**INFO7275 16619 Advanced Database Management Sys SEC 01 - Fall 2016**

**-Sumeet Usturge**

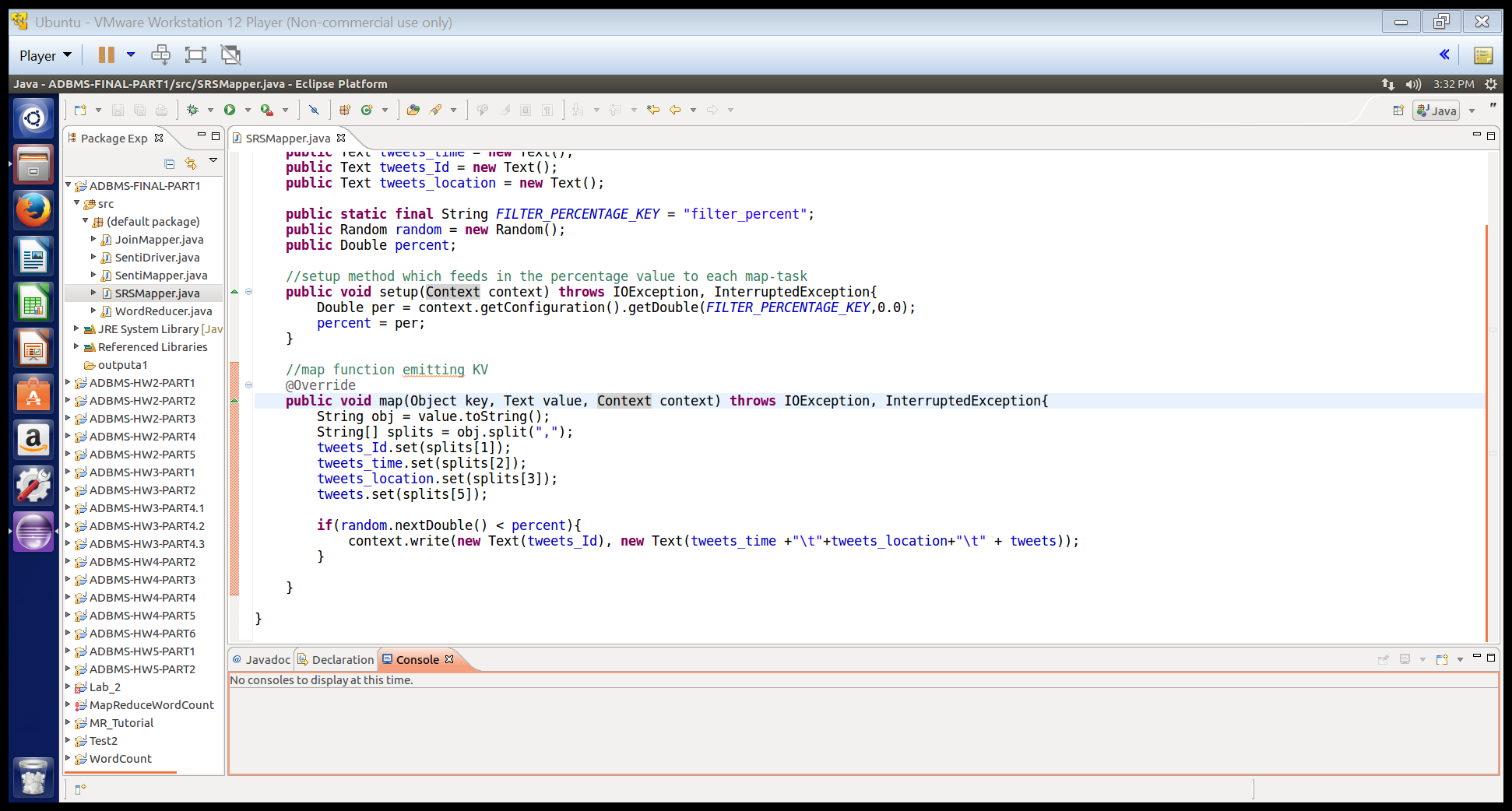
**NUID:1676003**

**Overview:**

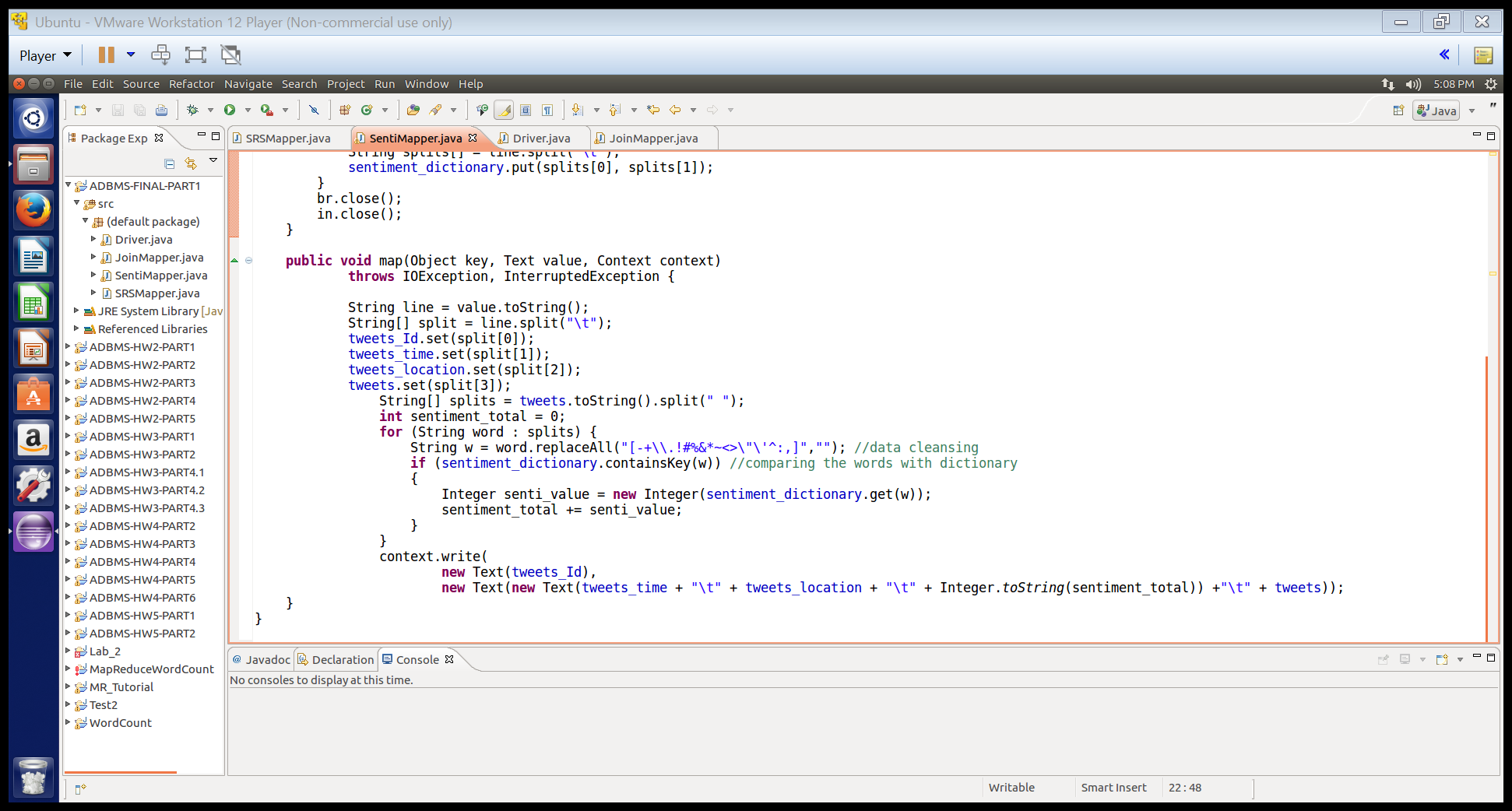
Besides United States presidential election, on 8th November 2016 there was another significant announcement that was made by Indian Govt on the same day that to discontinue current currency of value 500 and 1000 in order to fight against black money. Twitter reacts to this and there were mixed responses from people. I am doing sentimental analysis on these tweets to analyze the people’s reaction to this big move by the Indian Govt. The twitter data is download into HDFS using Apache-Flumes and is cleansed at local file system as per my requirement. Similar very small dataset can be found on <https://www.kaggle.com/arathee2/demonetization-in-india-twitter-data/version/2>

