Big Data- Hadoop – Assignment 4

HBase Basics:

Task 1:

Answer in your own words with example.

1. **What is NoSQL database?**

NoSQL refers to a database that is not based on SQL (Structured Query Language), which is the language most commonly associated with **relational** databases. Essentially, NoSQL data isn't relational, NoSQL databases usually do not have schema, and they come with looser consistency models than traditional relational databases do.

The term "NoSQL" refers to the fact that traditional relational databases are not adequate for all solutions, particularly ones involving large volumes of data. But the term has been extended to also mean "Not only SQL", indicating support for potential SQL-based interfaces even if the core database isn't relational.

Dozens of NoSQL data stores are available; the following are among the most popular:

- MongoDB. Open-source document database.
- CouchDB. Database that uses JSON for documents, JavaScript for MapReduce queries, and regular HTTP for an API.
- GemFire. Distributed data management platform providing dynamic scalability, high performance, and database-like persistence.
- Redis. Data structure server wherein keys can contain strings, hashes, lists, sets, and sorted sets.
- Cassandra. Database that provides scalability and high availability without compromising performance.
- memcached. Open source high-performance, distributed-memory, and object-caching system.
- Hazelcast. Open source highly scalable data distribution platform.
- HBase. Hadoop database, a distributed and scalable big data store.
- Mnesia. Distributed database management system that exhibits soft real-time properties.
- Neo4j. Open source high-performance, enterprise-grade graph database.

2. **How does data get stored in NoSQl database?**

There are various NoSQL Databases. Each one uses a different method to store data. Some might use column store, some document, some graph, etc., Each database has its own unique characteristics.

Key-value store: Key-value (KV) stores use the associative array (also known as a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.

The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented as an extension of it. This extension is computationally powerful, in that it can efficiently retrieve selective key ranges. Key-value stores can use consistency models ranging from eventual consistency to serializability. Some databases support ordering of keys. Examples of key-value storage are Berkeley DB, Dynamo and so on.

Document store: Each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode in some standard formats or encodings. Encodings in use include XML, YAML, and JSON as well as binary forms like BSON. Documents are addressed in the database via a unique key that represents that document. One of the other defining characteristics of a document-oriented database is that in addition to the key lookup performed by a key-value store, the database also offers an API or query language that retrieves documents based on their contents. Examples are Mongo DB.

Graph: This kind of database is designed for data whose relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them. The type of data could be social relations, public transport links, road maps, network topologies, etc. Example Apache Giraph

Wide-column stores: Wide-column stores organize data tables as columns instead of as rows. Wide-column stores can be found both in SQL and NoSQL databases. Wide-column stores can query large data volumes faster than conventional relational databases. A wide-column data store can be used for recommendation engines, catalogs, fraud detection and other types of data processing. Google BigTable, Cassandra and HBase are examples of wide-column stores.

3. **What is a column family in HBase?**

Columns in Apache HBase are grouped into column families. A column family defines shared features to all columns that are created within them (think of it almost as a sub-table within your larger table). You will notice that HBase columns are composed of a combination of the column family and column qualifier (or column key): 'family: qualifier'.

All column members of a column family have the same prefix. The colon character (:) delimits the column family from the column. The column family prefix must be composed of printable characters. The qualifying tail, the column family qualifier, can be made of any arbitrary bytes. Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up and running. For example, the columns *courses: history* and *courses: math* are both members of the courses column family.

4. How many maximum number of columns can be added to HBase table?

There is no limit on number of columns in HBase. Some shortcomings in the current HBase implementation do not properly support large number of column families in a single table. That number should be in low tens. Most of the time up to three column families should work fine without any significant performance drawback. Ideally you should go with a single column family.

A column family can have an arbitrary number of columns denoted by a column qualifier which is like a column's label. For example:

```
{
 "row1": {"1": {"color": "green",
             "size": 25},
        "2": {"weight": 52,
             "size": 18}
       },
 "row2": {"1": {"color": "blue"},
        "2": {"height": 192,
             "size": 43}
       }
 }
```

As you can see in the example above, the same column family (e.g., "1") in two rows can have different columns. In row "row1", it has columns "color" and "size", while in row "row2", it has only "color" column. It can also have a column that is none of the above. Since rows can have different columns in column families there is no a single way to query for a list of all columns in all column families. This means that you have to do a full table scan.

There is no specific limit on the number of columns in a column family. Actually, you can have millions of columns in the single column family.

