NIIT DILSUKHNAGAR CENTER

DT BIG DATA
with
HADOOP

Prepared By:
GuruPrasad Sunkara
Under the Guidence of Mr.Srikanth Sir

CONTEXT

Introduction to HADOOP

Today, we're surrounded by data. People upload videos, take pictures on their cell phones, text friends, update their Facebook status, leave comments around the web, click on ads, and so forth. Machines, too, are generating and keeping more and more data. The exponential growth of data first presented challenges to cutting-edge businesses such as Google, Yahoo, Amazon, and Microsoft. They needed to go through terabytes and petabytes of data to figure out which websites were popular, what books were in demand, and what kinds of ads appealed to people. Existing tools were becoming inadequate to process such large data sets. Google was the first to publicize *Map Reduce* a system they had used to scale their data processing needs. This system aroused a lot of interest because many other businesses were facing similar scaling challenges, and it wasn't feasible for everyone to reinvent their own proprietary tool. Doug Cutting saw an opportunity and led the charge to develop an open source version of this Map Reduce system called Hadoop . Soon after, Yahoo and others rallied around to support this effort. Today, Hadoop is a core part of the computing infrastructure for many web companies, such as Yahoo , Facebook , LinkedIn , and Twitter. Many more traditional businesses, such as media and telecom, are beginning to adopt this system too.

Hadoop is an open source framework for writing and running distributed applications that process large amounts of data. Distributed computing is a wide and varied field, but the key distinctions of Hadoop are that it is

■ *Accessible*—Hadoop runs on large clusters of commodity machines or on cloud computing services such as Amazon's Elastic Compute Cloud (EC2).

■ *Robust*—Because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.

■ *Scalable*—Hadoop scales linearly to handle larger data by adding more nodes to the cluster.

■ *Simple*—Hadoop allows users to quickly write efficient parallel code.

Hadoop's accessibility and simplicity give it an edge over writing and running large distributed programs. Even college students can quickly and cheaply create their own Hadoop cluster. On the other hand, its robustness and scalability make it suitable for even the most demanding jobs at Yahoo and Facebook. These features make Hadoop popular in both academia and industry.

BIGDATA

Big Data:-

Big Data is a term that describes the large volume of data – both structured and unstructured .The early 2000s when industry analyst Doug Laney articulated the now-mainstream definition of big data as the four 'V's but we consider three 'V's .

1. Volume
2. Velocity
3. Variety
4. Variability

Volume:-

Organizations collect data from a variety of sources, including business transactions, social media and information from sensor or machine-to-machine data. In the past, storing it would've been a problem – but new technologies (such as Hadoop) have eased the burden.

Velocity:-

Data streams in at an unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time.

Variety:-

Data comes in all types of formats – from structured, numeric data in traditional databases to unstructured text documents, email, video, audio, stock ticker data and financial transactions.

Variability:-

In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Is something trending in social media? Daily, seasonal and event-triggered peak data loads can be challenging to manage. Even more so with unstructured data.

What comes under Big Data?

- Social Media
- Government Sector
- Private Sector

  And many more

Big –Data Challenges:-

- It will store huge amount of data (large :- it may be TB(or)PB(or)EB and more
- It has high computation efficiency
- In this there is no any data loss because of replication
- It is cost effective because we are using commodity hardware.

Difference between Hadoop and RDBMS:-

HADOOP	RDBMS
Hadoop will work extremely with huge amount of data like Peta Bytes (or)Tera Bytes	It work extremely in Giga Bytes
It work in dynamic schema and support files many different formats	In this schema is very strict and not so flexible not handle multiple formats
It scale vertically and It's cost effective because it work on commodity computers	Scaling is not vertically and its coaster

HADOOP

The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how the name came about:
The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term.

Subprojects and "contrib" modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the jobtracker keeps track of MapReduce jobs.

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS. NDFS and the MapReduce implementation in Nutch were applicable beyond the realm of search, and 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 (see sidebar). This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster.†

Hadoop:-

Hadoop is framework for distributed processing of large datasets across cluster of commodity computers.