**Design Patterns:**

To begin with I am implementing Simple Random Sampling design pattern on my dataset to get the random rows to perform analysis. The output for this is then fed into the mapper of Sentiment Analysis.

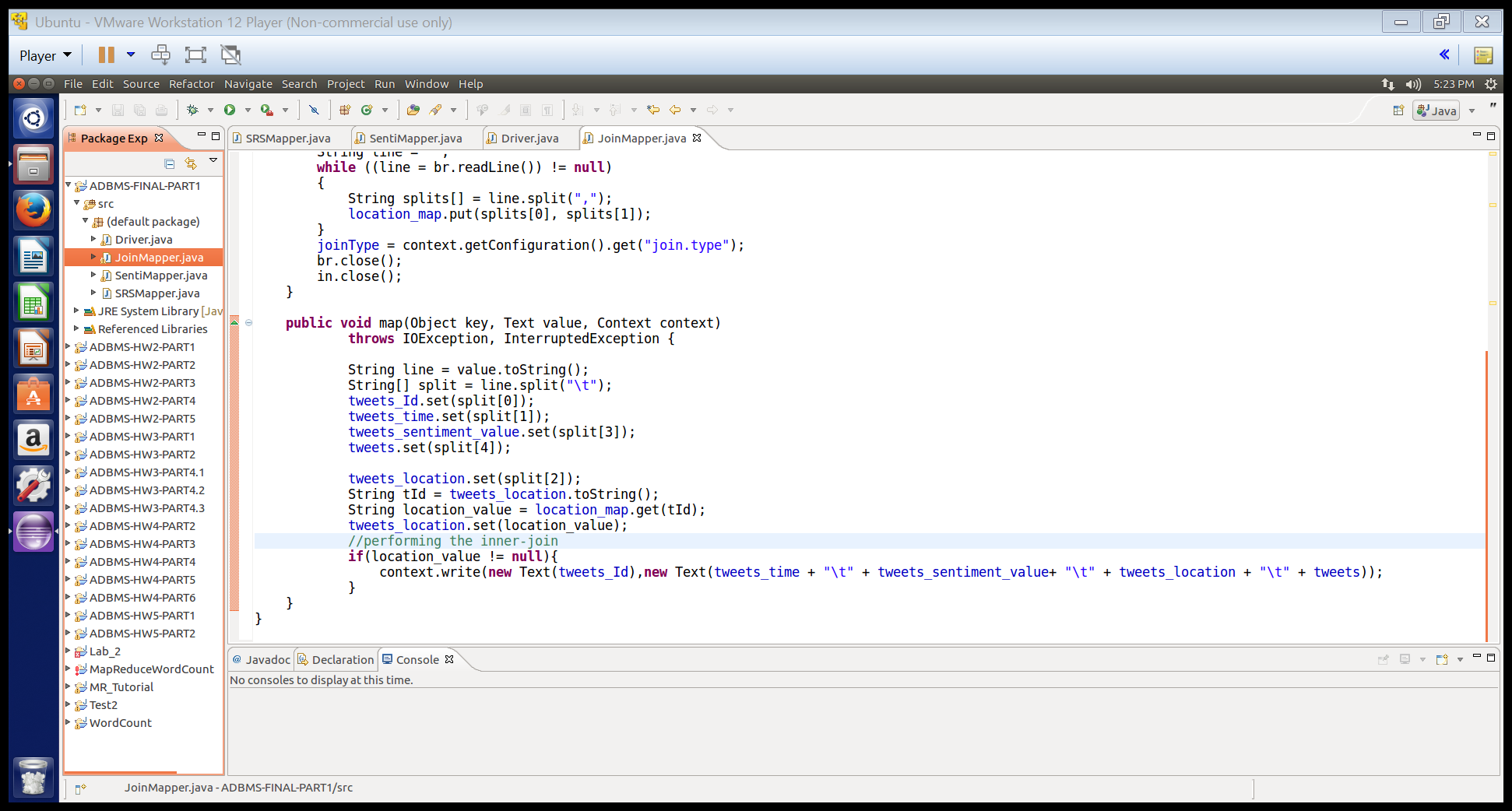


Performing sentimental analysis on the random data tweets that are fed into mapper of this algorithm. Each tweet is split into words with delimiter space and any special character associated with it is replaced. Then each word is matched against the word in the dictionary. The word dictionary is added to the distributed cache to make it available to each mapper task. The sentiment value assigned to it is calculated and summed for the entire tweet.





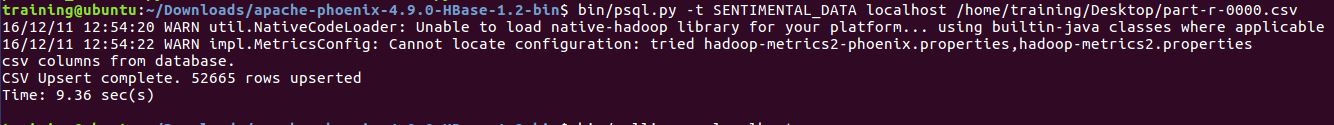
Using Replicated join pattern the location id is matched with the location data-set and the corresponding location value is replaced for the location id in the input file and emitted. Purpose for joining using replicated join is that it is quick. The input data is enriched with the data stored in memory. There is no reducer involved and so no overhead for shuffling and partitioning. This makes the join fast.