5. Why columns are not defined at the time of table creation in HBase?

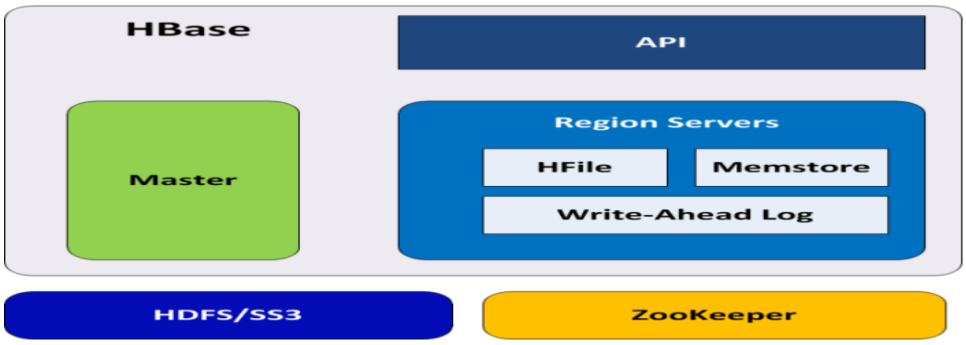
One of the advantages of using hive is only the column family needs to be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up and running. The creation of columns at run time helps to add any type of column and avoids unnecessary recreation of table.

Columns are usually physically co-located in column families. A column qualifier is an index for a given data and it is added to a column family. Data within a column family is addressed via the column qualifier. Since column qualifiers are mutable and they may vary between rows. They do not have data types and they are always treated as arrays of bytes, hence there is no need of defining columns at table creation.

6. How does data get managed in HBase?

HBase is built upon distributed filesystems with file storage distributed across commodity machines. The distributed file systems HBase works with include

- Hadoop's Distributed File System (HDFS) and
- Amazon's Simple Storage Service (SS3).



HDFS provides a scalable and replicated storage layer for HBase. It guarantees that data is never lost by writing the changes across a configurable number of physical servers.

The data is stored in HFiles, which are ordered immutable key/value maps. Internally, the HFiles are sequences of blocks with a block index stored at the end. The block index is loaded when the HFile is opened and kept in memory. 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.

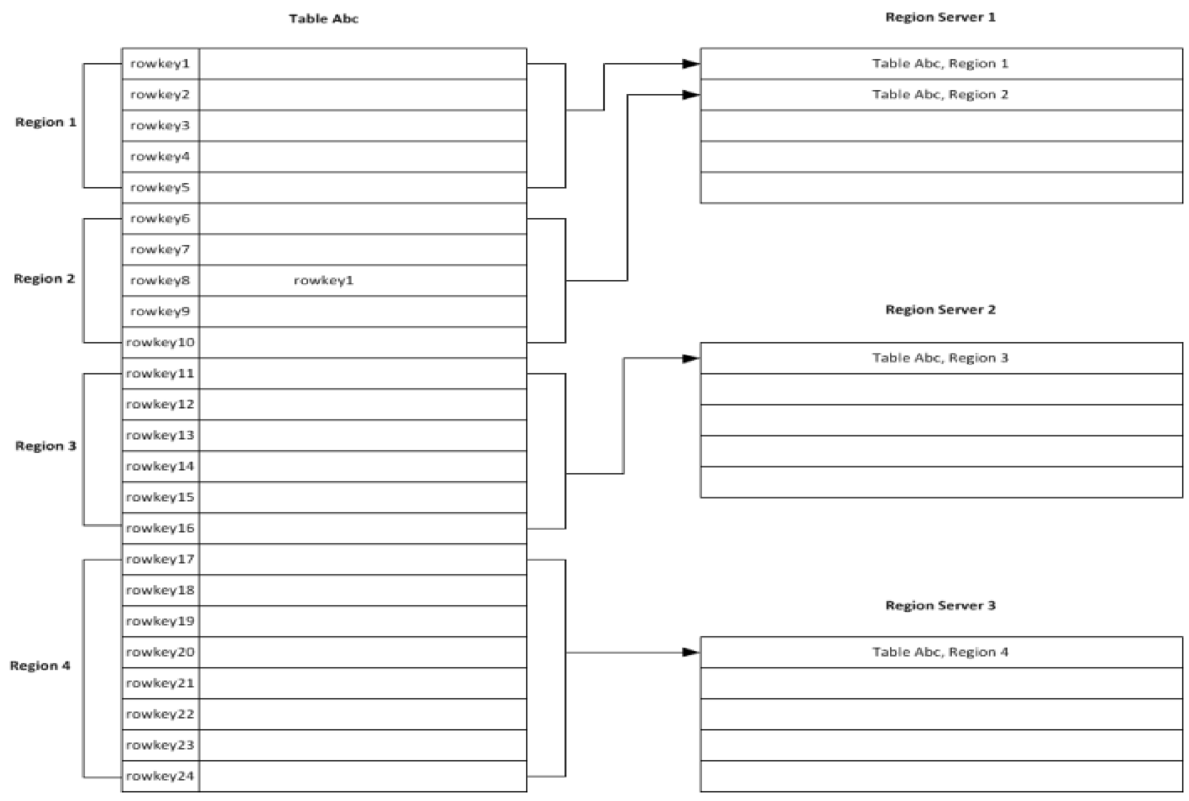
Since every HFile has a block index, lookups can be performed with a single disk seek. First, HBase does a binary search in the in-memory block index to find a block containing the given key and then the block is read from disk.

