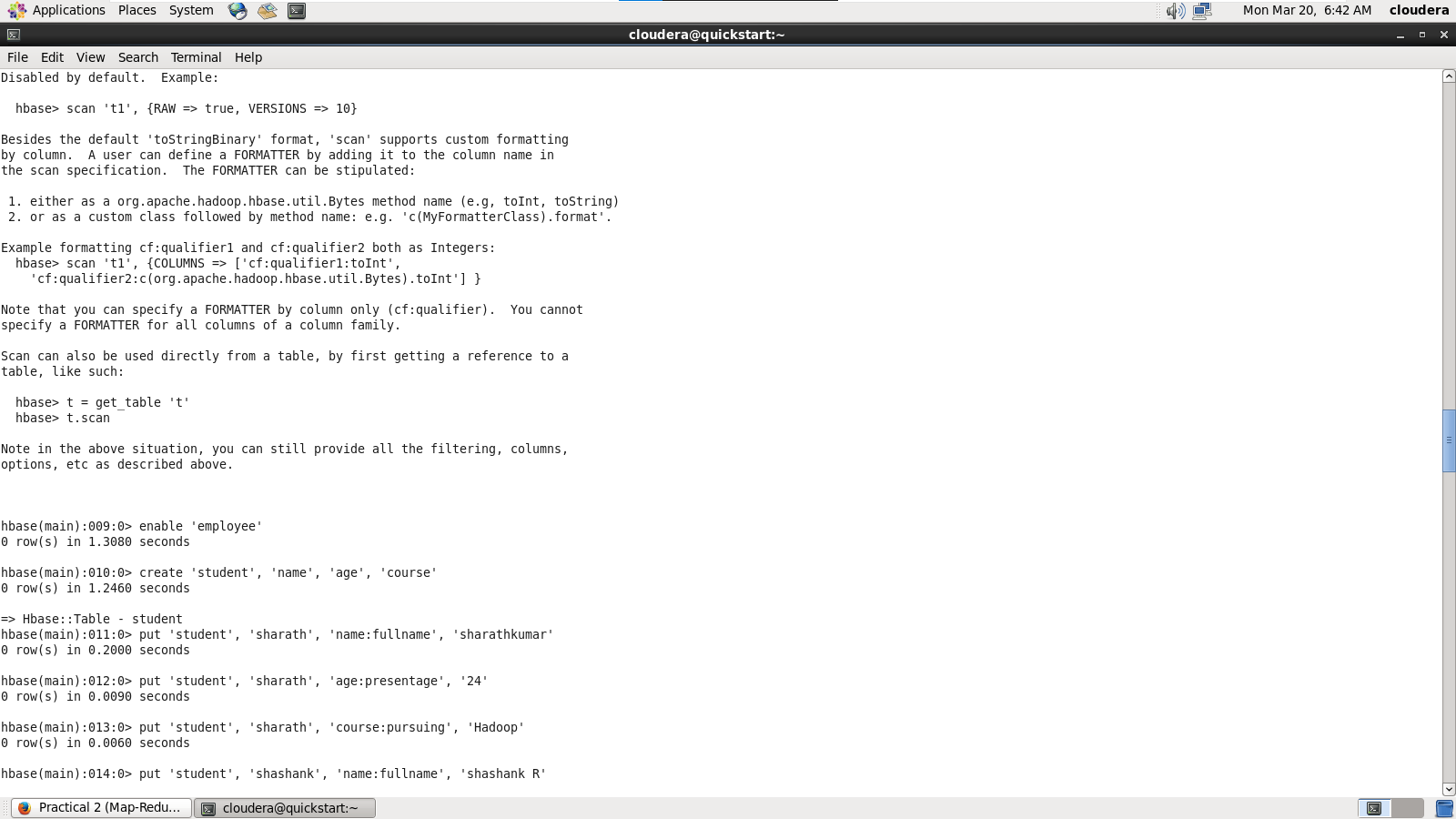
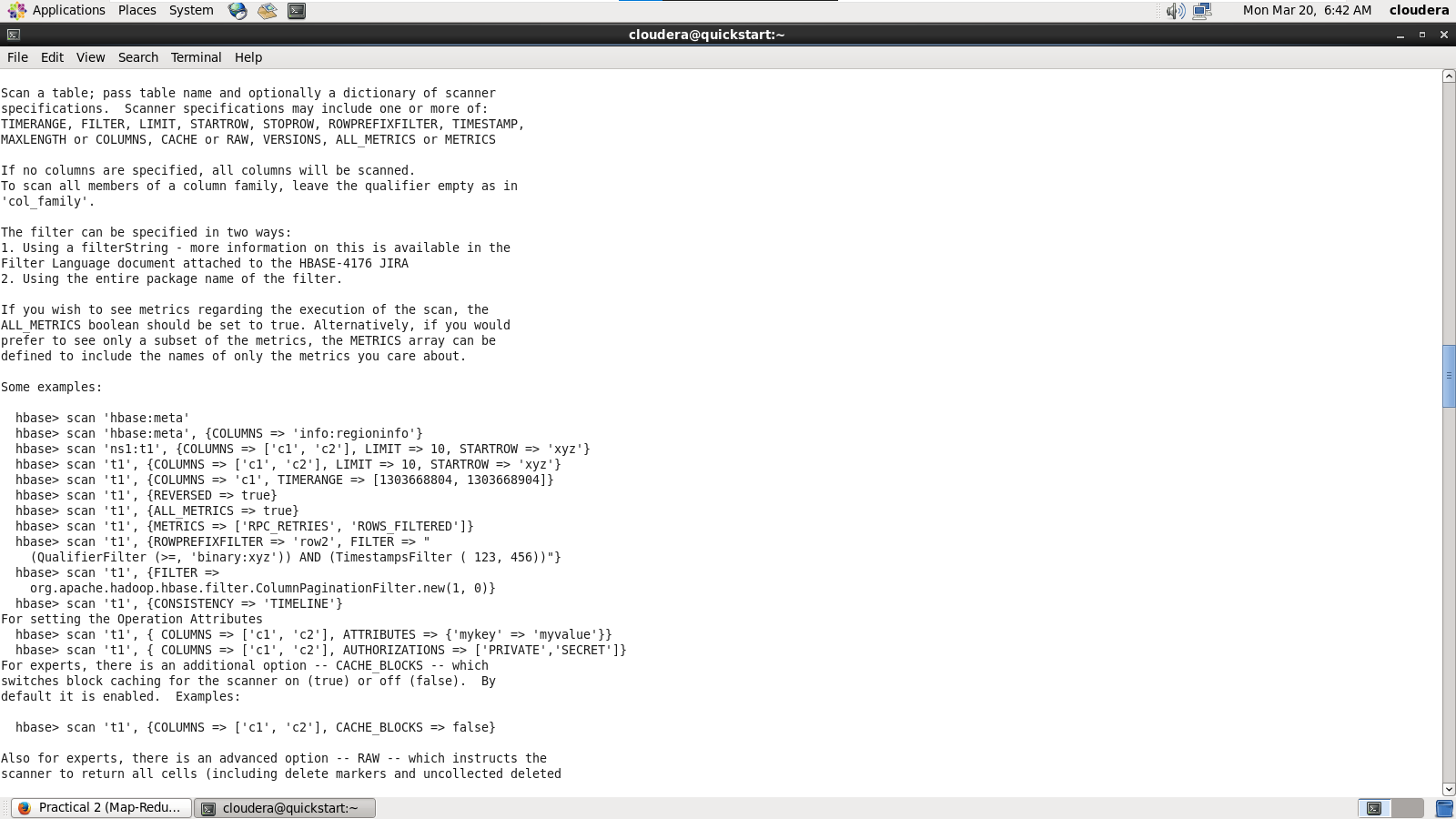
