**H1-B PROJECT**

**Nupur Chauhan**

**S170020100444**

**Batch Code: S180017**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Contents** | **Page** |
| **1** | **Acknowledgement** | **3** |
| **2** | **Abstract** | **4** |
| **3** | **Big Data**  **3.1 What is Big Data?**  **3.2 Big data characteristics**  **3.3 Type of Big Data**  **3.4 Examples of Big Data**  **3.5 Applications of Big Data**  **3.6 Challenges with Big Data** | **5-8** |
| **4** | **Hadoop**  **4.1 Hadoop to Rescue**  **4.2 History of Hadoop**  **4.5 Modules in Hadoop** | **9-10** |
| **5** | **HDFS**  **5.1 Advantages of HDFS**  **5.2 HDFS Architecture** | **11-14** |
| **6** | **Project Implementation**  **6.1 Assumptions**  **6.2 Prerequisites** | **15-16** |
| **7** | **Steps for Conversion** | **17-18** |
| **8** | **Mapreduce**  **8.1 Mapper**  **8.2 What does mapper do?**  **8.3 Reducer**  **8.4 Phases of reducer**  **8.5 Task:1a**  **8.6 Task:2a**  **8.7 Task:3**  **8.8 Task:4** | **19-41** |
| **9** | **Hive**  **9.1 Task:2b**  **9.2 Task:5**  **9.3 Task:7**  **9.4 Task:8** | **42-72** |
| **10** | **Pig**  **10.1 Features of Pig**  **10.2 Task:1b**  **10.3 Task:6**  **10.4 Task:9**  **10.5 Task:10** | **73-83** |
| **11** | **Sqoop**  **11.1 Task:11** | **84-86** |
| **12** | **References** | **87** |

**ACKNOWLEDGEMENT**

**I have taken efforts in the project. However, it would not have been possible without the kind support of many individuals and organization. I would like to extend my sincere thanks to all of them.**

**I am highly indebted to Mr. Annu Sharma and Mr. Sandeep Agarwal for their guidance and constant supervision as well as providing necessary information regarding the project and also for their support in completing the project.**

**I would like to express my gratitude towards Mr. Shakun Tyagi for his cooperation, encouragement and providing all necessary support at the earliest possible.**

**My special thanks and appreciation goes to NIIT South Ex for providing the necessary infrastructure and sound environment for hassle free learning and successful project completion.**

**ABSTRACT**

The H1B is an employment-based, nonimmigrant visa category for temporary foreign workers in the United States. For a foreign national to apply for H1B visa, an US employer must offer a job and petition for H1B visa with the US immigration department. This is the most common visa status applied for and held by international students once they complete college/ higher education (Masters, Ph.D.) and work in a full-time position. We will be performing analysis on the H1B visa applicants between the years 2011-2016. After analyzing the data, we can derive the following facts.

1 a) Is the number of petitions with Data Engineer job title increasing over time?  
 b) Find top 5 job titles who are having highest growth in applications.  
2 a) Which part of the US has the most Data Engineer jobs for each year?  
 b) find top 5 locations in the US who have got certified visa for each year.  
3)Which industry has the most number of Data Scientist positions?  
4)Which top 5 employers file the most petitions each year?  
5) Find the most popular top 10 job positions for H1B visa applications for each year?  
6) Find the percentage and the count of each case status on total applications for each year.   
Create a graph depicting the pattern of all the cases over the period of time.  
7) Create a bar graph to depict the number of applications for each year  
8) Find the average Prevailing Wage for each Job for each Year (take part time and full time separate).  
Arrange the output in descending order.  
9) Which are employers along with the number of petitions who have the success rate more than 70%   
in petitions and total petitions filed more than 1000?  
10) Which are the job positions along with the number of petitions which have the success rate more than 70%  
in petitions and total petitions filed more than 1000?  
11) Export result for question no 10 to MySql database.

**We have use the following four tools to derive the above facts - Mapreduce, Hive, Pig, and Sqoop.**

**BIG DATA**

## **What is Big Data?**

Big Data is a term used for a collection of data sets that are large and complex, which is difficult to store and process using available database management tools or traditional data processing applications. The challenge includes capturing, curating, storing, searching, sharing, transferring, analyzing and visualization of this data.

## 

## 

## **Big Data Characteristics**

The five characteristics that define Big Data are: Volume, Velocity, Variety, Veracity and Value.

1. **VOLUME**: Volume refers to the ‘amount of data’, which is growing day by day at a very fast pace. The size of data generated by humans, machines and their interactions on social media itself is massive. Researchers have predicted that 40 Zettabytes (40,000 Exabytes) will be generated by 2020, which is an increase of 300 times from 2005.
2. **VELOCITY**: Velocity is defined as the pace at which different sources generate the data every day. This flow of data is massive and continuous. There are 1.03 billion Daily Active Users (Facebook DAU) on Mobile as of now, which is an increase of 22% year-over-year. This shows how fast the number of users are growing on social media and how fast the data is getting generated daily. If we are able to handle the velocity, we will be able to generate insights and take decisions based on real-time data.
3. **VARIETY**: As there are many sources which are contributing to Big Data, the type of data they are generating is different. It can be structured, semi-structured or unstructured. Hence, there is a variety of data which is getting generated every day. Earlier, we used to get the data from excel and databases, now the data are coming in the form of images, audios, videos, sensor data etc. as shown in below image. Hence, this variety of unstructured data creates problems in capturing, storage, mining and analyzing the data.
4. **VERACITY**: Veracity refers to the data in doubt or uncertainty of data available due to data inconsistency and incompleteness.Data available can sometimes get messy and maybe difficult to trust. With many forms of big data, quality and accuracy are difficult to control like Twitter posts with hashtags, abbreviations, typos and colloquial speech. The volume is often the reason behind for the lack of quality and accuracy in the data.
5. **VALUE**: It is all well and good to have access to big data but unless we can turn it into value it is useless.

## 

## 

## **Types of Big Data**

Big Data could be of three types:

* Structured
* Semi-Structured
* Unstructured

1. **Structured**: The data that can be stored and processed in a fixed format is called as Structured Data. Data stored in a relational database management system (RDBMS) is one example of ‘structured’ data. It is easy to process structured data as it has a fixed schema. Structured Query Language (SQL) is often used to manage such kind of Data.
2. **Semi-Structured**: Semi-Structured Data is a type of data which does not have a formal structure of a data model, i.e. a table definition in a relational DBMS, but nevertheless it has some organizational properties like tags and other markers to separate semantic elements that makes it easier to analyze. XML files or JSON documents are examples of semi-structured data.
3. **Unstructured**: The data which have unknown form and cannot be stored in RDBMS and cannot be analyzed unless it is transformed into a structured format is called as unstructured data. Text Files and multimedia contents like images, audios, videos are example of unstructured data. The unstructured data is growing quicker than others, experts say that 80 percent of the data in an organization are unstructured.

## 

## 

## **Examples of Big Data**

Daily we upload millions of bytes of data. 90 % of the world’s data has been created in last two years.

* Walmart handles more than 1 million customer transactions every hour.
* Facebook stores, accesses, and analyzes 30+ Petabytes of user generated data.
* 230+ millions of tweets are created every day.
* More than 5 billion people are calling, texting, tweeting and browsing on mobile phones worldwide.
* YouTube users upload 48 hours of new video every minute of the day.
* Amazon handles 15 million customer click stream user data per day to recommend products.
* 294 billion emails are sent every day. Services analyses this data to find the spams.
* Modern cars have close to 100 sensors which monitors fuel level, tire pressure etc. , each vehicle generates a lot of sensor data.

## 

## 

## 

## 

## **Applications of Big Data**

Almost all the industries today are leveraging Big Data applications in one or the other way.

1. **Smarter Healthcare**: Making use of the petabytes of patient’s data, the organization can extract meaningful information and then build applications that can predict the patient’s deteriorating condition in advance.
2. **Telecom:** Telecom sectors collects information, analyzes it and provide solutions to different problems. By using Big Data applications, telecom companies have been able to significantly reduce data packet loss, which occurs when networks are overloaded, and thus, providing a seamless connection to their customers.
3. **Retail:** Retail has some of the tightest margins, and is one of the greatest beneficiaries of big data. The beauty of using big data in retail is to understand consumer behavior. Amazon’s recommendation engine provides suggestion based on the browsing history of the consumer.
4. **Traffic control**: Traffic congestion is a major challenge for many cities globally. Effective use of data and sensors will be key to managing traffic better as cities become increasingly densely populated.
5. **Manufacturing:** Analyzing big data in the manufacturing industry can reduce component defects, improve product quality, increase efficiency, and save time and money.
6. **Search Quality**: Every time we are extracting information from google, we are simultaneously generating data for it. Google stores this data and uses it to improve its search quality.

## 

## 

## **Challenges with Big Data**

1. **Data Quality** – The problem here is the 4th V i.e. Veracity. The data here is very messy, inconsistent and incomplete. Dirty data cost $600 billion to the companies every year in the United States.
2. **Discovery** – Finding insights on Big Data is like finding a needle in a haystack. Analyzing petabytes of data using extremely powerful algorithms to find patterns and insights are very difficult.
3. **Storage** – The more data an organization has, the more complex the problems of managing it can become. The question that arises here is “Where to store it?”. We need a storage system which can easily scale up or down on-demand.
4. **Analytics** – In the case of Big Data, most of the time we are unaware of the kind of data we are dealing with, so analyzing that data is even more difficult.
5. **Security** – Since the data is huge in size, keeping it secure is another challenge. It includes user authentication, restricting access based on a user, recording data access histories, proper use of data encryption etc.
6. **Lack of Talent** – There are a lot of Big Data projects in major organizations, but a sophisticated team of developers, data scientists and analysts who also have sufficient amount of domain knowledge is still a challenge.

**HADOOP**

## 

## **Hadoop to Rescue**

Hadoop is an open-source software framework used for storing and processing Big Data in a distributed manner on large clusters of commodity hardware. Hadoop is licensed under the Apache v2 license. Hadoop was developed, based on the paper written by Google on MapReduce system and it applies concepts of functional programming. Hadoop is written in the Java programming language and ranks among the highest-level Apache projects. Hadoop was developed by Doug Cutting and Michael J. Cafarella.

Hadoop with its distributed processing, handles large volumes of structured and unstructured data more efficiently than the traditional enterprise data warehouse. Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to handle thousands of terabytes of data. Organizations are adopting Hadoop because it is an open source software and can run on commodity hardware (your personal computer). The initial cost savings are dramatic as commodity hardware is very cheap. As the organizational data increases, you need to add more & more commodity hardware on the fly to store it and hence, Hadoop proves to be economical. Additionally, Hadoop has a robust Apache community behind it that continues to contribute to its advancement.

**History of Hadoop**

Hadoop was created by Doug Cutting who had created the Apache Lucene which is origin in Apache Nutch.Hadoop is a part of Apache Lucene Project.Actually Apache Nutch was started in 2002 for working crawler and search system.Nutch Architecture would not scale up to billions of pages on the web. In 2003 google had published one Architecture called Google Distributed File System(GFS),which was solve the storage need for the very large files generated as a part of the web crawl and indexing process.

In 2004 based on GFS architecture Nutch was implementing open source called the Nutch Distributed File System (NDFS).In 2004 google was published Mapreduce,In 2005 Nutch developers had working on Mapreduce in Nutch Project.Most of the Algorithms had been ported to run using mapreduce and NDFS.

In February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop.At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale. This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster.

In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time, Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the New York Times.

In April 2008, Hadoop broke a world record to become the fastest system to sort a terabyte of data. Running on a 910-node cluster, Hadoop sorted one terabyte in 209 seconds (just under 3½ minutes), beating the previous year’s winner of 297 seconds.

**Modules in Hadoop**

The project includes the following modules

1. **Hadoop Common:** The common utilities that support the other Hadoop modules.
2. **Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
3. **Hadoop YARN:** A framework for job scheduling and cluster resource management.
4. **Hadoop MapReduce:** A YARN-based system for parallel processing of large data sets.

**HDFS**

Hadoop Distributed file system or HDFS is a Java based distributed file system that allows you to store large data across multiple nodes in a Hadoop cluster.

If one has ten machines or ten computers with a hard drive of 1 TB on each machine and HDFS as a platform on top of these ten machines, one will get HDFS as a storage service. Hadoop Distributed File System is distributed in such a way that every machine contributes their individual storage for storing any kind of data.

**Advantages of HDFS**

**1. Distributed Storage:**

When we access Hadoop Distributed file system from any of the ten machines in the Hadoop cluster, we will feel as if we have logged into a single large machine which has a storage capacity of 10 TB. It means that we can store a single large file of 10 TB which will be distributed over the ten machines (1 TB each). So, it is not limited to the physical boundaries of each individual machine.