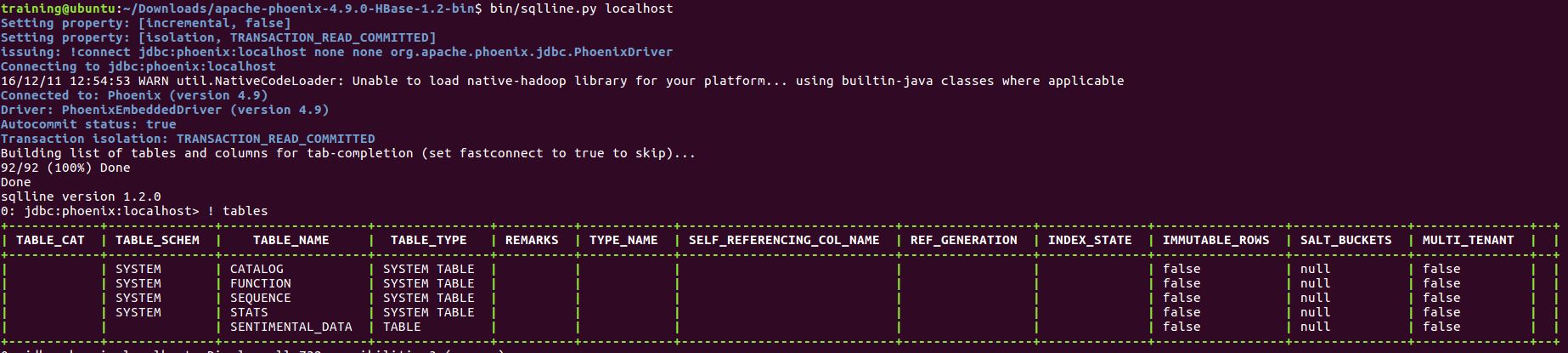
In the driver the jobs are controlled using org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob. The chaining is performed and every job is dependent on the job preceding it. If the preceding job fails, then successive job also fails.

**HBASE and Apache Phoenix:**

The output data is copied to local using copyToLocal from HDFS and this bulk data is fed into HBASE using Apache Phoenix as shown in the below screenshot:



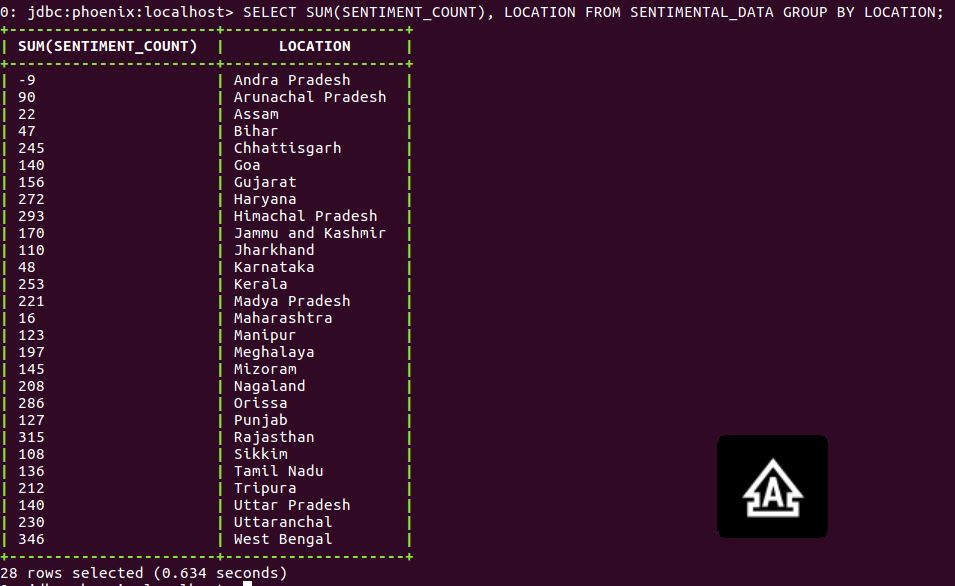
Checking if the table is properly created in HBASE from Phoenix sqlline:



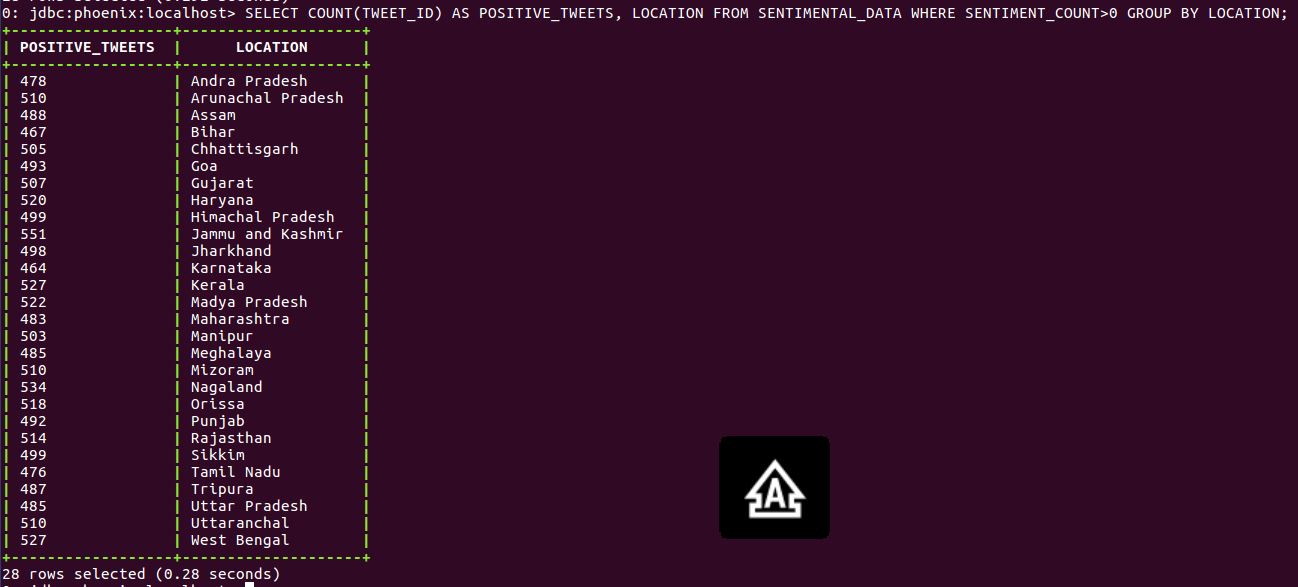
Retrieving the table from Apache Phoenix using SQL like command:

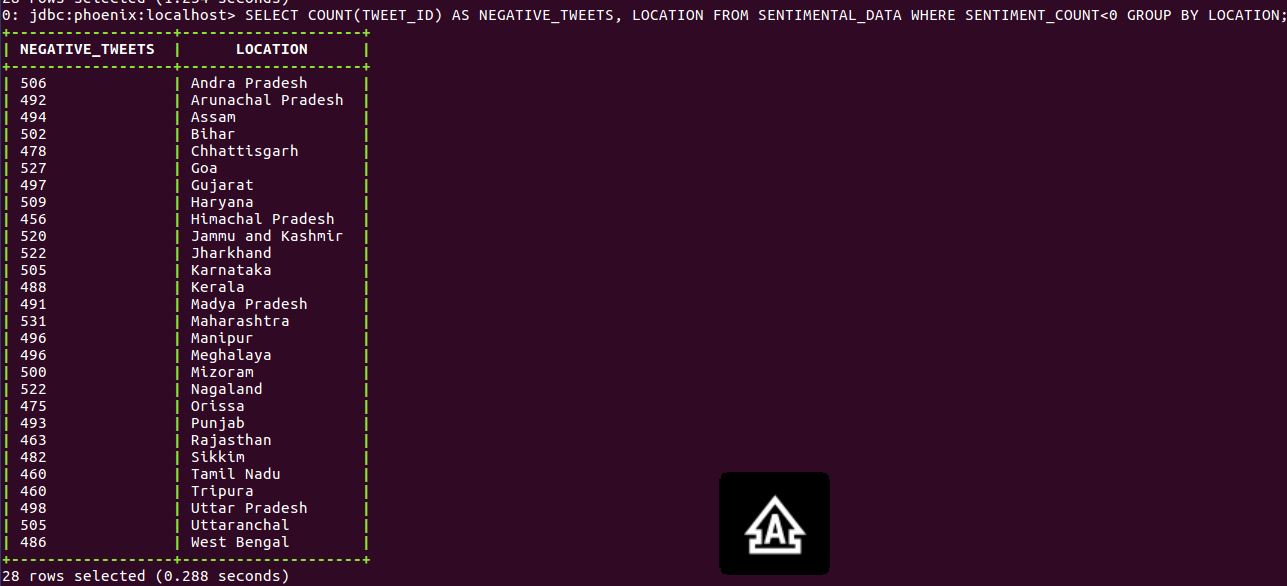


Performing queries like sentiments in each state:

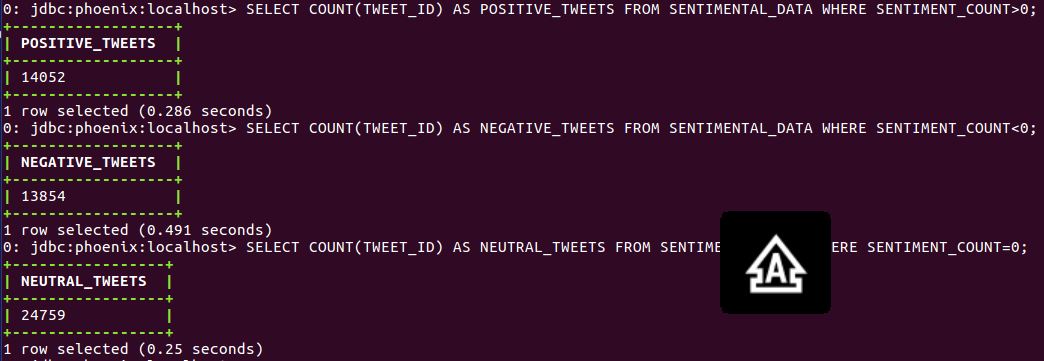


Counting the total positive tweets, negative and neutral tweets grouped by location:





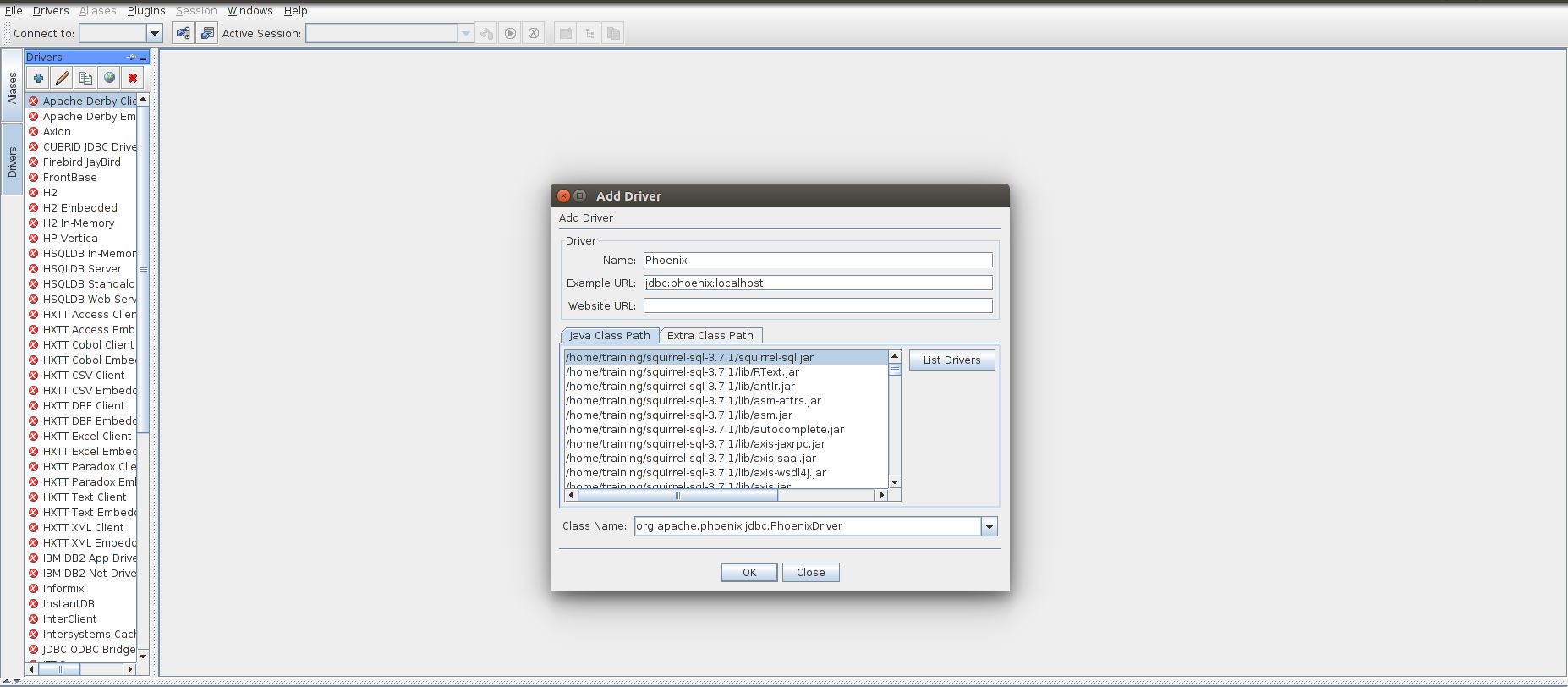
Total number of positive, negative and neutral tweets on the given data:

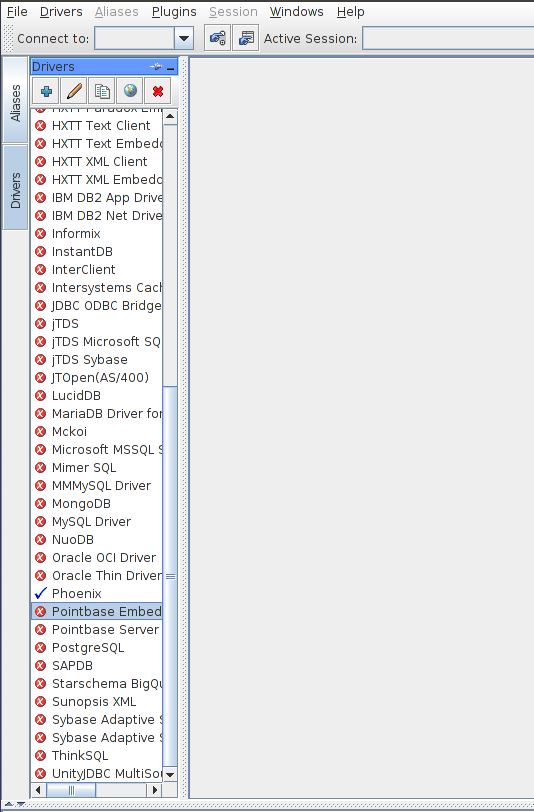


**SQuirreL Configuration:**

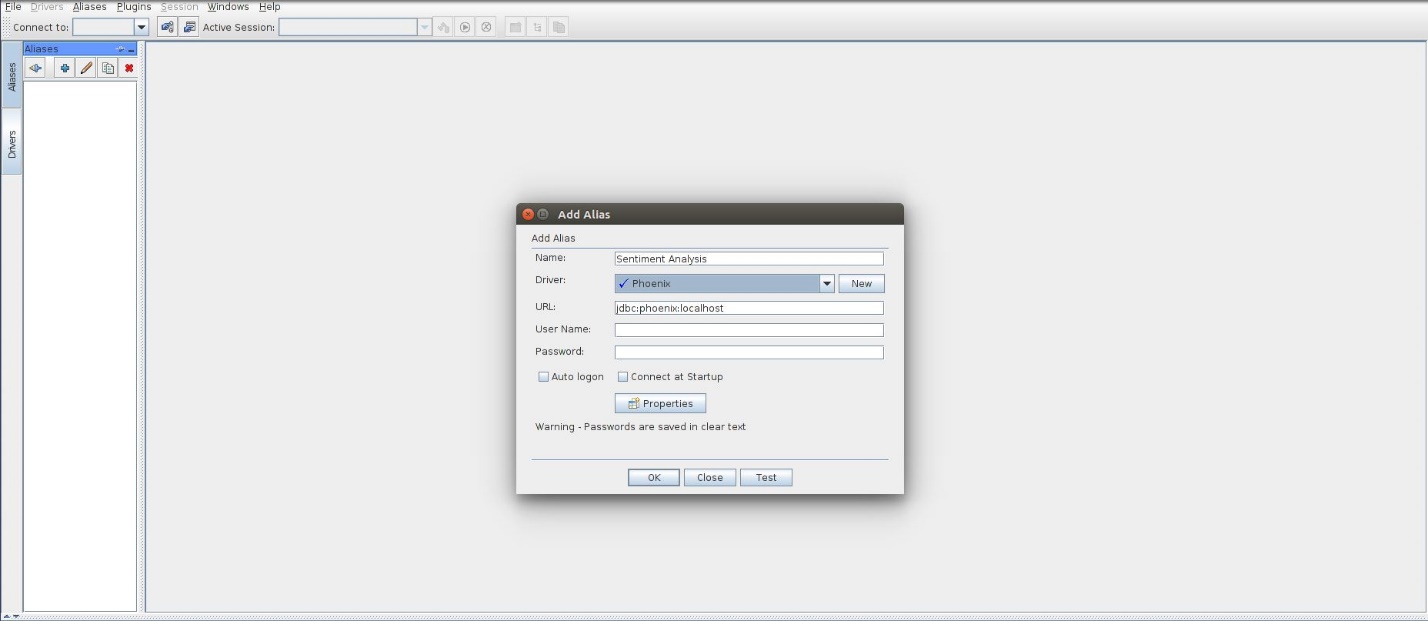
SQuirreL SQL Client is a graphical Java program that will allow you to view the structure of a JDBC compliant database, browse the data in tables, issue SQL commands etc, see Getting Started and Introduction.

Below are the screenshots for configuring the SQuirreL for performing some visualization over the final output from the mapper.

