

# Assignment-11

## Task1

Explain the below concepts with an example in brief.

- **Nosql Databases:**

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean “not SQL” or “not only SQL.” As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

- **Types of Nosql Databases:**

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

**Key-value data stores:** Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.

**Document stores:** Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.

**Wide-column stores:** Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

**Graph stores:** A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

## ● **CAP Theorem:**

CAP describes that before choosing any Database (Including distributed database), Basing on your requirement we have to choose only two properties out of three.

### ➤ **Consistency**

Whenever you read a record (or data), consistency guaranties that it will give same data how many times you read. Simply we can say that each server returns the right response to each request, thus the system will be always consistent whenever you read or write data into that.

### ➤ **Availability**

Availability simply means that each request eventually receives a response (even if it's not the latest data or consistent across the system or just a message saying the system isn't working).

### ➤ **Partition Tolerance**

Partition Tolerance means that the cluster continues to function even if there is a "partition" (communications break) between two nodes (both nodes are up, but can't communicate).

One property should be scarified among three, so you have to choose combination of CA or CP or AP.

### ➤ **Consistency – Partition Tolerance:**

System will give you a consistent data and it will be distributed across the cluster. But it becomes unavailable when a node goes down.

➤ **Availability – Partition Tolerance:**

System will respond even if nodes are not communicating with each other (nodes are up and running but not communicating). But there is no guaranty that all nodes will have same data.

➤ **Consistency – Availability:**

System will give you a consistent data and you can write/read data to/from any node, but data partitioning will not be sync when develop a partition between nodes (RDBMS systems such as MySQL are of CA combination systems.)

## ● **HBase Architecture**

HBase architecture consists mainly of four components

HMaster

HRegionserver

HRegions

Zookeeper

### **HMaster:**

HMaster is the implementation of Master server in HBase architecture. It acts like monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

Plays a vital role in terms of performance and maintaining nodes in the cluster.

HMaster provides admin performance and distributes services to different region servers.

HMaster assigns regions to region servers.

HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.

When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.

Table ( createTable, removeTable, enable, disable)

ColumnFamily (add Column, modify Column)

Region (move, assign)

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn check the health status of region servers.

In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

### **HRegions Servers:**

When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

HMaster can get into contact with multiple HRegion servers and performs the following functions.

Hosting and managing regions

Splitting regions automatically

Handling read and writes requests

Communicating with the client directly

### **HRegions:**

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

## **ZooKeeper:**

In Hbase, Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes. If the client wants to communicate with regions, the servers client has to approach ZooKeeper first.

It is an open source project, and it provides so many important services.

Services provided by ZooKeeper

Maintains Configuration information

Provides distributed synchronization

Client Communication establishment with region servers

Provides ephemeral nodes for which represent different region servers

Master servers usability of ephemeral nodes for discovering available servers in the cluster

To track server failure and network partitions

Master and HBase slave nodes (region servers) registered themselves with ZooKeeper. The client needs access to ZK (zookeeper) quorum configuration to connect with master and region servers.

## **● HBase vs RDBMS**

<b>HBASE</b>	<b>RDBMS</b>
Schema-less in database.	Having fixed schema in database.
Column oriented database.	Row oriented data store.

Designed to store De-normalized data.

Designed to store Normalized data.

Wide and sparsely populated tables present in Hbase.

Contains thin tables in database.

Supports automatic partitioning.

Has no built in support for partitioning.

Well suited for OLAP systems.

Well suited for OLTP systems.

Read only relevant data from database.

To retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required.

Structured and semi structure data can be stored and processed using Hbase.

Structured data can be stored and processed using an RDBMS.

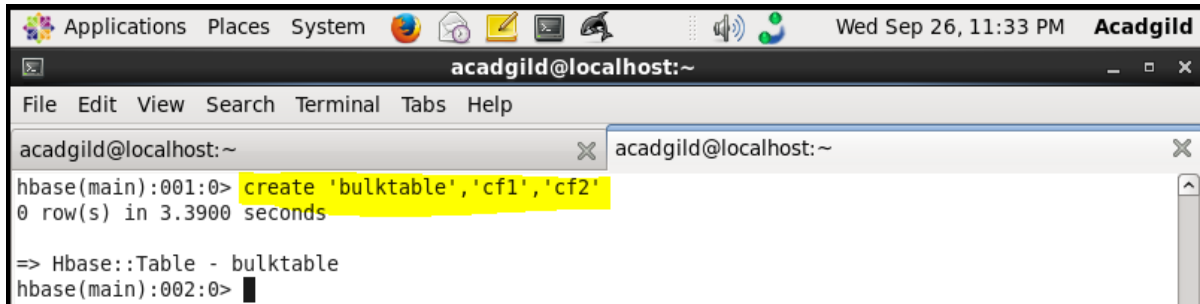
Enables aggregation over many rows and columns.