Benefit's of HDFS:-

- Support distributed processing Blocks(not as whole size).
- Handle failures by replicating the blocks .
- Able to support future expansion.
- It is cost effective.

HADOOP DISTRIBUTED FILE SYSTEM

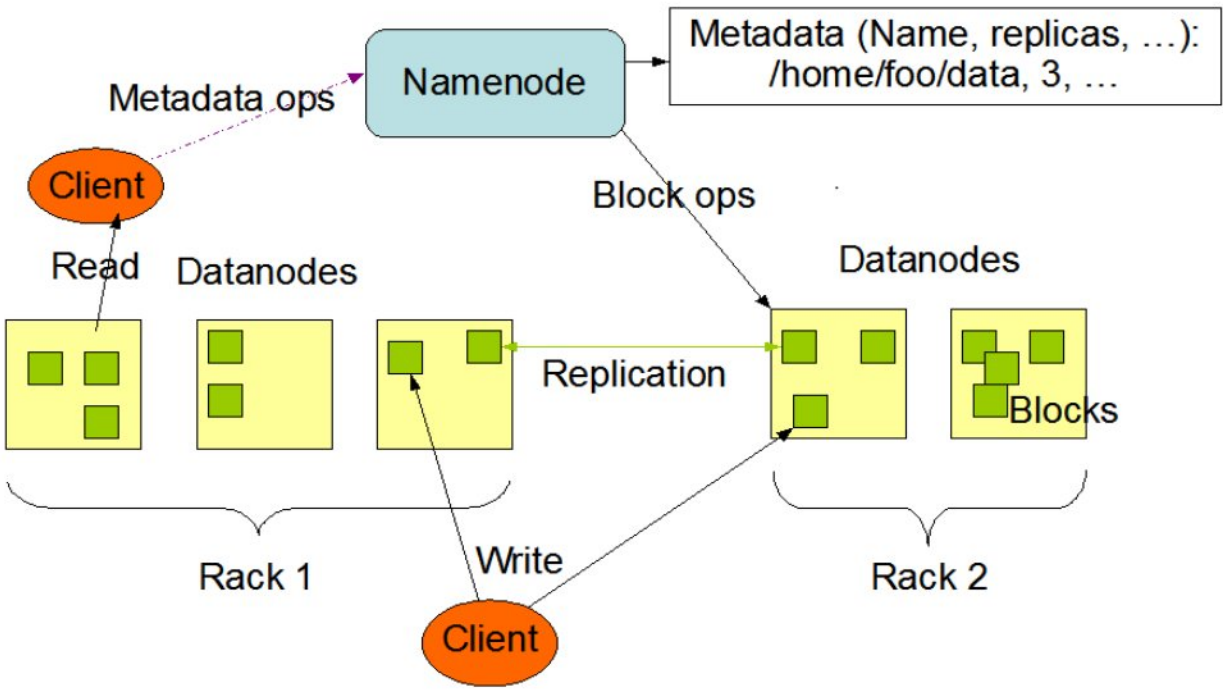
Functions of File System :-

- File Systems Control how data is stored and retrieved
- File System has Metadata about files and systems
- Permission and security.
- Manage store space efficiently.

HDFS(Hadoop Distributed File System) :-

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is now an Apache Hadoop subproject.

HDFS Architecture



Data Node :-

- A node where the blocks are physically stored as named as DataNode.
- Each Data Node knows the list of blocks. It is responsible for read and write requests from the file systems client.
- DataNode is not take of others DataNodes and other blocks.
- Data Node is also called as Slave Nodes.

Name Node :-

- Name Node keeps track of all the files and datasets in hdfs.
- Name Node has list of all the blocks and the locations of block.
- Name Node is also known as Master Node.
- If Name Node is being down in cluster there is no way we can look of files in our Hadoop cluster. Name Node also has the "METADATA".
- It will store metadata of files and folders in disk.
- It will store block locations in memory.
- Name Node is a powerful Node.
- If Name Node is failure we have a recovery storage strategy like "secondary Name Node".