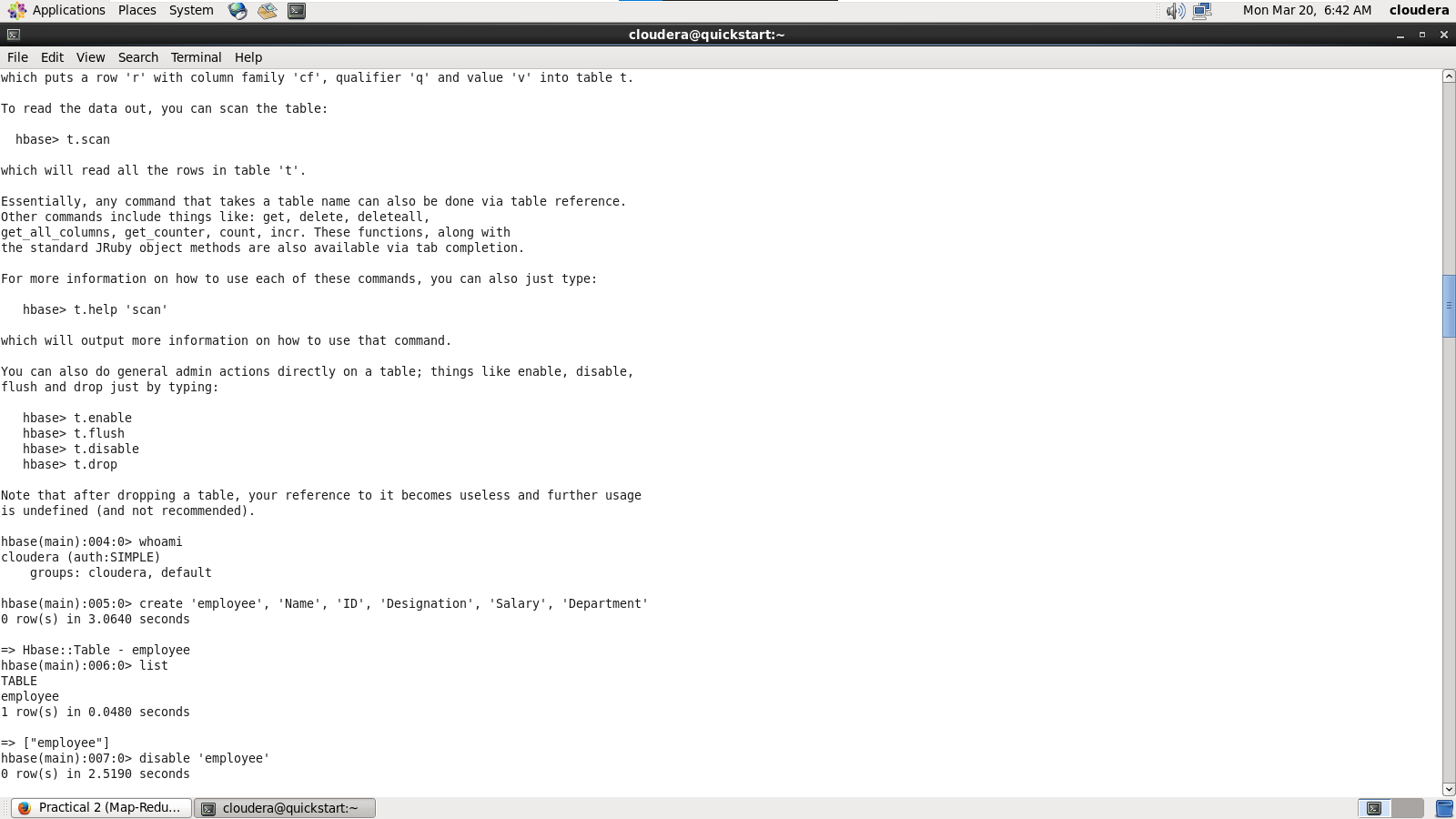
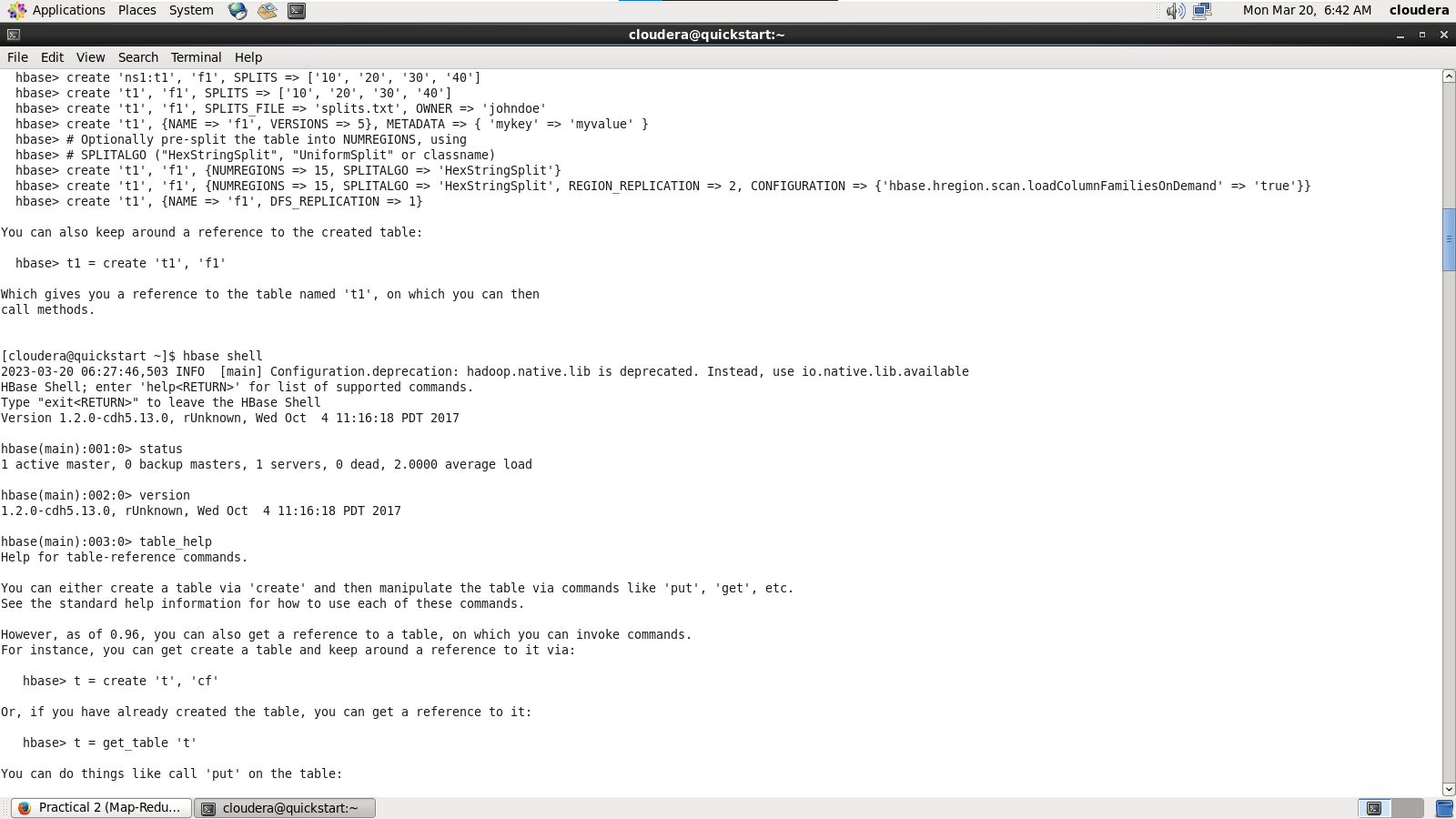