Aggregation is an expensive operation.

## Task-2:

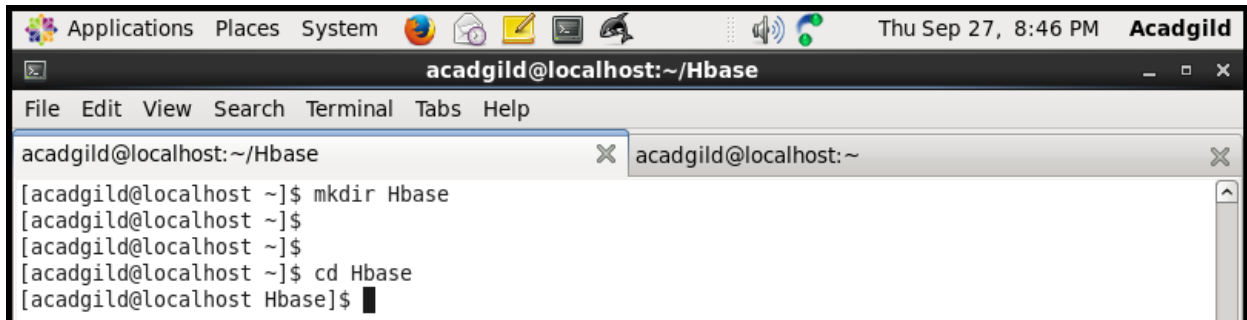
### Implementation of importing TSV data from HDFS into Hbase:

Created a table with name 'bulktable' having two column families.

A terminal window titled 'acadgild@localhost:~' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the command 'create 'bulktable','cf1','cf2'' being executed in the hbase(main) prompt. The output indicates '0 row(s) in 3.3900 seconds' and shows the table 'bulktable' has been created. The prompt returns to hbase(main):002:0>.

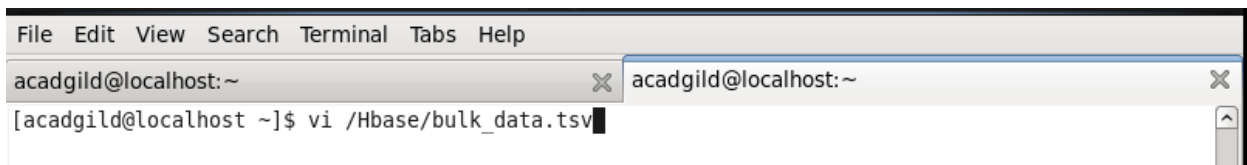
```
acadgild@localhost:~  
File Edit View Search Terminal Tabs Help  
acadgild@localhost:~  
hbase(main):001:0> create 'bulktable','cf1','cf2'  
0 row(s) in 3.3900 seconds  
=> Hbase::Table - bulktable  
hbase(main):002:0>
```

Created Hbase directory in local file system.

A terminal window titled 'acadgild@localhost:~/Hbase' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the commands 'mkdir Hbase', 'cd Hbase', and the current directory is now 'Hbase'.