METADATA:-

Meta data has all the bellow data

- It has all the owner's details.
- It has replication factor details.
- Created and Modified date and time etc.

Secondary Name Node:-

- Secondary Name Node has stored all the Name Node data.
- If Name Node failure that name will recovery by using this Secondary Name Node.
- Secondary Name Node is like replication of Name Node.

CHUNK:-

Chunk is nothing blocks

RACK:-

Collection of Nodes(30 r 40) connected in a single network is called Rack.

CLUSTER:-

Group of racks connected to a single Network.

DATA CENTERAL:-

It is physical location where is clusters are housed.

INTRODUCTION TO MAPREDUCE

Map Reduce:-

It is a programming model for disturbed computing .It is not a programming language. Which we can use to process huge datasets in a distributed fashion. It can be implemented any programming language (java, Pyton, Scala, C++ and more). Map reduce has three phases these are

1) MAP PHASE
2) SHUFFLE PHASE
3) REDUCE PHASE

Map Phase:-

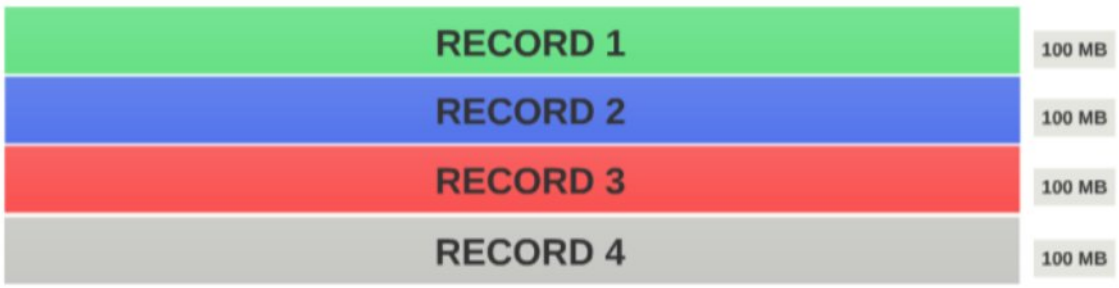
Map Phase input is records and output will be the Key-Value Pair. Mapper Phase consists there components

- Input Split
- Mappers
- Key-Value Pair

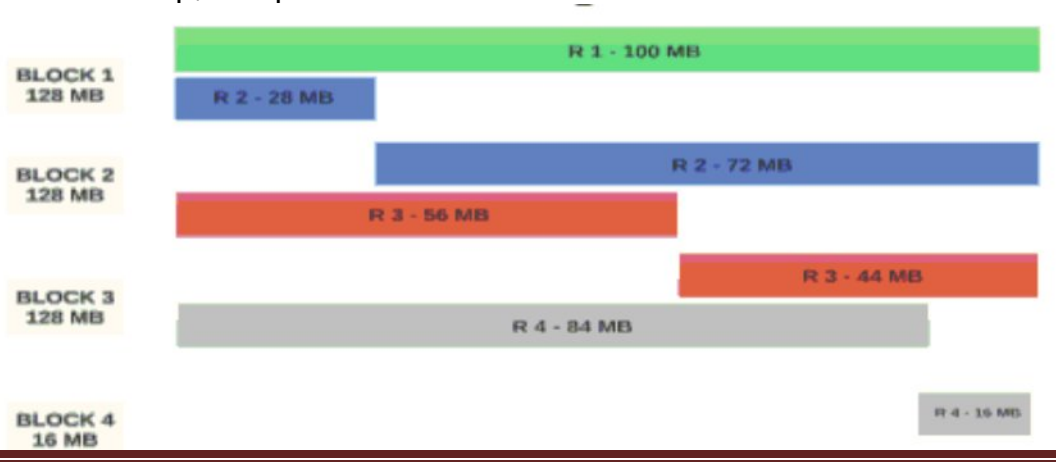
Input Split:-

- Chunks are called as Input Splits.
- Input split is not same as a block.