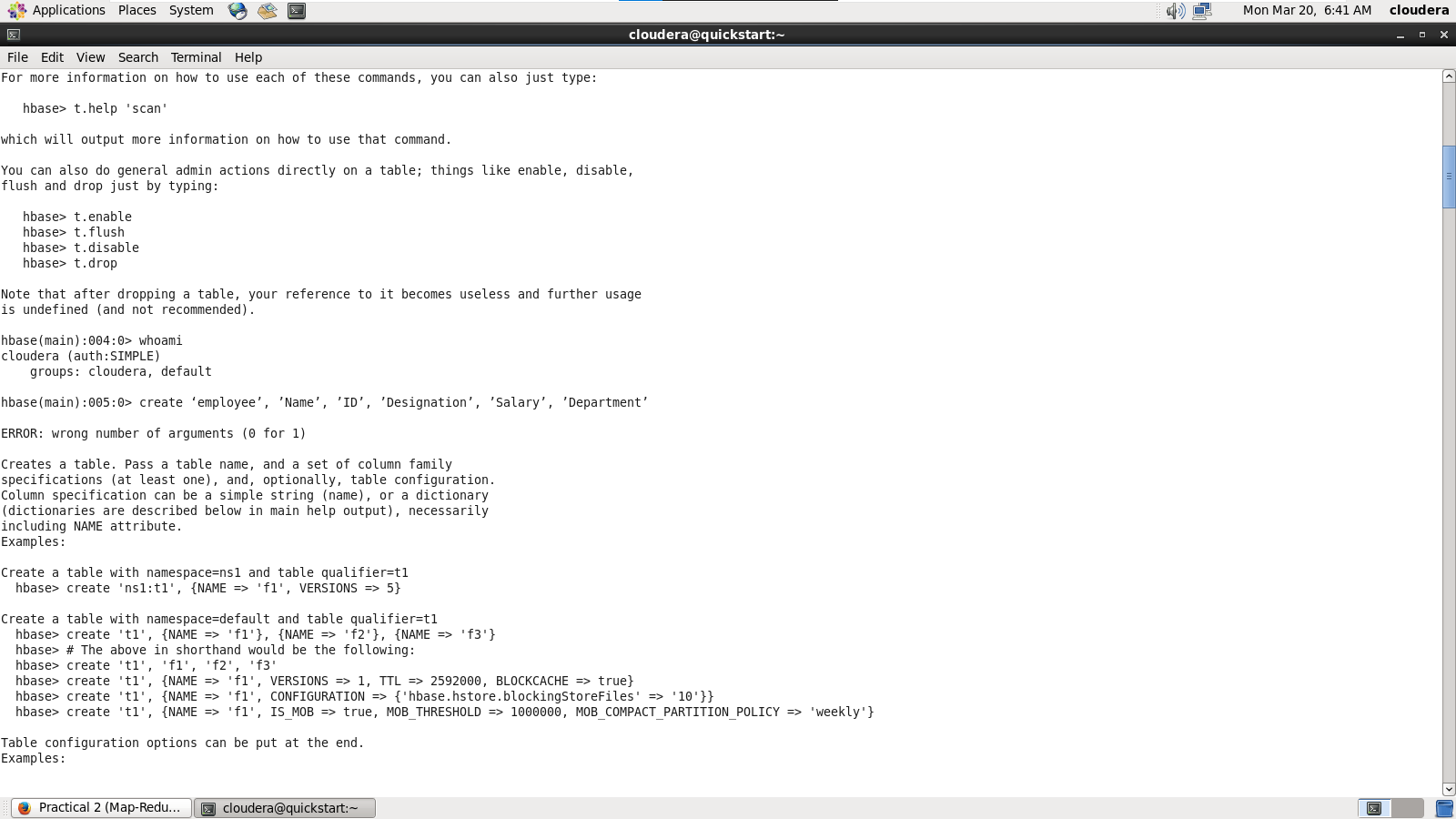
scan ‘student’