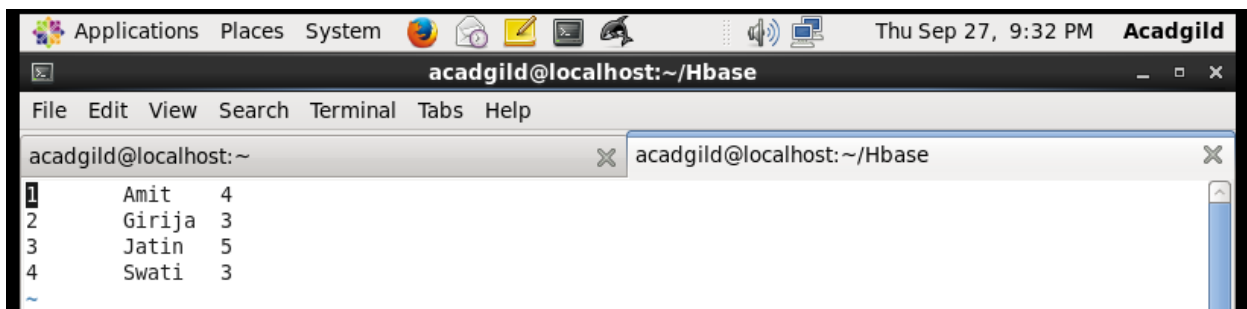
```
acadgild@localhost:~/Hbase  
File Edit View Search Terminal Tabs Help  
acadgild@localhost:~/Hbase  
[acadgild@localhost ~]$ mkdir Hbase  
[acadgild@localhost ~]$  
[acadgild@localhost ~]$  
[acadgild@localhost ~]$ cd Hbase  
[acadgild@localhost Hbase]$
```

Created a TSV file in Hbase directory as shown below.

A terminal window titled 'acadgild@localhost:~' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the command 'vi /Hbase/bulk\_data.tsv' being executed.

```
acadgild@localhost:~  
File Edit View Search Terminal Tabs Help  
acadgild@localhost:~  
[acadgild@localhost ~]$ vi /Hbase/bulk_data.tsv
```

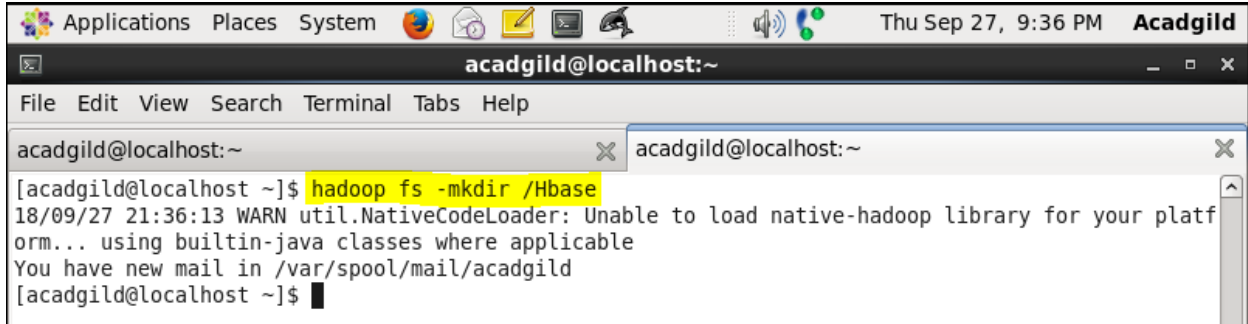
Added the below data into the file.

A terminal window titled 'acadgild@localhost:~/Hbase' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows a text editor (vi) with the following data entered into the file 'bulk\_data.tsv':

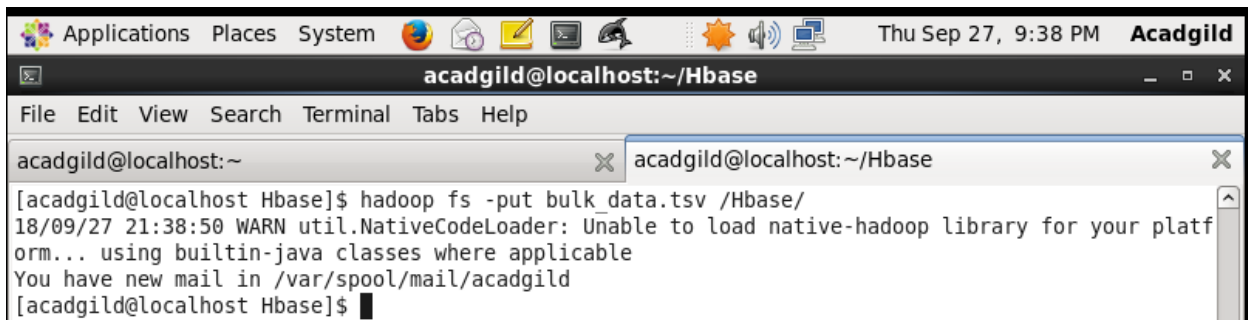
1	Amit	4
2	Girija	3
3	Jatin	5
4	Swati	3

```
acadgild@localhost:~/Hbase  
File Edit View Search Terminal Tabs Help  
acadgild@localhost:~  
1 Amit 4  
2 Girija 3  
3 Jatin 5  
4 Swati 3  
~
```

