TASK-1

Q. NoSql Database-

Ans. The database technology can be divided into two types in terms of how they are built-

a. Relational Database
b. Non-Relational Database

Relational databases are structured, like phone books that store phone numbers and addresses. Non-relational databases are document-oriented and distributed, like file folders that hold everything from a person's address and phone number to their Facebook likes and online shopping preferences.

We call them SQL and NoSQL, referring to whether or not they're written solely in structured query language (SQL).

NOSQL DATABASES: NON-RELATIONAL & DISTRIBUTED DATA

If our data requirements aren't clear at the outset or if we're dealing with massive amounts of unstructured data, we may not have the luxury of developing a relational database with clearly defined schema. Unstructured data from the web can include sensor data, social sharing, personal settings, photos, location-based information, online activity, usage metrics, and more. Trying to store, process, and analyze all of this unstructured data led to the development of schema-less alternatives to SQL.

Taken together, these alternatives are referred to as NoSQL, meaning "Not only SQL." Now today NoSQL databases have become the first alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. They go well beyond the more widely understood legacy, relational databases (such as Oracle, SQL Server and DB2 databases) in satisfying the needs of today's modern business applications. A very flexible and schema-less data model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are "not only" SQL typically characterize this technology.

Q. Types of NoSql database-

Ans. Types of NoSQL Databases

There are four general types of NoSQL databases, each with their own specific attributes:

- **Graph database** – Based on graph theory, these databases are designed for data whose relations are well represented as a graph and has elements which are interconnected, with an undetermined number of relations between them. Graph structures are used with edges, nodes and properties which provides index-free adjacency. Data can be easily transformed from one model to the other using a Graph Base NoSQL database. Examples include: Neo4j and Titan.

- **Key-Value store** – It has a Big Hash Table of keys & values. The key value type basically, uses a hash table in which there exists a unique key and a pointer to a particular item of data. A bucket is a logical group of keys – but they don't physically group the data. There can be identical keys in different buckets.

  Performance is enhanced to a great degree because of the cache mechanisms that accompany the mappings. There is no complexity around the Key Value Store database model as it can be implemented in a breeze.

  These databases are designed for storing data in a schema-less way. In a key-value store, all of the data within consists of an indexed key and a value, hence the name. Examples of this type of database include: Cassandra, DyanmoDB, Azure Table Storage (ATS), Riak, BerkeleyDB.

- **Column store** – In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows. Each storage block contains data from only one column (also known as wide-column stores) instead of storing data in rows, these databases are designed for storing data tables as sections of columns of data, rather than as rows of data. While this simple description sounds like the inverse of a standard database, wide-column stores offer very high performance and a highly scalable architecture. Examples include: HBase, BigTable and HyperTable.

- **Document database** – Stores documents made up of tagged elements. Expands on the basic idea of key-value stores where "documents" contain more complex in that they contain data and each document is assigned a unique key, which is used to retrieve the document. The data which is a collection of key value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as "documents") provide some structure and encoding of the managed data. These are designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Examples include: MongoDB and CouchDB.
  One key difference between a key-value store and a document store is that the latter embeds attribute metadata associated with stored content, which essentially provides a way to query the data based on the contents.

Q. CAP Theorem-

Ans. CAP stands for **Consistency**, **Availability** and **Partition Tolerance**. Now CAP theorem reiterates the need to find balance between Consistency, Availability and Partition tolerance for any distributed system.

Consistency means all the nodes see the same data at the same time. Which means whenever we read a record (or data), consistency guaranties that it will give same data how many times we read. Simply we can say that each server returns the right response to each request, thus the system will be always consistent whenever we read or write data into that.
Availability implies that every request receives a response about whether it was successful or failed. It's more of a handshaking mechanism in computer network methodology (even if it's not the latest data or consistent across the system or just a message saying the system isn't working).
Coming to **partition tolerance**, the system continues to operate despite arbitrary message loss or failure of part of the system. Systems with partition tolerance feature works well despite physical network partitions. According to CAP Theorem distributed systems can satisfy any two features at the same time but not all three features. Traditional systems like RDBMS provide consistency and availability. Column oriented databases like MongoDB, Hbase and Big Table provide features consistency and partition tolerance.