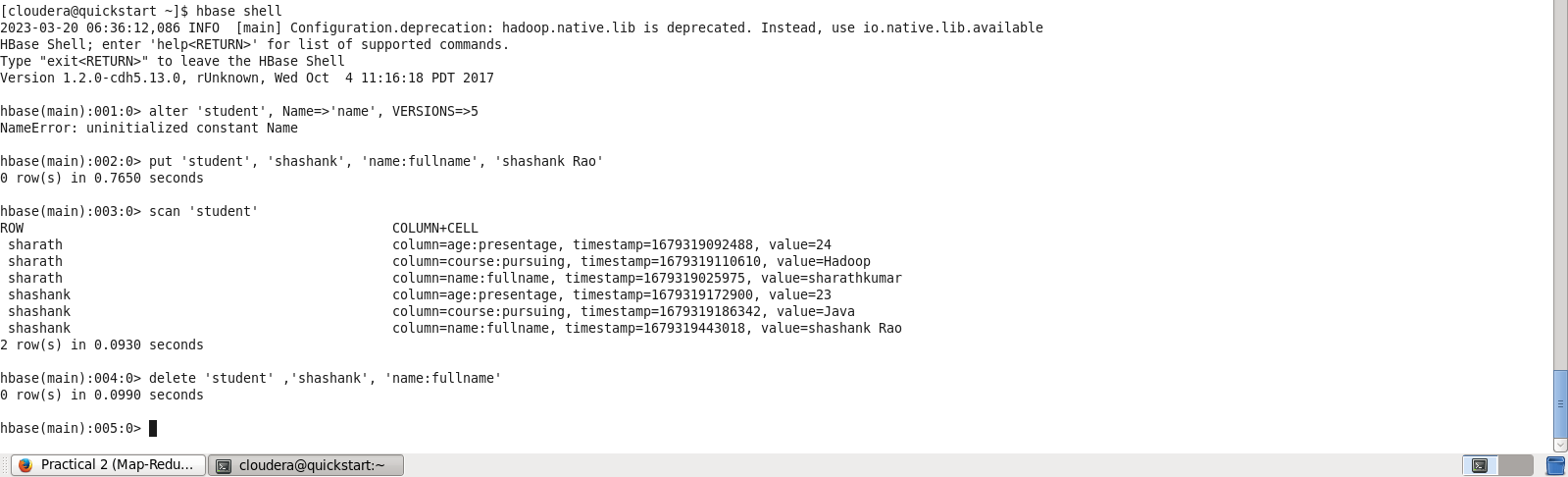
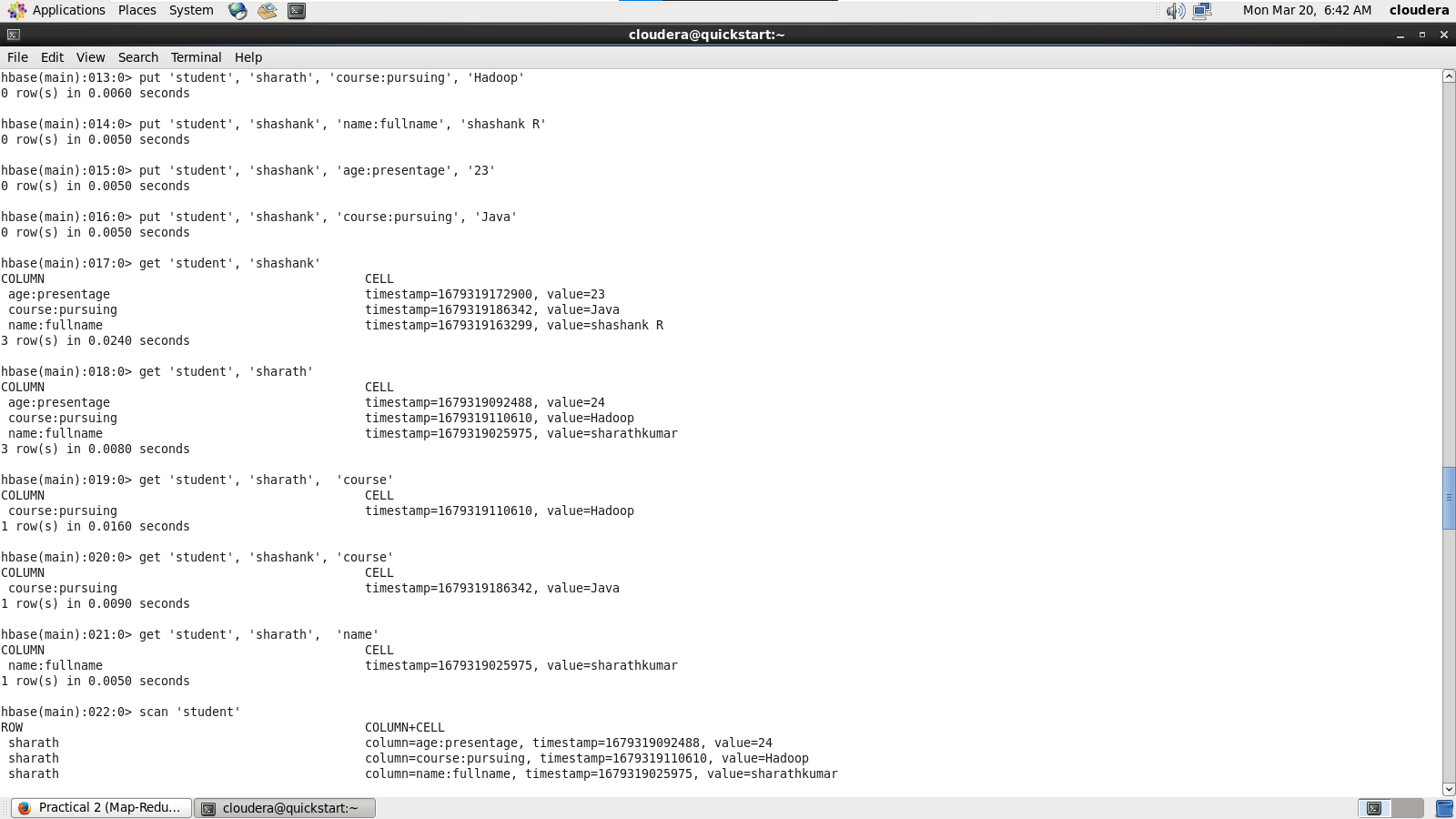
//Delete