**2. Distributed & Parallel Computation:**

Because the data is divided across the machines, it allows us to take advantage of Distributed and Parallel Computation. If it takes 43 minutes to process 1 TB file on a single machine. Each of the nodes is working with a part of the 1 TB file in parallel. Therefore, the work which was taking 43 minutes before, gets finished in just 4.3 minutes now as the work got divided over ten machines.

### **3. Horizontal Scalability:**

There are two types of scaling: vertical and horizontal. In vertical scaling (scale up), we increase the hardware capacity of your system. In other words, we procure more RAM or CPU and add it to our existing system to make it more robust and powerful. But there are challenges associated with vertical scaling or scaling up:

* There is always a limit to which we can increase your hardware capacity. So, we can’t keep on increasing the RAM or CPU of the machine.
* In vertical scaling, we stop our machine first. Then we increase the RAM or CPU to make it a more robust hardware stack. After we have increased our hardware capacity, we restart the machine. This down time when we are stopping your system becomes a challenge.

In case of horizontal scaling (scale out), we add more nodes to existing cluster instead of increasing the hardware capacity of individual machines. And most importantly, we can add more machines on the go i.e. without stopping the system. Therefore, while scaling out we don’t have any downtime or green zone, nothing of such sort. At the end of the day, we will have more machines working in parallel to meet your requirements.

**HDFS ARCHITECTURE**

Apache HDFS or Hadoop Distributed File System is a block-structured file system where each file is divided into blocks of a predetermined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

* **NAMENODE**

NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes). NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients.The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

### Functions of NameNode:

* It is the master daemon that maintains and manages the DataNodes (slave nodes)
* It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
  + FsImage: It contains the complete state of the file system namespace since the start of the NameNode.
  + EditLogs: It contains all the recent modifications made to the file system with respect to the most recent FsImage.
* It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
* It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
* It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
* The NameNode is also responsible to take care of the replication factor of all the blocks which we will discuss in detail later in this HDFS tutorial blog.
* In case of the DataNode failure, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

## 

## **DataNode:**

DataNodes are the slave nodes in HDFS. Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. The DataNode is a block server that stores the data in the local file ext3 or ext4.

### *Functions of DataNode:*

* These are slave daemons or process which runs on each slave machine.
* The actual data is stored on DataNodes.
* The DataNodes perform the low-level read and write requests from the file system’s clients.
* They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

## **Secondary NameNode:**

Apart from these two daemons, there is a third daemon or a process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. Secondary namenode is not a backup Namenode.

### Functions of Secondary NameNode:

* The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system.
* It is responsible for combining the EditLogs with FsImage from the NameNode.
* It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.

Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

**PROJECT IMPLEMENTATION**

**Assumptions**

1. Hadoop cluster is installed and running
2. Pig, Hive and Sqoop are installed

**Prerequisites**

The h1b data has nearly 3 million records.

It is present in the JSON format which has been converted into CSV format using Hive.

The columns in the dataset include:

1. CASE\_STATUS:

Status associated with the last significant event or decision.

Valid values include "Certified","Certified-Withdrawn","Denied"and"Withdrawn".

Certified: Employer filed the LCA, which was approved by DOL

Certified Withdrawn: LCA was approved but later withdrawn by employer

Withdrawn: LCA was withdrawn by employer before

Denied: LCA was denied by DOL

2. EMPLOYER\_NAME: Name of employer submitting labour condition application.

3. SOC\_NAME: the Occupational name associated with the SOC\_CODE. SOC\_CODE is the occupational code associated with the job being requested for temporary labour condition, as classified by the Standard Occupational Classification (SOC) System.

4. JOB\_TITLE: Title of the job

5. FULL\_TIME\_POSITION:

Y = Full Time Position;

N = Part Time Position

6. PREVAILING\_WAGE: Prevailing Wage for the job being requested for temporary labour condition. The wage is listed at annual scale in USD. The prevailing wage for a job position is defined as the average wage paid to similarly employed workers in the requested occupation in the area of intended employment. The prevailing wage is based on the employer’s minimum requirements for the position.

7. YEAR: Year in which the H1B visa petition was filed

8. WORKSITE: City and State information of the foreign worker’s intended area of employment

9. lon: longitude of the Worksite

10. lat: latitude of the Worksite

**STEPS FOR CONVERSION**

hive(default)> create database final;

hive(default)> use final;

hive(final)> CREATE TABLE h1b\_applications(s\_no int,case\_status string, employer\_name string, soc\_name string, job\_title string, full\_time\_position string,prevailing\_wage bigint,year string, worksite string, longitude double, latitude double )

> ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'

> WITH SERDEPROPERTIES (

> "separatorChar" = ",",

> "quoteChar" = "\"")

> STORED AS TEXTFILE;

hive(final)> load data local inpath '/home/hduser/h1b.csv' overwrite into table h1b\_applications;

hive(final)> CREATE TABLE h1b\_app2(s\_no int, case\_status string, employer\_name string, soc\_name string, job\_title string, full\_time\_position string,prevailing\_wage bigint,year string, worksite string, longitude double, latitude double )

> row format delimited

> fields terminated by '\t'

> STORED AS TEXTFILE;

hive(final)> INSERT OVERWRITE TABLE h1b\_app2 SELECT regexp\_replace(s\_no, "\t", ""), regexp\_replace(case\_status,"\t",""),regexp\_replace(employer\_name,"\t",""), regexp\_replace(soc\_name, "\t", ""), regexp\_replace(job\_title, "\t", ""), regexp\_replace(full\_time\_position, "\t", ""), prevailing\_wage, regexp\_replace(year, "\t", ""), regexp\_replace(worksite, "\t", ""), regexp\_replace(longitude, "\t", ""), regexp\_replace(latitude, "\t","") FROM h1b\_applications where case\_status != "NA";

hive(final)> CREATE TABLE h1b\_final(s\_no int,case\_status string, employer\_name string, soc\_name string, job\_title string, full\_time\_position string,prevailing\_wage bigint,year string, worksite string, longitude double, latitude double )

> row format delimited

> fields terminated by '\t'

> STORED AS TEXTFILE;

hive(final)> INSERT OVERWRITE TABLE h1b\_final SELECT s\_no,

> case when trim(case\_status) = "PENDING QUALITY AND COMPLIANCE REVIEW - UNASSIGNED" then "DENIED"

> when trim(case\_status) = "REJECTED" then "DENIED"

> when trim(case\_status) = "INVALIDATED" then "DENIED"

> else case\_status end,

> employer\_name, soc\_name, job\_title, full\_time\_position,

> case when prevailing\_wage is null then 100000

> else prevailing\_wage end,

> year, worksite, longitude, latitude FROM h1b\_app2;

**MAPREDUCE**

MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.

* MapReduce consists of two distinct tasks – Map and Reduce.
* As the name MapReduce suggests, reducer phase takes place after mapper phase has been completed.
* So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
* The output of a Mapper or map job (key-value pairs) is input to the Reducer.
* The reducer receives the key-value pair from multiple map jobs.
* Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

**MAPPER**

Mapper task processes each input record and it generates a new <key, value> pairs. The <key, value> pairs can be completely different from the input pair. In mapper task, the output is the full collection of all these <key, value> pairs. Before writing the output for each mapper task, partitioning of output take place on the basis of the key and then sorting is done. This partitioning specifies that all the values for each key are grouped together.

MapReduce frame generates one map task for each InputSplit generated by the InputFormat for the job.

Mapper only understands <key, value> pairs of data, so before passing data to the mapper, data should be first converted into <key, value> pairs.

**HOW DOES MAPPER WORK?**

* **InputSplits** converts the physical representation of the block into logical for the mapper. To read the 100MB file, two InputSlits are required. One InputSplit is created for each block and one RecordReader and one mapper are created for each InputSplit. InputSplits do not always depend on the number of blocks, we can customize the number of splits for a particular file by setting mapred.max.split.size property during job execution.
* **RecordReader’s** responsibility is to keep reading/converting data into key-value pairs until the end of the file. Byte offset (unique number) is assigned to each line present in the file by RecordReader. Further, this key-value pair is sent to the mapper. The output of the mapper program is called as intermediate data (key-value pairs which are understandable to reduce).

**REDUCER**

The Reducer process the output of the mapper. After processing the data, it produces a new set of output. At last HDFS stores this output data.

Hadoop Reducer takes a set of an intermediate key-value pair produced by the mapper as the input and runs a Reducer function on each of them. One can aggregate, filter, and combine this data (key, value) in a number of ways for a wide range of processing. Reducer first processes the intermediate values for particular key generated by the map function and then generates the output (zero or more key-value pair).

One-one mapping takes place between keys and reducers. Reducers run in parallel since they are independent of one another. The user decides the number of reducers. By default number of reducers is 1.

## **Phases of Reducer**

There are 3 phases of Reducer in Hadoop MapReduce.

**1. Shuffle Phase**

In this phase, the sorted output from the mapper is the input to the Reducer. In Shuffle phase, with the help of HTTP, the framework fetches the relevant partition of the output of all the mappers.

### **2. Sort Phase**

In this phase, the input from different mappers is again sorted based on the similar keys in different Mappers. The shuffle and sort phases occur concurrently.

### **3. Reduce Phase**

In this phase, after shuffling and sorting, reduce task aggregates the key-value pairs. The OutputCollector.collect() method, writes the output of the reduce task to the Filesystem. Reducer output is not sorted.

**Task(1a) Is the number of petitions with Data Engineer job title increasing over time?**

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

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

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

public class question1a {

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

{

Configuration conf= new Configuration();

Job job= Job.getInstance(conf,"Question 1a");

job.setJarByClass(question1a.class);

job.setMapperClass(Question1aMapper.class);

job.setReducerClass(Question1aReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

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

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

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

}

}

**MAPPER CLASS**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

Public class question1aMapper extends Mapper<LongWritable,Text,Text,LongWritable>{

LongWritable one =new LongWritable(1);

public void map(LongWritable key,Text values,Context context) throws IOException, InterruptedException

{

if(key.get()> 0)

{

String [] token=values.toString().split("\t");

if(token[4]!=null && token[4].contains("DATA ENGINEER") && token[7]!=null && !token[7].equals("NA"))

{ Text answer= new Text("DATA ENGINEER"+","+token[7]);

context.write(answer,one);

}

}

}

}

**REDUCER CLASS**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class Question1aReducer extends Reducer <Text,LongWritable,Text,LongWritable>

{ LongWritable SUM=new LongWritable(0); int i=0;

String[] years={"2011","2012","2013","2014","2015","2016"};

long [] arr=new long[6];

public void reduce(Text key,Iterable<LongWritable> values ,Context context) throws IOException, InterruptedException

{

long sum=0;

for(LongWritable val:values)

sum+=val.get();

arr[i++]=sum;

}

public void cleanup(Context context) throws IOException, InterruptedException

{

for (int i=0;i<6;i++)

if (i==0)

context.write(new Text(years[i]), new LongWritable(0));

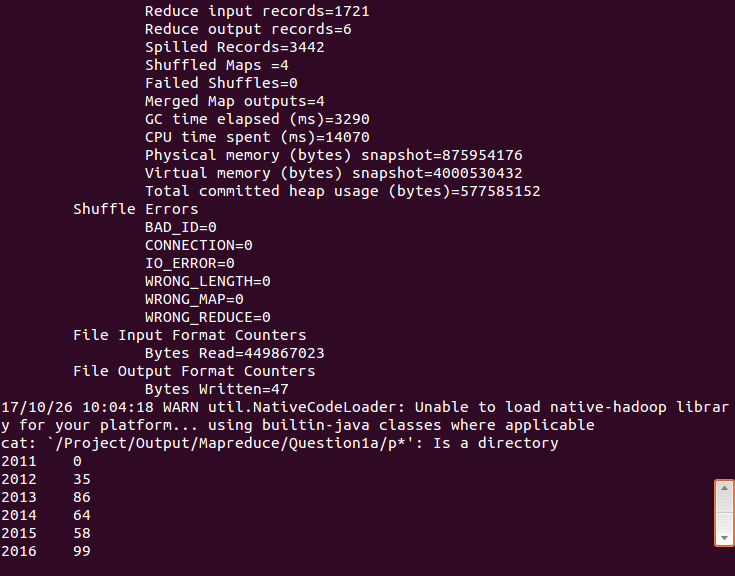
else

context.write(new Text(years[i]),new LongWritable((arr[i]-arr[i-1])\*100/arr[i-1]));

}

}

**Output:**

****

**Thus, we can see that the number of petitions has grown over the year till 2013 after which it dropped by 22% which further decreased by 6% in 2015 and finally increased by 41% in 2016.**

**Task(2a) Which part of the US has the most Data Engineer jobs for each year?**

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

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

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

public class Question2a

{

public static void main(String args[]) throws IOException, InterruptedException, ClassNotFoundException