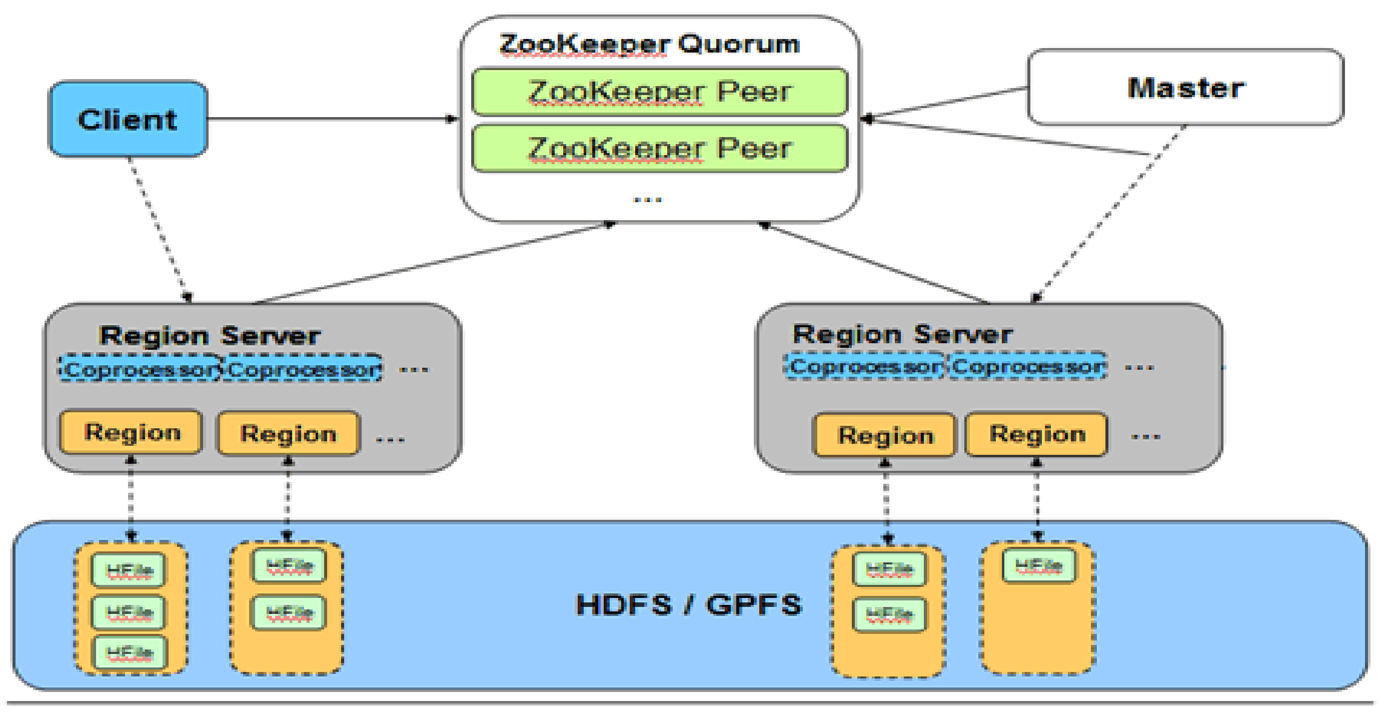
One property should be scarified among three, so you have to choose combination of CA or CP or AP.
Consistency – Partition Tolerance:
System will give a consistent data and it will be distributed across the cluster. But it becomes unavailable when a node goes down.
Availability – Partition Tolerance:
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 a consistent data and we 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.)

Q. HBase Architecture-

Ans. HBase combines the scalability of Hadoop by running on the Hadoop Distributed File System (HDFS), with real-time data access as a **key/value** store and deep analytic capabilities of Map Reduce. HBase is built on top of the distributed file system (DFS), which can store large files. HBase provides fast record lookups and updates for large tables.