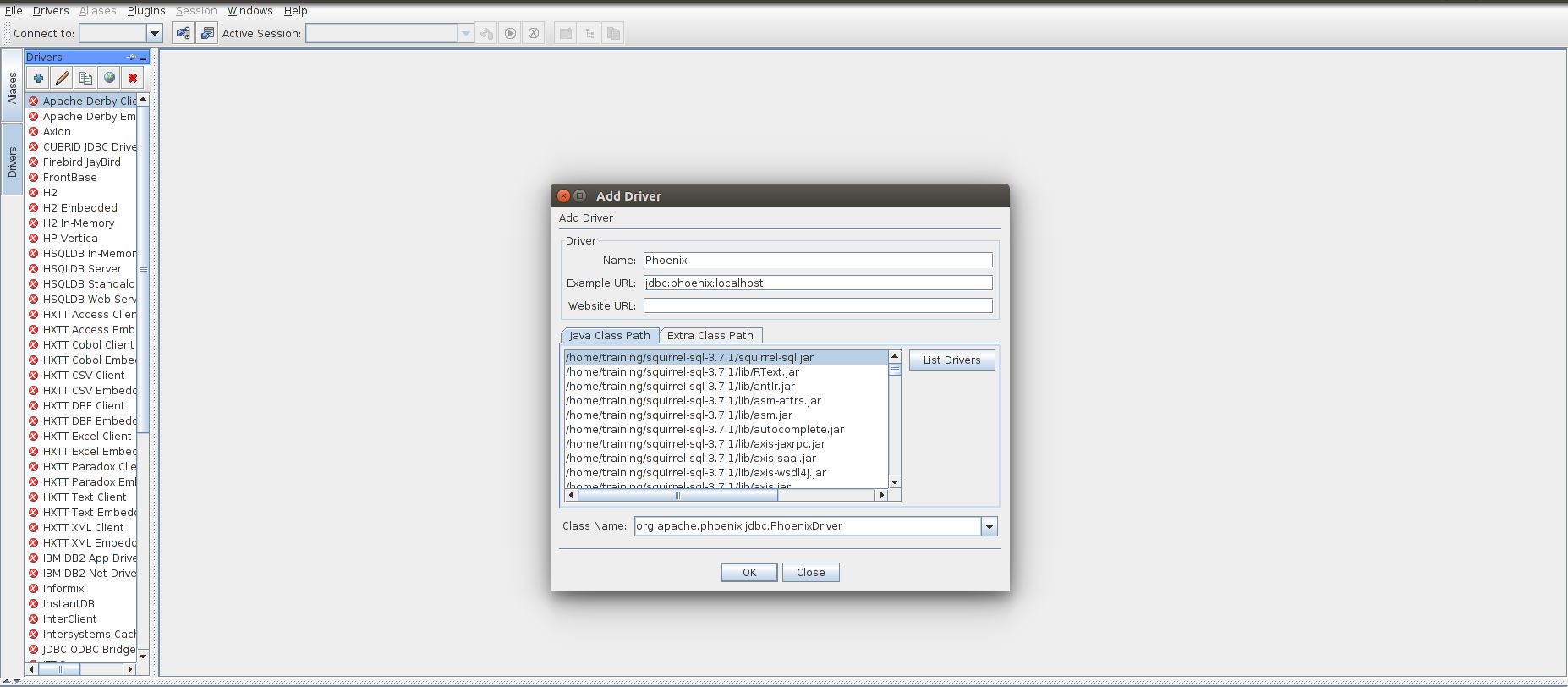


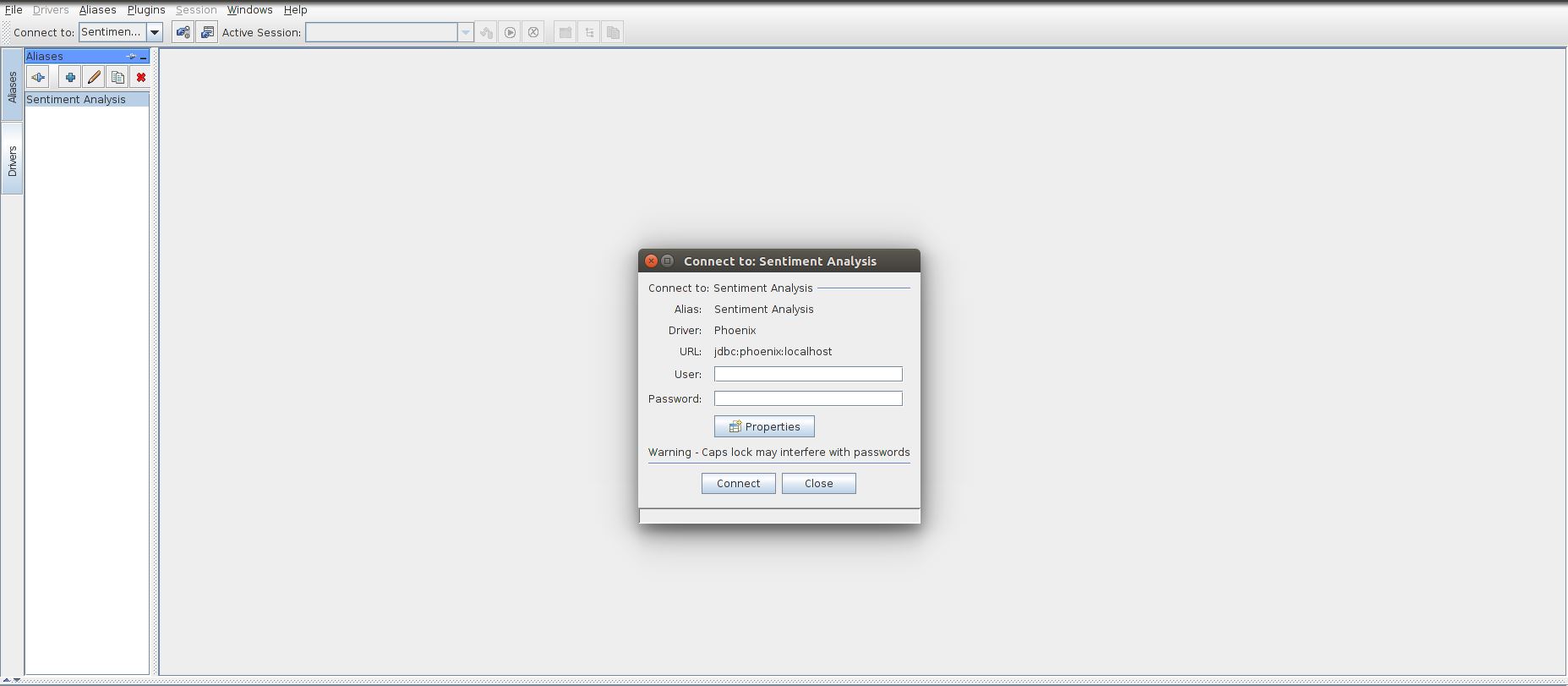


Creating alias below for the driver that you have setup as shown in the above picture