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Top 5 Data Engineer in a worksite");

job.setJarByClass(Question2a.class);

job.setMapperClass(Question2aMapper.class);

job.setPartitionerClass(Question2aPartitioner.class);

job.setReducerClass(Question2aReducer.class);

job.setNumReduceTasks(7);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

**MAPPER CLASS**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class Question2aMapper extends Mapper < LongWritable, Text, Text, LongWritable > {

LongWritable one = new LongWritable(1);

public void map(LongWritable key, Text values, Context context) throws IOException,

InterruptedException {

if (key.get() > 0)

{

String[] token = values.toString().split("\t");

if (token[4] != null && token[4].contains("DATA ENGINEER") && token[8] != null && !token[8].equals("NA")) {

Text answer = new Text(token[8].replaceAll("\"", "") + "\t" + token[7]);

context.write(answer, one);

}

}

}

}

**PARTITIONER CLASS**

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Partitioner;

public class Question2aPartitioner extends

Partitioner < Text, LongWritable > {

@Override

public int getPartition(Text key, LongWritable value, int numReduceTasks) {

String[] str = key.toString().split("\t");

if (str[1].equals("2011"))

return 0;

if (str[1].equals("2012"))

return 1;

if (str[1].equals("2013"))

return 2;

if (str[1].equals("2014"))

return 3;

if (str[1].equals("2015"))

return 4;

if (str[1].equals("2016"))

return 5;

else

return 6;

}

}

**REDUCER CLASS**

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class Question2aReducer extends Reducer<Text,LongWritable,NullWritable,Text>

{

private TreeMap<LongWritable, Text> Top5DataEngineer = new TreeMap<LongWritable, Text>();

long sum=0;

public void reduce(Text key,Iterable <LongWritable> values,Context context) throws IOException, InterruptedException

{

sum=0;

for(LongWritable val:values)

{

sum+=val.get();

}

Top5DataEngineer.put(new LongWritable(sum),new Text(key+","+sum));

if (Top5DataEngineer.size()>5)

Top5DataEngineer.remove(Top5DataEngineer.firstKey());

}

protected void cleanup(Context context)throws IOException, InterruptedException

{

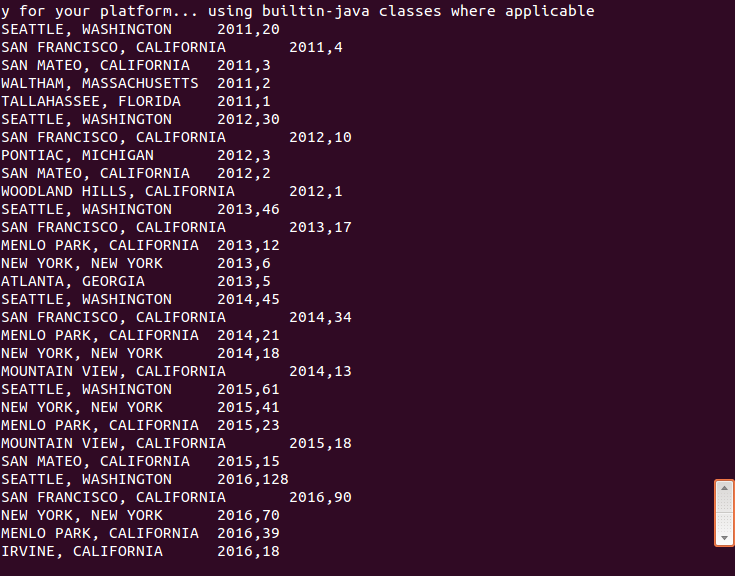
for (Text t : Top5DataEngineer.descendingMap().values())

context.write(NullWritable.get(), t);

}

}

**Output:**

**Thus, we can conclude that Seattle had the highest number of Data Engineer posts in the US for the all years with**

1. **20 in 2011**
2. **30 in 2012**
3. **46 in 2013**
4. **45 in 2014**
5. **61 in 2015**
6. **128 in 2016**

**Task 3: Which industry has the most number of Data Scientist positions?**

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class Question3Reducer extends Reducer < Text, LongWritable, NullWritable, Text > {

private TreeMap < LongWritable,

Text > DataScientistJobs = new TreeMap < LongWritable,

Text > ();

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

InterruptedException {

long sum = 0;

for (LongWritable val: values)

sum += val.get();

DataScientistJobs.put(new LongWritable(sum), new Text(key.toString().replaceAll("\"", "") + "," + sum));

if (DataScientistJobs.size() > 5)

DataScientistJobs.remove(DataScientistJobs.firstKey());

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

for (Text t: DataScientistJobs.descendingMap().values())

context.write(NullWritable.get(), t);

}

}

**MAPPER CLASS**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class Question3Mapper extends Mapper < LongWritable, Text, Text, LongWritable > {

LongWritable one = new LongWritable(1);

public void map(LongWritable key, Text values, Context context) throws IOException,

InterruptedException {

if (key.get() > 0) {

String[] token = values.toString().split("\t");

if (token[4].contains("DATA SCIENTIST")) {

Text answer = new Text(token[3].replaceAll("\"", ""));

context.write(answer, one);

}

}

}

}

**REDUCER CLASS**

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class Question3Reducer extends Reducer < Text, LongWritable, NullWritable, Text > {

private TreeMap < LongWritable,

Text > DataScientistJobs = new TreeMap < LongWritable,

Text > ();

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

InterruptedException {

long sum = 0;

for (LongWritable val: values)

sum += val.get();

DataScientistJobs.put(new LongWritable(sum), new Text(key.toString().replaceAll("\"", "") + "," + sum));

if (DataScientistJobs.size() > 5)

DataScientistJobs.remove(DataScientistJobs.firstKey());

}

protected void cleanup(Context context) throws IOException,

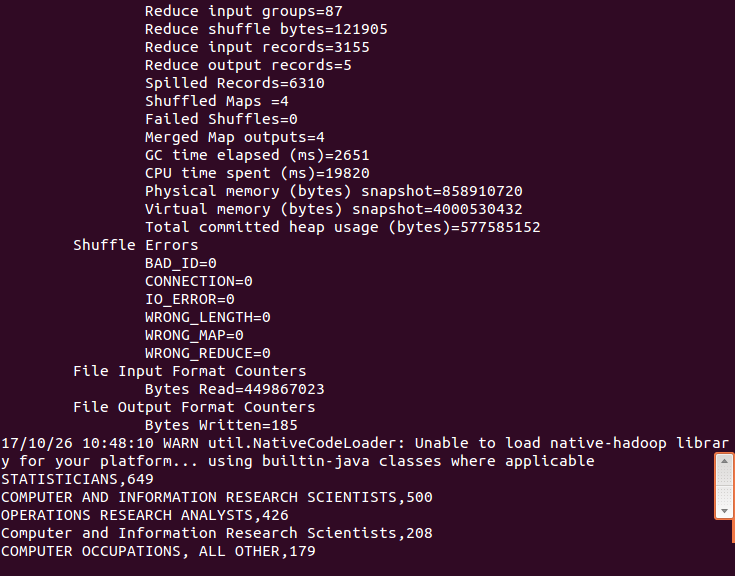
InterruptedException {

for (Text t: DataScientistJobs.descendingMap().values())

context.write(NullWritable.get(), t);

}

}

**Output: **

**Thus, the industry(soc\_name) which has the highest number of Data Scientist jobs is Statisticians(649).**

**Task4: Which top 5 employers file the most petitions each year?**

import java.io.IOException;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

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

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

public class Question4 {

public static void main(String args[]) throws IOException, InterruptedException, ClassNotFoundException

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Top 5 Employers");

job.setJarByClass(Question4.class);

job.setMapperClass(Question4Mapper.class);

job.setPartitionerClass(Question4Partitioner.class);

job.setReducerClass(Question4Reducer.class);

job.setNumReduceTasks(7);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(LongWritable.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.class);

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

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

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

}

}

**MAPPER CLASS**

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class Question4Mapper extends Mapper < LongWritable, Text, Text, LongWritable > {

LongWritable one = new LongWritable(1);

public void map(LongWritable key, Text value, Context context) throws IOException,

InterruptedException {

if (key.get() > 0)

{

String[] token = value.toString().split(",(?=([^\"]\*\"[^\"]\*\")\*[^\"]\*$)");

if (!token[1].equals("NA") && token[1] != null && !token[2].equals("NA") && token[2] != null && !token[7].equals("NA") && token[7] != null) {

Text answer = new Text(token[2].replaceAll("\"", "") + "\t" + token[7]);

context.write(answer, one);

}

}

}

}

**PARTITIONER CLASS**

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Partitioner;

public class Question4Partitioner extends

Partitioner < Text, LongWritable > {

@Override

public int getPartition(Text key, LongWritable value, int numReduceTasks) {

String[] str = key.toString().split("\t");

if (str[1].equals("2011"))

return 0;

if (str[1].equals("2012"))

return 1;

if (str[1].equals("2013"))

return 2;

if (str[1].equals("2014"))

return 3;

if (str[1].equals("2015"))

return 4;

if (str[1].equals("2016"))

return 5;

else

return 6;

}

}

**REDUCER CLASS**

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class Question4Reducer extends Reducer < Text, LongWritable, NullWritable, Text > {

private TreeMap < LongWritable,

Text > Top5Employers = new TreeMap < LongWritable,

Text > ();

long sum = 0;

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

InterruptedException {

sum = 0;

for (LongWritable val: values) {

sum += val.get();

}

Top5Employers.put(new LongWritable(sum), new Text(key + "," + sum));

if (Top5Employers.size() > 5)

Top5Employers.remove(Top5Employers.firstKey());

}

protected void cleanup(Context context) throws IOException,

InterruptedException {

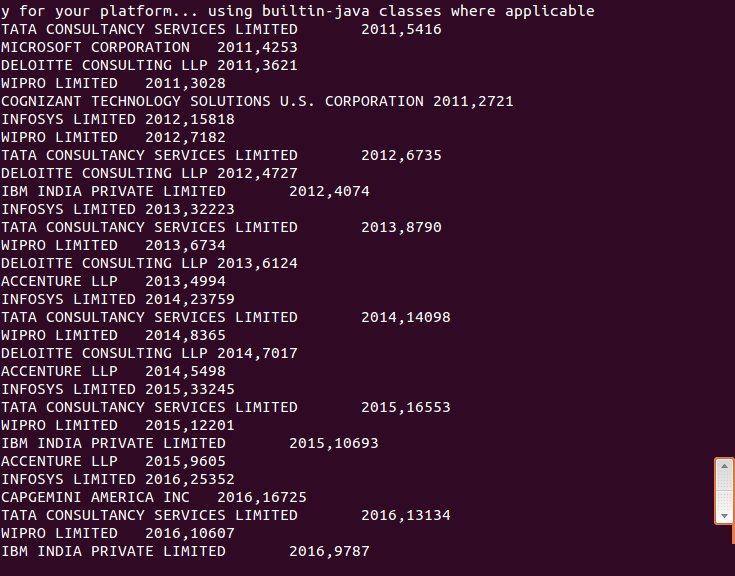
for (Text t: Top5Employers.descendingMap().values())

context.write(NullWritable.get(), t);

}

}

**Output:**

****

**Thus, top 5 employers who file the most number of petitions each year are:**

1. **For 2011**

* **Tata Consultancy Services Limited**
* **Microsoft Corporation**
* **Deloitte Consulting LLP**
* **Wipro Limited**
* **Cognizant Technology Solutions U.S. Corporation**

**2. For 2012**

* **Infosys Limited**
* **Wipro Limited**
* **Tata Consultancy Services Limited**
* **Deloitte Consulting LLP**
* **IBM India Private Limited**

**3. For 2013**

* **Infosys Limited**
* **Tata Consultancy Services Limited**
* **Wipro Limited**
* **Deloitte Consulting LLP**
* **Accenture LLP**

**4. For 2014**

* **Infosys Limited**
* **Tata Consultancy Services Limited**
* **Wipro Limited**
* **Deloitte Consulting LLP**
* **Accenture LLP**

**5. For 2015**

* **Infosys Limited**
* **Tata Consultancy Services Limited**
* **Wipro Limited**
* **IBM India Private Limited**
* **Accenture LLP**

**6. For 2016**

* **Infosys Limited**
* **Capgemini America INC**
* **Tata Consultancy Services Limited**
* **Wipro Limited**
* **IBM India Private Limited**

**HIVE**

Apache Hive is a data warehouse system built on top of Hadoop and is used for analyzing structured and semi-structured data. Hive abstracts the complexity of Hadoop MapReduce. Basically, it provides a mechanism to project structure onto the data and perform queries written in HQL (Hive Query Language) that are similar to SQL statements. Internally, these queries or HQL gets converted to map reduce jobs by the Hive compiler. Therefore, you don’t need to worry about writing complex MapReduce programs to process your data using Hadoop. It is targeted towards users who are comfortable with SQL. Apache Hive supports Data Definition Language (DDL), Data Manipulation Language (DML) and User Defined Functions (UDF).