Below is the Architecture of HBase Cluster-

HBase Cluster Architecture



Hbase architecture consists of mainly HMaster, HRegionserver, HRegions and Zookeeper. Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. If the client wants to communicate with regions servers, client has to approach Zookeeper

The ZooKeeper cluster acts as a coordination service for the entire HBase cluster.

HBase contains two primary services:

- **Master server** It is also known as HMASTER. HMaster in Hbase plays vital role in terms of performance and maintaining nodes in the cluster. It provides admin performance and distributes services to different region servers. HMaster assigns regions to region servers. The master server co-ordinates the cluster and performs administrative operations, such as assigning regions and balancing the loads. The HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster. When client wants to change any schema and to change any Meta data operations, HMaster takes responsibility for these operations.

- **Region server** The region servers do the real work. A subset of the data of each table is handled by each region server. Clients talk to region servers to access data in HBase. Other HBase architectural components include the client library (API), at least one master server, and many region servers. The region servers can be added or removed while the system is up and running to accommodate increased workloads. The master is responsible for assigning regions to region servers. It uses Apache ZooKeeper, a distributed coordination service, to facilitate that task. Data is partitioned and replicated across a number of regions located on region servers.

  Regions Region servers manage a set of regions. An HBase table is made up of a set of regions. Regions are the basic unit of work in HBase. It is what is used as a split by the map reduce framework. The region contains store objects that correspond to column families. There is one store instance for each column family. Store objects create one or more StoreFiles, which are wrappers around the actual storage file called the HFile. The region also contains a MemStore, which is in-memory storage and is used as a write cache. Rows are written to the MemStore. The data in the MemStore is ordered. If the MemStore becomes full, it is persisted to an HFile on disk.
  To improve performance, it is important to get an even distribution of data among regions, which ensures the best parallelism in map tasks.

  HFiles- HFiles are the physical representation of data in HBase. Clients do not read HFiles directly but go through region servers to get to the data. HBase internally puts the data in indexed StoreFiles that exist on HDFS for high-speed lookups. The default block size is 64 KB but it can be changed since it is configurable. HBase API can be used to access specific values and also scan ranges of values given a start and end key. When the data in memory exceeds a given maximum value, it is flushed as an HFile to disk and after that the commit logs are discarded up to the last unflushed modification. The system can continue to serve readers and writers without blocking them while it is flushing the memstore to disk. This is done by rolling the memstore in memory where the new empty one is taking the updates and the old full one is transferred into an HFile. At the same time, no sorting or other special processing has to be performed since the data in the memstores is already sorted by keys matching what HFiles represent on disk.

An HBase table contains *column families*, which are the logical and physical grouping of columns. There are *column qualifiers* inside of a column family, which are the columns. Column families contain columns with time stamped versions. Columns only exist when they are inserted, which makes HBase a sparse database. All column members of the same column family have the same column family prefix. Each column value is identified by a key. The *row key* is the implicit primary key. Rows are sorted by the *row key*.

Q. HBase vs RDBMS-

Ans. Hadoop and RDBMS are varying concepts of processing, retrieving and storing the data or information. While Hadoop is an open-source Apache project, RDBMS stands for Relational Database Management System. Hadoop framework has been written in Java which makes it scalable and makes it able to support applications that call for high performance standards. Hadoop framework enables the storage of large amounts of data on files systems of multiple computers. Hadoop is configured to allow scalability from a single computer node to several thousands of nodes or independent workstations in a manner that the individual nodes utilize local computer storage CPU processing power and memory.

Traditional databases or RDBMS' have "ACID" properties which is an acronym that stands for Atomicity, Consistency, Isolation and Durability. These properties are not found at all in Hadoop. For instance if one has to script code for taking money from one particular bank to deposit the same into another bank then, they will have to painstakingly code all scenarios that may occur such as what happens when money is taken out but a failure results before it is moved into another account.

HBase is a distributed, column-oriented data storage system. It picks up where Hadoop left off by providing random reads and writes on top of HDFS. It has been designed from the ground up with a focus on scale in every direction: tall in numbers of rows (billions), wide in numbers of columns (millions), and to be horizontally partitioned and replicated across thousands of commodity nodes automatically.