delete ‘student’, ‘shashank’, ‘name:fullname’

**OUTPUT**







**Practical 5: Install Hive and Use Hive to Create and Store Structured Databases**

Commands:

cat > /home/cloudera/employee.txt

cat  /home/cloudera/employee.txt

sudo -u hdfs hadoop fs -put /home/cloudera/employee.txt /inputdirectroy

hdfs dfs -ls /inputdirectory

hadoop fs -cat  /inputdirectory/employee.txt

hive

show databases;

create database organization;

show databases;

use organization;

show tables;

hive> create table employee(

    > id int,

    > name string,

    > city string,

    > department string,

    > salary int,

    > domain string)

    > row format delimited

    > fields terminated by '~';

show tables;

select \* from employee;

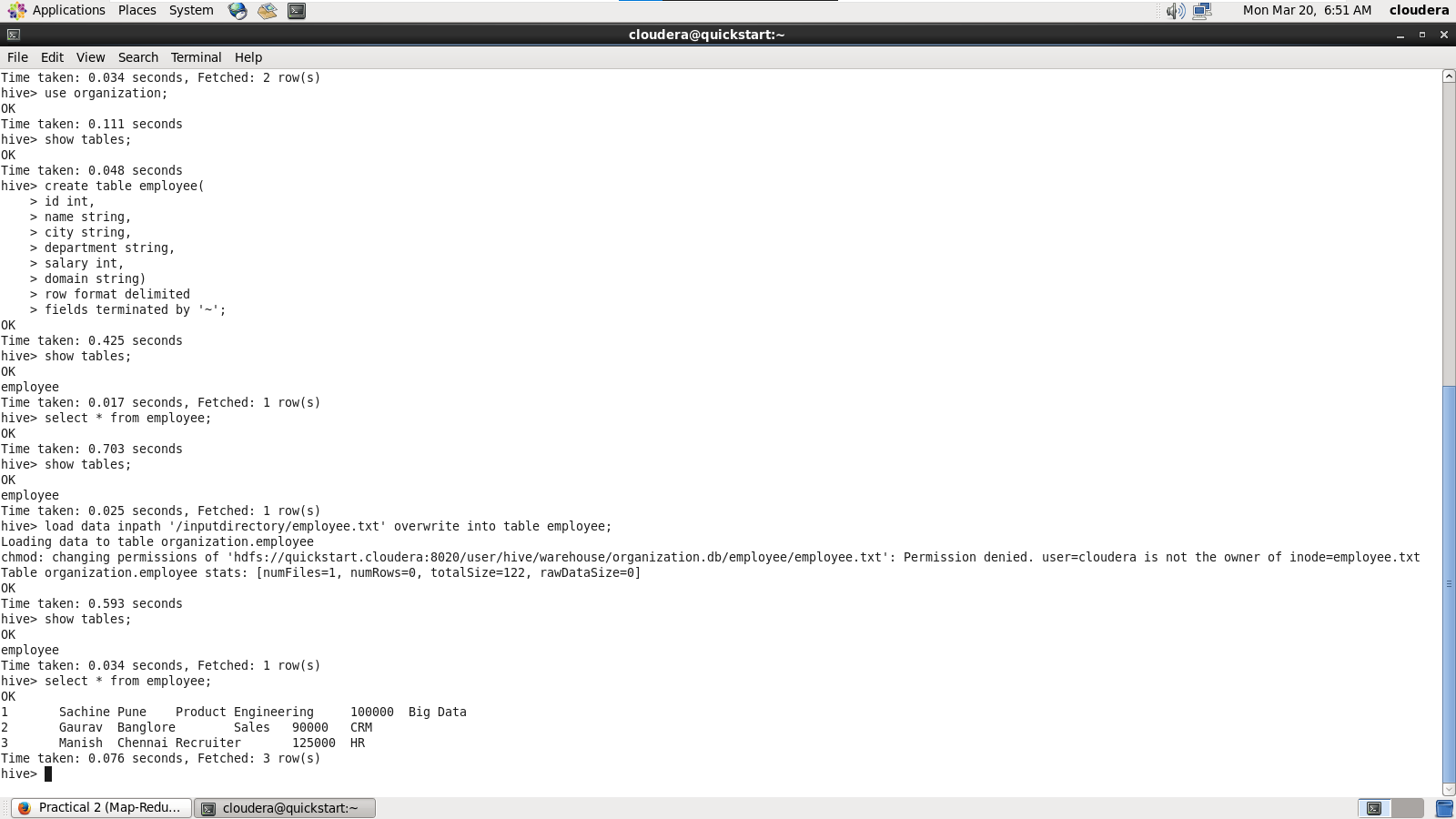
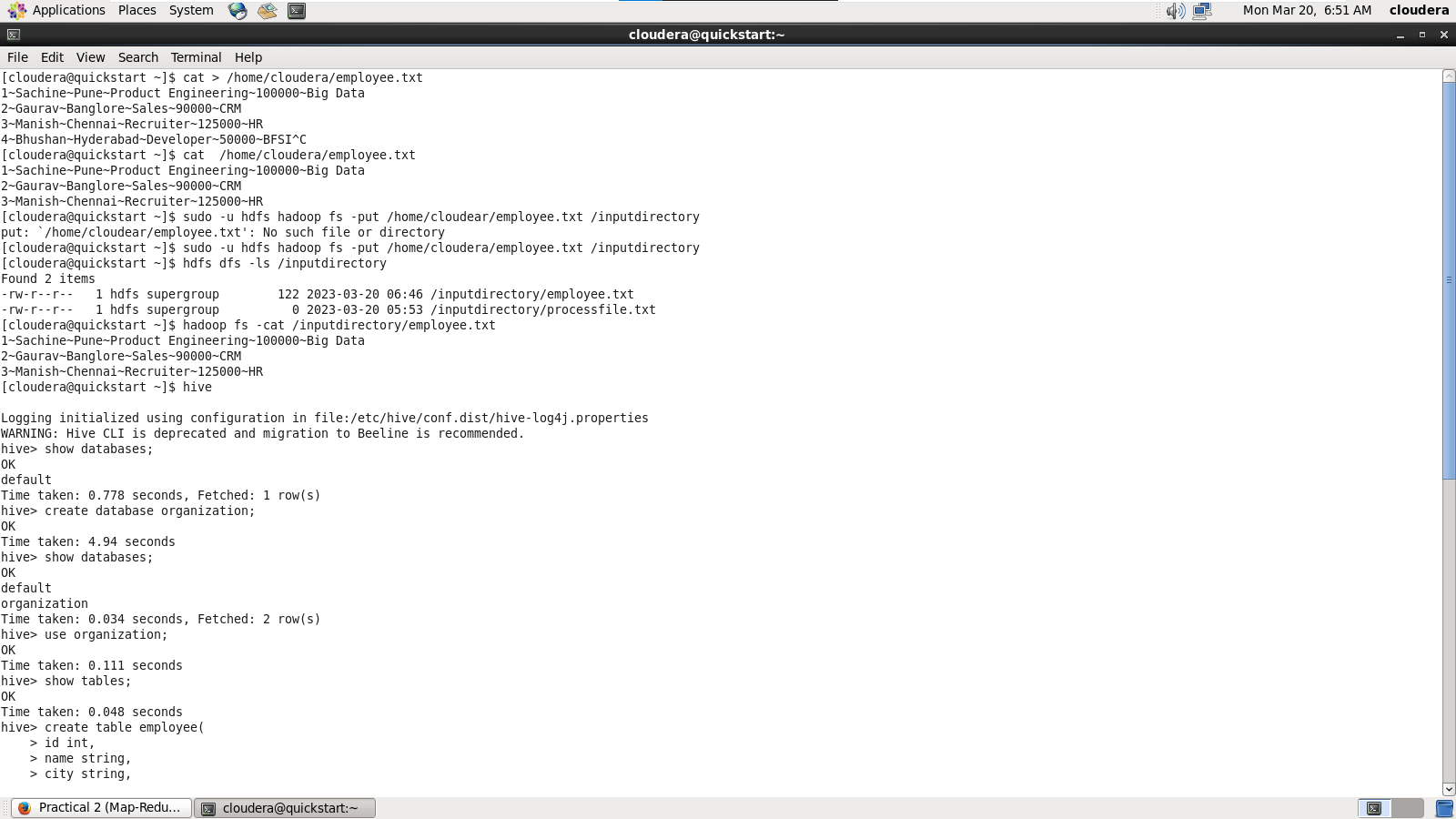
show tables;

load data inpath '/inputdirectory/employee.txt' overwrite into table employee;

show tables;

select \* from employee;

**OUTPUT**



**Practical 6: Construct Different Types of K-Shingles for Given Document**

Code:

# Install necessary packages

install.packages("tm")

require("tm")

install.packages("devtools")

# Define function to read an integer and create shingles from a file

readinteger <- function() {

  n <- readline(prompt = "Enter value of k-1: ")  # Prompt user for input

  k <- as.integer(n)  # Convert input to integer

  u1 <- readLines("C:/Users/asif0/Documents/File1.txt")  # Read in file

  Shingle <- 0  # Initialize variable for storing shingles

  i <- 0  # Initialize loop counter

  while (i < nchar(u1) - k + 1) {  # Loop through file, creating shingles

    Shingle[i] <- substr(u1, start = i, stop = i + k)  # Extract shingle from file

    print(Shingle[i])  # Print shingle to console

    i <- i + 1  # Increment loop counter

  }

}

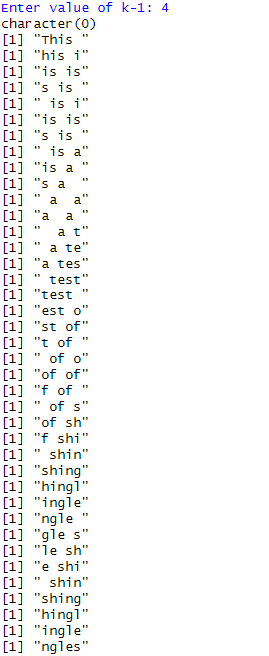
# Call readinteger function if running interactively

if (interactive()) {

  readinteger()

}

**OUTPUT**



**Practical 7: Measuring Similarity Among Documents and Detecting Passages Which Have Been Reused**

Codes:

# Install necessary packages

install.packages("tm")

require("tm")

install.packages("ggplot2")

install.packages("textreuse")

install.packages("devtools")

# Load in corpus and preprocess text

my.corpus <- Corpus(DirSource("C:/Users/asif0/Documents/New folder"))  # Load in corpus from directory

my.corpus <- tm\_map(my.corpus, removeWords, stopwords("english"))  # Remove stop words from corpus

# Create term-document matrix

my.tdm <- TermDocumentMatrix(my.corpus)  # Create term-document matrix from corpus

#inspect(my.tdm)  # Inspect term-document matrix (optional)

# Create document-term matrix

my.dtm <- DocumentTermMatrix(my.corpus, control = list(weighting = weightTfIdf, stopwords = TRUE))  # Create document-term matrix from corpus, using TF-IDF weighting and removing stop words

#inspect(my.dtm)  # Inspect document-term matrix (optional)

# Convert document-term matrix to data frame and scale data

my.df <- as.data.frame(inspect(my.tdm))  # Convert document-term matrix to data frame

my.df.scale <- scale(my.df)  # Scale data using z-score normalization

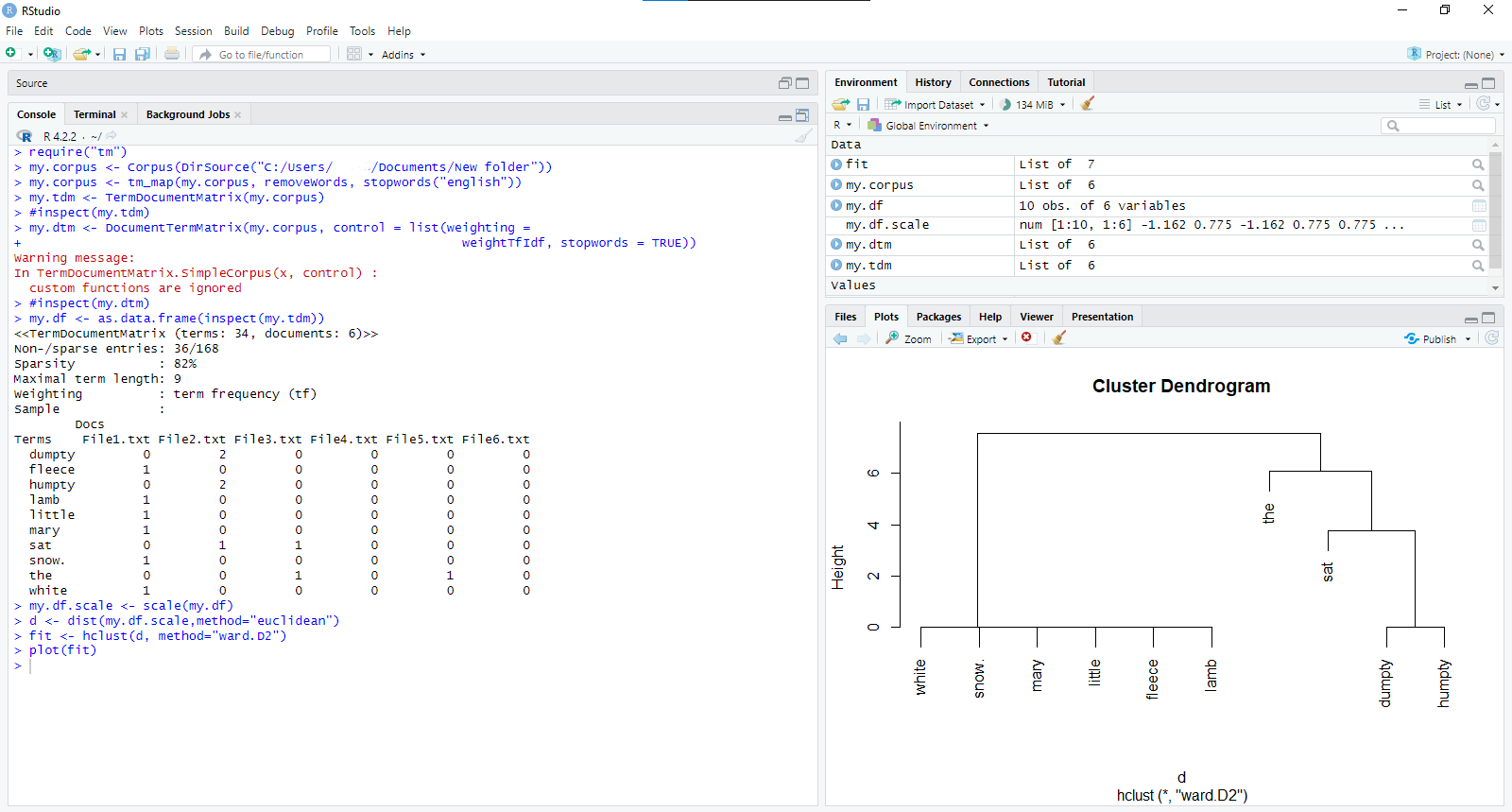
# Perform hierarchical clustering and plot dendrogram