SQL + Hadoop MapReduce = HiveQL

**Task(2b) Find top 5 locations in the US that have got certified visa for each year.**

select worksite,count(case\_status) AS count,year from final.h1b\_final where year ='2011' and case\_status='CERTIFIED' group by worksite,year order by count desc limit 5;

select worksite,count(case\_status) AS count,year from final.h1b\_final where year ='2012' and case\_status='CERTIFIED' group by worksite,year order by count desc limit 5;

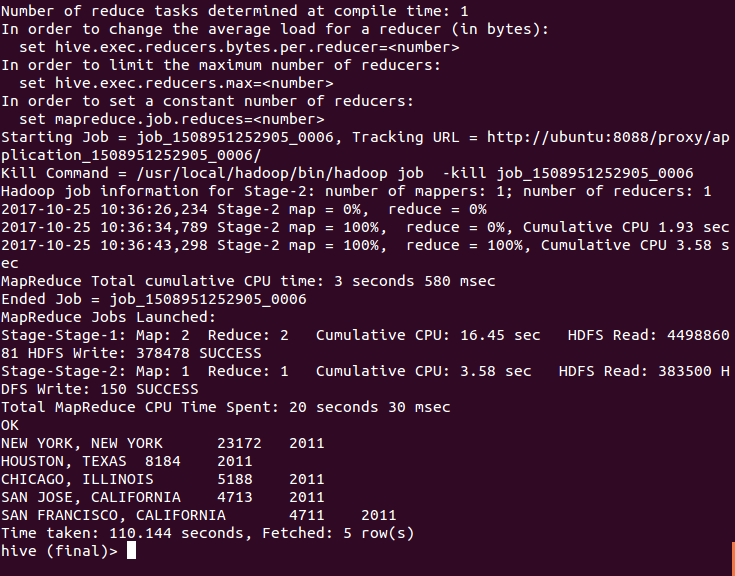
select worksite,count(case\_status) AS count,year from final.h1b\_final where year ='2013' and case\_status='CERTIFIED' group by worksite,year order by count desc limit 5;

select worksite,count(case\_status) AS count,year from final.h1b\_final where year ='2014' and case\_status='CERTIFIED' group by worksite,year order by count desc limit 5;

select worksite,count(case\_status) AS count,year from final.h1b\_final where year ='2015' and case\_status='CERTIFIED' group by worksite,year order by count desc limit 5;

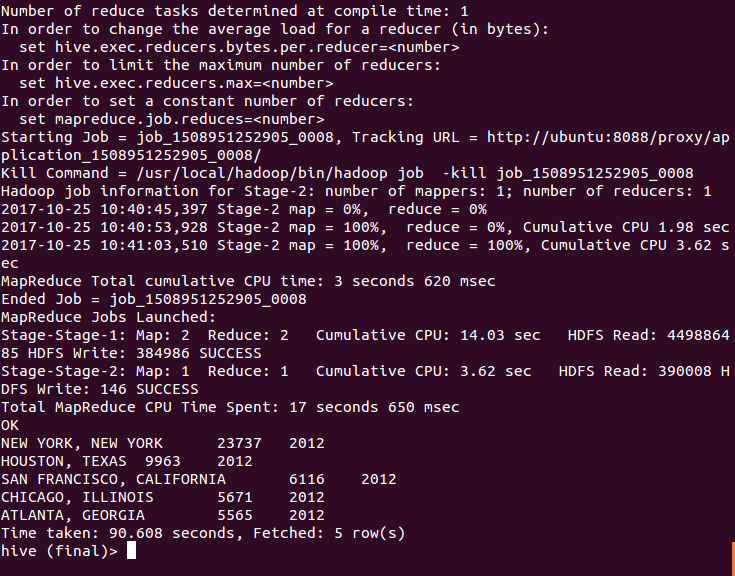
select worksite,count(case\_status) AS count,year from final.h1b\_final where year ='2016' and case\_status='CERTIFIED' group by worksite,year order by count desc limit 5;

**Output:**

****

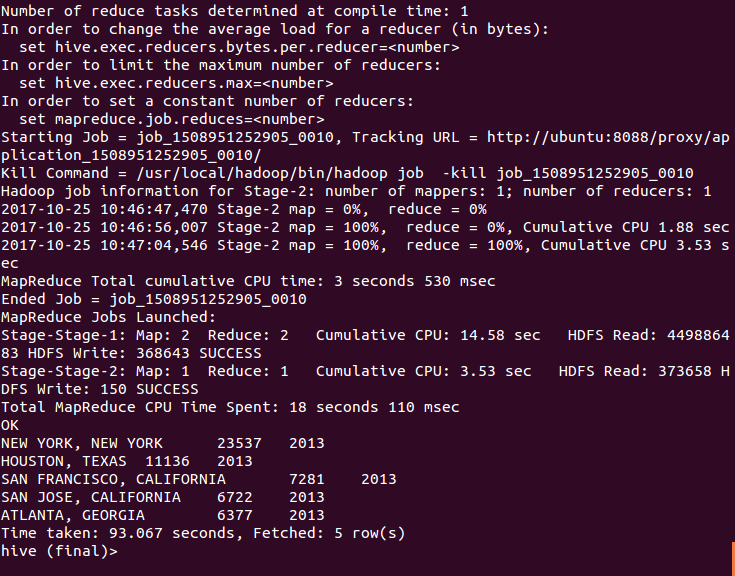
**Thus, for the year 2011, the top 5 locations in the US that have got certified visa are:**

1. **New York**
2. **Houston**
3. **Chicago**
4. **San Jose**
5. **San Francisco**

****

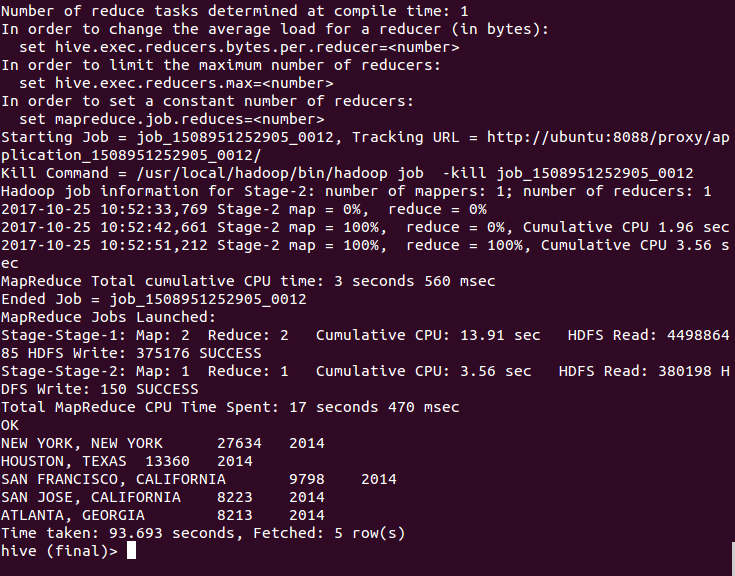
**Thus, for the year 2012, top 5 locations in the US that have got certified visa are:**

1. **New York**
2. **Houston**
3. **San Francisco**
4. **Chicago**
5. **Atlanta**



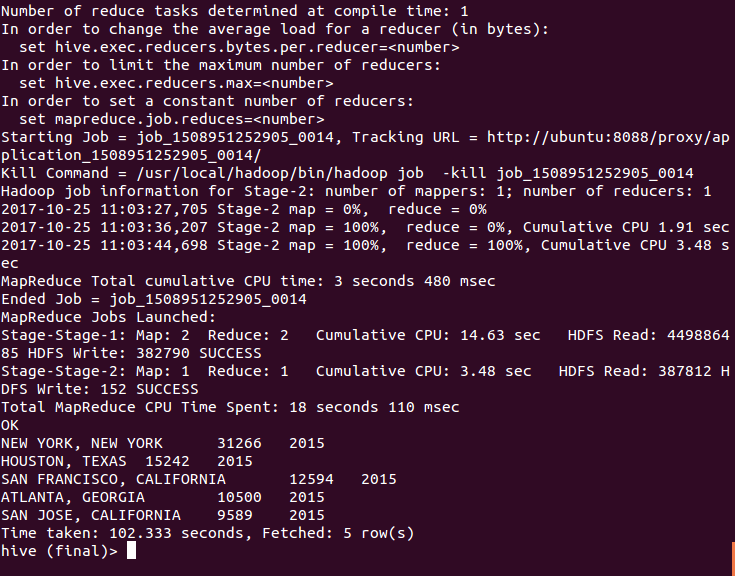
**Thus, for the year 2013, the top 5 locations in the US that have got certified visa are:**

1. **New York**
2. **Houston**
3. **San Francisco**
4. **San Jose**
5. **Atlanta**



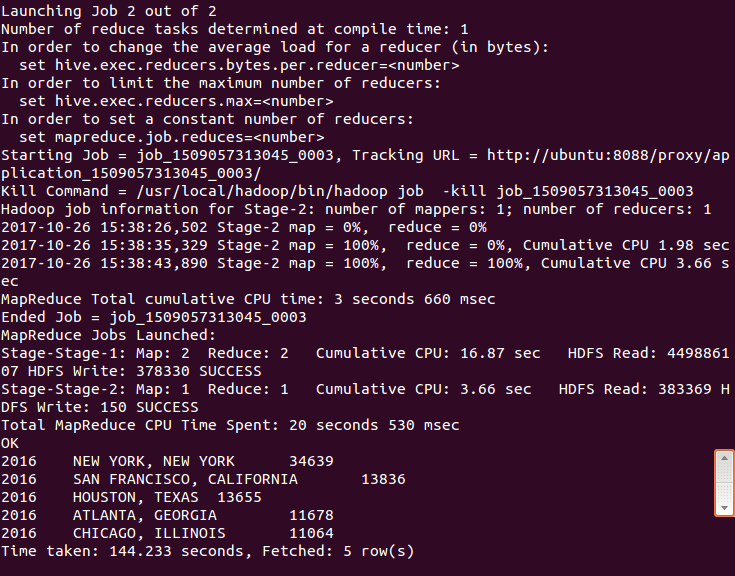
**Thus, for the year 2014**, **top 5 locations in the US that have got certified visa are:**

1. **New York**
2. **Houston**
3. **San Francisco**
4. **San Jose**
5. **Atlanta**

****

**Thus, for the year 2015**, **top 5 locations in the US that have got certified visa are:**

1. **New York**
2. **Houston**
3. **San Francisco**
4. **Atlanta**
5. **San Jose**



**Thus, for the year 2013, the top 5 locations in the US that have got certified visa are:**

1. **New York**
2. **San Francisco**
3. **Houston**
4. **Atlanta**
5. **Chicago**

**Task(5) Find the most popular top 10 job positions for H1B visa applications for each year.**

select job\_title,year,count(case\_status ) as count from final.h1b\_final where year = 2011 group by job\_title,year order by count desc limit 10;

select job\_title,year,count(case\_status ) as count from final.h1b\_final where year = 2012 group by job\_title,year order by count desc limit 10;

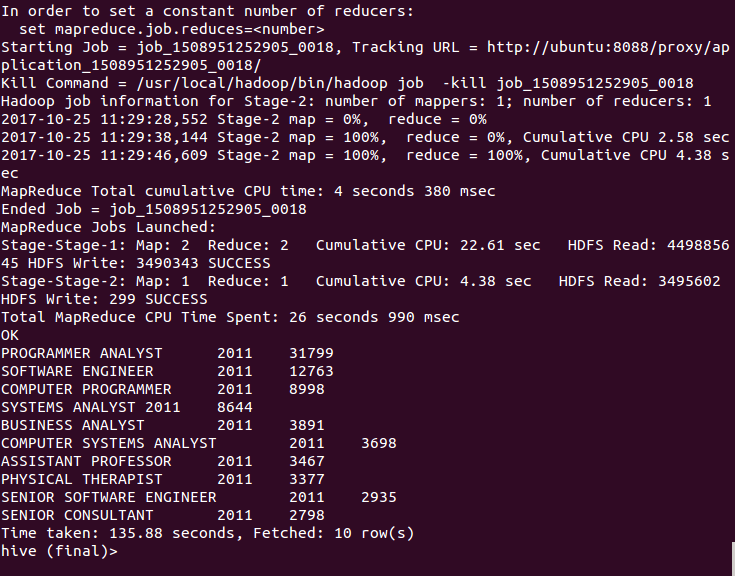
select job\_title,year,count(case\_status ) as count from final.h1b\_final where year = 2013 group by job\_title,year order by count desc limit 10;

select job\_title,year,count(case\_status ) as count from final.h1b\_final where year = 2014 group by job\_title,year order by count desc limit 10;

select job\_title,year,count(case\_status ) as count from final.h1b\_final where year = 2015 group by job\_title,year order by count desc limit 10;