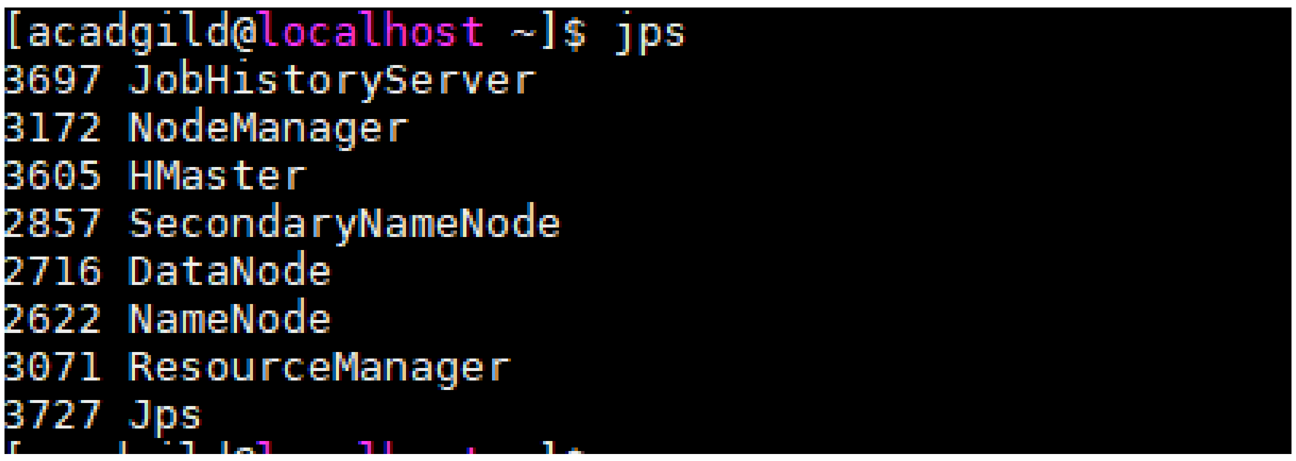
However Schema/Database in RDBMS can be compared to namespace in Hbase. A table in RDBMS can be compared to column family in Hbase. A record (after table joins) in RDBMS can be compared to a record in Hbase. A collection of tables in RDBMS can be compared to a table in Hbase..

The basic difference between HBase and RDBMS are as follows-

H Base	RDBMS
1. Column-oriented	1. Row-oriented(mostly)
2. Flexible schema, add columns on the Fly	2. Fixed schema
3. Good with sparse tables.	3. Not optimized for sparse tables.
4. No query language	4. SQL
5. Wide tables	5. Narrow tables
6. Joins using MR – not optimized	6. optimized for Joins(small, fast ones)
7. Tight – Integration with MR	7. Not really
8. De-normalize your data.	8. Normalize as you can
9. Horizontal scalability-just add hard war.	9. Hard to share and scale.
10. Consistent	10. Consistent
11. No transactions.	11. transactional
12. Good for semi-structured data as well as structured data.	12. Good for structured data.

TASK-2- Below is the use case described in blog-

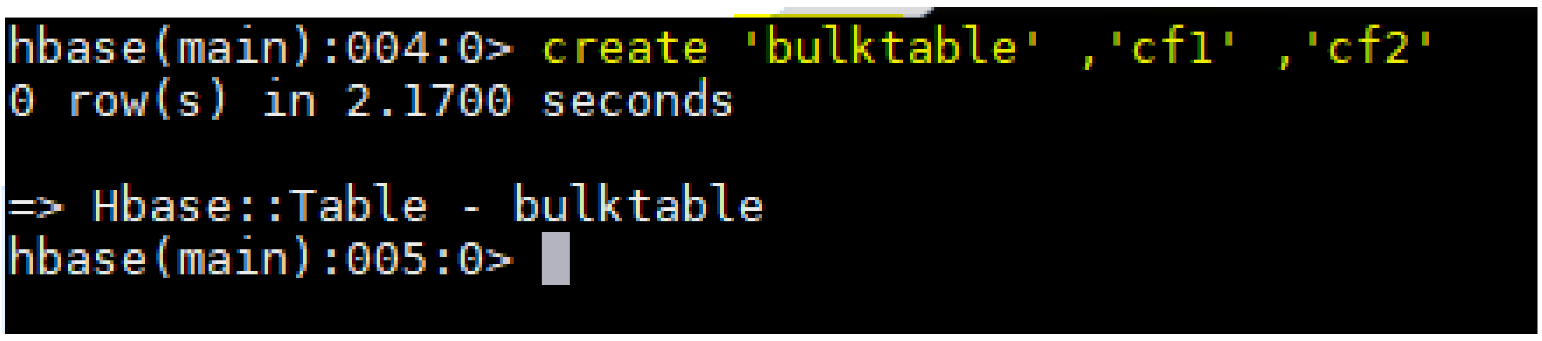
At first we make sure that all daemons are up and running-



```
[acadgild@localhost ~]$ jps
3697 JobHistoryServer
3172 NodeManager
3605 HMaster
2857 SecondaryNameNode
2716 DataNode
2622 NameNode
3071 ResourceManager
3727 Jps
```