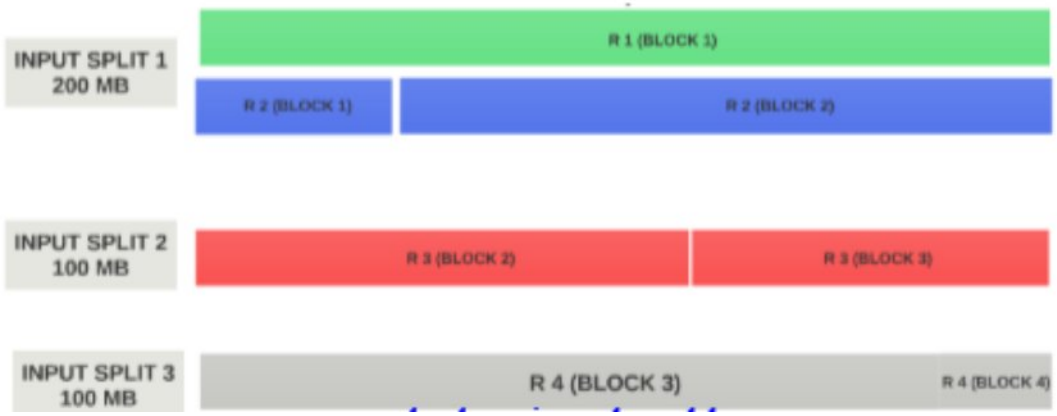
Let us consider we have four records each of size 100 Mb



- Block size is 64 Mb by default in Apache Hadoop and 128 Mb by default in Cloudera Hadoop, except last block.



- Block can end before the record end.
- Then the inputSplit will be



MAPPER:-

- It will map all the value to the key.
- No.Of Mappers = No.Of InputSplits

Key-Value Pair:-

- The output of the Mapper-Phase is Key-Value Pair.
- In this every key has one value

SHUFFLE PHASE:-

Input to the shuffle phase is Key-Value pair and Out put to the Shuffle is Key-Values. It has two components in this phase

- Sort
- Shuffle

SORT:-

- Input to the sort is key-Value from
- Sort will sort the data it means it will keep all the values of a key at a place.

Shuffle:-

- Output of the sorting is input to the Shuffle and it will perform task like it will give key and many values as output.
- Shuffle phase is also known as Merge

REDUCER PHASE:-

- Reduce will do all the reducer performance as per the logic.

DATA FLOW:-

INPUT SPLIT → MAPPER → MAPPER OUTPUT → SORT → COPY → SHUFFLE(Merge) → REDUCER →FINAL OUTPUT

MAPREDUCE PROGRAM:-

In Map reduce program we have to write 3 classes in the program

- Mapper Program
- Reducer Program
- Driver Program

Mapper Program:-

- In mapper program we are going to Mapper Phase in this program

Reducer Program:-

- All the Shuffle and Reducer Phase programs are written under this program.

Driver Program:-

- We will write main program under this program.
- ALL the mapper and reducer programs relations are written under this program.

Use Case:-
DIVORCED (MODULE):-
1) Total number of people who are educated and they got divorced

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class divorced1
{
	public static class Mymapper1 extends
			Mapper<LongWritable, Text, Text, Text>
	 {
		protected void map(LongWritable key, Text value, Context context)
				throws IOException, InterruptedException
		{
			String[] str = value.toString().split(",");
			if((!str[1].equals("Less than 1st
Grade")||!str[1].equals("Children"))&&str[2].equals("Divorced"))
			{
			context.write(new Text("all"),new Text("1"));
			}
		}

	}

	public static class Myreducer1 extends
			Reducer<Text, Text, Text, DoubleWritable>
	{
		protected void reduce(Text key, Iterable<Text> values, Context context
			throws IOException, InterruptedException
		 {
			double sum=0;
```

```
                    for (Text value : values)
                    {
                            long lol=Long.parseLong(value.toString());
                            sum+=lol;
                    }

                    context.write(key,new DoubleWritable(sum));