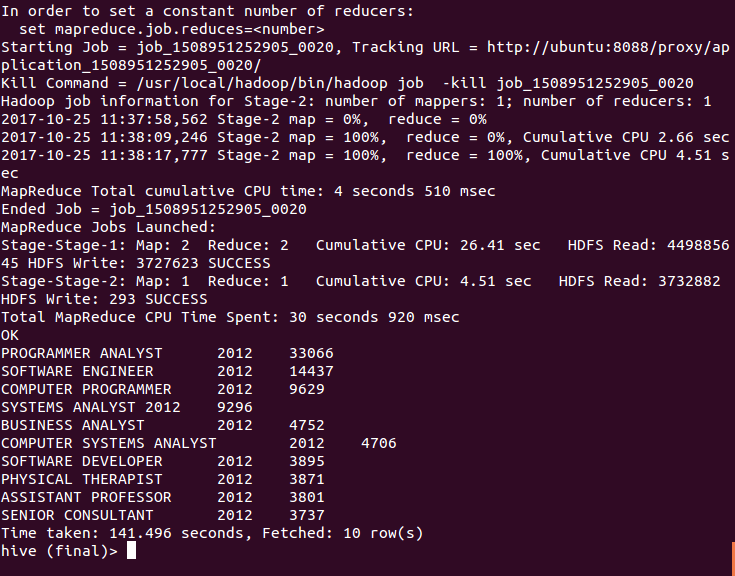
select job\_title,year,count(case\_status ) as count from final.h1b\_final where year = 2016 group by job\_title,year order by count desc limit 10;

**Output:**

****

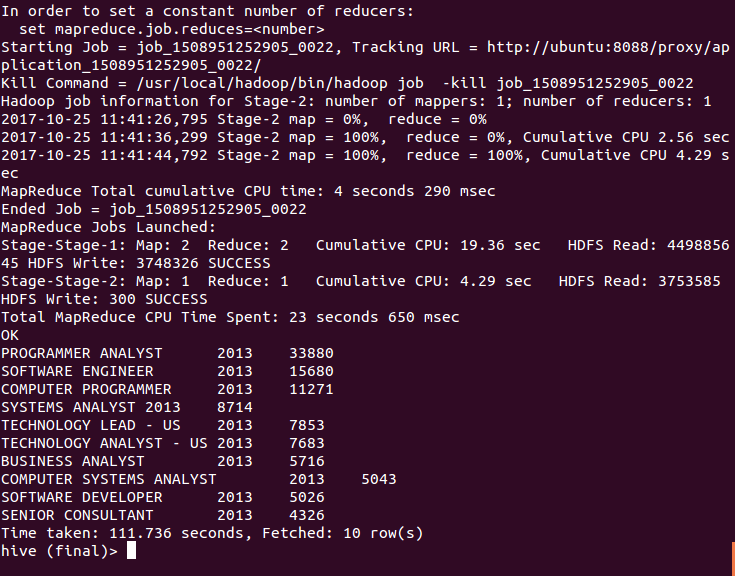
**Thus, the most popular top 10 job positions for H1B visa applications for the year 2011 are:**

1. **Programmer Analyst**
2. **Software Engineer**
3. **Computer Programmer**
4. **System Analyst**
5. **Business Analyst**
6. **Computer Systems Analyst**
7. **Assistant Professor**
8. **Physical Therapist**
9. **Senior Software Engineer**
10. **Senior Consultant**



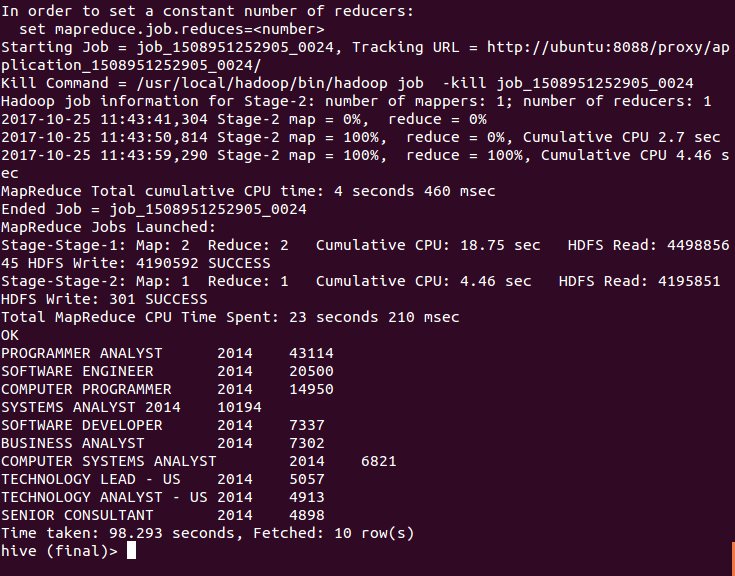
**Thus,** **the most popular top 10 job positions for H1B visa applications for the year 2012 are:**

1. **Programmer Analyst**
2. **Software Engineer**
3. **Computer Programmer**
4. **Systems Analyst**
5. **Business Analyst**
6. **Computer Systems Analyst**
7. **Software Developer**
8. **Physical Therapist**
9. **Assistant Professor**
10. **Senior Consultant**

****

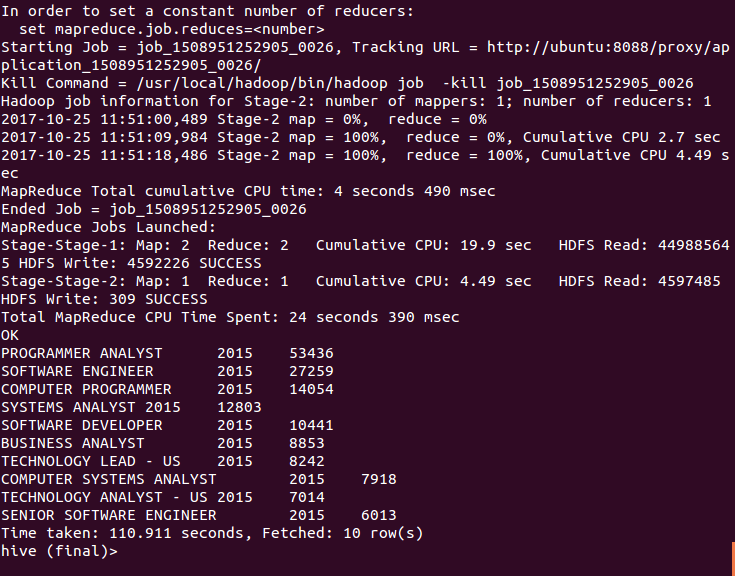
**Thus,** **the most popular top 10 job positions for H1B visa applications for the year 2013 are:**

1. **Programmer Analyst**
2. **Software Engineer**
3. **Computer Programmer**
4. **Systems Analyst**
5. **Technology Lead**
6. **Technology Analyst - US**
7. **Business Analyst**
8. **Computer Systems Analyst**
9. **Software Developer**
10. **Senior Consultant**

****

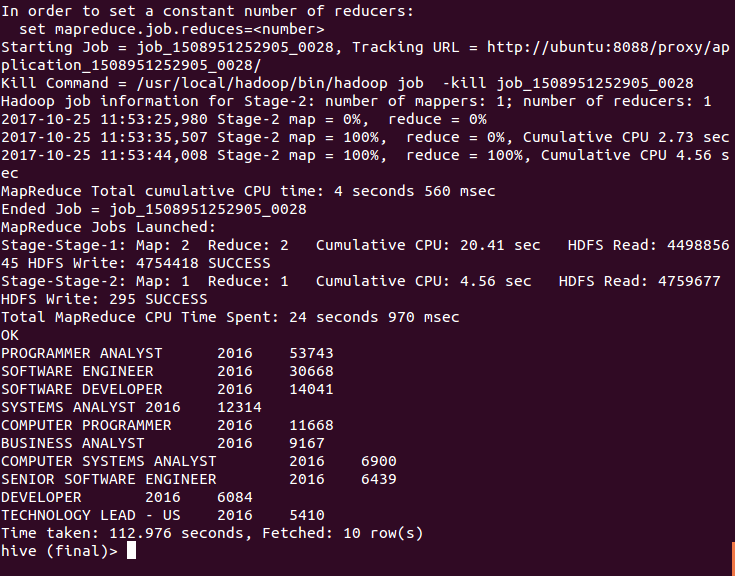
**Thus,** **the most popular top 10 job positions for H1B visa applications for the year 2014 are:**

1. **Programmer Analyst**
2. **Software Engineer**
3. **Computer Programmer**
4. **Systems Analyst**
5. **Software Developer**
6. **Business Analyst**
7. **Computer Systems Analyst**
8. **Technology Lead - US**
9. **Technology Analyst - US**
10. **Senior Consultant**

****

**Thus,** **the most popular top 10 job positions for H1B visa applications for the year 2015 are:**

1. **Programmer Analyst**
2. **Software Engineer**
3. **Computer Programmer**
4. **Systems Analyst**
5. **Software Developer**
6. **Business Analyst**
7. **Technology Lead - US**
8. **Computer Systems Analyst**
9. **Technology Analyst - US**
10. **Senior Software Engineer**

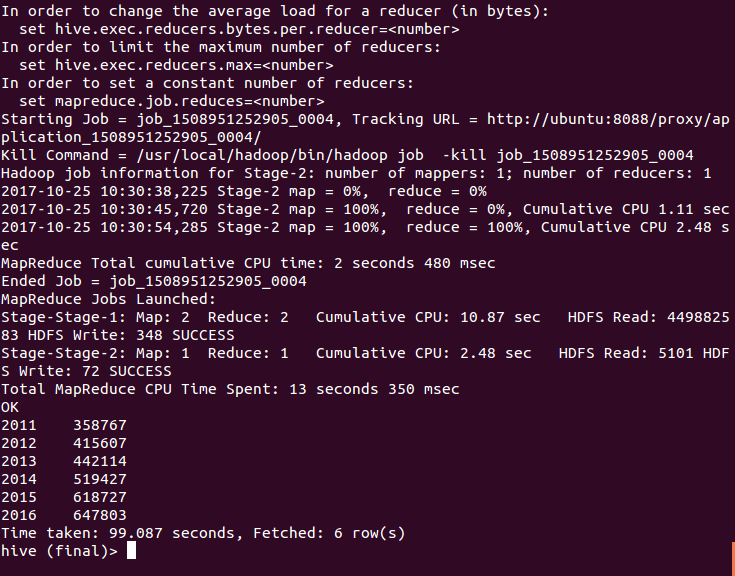
****

**Thus,** **the most popular top 10 job positions for H1B visa applications for the year 2016 are:**

1. **Programmer Analyst**
2. **Software Engineer**
3. **Software Developer**
4. **Systems Analyst**
5. **Computer Programmer**
6. **Business Analyst**
7. **Computer Systems Analyst**
8. **Senior Software Engineer**
9. **Developer**
10. **Technology Lead - US**

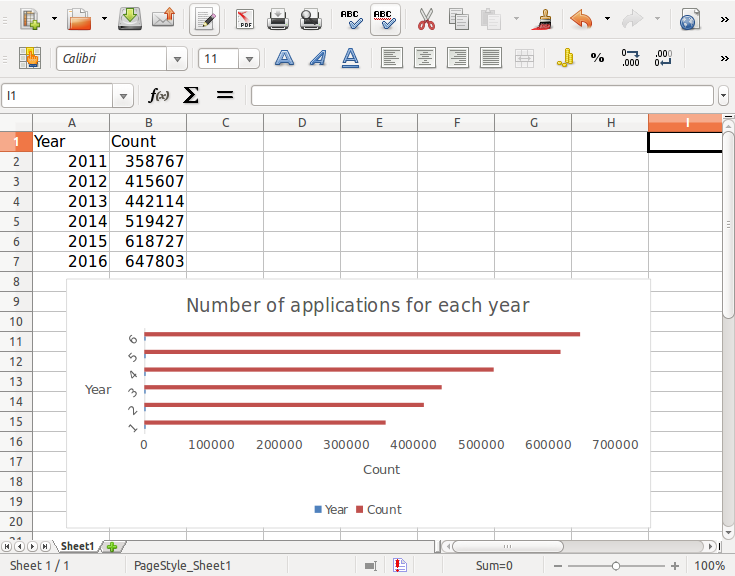
**Task(7) Create a bar graph to depict the number of applications for each year**

select year,count(\*) from final.h1b\_final group by year order by year;

**Thus, the total number of applications**

1. **For the year 2011 is 358767**
2. **For the year 2012 is 415607**
3. **For the year 2013 is 442114**
4. **For the year 2014 is 519427**
5. **For the year 2015 is 618727**
6. **For the year 2016 is 647803**

**Graph depicting total number of applications for each year**

****

**Task(8) Find the average Prevailing Wage for each job for each year (take part time and full time separate). Arrange output in descending order.**