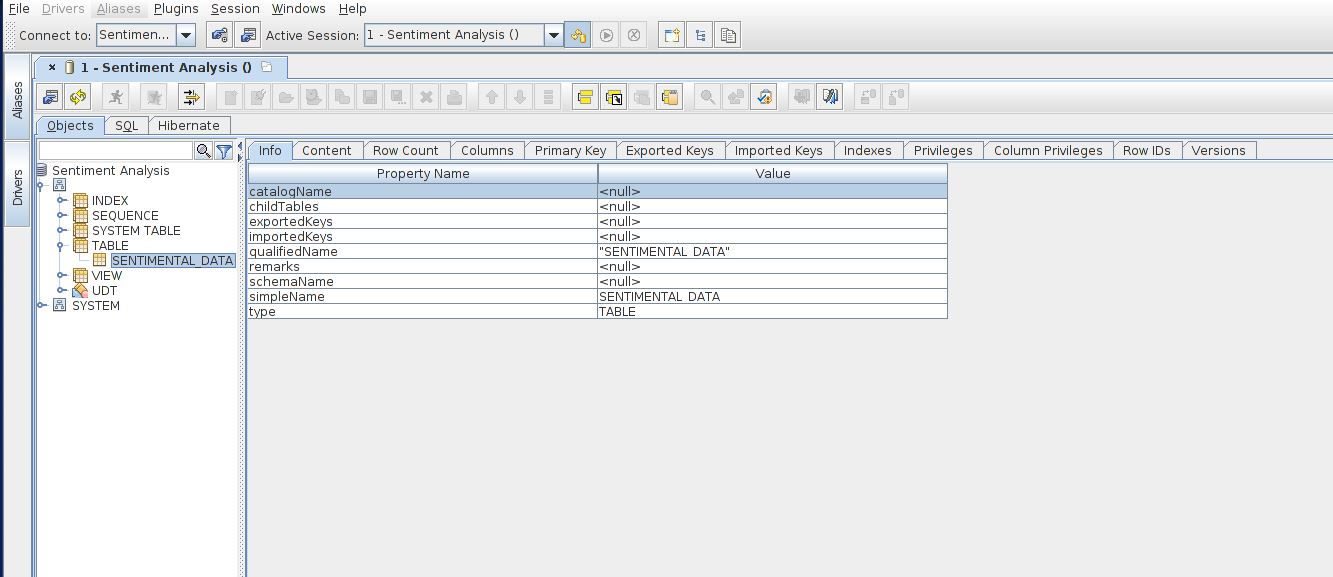


Connecting to the driver that you have created to access the table from HBASE

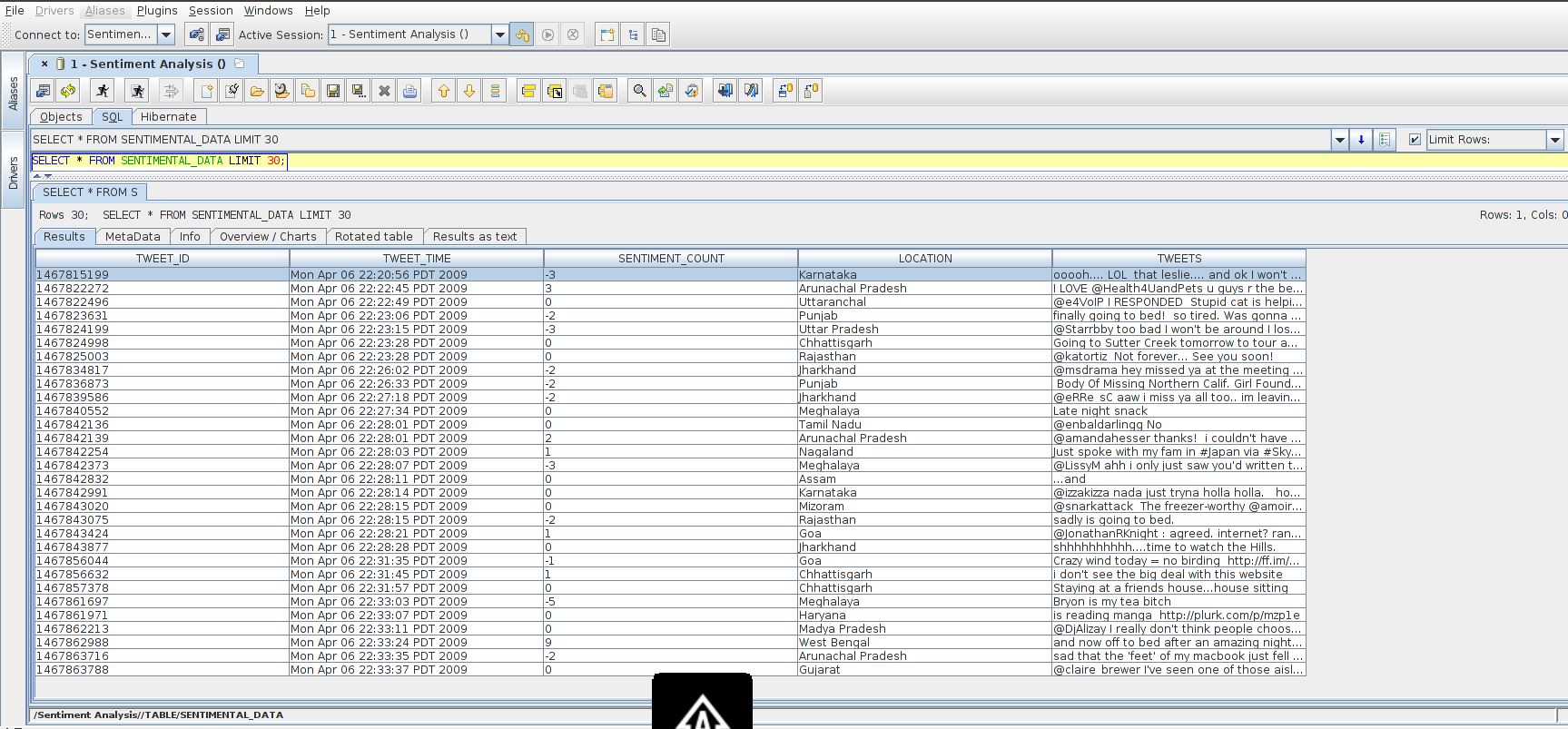




The table that you created in the HBASE

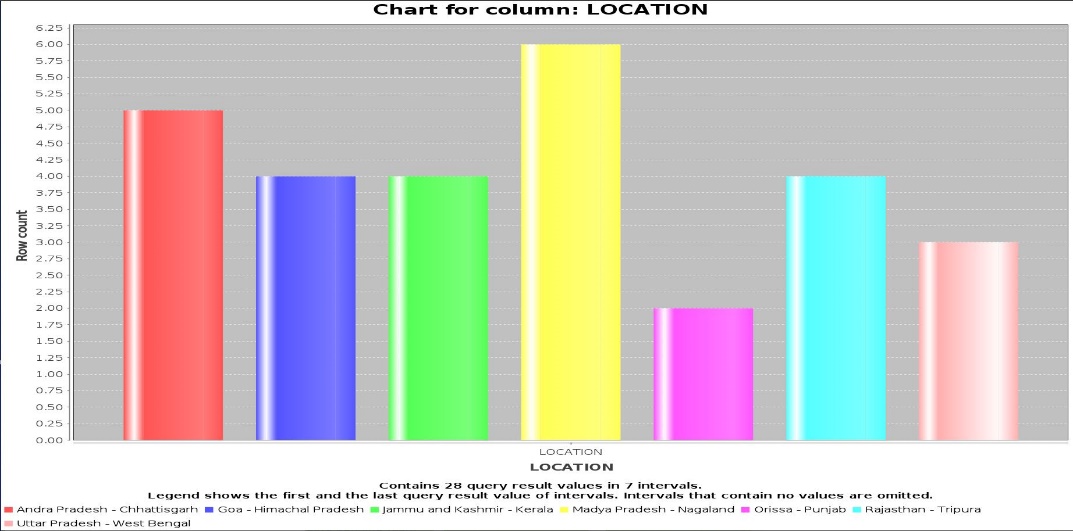
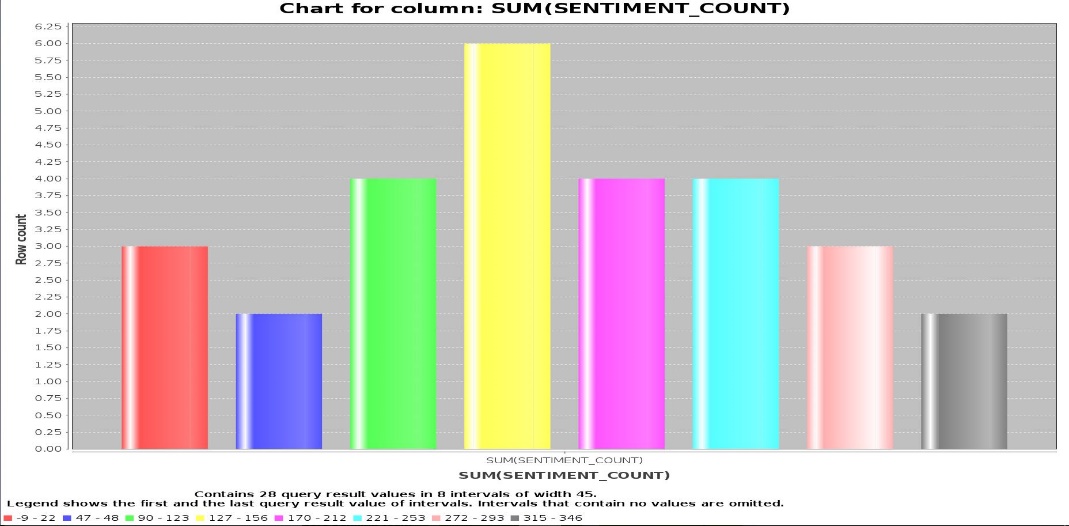


Performing SELECT \* FROM TABLE LIMIT 30; Just as in case of SQL over HBASE.

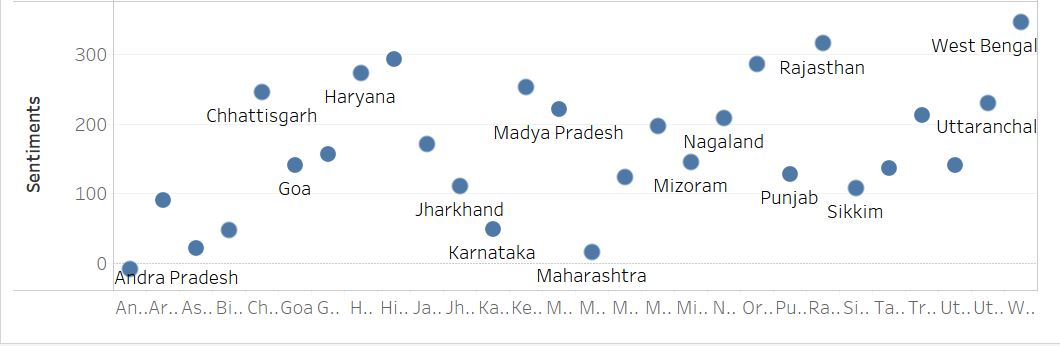


**Reporting and Visualization using BI tools:**

Visualization performed using Squirrel over the data:



Performing BI by importing the data into tableau. This shows the sentiments of the people from different states over the demonetization by the Indian Govt.