            }
    }

    public static void main(String[] args) throws IOException,
                    ClassNotFoundException, InterruptedException
    {
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf);
            job.setJarByClass(divorced1.class);
            job.setMapperClass(Mymapper1.class);
            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);
            job.setReducerClass(Myreducer1.class);
            FileInputFormat.addInputPath(job, new Path("/census"));
            FileOutputFormat.setOutputPath(job, new Path("/outputforedu"));
            job.waitForCompletion(true);

    }

}
```

RESULT:-

EDUCATED AND MARRIED = 338221

2) Finding Of Divorced and Non Dvorced percentage from all the records

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class divorced2{
	public static class Mymapper1 extends
			Mapper<LongWritable, Text, Text, Text> {
		protected void map(LongWritable key, Text value, Context context)
				throws IOException, InterruptedException {
			String[] str = value.toString().split("\t");
		context.write(new Text("all"),new Text(str[1]));
		}
	}

	public static class Myreducer1 extends Reducer<Text, Text, Text,
DoubleWritable>
{
	protected void reduce(Text key, Iterable<Text> values, Context context)
				throws IOException, InterruptedException
{
	double num1=0;double num2=0;int count=0;
		for (Text value : values)
	{
		if(count==0)
		{
			num1=Double.parseDouble(value.toString());
			count++;
		}
		else
		{
				num2=Double.parseDouble(value.toString());
		}
	}
```

```
        context.write(new Text("nondivorced"), new
        DoubleWritable((num1*100)/(num1+num2)));
        context.write(new Text("divorced"),new
        DoubleWritable((num2*100)/(num1+num2)));
                }
        }

public static void main(String[] args) throws IOException,
                        ClassNotFoundException, InterruptedException
 {
Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        job.setJarByClass(divorced2.class);
        job.setMapperClass(Mymapper1.class);
        job.setReducerClass(Myreducer1.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path("/outputforguru"));
        FileOutputFormat.setOu tputPath(job, new Path(args[0]));
        job.waitForCompletion(true);

        }

}
```

RESULT:-

nondivorced – 93.6
divorced - 6.36

3) Find the total educated and uneducated percentage from the given record.

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class education3
{
	public static class Mymapper1 extends
			Mapper<LongWritable, Text, Text, Text>
	{
		protected void map(LongWritable key, Text value, Context context)
				throws IOException, InterruptedException
		{
			String[] str = value.toString().split("\t");
		context.write(new Text("all"),new Text(str[1]));

		}
	}

	public static class Myreducer1 extends
			Reducer<Text, Text, Text, DoubleWritable>
	{
	    protected void reduce(Text key, Iterable<Text> values, Context context)
				throws IOException, InterruptedException
		{
			double num1=0;double num2=0;int count=0;
			for (Text value : values)
			{
				if(count==0)
				{
				num1=Double.parseDouble(value.toString());
				count++;
				}
```

```
                                else
                                {
                                        num2=Double.parseDouble(value.toString());
                                }
                        }
context.write(new Text("educated"),new
DoubleWritable((num1*100)/(num1+num2)));
context.write(new Text("non-educated"),new
DoubleWritable((num2*100)/(num1+num2)));
                }
        }

        public static void main(String[] args) throws IOException,
                        ClassNotFoundException, InterruptedException
{
                Configuration conf = new Configuration();
                Job job = Job.getInstance(conf);
                job.setJarByClass(educatio3.class);
                job.setMapperClass(Mymapper1.class);
                job.setReducerClass(Myreducer1.class);
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(Text.class);
                FileInputFormat.addInputPath(job, new Path("/outputforedu"));
                FileOutputFormat.setOutputPath(job, new Path(args[0]));
                job.waitForCompletion(true);

        }

}
```

RESULT:-

Educated = 75.8
Non – educated =24.12

INTRODUCTION TO PIG

PIG:-

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

- Pig was developed by yahoo to make map reduce accessible to any one .
- By using pig latin language we can work on pig

USE CASES

TaxFiler (Module):-

1) Find out the top ten taxpayers and display how many weeks they are working and find average income of them