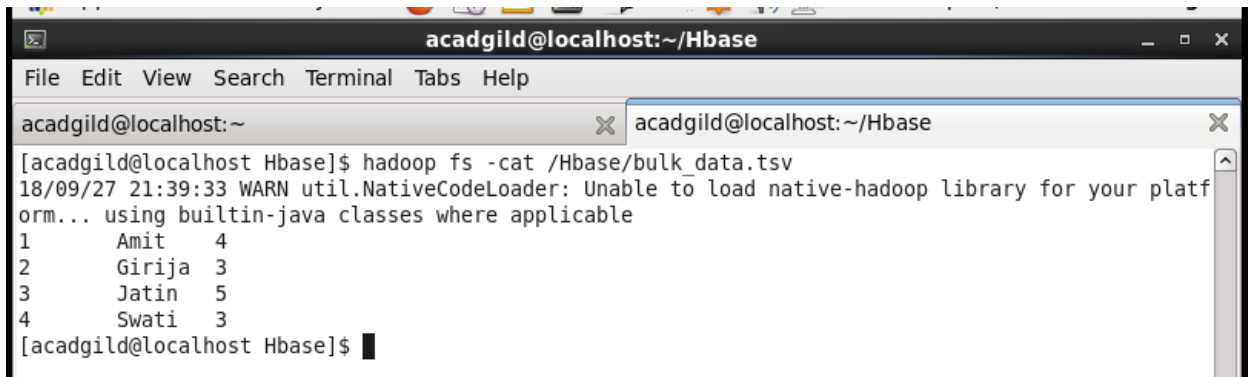
Created a directory with name Hbase in HDFS

A terminal window titled 'acadgild@localhost:~' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the command 'hadoop fs -mkdir /Hbase' being executed. The output includes a timestamp '18/09/27 21:36:13', a warning message 'WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable', and a mail notification 'You have new mail in /var/spool/mail/acadgild'. The prompt returns to '[acadgild@localhost ~]\$'.

Imported the bulk\_data.tsv file into Hbase directory in HDFS.

A terminal window titled 'acadgild@localhost:~/Hbase' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the command 'hadoop fs -put bulk\_data.tsv /Hbase/'. The output includes a timestamp '18/09/27 21:38:50', a warning message 'WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable', and a mail notification 'You have new mail in /var/spool/mail/acadgild'. The prompt returns to '[acadgild@localhost Hbase]\$'.

The contents of the file is viewed using the cat command.

A terminal window titled 'acadgild@localhost:~/Hbase' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the command 'hadoop fs -cat /Hbase/bulk\_data.tsv'. The output includes a timestamp '18/09/27 21:39:33', a warning message 'WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable', and the contents of the file: a table with 4 rows and 3 columns. The prompt returns to '[acadgild@localhost Hbase]\$'.