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='Y' and year='2011' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='N' and year='2011' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='Y' and year='2012' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='N' and year='2012' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='Y' and year='2013' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='N' and year='2013' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='Y' and year='2014' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='N' and year='2014' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='Y' and year='2015' group by job\_title,full\_time\_position,year order by average desc;

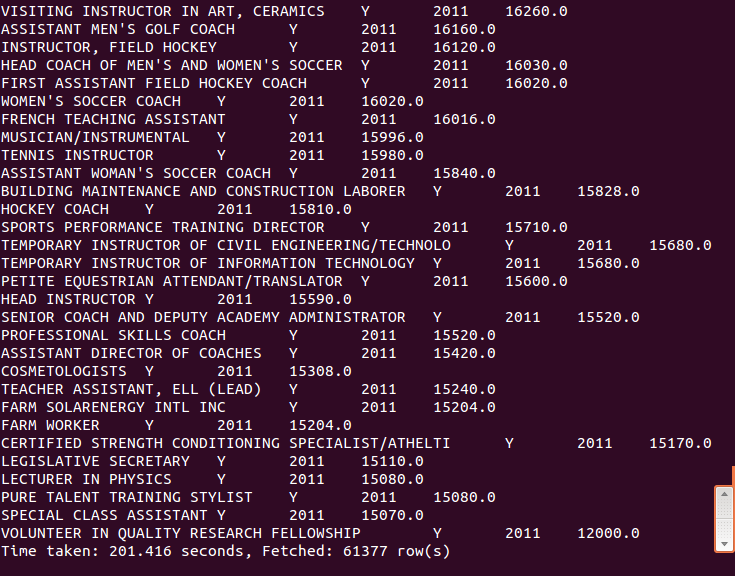
select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='N' and year='2015' group by job\_title,full\_time\_position,year order by average desc;

select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='Y' and year='2016' group by job\_title,full\_time\_position,year order by average desc;

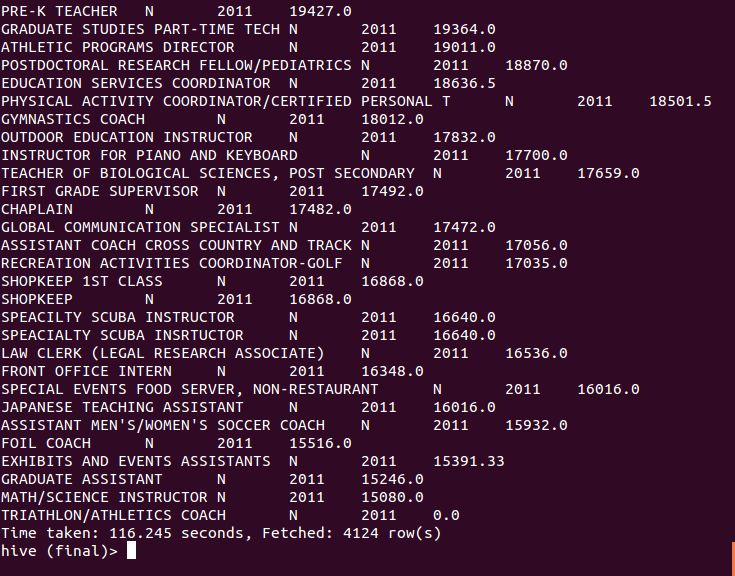
select job\_title,full\_time\_position,year,ROUND(avg(prevailing\_wage),2) as average from final.h1b\_final where full\_time\_position ='N' and year='2016' group by job\_title,full\_time\_position,year order by average desc;