```
--loading data into pig
loadingdata = LOAD '/home/hduser/census' USING PigStorage(',') as (age : int,education : chararray,martialstatus : chararray,gender : chararray,taxfilerstatus : chararray,income : double,parents : chararray,countryofbirth : chararray,citizenship : chararray,weekworked : int);
--filtering taxpayers data
findingtaxfiler = FILTER loadingdata BY taxfilerstatus!='Nonfiler';
--ordering the data by income
toptentaxfilers = ORDER findingtaxfiler BY income;
--taking top 10 records
limitingtotopten = LIMIT toptentaxfilers 10;
--grouping all to find their average
foraverage = GROUP limitingtotopten all;
--finding average
findingaverage = FOREACH foraverage GENERATE AVG(limitingtotopten.income) as averagetop;
--showing output
dump findingaverage;
```

RESULT:-

AVERAGE INCOME OF TOP 10 TAXPAYERS = 11902.97

2) Taxfiler with same country and finding their count

```
--loading data into pig
loadingdata = LOAD '/home/hduser/census' USING PigStorage(',') as (age : int,education : chararray,martialstatus : chararray,gender : chararray,taxfilerstatus : chararray,income : double,parents : chararray,countryofbirth : chararray,citizenship : chararray,weekworked : int);
--filtering taxpayers data
findingtaxfiler = FILTER loadingdata BY taxfilerstatus!='Nonfiler';
--grouping all to find the count
forfinding = GROUP findingtaxfiler BY citizenship;
--counting taxpayers by their ciizenship
finalcount = FOREACH forfinding GENERATE group,COUNT(findingtaxfiler.citizenship),findingtaxfiler.citizenship as finnalcounting;
--showing output
dump finalcount;
```

RESULT:-

Foreign born- Not a citizen of U S 26930

Foreign born- U S citizen by naturalization 14405

Native- Born abroad of American Parent(s) 3642

Native- Born in Puerto Rico or U S Outlying 2466

Native- Born in the United States 325237

3) Tax filer according to gender

```
--loading data in to pig
loadingdata = LOAD '/home/hduser/census' USING PigStorage(',') as (age : int,education : chararray,martialstatus : chararray,gender : chararray,taxfilerstatus : chararray,income : double,parents : chararray,countryofbirth : chararray,citizenship : chararray,weekworked : int);
--filtering taxpayers details
findingtaxfilergender = FILTER loadingdata BY taxfilerstatus!='Nonfiler';
--filter male taxpayers data
filteringformale = FILTER findingtaxfilergender BY gender=='Male';
--grouping male taxpayers
malegrouping = GROUP filteringformale BY gender;
--counting total male details
sumforpermale = FOREACH malegrouping GENERATE COUNT(filteringformale.gender) as malecount,1 as keyforjoin;
--filter female taxpayers
filteringforfemale = FILTER findingtaxfilergender BY gender=='Female';
--grouping female taxpayers
femalegrouping = GROUP filteringforfemale BY gender;
--counting female taxpayers
sumforperfemale = FOREACH femalegrouping GENERATE COUNT(filteringforfemale.gender) as femalecount,1 as keyforjoin;
--joining the needed data to a single bag
joininggender = JOIN sumforperfemale BY keyforjoin,sumforpermale BY keyforjoin;
--calculating percentage
finalpercentage = FOREACH joininggender GENERATE (malecount*100)/(malecount+femalecount) as malepercentage,(femalecount*100)/(malecount+femalecount) as femalepercentage;
--showing result
dump finalpercentage;
```

RESULT:-

malepercentage	femalepercentage
48	51

INTRODUCTION TO HIVE

HIVE:-

Hive is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop compatible file systems. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

USE CASE:-

Education Module:-

1) Find out the how many are literate and how many are Illiterate from the list?