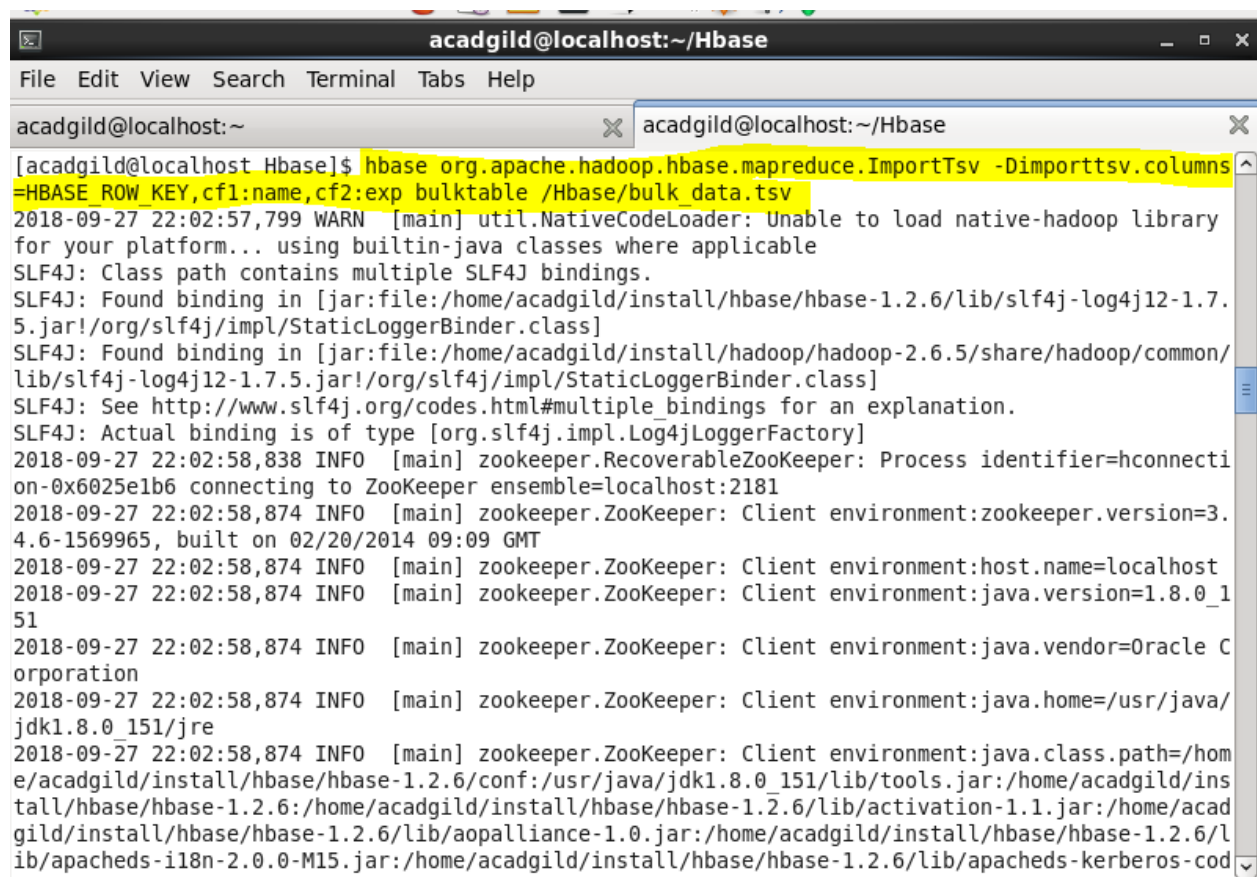


Using the below command, data is imported into the Hbase table.