**Output:**

**For year 2011**

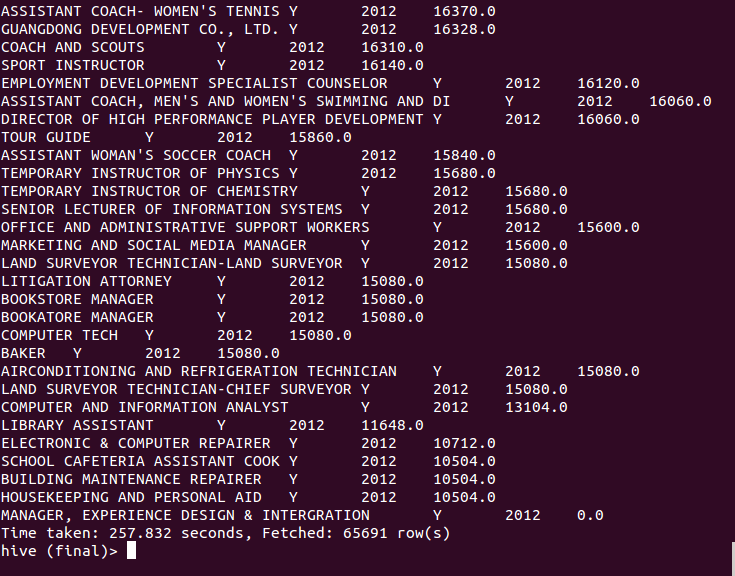


**Full time**

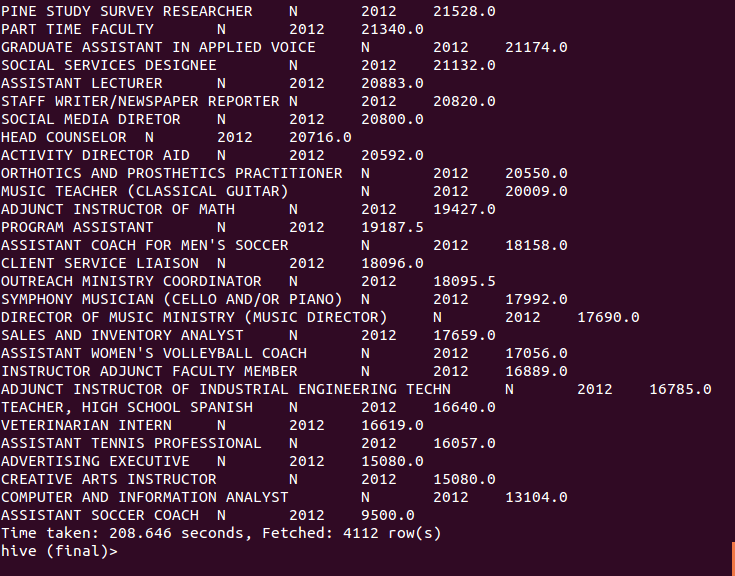
****

**Part Time**

**For the year 2012**

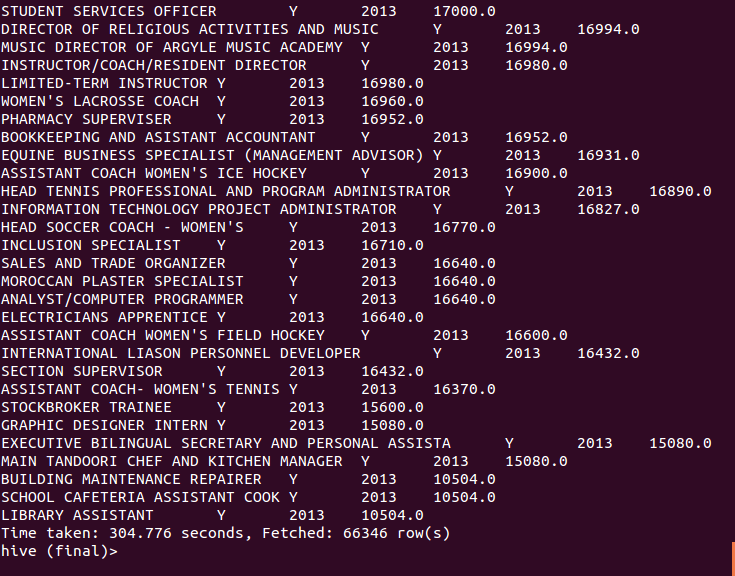
****

**Full time**

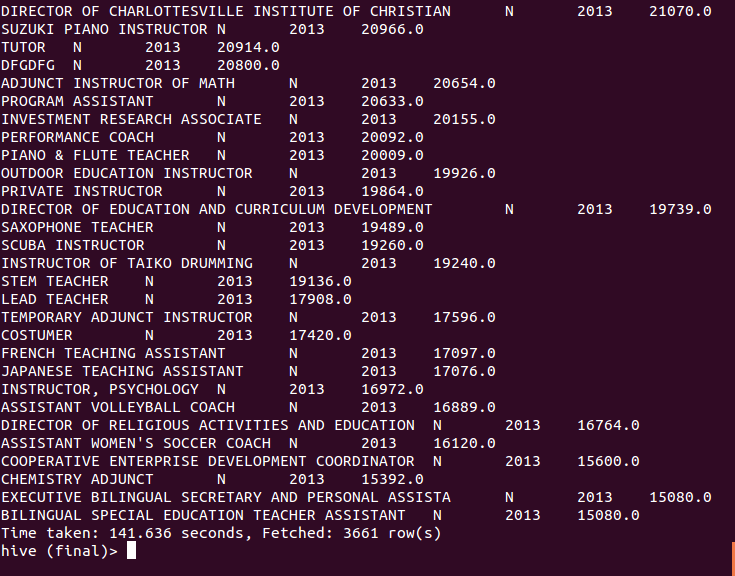
****

**Part time**

**For the year 2013**

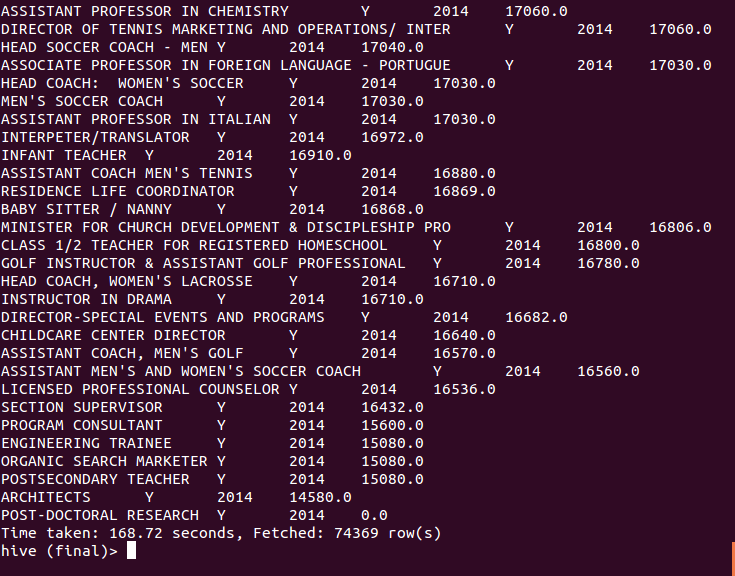
****

**Full time**

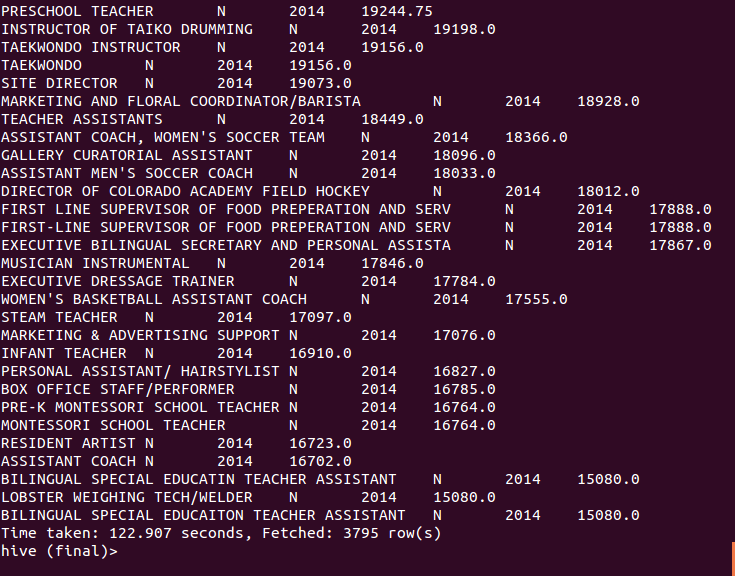
****

**Part Time**

**For the year 2014**

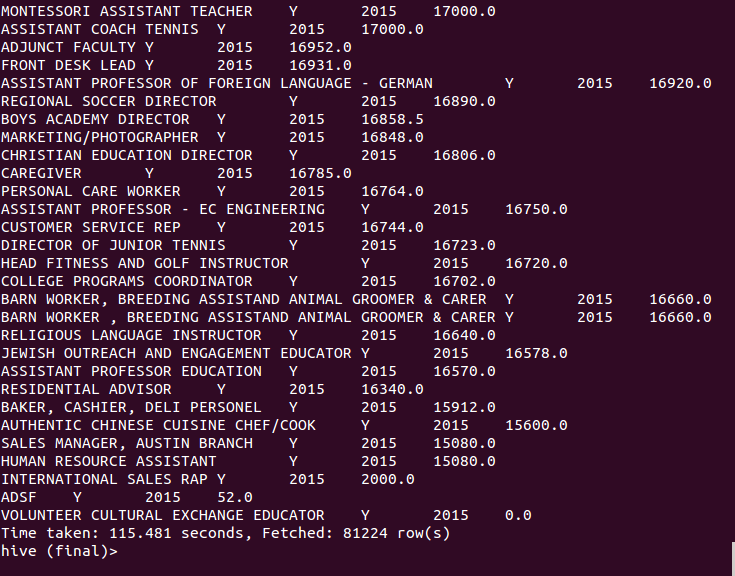
****

**Full time**

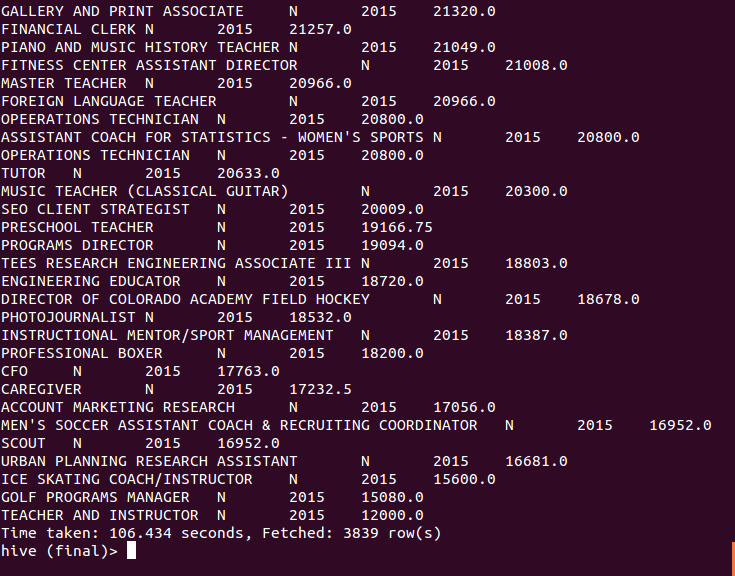
****

**Part time**

**For the year 2015**

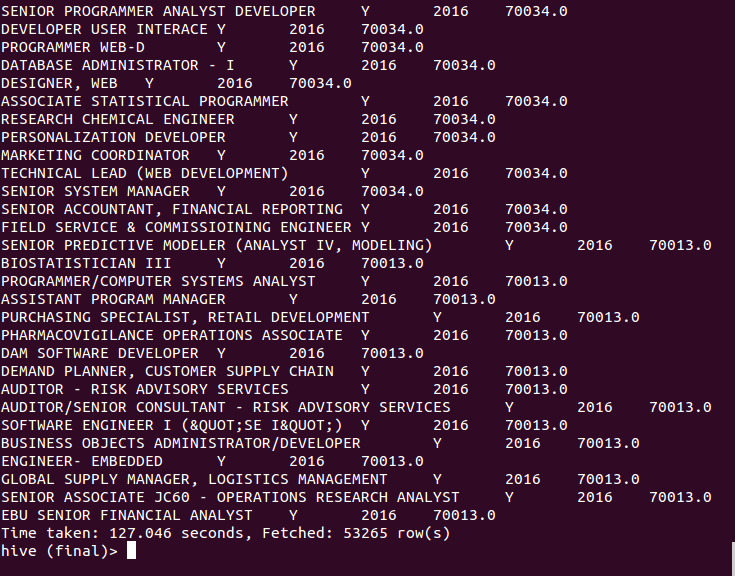
****

**Full time**

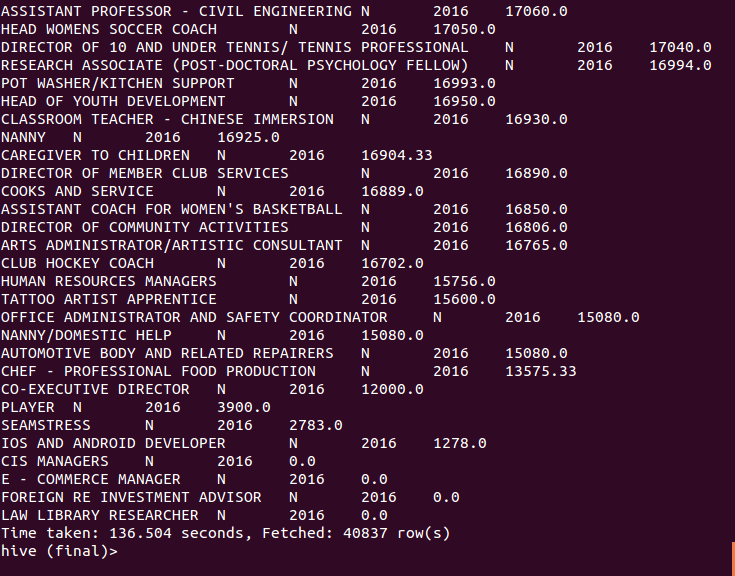
****

**Part time**

**For the year 2016**

****

**Full time**

****

**Part time**

**PIG**

Apache Pig is a platform, used to analyze large data sets representing them as data flows. It is designed to provide an abstraction over MapReduce, reducing the complexities of writing a MapReduce program. We can perform data manipulation operations very easily in Hadoop using Apache Pig.

The features of Apache pig are:

* Pig enables programmers to write complex data transformations without knowing Java.
* Apache Pig has two main components – the Pig Latin language and the Pig Run-time Environment, in which Pig Latin programs are executed.
* For data analysis, Pig gives a simple data flow language known as Pig Latin which has functionalities similar to SQL like join, filter, limit etc.
* Developers who are working with scripting languages and SQL, leverages Pig Latin. This gives developers ease of programming with Apache Pig. Pig Latin provides various built-in operators like join, sort, filter, etc to read, write, and process large data sets. Thus it is evident, Pig has a rich set of operators.
* Programmers write scripts using Pig Latin to analyze data and these scripts are internally converted to Map and Reduce tasks by Pig MapReduce Engine. Before Pig, writing MapReduce tasks was the only way to process the data stored in HDFS.
* If a programmer wants to write custom functions which is unavailable in Pig, Pig allows them to write User Defined Functions (UDF) in any language of their choice like Java, Python, Ruby, Jython, JRuby etc and embed them in Pig script written in Pig Latin. This provides extensibility to Apache Pig.
* Pig can process any kind of data, i.e. structured, semi-structured or unstructured data, coming from various sources. Apache Pig handles all kinds of data.
* Approximately, 10 lines of pig code is equal to 200 lines of MapReduce code.
* It can handle inconsistent schema (in case of unstructured data).
* Apache Pig extracts the data, performs operations on that data and dumps the data in the required format in HDFS i.e. ETL (Extract Transform Load).
* Apache Pig automatically optimizes the tasks before execution, i.e. automatic optimization.
* It allows programmers and developers to concentrate upon the whole operation irrespective of creating mapper and reducer functions separately.

**Task(1b) Find top 5 job titles who are having highest average growth in applications.**

REGISTER '/home/hduser/ExternalJars/piggybank-0.13.0.jar'; **--Register external jar 'Piggy Bank.jar'**

data = LOAD '/user/hive/warehouse/final.db/h1b\_final’ USING PigStorage() as

(s\_no:int,case\_status:chararray,employer\_name:chararray,soc\_name:chararray, job\_title:chararray,full\_time\_position:chararray,prevailing\_wage:int,year:chararray, worksite:chararray, longitude:double, latitude:double); **--Load data**

noheader= filter data by $0>=1; **--Remove header**

cleansed= filter noheader by $7 matches '2011'; **--filtering dataset by year**

a= group cleansed by $4;  **--grouping by job**

year\_2011 = foreach a generate group,COUNT($1); **--generate year,job,count**

--dump year\_2011;

cleansed= filter noheader by $7 matches '2012';

a= group cleansed by $4;

year\_2012= foreach a generate group,COUNT($1);

--dump year\_2012;

cleansed= filter noheader by $7 matches '2013';

a= group cleansed by $4;

year\_2013= foreach a generate group,COUNT($1);

--dump year\_2013;

cleansed= filter noheader by $7 matches '2014';

a= group cleansed by $4;

year\_2014= foreach a generate group,COUNT($1);

--dump year\_2014;

cleansed= filter noheader by $7 matches '2015';

a= group cleansed by $4;

year\_2015= foreach a generate group,COUNT($1);

--dump year\_2015;

cleansed= filter noheader by $7 matches '2016';

a= group cleansed by $4;

year\_2016= foreach a generate group,COUNT($1);

--dump year\_2016;

joined= join year\_2011 by $0, year\_2012 by $0, year\_2013 by $0, year\_2014 by $0, year\_2015 by $0, year\_2016 by $0;

--dump joined;

year\_wise\_applications= foreach joined generate $0,$1,$3,$5,$7,$9,$11;

--dump year\_wise\_applications;

**--generate progressive growth**

progressive\_growth= foreach year\_wise\_applications generate $0,

(float)($6-$5)\*100/$5,(float)($5-$4)\*100/$4,

(float)($4-$3)\*100/$3,(float)($3-$2)\*100/$2,

(float)($2-$1)\*100/$1;

--dump progressive\_growth;

**--average progressive growth**

avg\_progressive\_growth=foreach progressive\_growth generate $0, ($1+$2+$3+$4+$5)/5;

--dump avg\_progressive\_growth;

**---ordered progressive growth**

ordered\_avg\_growth= order avg\_progressive\_growth by $1 desc;

--dump ordered\_avg\_growth;

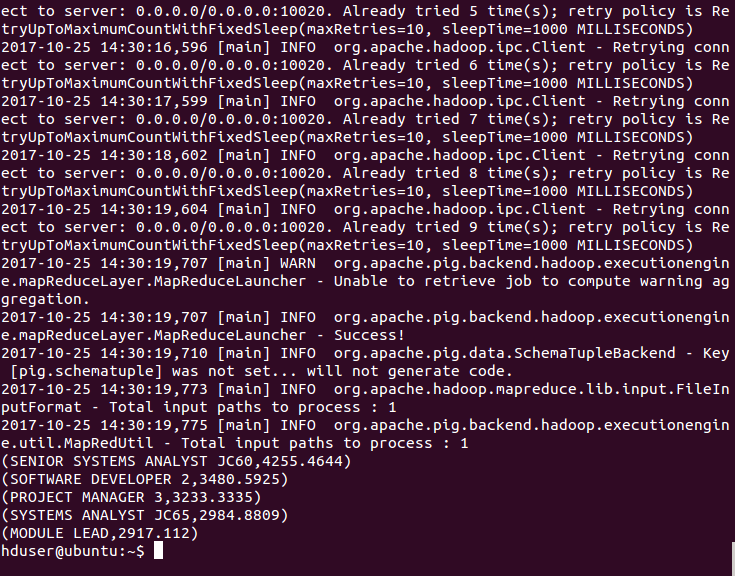
**--display top 5**

final = limit ordered\_avg\_growth 5;

dump final;

The above commands are written and saved in the file - **question1b.pig** and then executed.