STEP-1-

We then create a table named ‘bulktable’ in hbase with column families cf1 and cf2

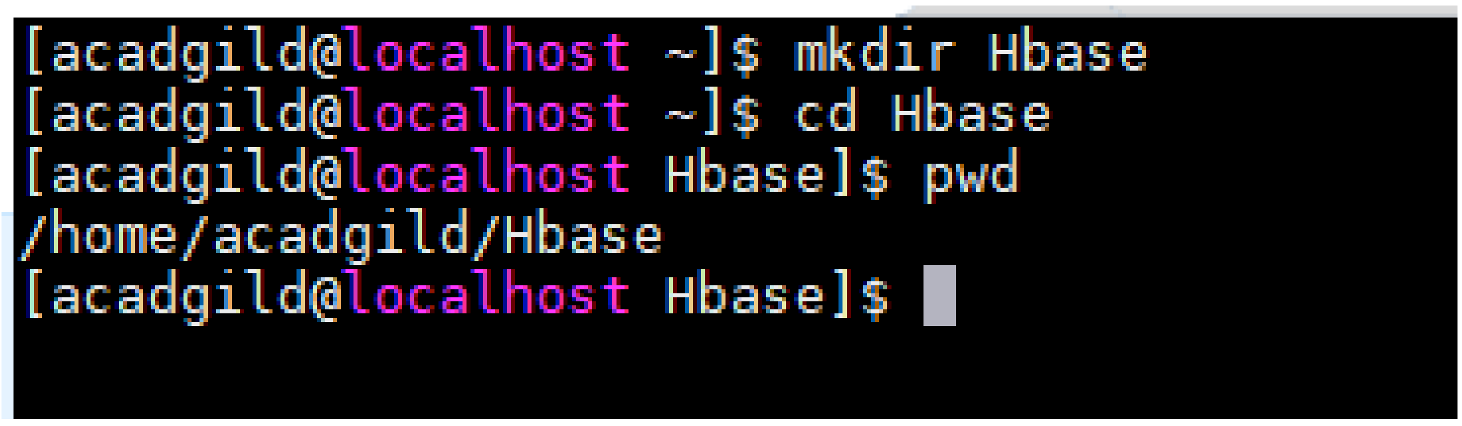


```
hbase(main):004:0> create 'bulktable' ,'cf1' ,'cf2'
0 row(s) in 2.1700 seconds

=> Hbase::Table - bulktable
hbase(main):005:0>
```