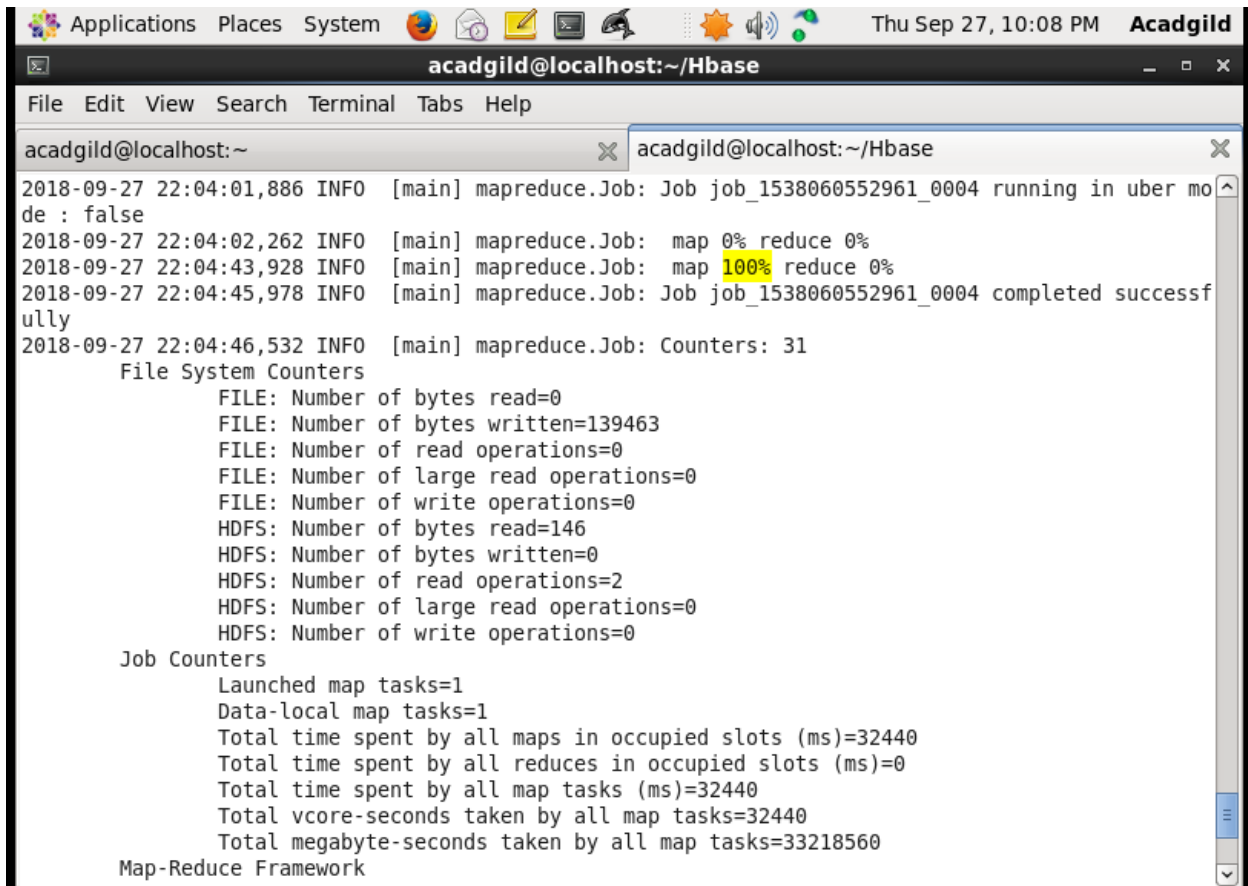
**Hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -  
Dimporttsv.columns=HBASE\_ROW\_KEY,cf1:name,cf2:exp  
bulktable/hbase/bulk\_data.tsv**

Usage: importtsv -Dimporttsv.columns=a,b,c <tablename> <inputdir>

Imports the given input directory of TSV data into the specified table.



```
acadgild@localhost:~/Hbase
File Edit View Search Terminal Tabs Help
acadgild@localhost:~ acadgild@localhost:~/Hbase
[acadgild@localhost Hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns
=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /Hbase/bulk_data.tsv
2018-09-27 22:02:57,799 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.
5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/
lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2018-09-27 22:02:58,838 INFO [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnecti
on-0x6025e1b6 connecting to ZooKeeper ensemble=localhost:2181
2018-09-27 22:02:58,874 INFO [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.
4.6-1569965, built on 02/20/2014 09:09 GMT
2018-09-27 22:02:58,874 INFO [main] zookeeper.ZooKeeper: Client environment:host.name=localhost
2018-09-27 22:02:58,874 INFO [main] zookeeper.ZooKeeper: Client environment:java.version=1.8.0_1
51
2018-09-27 22:02:58,874 INFO [main] zookeeper.ZooKeeper: Client environment:java.vendor=Oracle C
orporation
2018-09-27 22:02:58,874 INFO [main] zookeeper.ZooKeeper: Client environment:java.home=/usr/java/
jdk1.8.0_151/jre
2018-09-27 22:02:58,874 INFO [main] zookeeper.ZooKeeper: Client environment:java.class.path=/hom
e/acadgild/install/hbase/hbase-1.2.6/conf:/usr/java/jdk1.8.0_151/lib/tools.jar:/home/acadgild/ins
tall/hbase/hbase-1.2.6:/home/acadgild/install/hbase/hbase-1.2.6/lib/activation-1.1.jar:/home/acad
gild/install/hbase/hbase-1.2.6/lib/aopalliance-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/l
ib/apacheds-i18n-2.0.0-M15.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-kerberos-cod
```