d <- dist(my.df.scale, method = "euclidean")  # Calculate distance matrix using Euclidean distance

fit <- hclust(d, method = "ward")  # Perform hierarchical clustering using Ward's method

plot(fit)  # Plot dendrogram

OUTPUT:



**Practical 8: Compute n-moment**

Codes:

import java.io.\*;

import java.util.\*;

public class n\_moment {

    public static void main(String args[]) {

        int n = 15; // Total number of elements in the stream

        String stream[] = {"a", "b", "c", "b", "d", "a", "c", "d", "a", "b", "d", "c", "a", "a", "b"};

        int zero\_moment = 0, first\_moment = 0, second\_moment = 0, count = 1, flag = 0;

        ArrayList<Integer> arrlist = new ArrayList(); // Creating a new ArrayList

        System.out.println("Arraylist elements are::");

        for (int i = 0; i < 15; i++) {

            System.out.println(stream[i] + " "); // Printing the elements of the stream

        }

        Arrays.sort(stream); // Sorting the elements of the stream

        for (int i = 1; i < n; i++) {

            if (stream[i] == stream[i - 1]) { // If current element is same as previous element

                count++; // Increment the count

            } else {

                // System.out.println("Hello"+i);

                arrlist.add(count); // Add the count to the ArrayList

                count = 1; // Reset the count

            }

        }

        arrlist.add(count); // Add the last count to the ArrayList

        zero\_moment = arrlist.size(); // Zeroth moment is the size of the ArrayList

        System.out.println("\n\n\nValue of Zeroth moment for given stream::" + zero\_moment);

        for (int i = 0; i < arrlist.size(); i++) {

            first\_moment += arrlist.get(i); // Summing up all the elements in the ArrayList

        }

        System.out.println("\n\nValue of First moment for given stream::" + first\_moment);

        for (int i = 0; i < arrlist.size(); i++) {

            int j = arrlist.get(i);

            second\_moment += (j \* j); // Computing the second moment by summing up the squares of all elements in the ArrayList

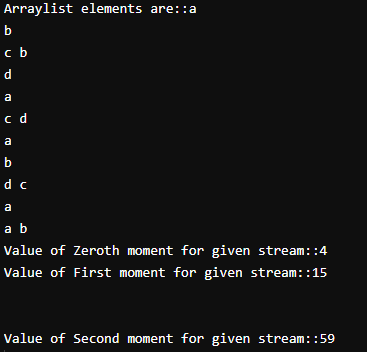
        }

        System.out.println("\n\nValue of Second moment for given stream::" + second\_moment);

    }

}

**OUTPUT**



**Practical 9: Alon-Matias-Szegedy Algorithm**

Codes:

import java.io.\*;

import java.util.\*;

class AMSA {

  public static int findCharCount(String stream, char XE, int random, int n) {

    int countoccurance = 0;

    for (int i = random; i < n; i++) {

      if (stream.charAt(i) == XE) {

        countoccurance++;

      }

    }

    return countoccurance;

  }

  public static int estimateValue(int XV1, int n) {

    int ExpValue;

    ExpValue = n \* (2 \* XV1 - 1);

    return ExpValue;

  }

  public static void main(String args[]) {

    int n = 15;

    String stream = "abcbdacdabdcaab";

    int random1 = 3, random2 = 8, random3 = 13;

    char XE1, XE2, XE3;

    int XV1, XV2, XV3;

    int ExpValuXE1, ExpValuXE2, ExpValuXE3;

    int apprSecondMomentValue;

    // Select three random characters from the stream

    XE1 = stream.charAt(random1 - 1);

    XE2 = stream.charAt(random2 - 1);

    XE3 = stream.charAt(random3 - 1);

    // Count the number of occurrences of each character in the stream

    XV1 = findCharCount(stream, XE1, random1 - 1, n);

    XV2 = findCharCount(stream, XE2, random2 - 1, n);

    XV3 = findCharCount(stream, XE3, random3 - 1, n);

    // Print the counts of the selected characters

    System.out.println(XE1 + "=" + XV1 + " " + XE2 + "=" + XV2 + " " + XE3 + "=" + XV3);

    // Estimate the expected value for each selected character

    ExpValuXE1 = estimateValue(XV1, n);

    ExpValuXE2 = estimateValue(XV2, n);

    ExpValuXE3 = estimateValue(XV3, n);

    // Print the expected values for each selected character

    System.out.println("Expected value for" + XE1 + " is::" + ExpValuXE1);

    System.out.println("Expected value for" + XE2 + " is::" + ExpValuXE2);

    System.out.println("Expected value for" + XE3 + " is::" + ExpValuXE3);

    // Compute the approximate second moment value using Alon-Matias-Szegedy algorithm

    apprSecondMomentValue = (ExpValuXE1 + ExpValuXE2 + ExpValuXE3) / 3;

    System.out.println("approximate second moment value using alon-matis-szegedy is::" + apprSecondMomentValue);

  }

}

**OUTPUT**