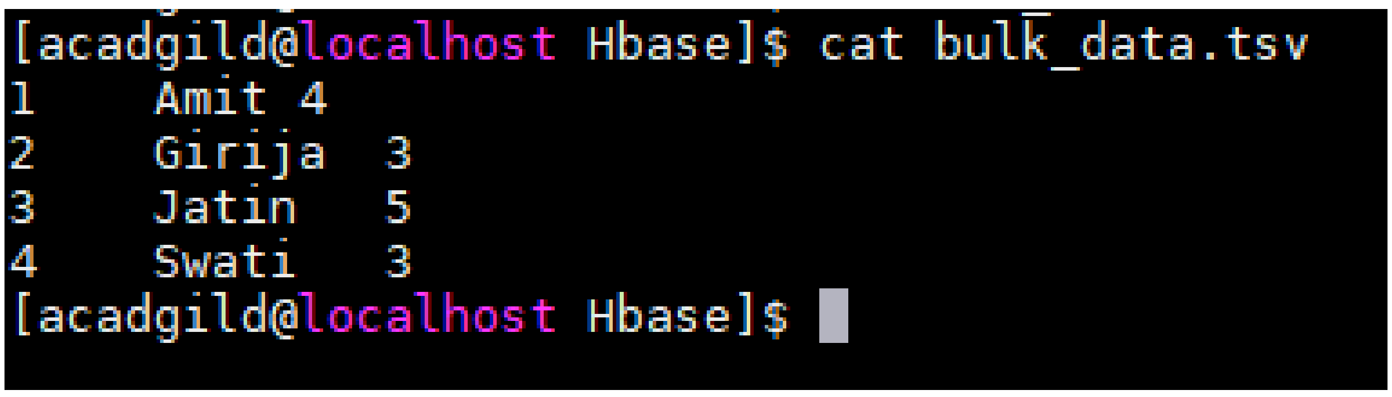
STEP-2

Create a file inside the HBase directory named bulk_data.tsv with tab separated data inside using below command in terminal.



```
[acadgild@localhost ~]$ mkdir Hbase
[acadgild@localhost ~]$ cd Hbase
[acadgild@localhost Hbase]$ pwd
/home/acadgild/Hbase
[acadgild@localhost Hbase]$
```

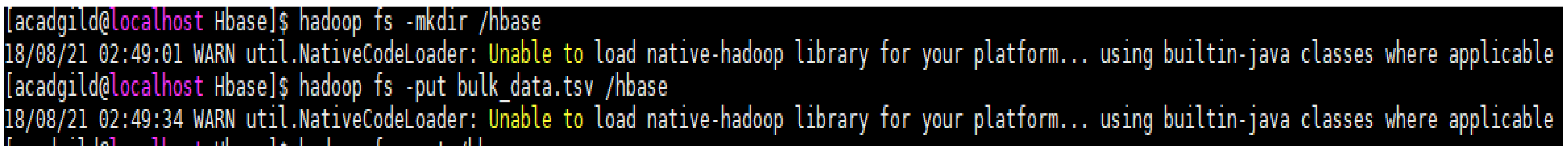
STEP-3



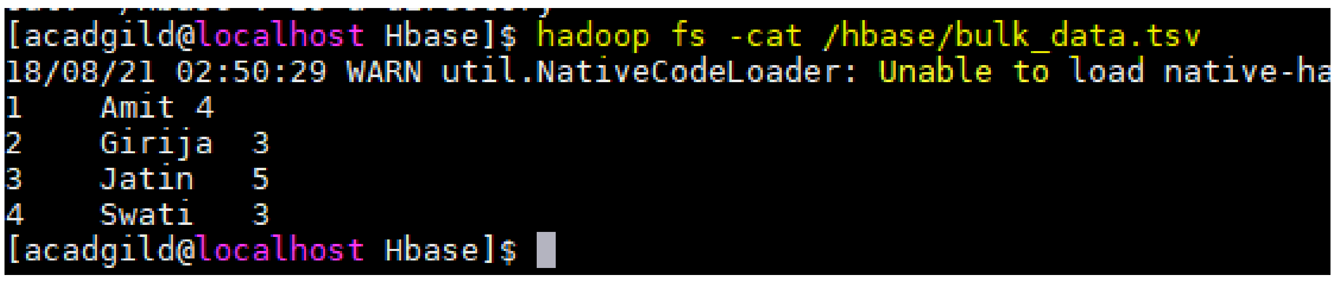
```
[acadgild@localhost Hbase]$ cat bulk_data.tsv
1     Amit  4
2     Girija  3
3     Jatin   5
4     Swati   3
[acadgild@localhost Hbase]$
```