```
//loading literate details
CREATE VIEW literate  as SELECT education,count(education) as literates FROM census WHERE education != 'Less than 1st grade' AND education != 'Children' GROUP BY education;
//loading illiterte details
CREATE VIEW illiterates  as SELECT  education,count(education) as illiterates FROM census WHERE education = 'Less than 1st grade'  OR education = 'Children' GROUP BY education;
//calcutating total literates and total illiterates
CREATE VIEW detailsofeducation as SELECT * FROM (SELECT SUM(literates) as totalliterates FROM literate) a JOIN (SELECT SUM(illiterates) as totalilliterates FROM illiterates) b;
//calculating total people and keeping all in a single view(sub table)
CREATE VIEW totaldetails as SELECT sum(totalliterates+totalilliterates) as totalpeople,totalliterates,totalilliterates FROM detailsofeducation GROUP by totalliterates , totalilliterates ;
//finding the percentage
CREATE VIEW percentage as SELECT totalpeople,(totalliterates*100)/totalpeople as literatepercentage,(totalilliterates*100)/totalpeople as illiteratepercentage FROM totaldetails;
```

RESULT:-

totalpeople	literatepercentage	illiteratepercentage
596523	75.8755320415139	24.124467958486093

2) In the literates list how many are married and how many are unmarried?

```
//loading literates married data
CREATE VIEW educated as SELECT education,maritalstatus,COUNT(maritalstatus) as c FROM census WHERE (education != 'Less than 1st grade' AND education != 'Children') AND Maritalstatus != 'Never married' GROUP BY maritalstatus,education;
//loading literates nevermarried data
CREATE VIEW eduunmarried as SELECT education,maritalstatus,COUNT(maritalstatus) as c FROM census WHERE (education != 'Less than 1st grade' AND education != 'Children') AND Maritalstatus = 'Never married' GROUP BY maritalstatus,education;
//counting educated married and unmarried details
CREATE VIEW edumaritaldetails as SELECT * FROM (SELECT SUM(c) as educatedandmarried FROM edumarried) a JOIN (SELECT SUM(c) as educatedandunmarried FROM eduunmarried) b;
//calculating percentage
CREATE VIEW educateddetails SELECT SUM(educatedandmarried+educatedandunmarried) as totaleducated,educatedandmarried,educatedandunmarried FROM edumaritaldetails GROUP BY educatedandmarried,educatedandunmarried;
```

RESULT:-

totaleducated	educatedandmarried	educatedandunmarried
455027	338195	116832

3) how many of non citizen people are studing their ms in us and tell me indian people percentage in that?

```
//loading all ms studied people data except us people
CREATE VIEW foreigner as SELECT citizenship,education,countryofbirth,count(education) as studentcount FROM census WHERE (citizenship = 'Native- Born in Puerto Rico or U S Outlying' OR citizenship = 'Foreign born- Not a citizen of U S') AND countryofbirth !='United-States' AND education = 'Masters degree(MA MS MEng MEd MSW MBA)' GROUP BY citizenship,education,countryofbirth;
//calculatind indian and foreigner peoples
CREATE VIEW indiansdetails as SELECT total,exceptindians,Indians FROM ( SELECT SUM(studentcount) as total FROM foreigner) a JOIN ( SELECT SUM(studentcount) as exceptindians FROM foreigner WHERE countryofbirth != 'India' ) b JOIN ( SELECT SUM(studentcount) as Indians FROM foreigner WHERE countryofbirth = 'India') c ;
//calculating their percentages
CREATE VIEW indianpercentage as SELECT total,exceptindians,(exceptindians*100)/total as forignerpercentage,Indians,(Indians*100)/total as indianpercentage FROM indiansdetails;
```

RESULT:-

total	exceptindians	forignerpercentage	indians	indianpercentage
1150	1028	89.3913043478261	122	10.608695652173912

4) Find the male and female education percentage of us citizens from their population ?

```
//loading us educated data
CREATE VIEW b as SELECT gender,COUNT(education) as educators FROM census WHERE (education != 'Less than 1st grade' AND education != 'Children') AND countryofbirth ='United-States' GROUP BY gender;
//calculating female and male percentage
SELECT total,(female*100)/total as femalepercentage,(male*100)/total as malepercentage FROM (SELECT sum(educators) as total FROM b) a JOIN (SELECT educators as female FROM b WHERE gender = 'Female') b JOIN (SELECT educators as male FROM b WHERE gender ='Male') c;
```

RESULT:-

total	femalepercentage	malepercentage
393112	53.176702822605264	46.823297177394736