**Output:**

****

**Thus, top 5 job titles who are having highest average growth in applications are:**

1. **Senior Systems Analyst**
2. **Software Developer**
3. **Project Manager**
4. **System Analyst**
5. **Module Lead**

**Task(6) Find the percentage and the count of each case status on total applications for each year. Create a graph depicting the pattern of All the cases over the period of time.**

REGISTER '/home/hduser/ExternalJars/piggybank-0.13.0.jar';

DEFINE CSVExcelStorage org.apache.pig.piggybank.storage.CSVExcelStorage;

data = LOAD '/home/hduser/DataSets/h1b.csv' USING CSVExcelStorage() as

(s\_no:int, case\_status:chararray, employer\_name:chararray, soc\_name:chararray,

Job\_title:chararray, full\_time\_position:chararray, prevailing\_wage:int, year:chararray,

Worksite:chararray, longitude:double, latitude:double);

noheader= filter data by $0>=1;

cleansed= filter noheader by $1 is not null and $1!='NA';

temp= group cleansed by $7;

count= foreach temp generate group,COUNT(cleansed.$1);

--dump count;

noheader= filter data by $0>=1;

cleansed= filter noheader by $7 is not null and $7!='NA';

temp= group cleansed by ($7,$1);

yearsofcount= foreach temp generate group,group.$0,COUNT($1);

--dump yearsofcount;

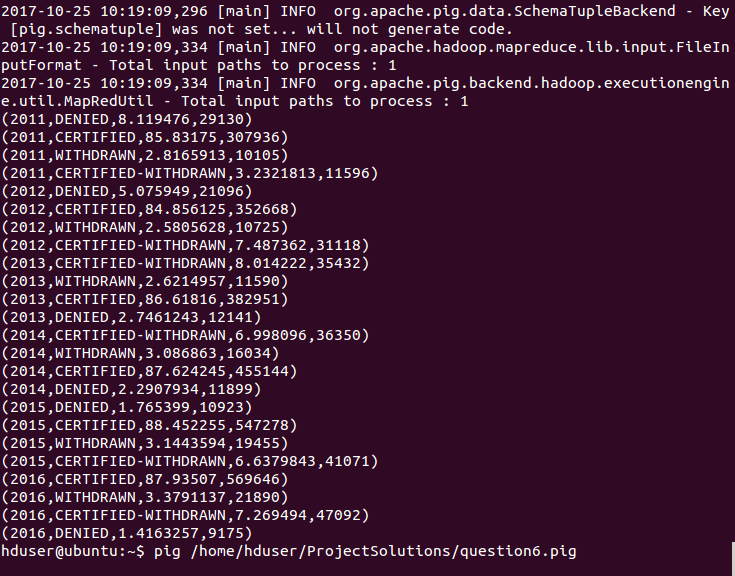
joined= join yearsofcount by $1,count by $0;

final= foreach joined generate FLATTEN($0),(float)($2\*100)/$4,$2;

dump final;

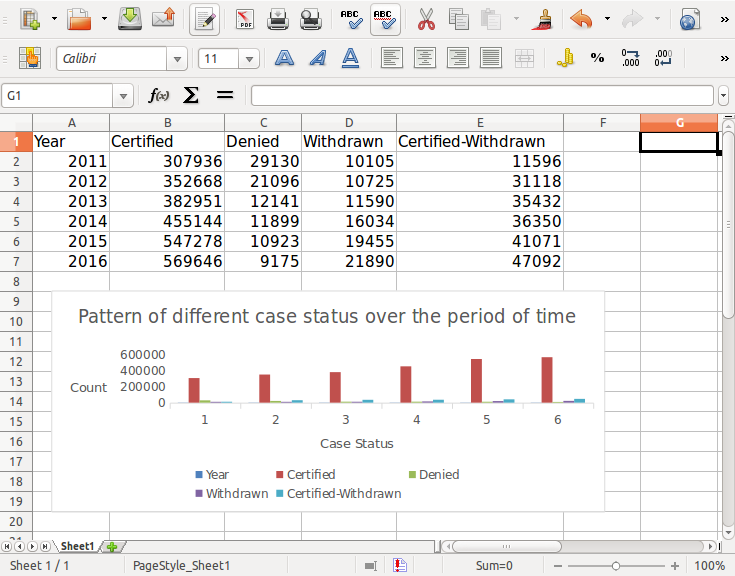
THE ABOVE COMMANDS ARE WRITTEN AND SAVED IN THE FILE - **question6.pig** AND THEN EXECUTED.

**Output:**

****

**Thus, the percentage and the count of each case status on total applications for each year is as above.**

**Graph depicting pattern of different case\_status over the period of time**



**Task(9) Which are employers who have the highest success rate in petitions more than 70% in petitions and total petitions filed more than 1000?**

REGISTER '/home/hduser/ExternalJars/piggybank-0.13.0.jar';

data = LOAD '/user/hive/warehouse/final.db/h1b\_final' USING PigStorage() as

(s\_no:int, case\_status:chararray, employer\_name:chararray, soc\_name:chararray, job\_title:chararray, full\_time\_position:chararray, prevailing\_wage:int, year:chararray,

Worksite:chararray, longitude:double,latitude:double);

noheader= filter data by $0>=1;

cleansed= filter noheader by $1 is not null and $1!='NA';

temp= group cleansed by $2;

total= foreach temp generate group,COUNT(cleansed.$1);

certified= filter noheader by $1 == 'CERTIFIED';

temp1= group certified by $2;

totalcertified= foreach temp1 generate group,COUNT(certified.$1);

**--Group by employername and count the case status = 'CERTIFIED'**

certified= filter noheader by $1 == 'CERTIFIED-WITHDRAWN';

temp2= group certified by $2;

totalcertifiedwithdrawn= foreach temp2 generate group,COUNT(certified.$1);

**--Group by employername and count the case status = 'CERTIFIED-WITHDRAWN'**

joined= join totalcertified by $0,totalcertifiedwithdrawn by $0,total by $0;

--dump joined;

joined= foreach joined generate $0,$1,$3,$5;

joined2= foreach joined generate $0,(float)($1+$2)\*100/($3),$3;

joined3= filter joined2 by $1>70 and $2>1000;

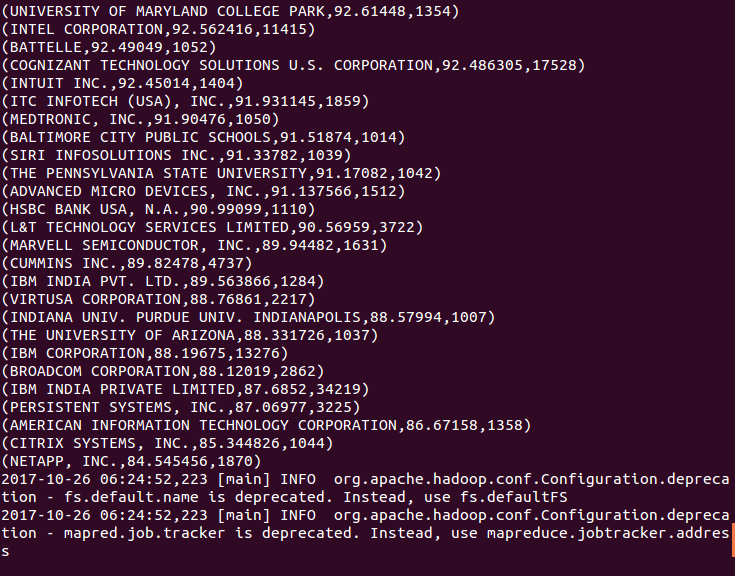
**--Filter by success-rate greater than 70% and petition count above 1000**

final= order joined3 by $1 DESC;

dump final;

THE ABOVE COMMANDS ARE WRITTEN AND SAVED IN THE FILE - **question9.pig** AND THEN EXECUTED.

**Output:**

****

**Thus,** **employers who have the highest success rate in petitions more than 70% in petitions and total petitions filed more than 1000 are as above.**

**Task(10) Which are the top 10 job positions that have success rate more than 70% in petitions and total petitions filed more than 1000?**

REGISTER '/home/hduser/ExternalJars/piggybank-0.13.0.jar';

data = LOAD '/user/hive/warehouse/final.db/h1b\_final' USING PigStorage() as

(s\_no:int, case\_status:chararray, employer\_name:chararray, soc\_name:chararray,

Job\_title:chararray, full\_time\_position:chararray, prevailing\_wage:int, year:chararray, worksite:chararray, longitude:double, latitude:double);

noheader= filter data by $0>=1;

**--Count Total Applications**

cleansed= filter noheader by $1 is not null and $1!='NA';

a= group cleansed by $4;

total= foreach a generate group,COUNT(cleansed.$1);

**--Count Total Applications who are 'CERTIFIED'**

certified= filter noheader by $1 == 'CERTIFIED';

b= group certified by $4;

totalcertified= foreach b generate group,COUNT(certified.$1);

**--Count Total Applications who are 'CERTIFIED-WITHDRAWN'**

certified= filter noheader by $1 == 'CERTIFIED-WITHDRAWN';

c= group certified by $4;

totalcertifiedwithdrawn= foreach c generate group,COUNT(certified.$1);

**--SUCCESS\_RATE=(CERTIFIED+CERTIFIED-WITHDRAWN)/TOTAL X 100**

joined= join totalcertified by $0,totalcertifiedwithdrawn by $0,total by $0;

joined1= foreach joined generate $0,$1,$3,$5;

joined2= foreach joined1 generate $0,(float)($1+$2)\*100/($3),$3;

joined\_filter= filter joined2 by $1>70 and $2>1000;

**--Filter by success-rate greater than 70% and petition count above 1000**

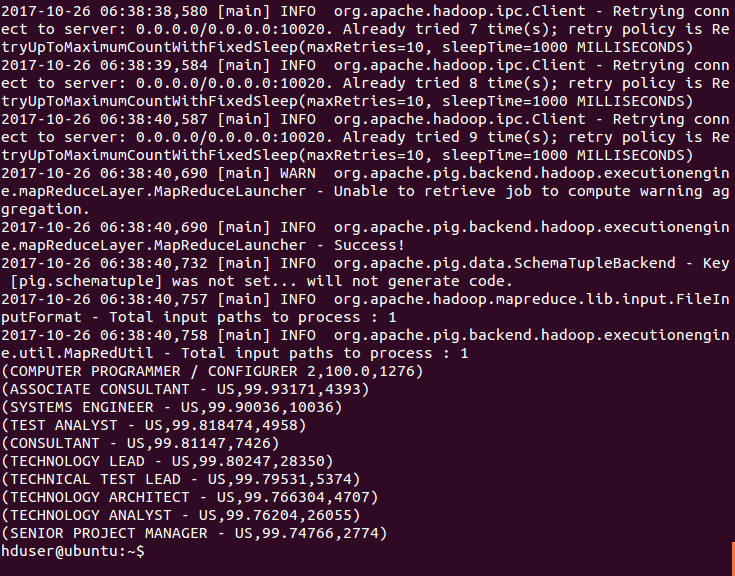
joined\_order= order joined\_filter by $1 DESC;

final = LIMIT joined\_order 10;

dump final;

THE ABOVE COMMANDS ARE WRITTEN AND SAVED IN THE FILE - **question10..pig** AND THEN EXECUTED.

**Output:**

****

**Thus, the top 10 job positions that have success rate more than 70% in petitions and total petitions filed more than 1000 are as above.**

**SQOOP**

Sqoop is a tool designed to transfer data between Hadoop and relational databases or mainframes. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle or a mainframe into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS.

Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance.

Sqoop provides a pluggable mechanism for optimal connectivity to external systems. The Sqoop extension API provides a convenient framework for building new connectors which can be dropped into Sqoop installations to provide connectivity to various systems. Sqoop itself comes bundled with various connectors that can be used for popular database and data warehousing systems.

**Task(11) Export result for question no 10 to MySql database.**

*The output of question 10 has been stored as question10.txt and uploaded on the HDFS in the directory /pig/question10*

**#Creating a database in mysql and a table in it**

mysql -u root -p'1234' -e 'create database if not exists final;

use final;

create table question11(job\_title varchar(100),success\_rate float,petitions int);';

**#Export command**

sqoop export --connect jdbc:mysql://localhost/final --username root --password '1234' --table question11 --update-mode allowinsert --export-dir /pig/question10/question10.txt --input-fields-terminated-by ',' ;

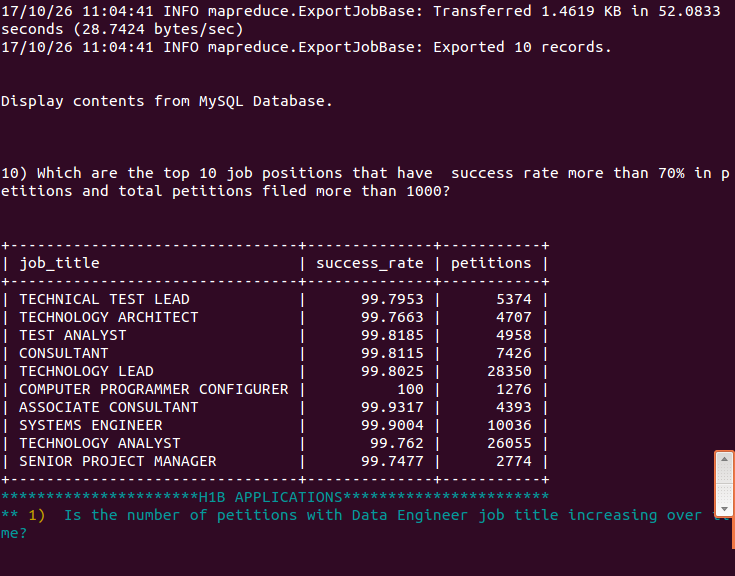
echo -e '\n\nDisplay contents from MySQL Database.\n\n'

echo -e '\n10) Which are the top 10 job positions that have success rate more than 70% in petitions and total petitions filed more than 1000?\n\n'

mysql -u root -p'1234' -e 'select \* from final.question11';

THE ABOVE COMMANDS ARE WRITTEN AND STORED IN THE FILE - **question11.sh** AND EXECUTED.

**Output:**

****

**REFERENCES**

1. https://pig.apache.org/docs/r0.9.1/udf.html

2.https://www.google.co.in/amp/s/www.edureka.co/blog/hadoop-tutorial/amp/

3. https://hortonworks.com/tutorial/hadoop-tutorial-getting-started-with-hdp/

4.https://sqoop.apache.org/docs/1.4.2/SqoopUserGuide.html

5.http://www.hadoopadmin.co.in/hive/how-to-remove-header-from-csv-during-loading-to-hive/