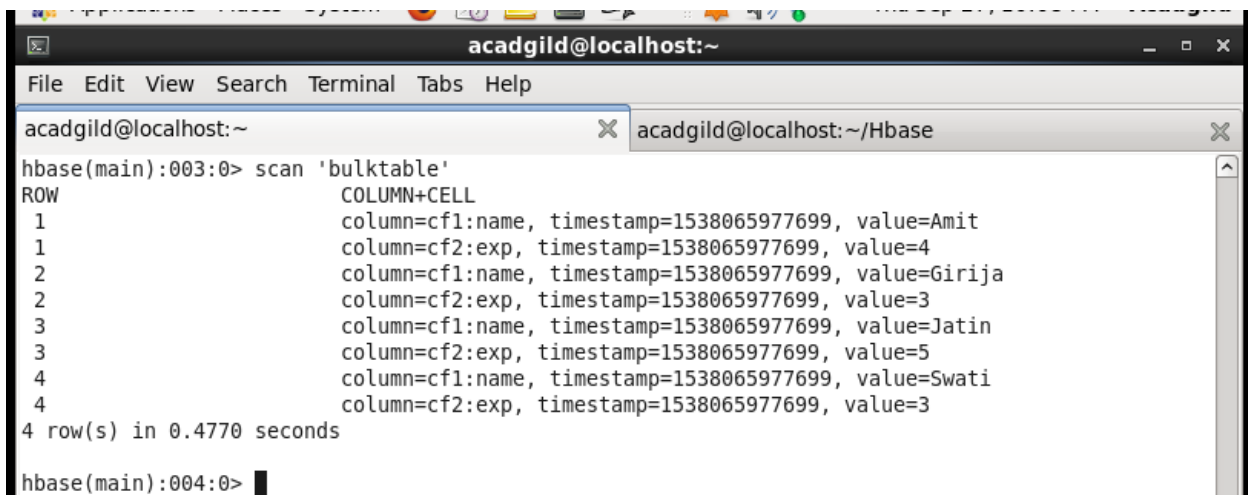


The screenshot shows a terminal window titled 'acadgild@localhost:~/Hbase'. The terminal displays the output of a Hadoop MapReduce job. The logs indicate that the job 'job\_1538060552961\_0004' has completed successfully. It shows the progress of map and reduce tasks, with the map tasks reaching 100% completion. The final output includes a summary of counters for the file system, HDFS, and the job itself.

```
2018-09-27 22:04:01,886 INFO [main] mapreduce.Job: Job job_1538060552961_0004 running in uber mode : false
2018-09-27 22:04:02,262 INFO [main] mapreduce.Job: map 0% reduce 0%
2018-09-27 22:04:43,928 INFO [main] mapreduce.Job: map 100% reduce 0%
2018-09-27 22:04:45,978 INFO [main] mapreduce.Job: Job job_1538060552961_0004 completed successfully
2018-09-27 22:04:46,532 INFO [main] mapreduce.Job: Counters: 31
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=139463
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=146
        HDFS: Number of bytes written=0
        HDFS: Number of read operations=2
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=0
    Job Counters
        Launched map tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=32440
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=32440
        Total vcore-seconds taken by all map tasks=32440
        Total megabyte-seconds taken by all map tasks=33218560
Map-Reduce Framework
```

Now the table has been checked whether the data has been imported successfully or not.

The below screenshot shows that data has been imported successfully.



The screenshot shows a terminal window titled 'acadgild@localhost:~/Hbase'. The terminal displays the output of a HBase scan operation on a table named 'bulktable'. The scan results show 4 rows of data, each with two columns: 'cf1:name' and 'cf2:exp'. The values for 'cf1:name' are Amit, Girija, Jatin, and Swati, and the values for 'cf2:exp' are 4, 3, 5, and 3 respectively. The scan took 0.4770 seconds to complete.

```
hbase(main):003:0> scan 'bulktable'
ROW COLUMN+CELL
1 column=cf1:name, timestamp=1538065977699, value=Amit
1 column=cf2:exp, timestamp=1538065977699, value=4
2 column=cf1:name, timestamp=1538065977699, value=Girija
2 column=cf2:exp, timestamp=1538065977699, value=3
3 column=cf1:name, timestamp=1538065977699, value=Jatin
3 column=cf2:exp, timestamp=1538065977699, value=5
4 column=cf1:name, timestamp=1538065977699, value=Swati
4 column=cf2:exp, timestamp=1538065977699, value=3
4 row(s) in 0.4770 seconds
hbase(main):004:0>
```