STEP-4

Our data should be present in HDFS while performing the import task to Hbase.
In real time projects, the data will already be present inside HDFS.
Here for our learning purpose, we copy the data inside HDFS using below commands in terminal.



```
[acadgild@localhost Hbase]$ hadoop fs -mkdir /hbase
18/08/21 02:49:01 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[acadgild@localhost Hbase]$ hadoop fs -put bulk_data.tsv /hbase
18/08/21 02:49:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```



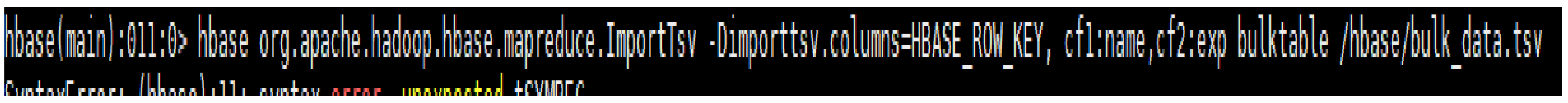
```
[acadgild@localhost Hbase]$ hadoop fs -cat /hbase/bulk_data.tsv
18/08/21 02:50:29 WARN util.NativeCodeLoader: Unable to load native-ha
1    Amit 4
2    Girija  3
3    Jatin   5
4    Swati   3
[acadgild@localhost Hbase]$
```

STEP-5

After the data is present now in HDFS.In terminal, we give the following command along with arguments<tablename> and <path of data in HDFS>

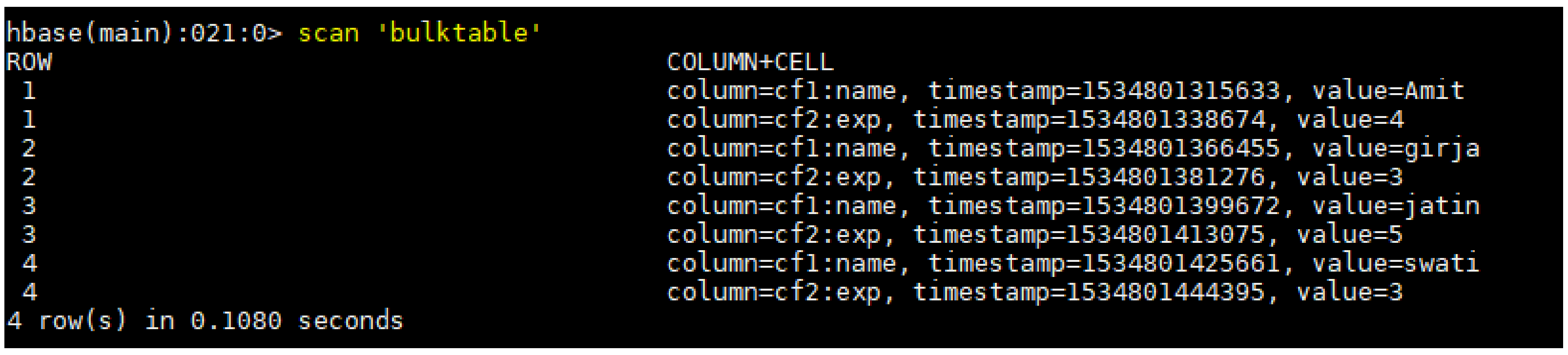
Command:

hbase org.apache.hadoop.hbase.mapreduce.ImportTsv –Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv



```
hbase(main):011:0> hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE_ROW_KEY, cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv
```

Below is the table after data insert



```
hbase(main):021:0> scan 'bulktable'
ROW                                         COLUMN+CELL
 1                                          column=cf1:name, timestamp=1534801315633, value=Amit
 1                                          column=cf2:exp, timestamp=1534801338674, value=4
 2                                          column=cf1:name, timestamp=1534801366455, value=girja
 2                                          column=cf2:exp, timestamp=1534801381276, value=3
 3                                          column=cf1:name, timestamp=1534801399672, value=jatin
 3                                          column=cf2:exp, timestamp=1534801413075, value=5
 4                                          column=cf1:name, timestamp=1534801425661, value=swati
 4                                          column=cf2:exp, timestamp=1534801444395, value=3
4 row(s) in 0.1080 seconds
```