**Source Code and JAR file:**

** **

**SRSMapper:**

import java.io.IOException;

import java.util.Random;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class SRSMapper extends Mapper<Object, Text, Text, Text>{

public Text tweets = new Text();

public Text tweets\_time = new Text();

public Text tweets\_Id = new Text();

public Text tweets\_location = new Text();

public static final String FILTER\_PERCENTAGE\_KEY = "filter\_percent";

public Random random = new Random();

public Double percent;

//setup method which feeds in the percentage value to each map-task

public void setup(Context context) throws IOException, InterruptedException{

Double per = context.getConfiguration().getDouble(FILTER\_PERCENTAGE\_KEY,0.0);

percent = per;

}

//map function emitting KV

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException{

String obj = value.toString();

String[] splits = obj.split(",");

tweets\_Id.set(splits[1]);

tweets\_time.set(splits[2]);

tweets\_location.set(splits[3]);

tweets.set(splits[5]);

if(random.nextDouble() < percent){

context.write(new Text(tweets\_Id), new Text(tweets\_time +"\t"+tweets\_location+"\t" + tweets));

}

}

}

**SentiMapper:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.URI;

import java.util.HashMap;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class SentiMapper extends Mapper<Object, Text, Text, Text> {

private URI[] files;

private HashMap<String, String> sentiment\_dictionary = new HashMap<String, String>();

public Text tweets\_Id = new Text();

public Text tweets = new Text();

public Text tweets\_time = new Text();

public Text tweets\_location = new Text();

//set up method which loads the small data-set from the cache having word-dictionary to perform map-side join

@Override

public void setup(Context context) throws IOException {

files = DistributedCache.getCacheFiles(context.getConfiguration());

Path path = new Path(files[0]);

FileSystem fs = FileSystem.get(context.getConfiguration());

FSDataInputStream in = fs.open(path);

BufferedReader br = new BufferedReader(new InputStreamReader(in));

String line = "";

while ((line = br.readLine()) != null)

{

String splits[] = line.split("\t");

sentiment\_dictionary.put(splits[0], splits[1]);

}

br.close();

in.close();

}

public void map(Object key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

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

tweets\_Id.set(split[0]);

tweets\_time.set(split[1]);

tweets\_location.set(split[2]);

tweets.set(split[3]);

String[] splits = tweets.toString().split(" ");

int sentiment\_total = 0;

for (String word : splits) {

String w = word.replaceAll("[-+\\.!#%&\*~<>\"\'^:,]",""); //data cleansing

if (sentiment\_dictionary.containsKey(w)) //comparing the words with dictionary

{

Integer senti\_value = new Integer(sentiment\_dictionary.get(w));

sentiment\_total += senti\_value;

}

}

context.write(

new Text(tweets\_Id),

new Text(new Text(tweets\_time + "\t" + tweets\_location + "\t" + Integer.toString(sentiment\_total)) +"\t" + tweets));

}

}

**JOINMAPPER:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.URI;

import java.util.HashMap;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class JoinMapper extends Mapper<Object, Text, Text, Text> {

private URI[] files;

private HashMap<String, String> location\_map = new HashMap<String, String>();

private String joinType = null;

public Text tweets\_Id = new Text();

public Text tweets = new Text();

public Text tweets\_time = new Text();

public Text tweets\_location = new Text();

public Text tweets\_sentiment\_value = new Text();

//set up method which loads the small data-set of location

@Override

public void setup(Context context) throws IOException {

files = DistributedCache.getCacheFiles(context.getConfiguration());

Path path = new Path(files[1]);

FileSystem fs = FileSystem.get(context.getConfiguration());

FSDataInputStream in = fs.open(path);

BufferedReader br = new BufferedReader(new InputStreamReader(in));

String line = "";

while ((line = br.readLine()) != null)

{

String splits[] = line.split(",");

location\_map.put(splits[0], splits[1]);

}

joinType = context.getConfiguration().get("join.type");

br.close();

in.close();

}

public void map(Object key, Text value, Context context)

throws IOException, InterruptedException {

String line = value.toString();

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

tweets\_Id.set(split[0]);

tweets\_time.set(split[1]);

tweets\_sentiment\_value.set(split[3]);

tweets.set(split[4]);

tweets\_location.set(split[2]);

String tId = tweets\_location.toString();

String location\_value = location\_map.get(tId);

tweets\_location.set(location\_value);

//performing the inner-join

if(location\_value != null){

context.write(new Text(tweets\_Id),new Text(tweets\_time + "\t" + tweets\_sentiment\_value+ "\t" + tweets\_location + "\t" + tweets));

}

}

}

**DRIVER:**

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.filecache.DistributedCache;

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

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

import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;

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

import org.apache.hadoop.util.GenericOptionsParser;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class Driver extends Configured implements Tool{

public static final String FILTER\_PERCENTAGE\_KEY = "filter\_percent";

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

ToolRunner.run(new Driver(), args);

}

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

Configuration conf = new Configuration();

//Job for SRS design pattern

Job job1 = Job.getInstance(conf, "Sample Random");

GenericOptionsParser parser = new GenericOptionsParser(conf, args); //to parse command line arguments generic to the Hadoop framework

String[] otherArgs = parser.getRemainingArgs();

if (otherArgs.length != 7) {

printErrorMessage();

}

Float filterPercentage = 0.0f;

try {

filterPercentage = Float.parseFloat(otherArgs[0]) / 100.0f;

} catch (NumberFormatException nfe) {

printErrorMessage();

}

job1.setJarByClass(Driver.class);

job1.setMapperClass(SRSMapper.class);

job1.setMapOutputKeyClass(Text.class);

job1.setMapOutputValueClass(Text.class);

job1.getConfiguration().setFloat(FILTER\_PERCENTAGE\_KEY, filterPercentage);

FileInputFormat.addInputPath(job1, new Path(otherArgs[1]));

FileOutputFormat.setOutputPath(job1, new Path(otherArgs[2]));

job1.waitForCompletion(true);

//Job for sentimental analysis

//DistributedCache.addCacheFile(new URI("/home/training/Desktop/twitter/AFINN.txt"),conf);

DistributedCache.addCacheFile(new URI("/tweet/AFINN.txt"),conf);

Job job2 = Job.getInstance(conf, "Sentiment");

job2.setJarByClass(Driver.class);

job2.setInputFormatClass(TextInputFormat.class);

job2.setMapperClass(SentiMapper.class);

job2.setMapOutputKeyClass(Text.class);

job2.setMapOutputValueClass(Text.class);

FileInputFormat.addInputPath(job2, new Path(args[3]));

FileOutputFormat.setOutputPath(job2, new Path(args[4]));

job2.waitForCompletion(true);

//Job for Join pattern

DistributedCache.addCacheFile(new URI("/Demonetization\_Join\_Input/Location.csv"),conf);

Job job3 = Job.getInstance(conf, "Join");

job3.setJarByClass(Driver.class);

job3.setInputFormatClass(TextInputFormat.class);

job3.setMapperClass(JoinMapper.class);

job3.setMapOutputKeyClass(Text.class);

job3.setMapOutputValueClass(Text.class);

FileInputFormat.addInputPath(job3, new Path(args[5]));

FileOutputFormat.setOutputPath(job3, new Path(args[6]));

job3.waitForCompletion(true);

return 0;

}

public void printErrorMessage(){

System.err.println("Arguments order: SimpleRandomSampling <percentage> <input data path> <output SRS path> " +

"<input Senti path> <output Senti path> <input Join path> <output Join path>");

ToolRunner.printGenericCommandUsage(System.err);

System.exit(1);

}

}

**APPENDIX:**

1. [**Overview**](#Overview)
2. [**Design Pattern and Algorithm**](#DesignPatterns)

* **Simple Random Sampling(SRS)**
* **Sentiment Analysis**
* **Distributed Cache**
* **Replicated Joins**
* **Job Chaining**

1. [**HBASE and Apache Phoenix**](#HBASEandApachePhoenix)
2. [**SQuirreL Configuration**](#SQuirreLConfiguration)
3. [**Reporting and Visualization using BI tools**](#Reporting)
4. [**Source Code and JAR file**](#SourceCode)