When data is updated it is first written to a commit log, called a write-ahead log (WAL) and then it is stored in the in-memory memstore. 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.

The write-ahead log (WAL) is used for recovery purposes only. Since flushing memstores to disk causes creation of HFiles, HBase has a housekeeping job that merges the HFiles into larger ones using compaction. Various compaction algorithms are supported.

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.



As mentioned above, assignment and distribution of regions to region servers is automatic. However manual management of regions is also possible. When a region's size reaches a pre-defined threshold, the region will automatically split into two child regions. The split happens along a row key boundary. A single region always manages an entire row. It means that a row is never divided.

7. What happens internally when new data gets inserted into HBase table?

HBase stores data in a form of a distributed sorted multidimensional persistence maps called Tables. HBase does not overwrite row values. It stores different values per row by time and column qualifier. A row key, column family and column qualifier form a cell that has a value and timestamp that represents the value's version. A timestamp is recorded for each value and it is the time on the region server when the value was written.

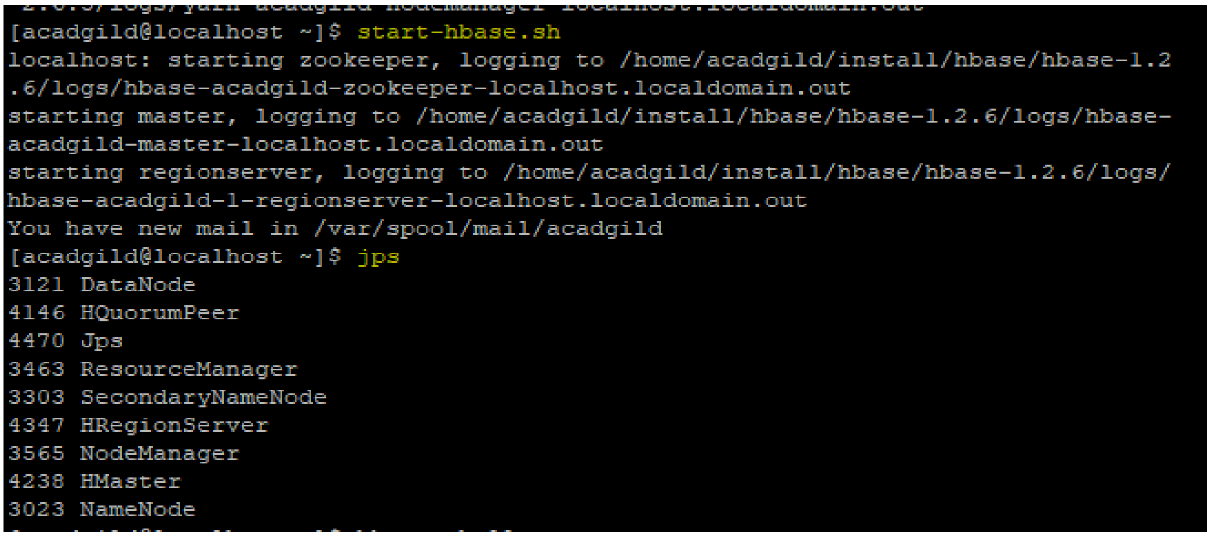
All cell's values are stored in a descending order by its timestamp. When values are retrieved and if the timestamp is not provided then HBase will return the cell value with the latest timestamp. If a timestamp is not specified during the write, the current timestamp is used. The maximum number of versions for a given column to store is part of the column schema. It is specified at table creation. It can be specified via alter table command as well.

The default value is 1. The minimum number of versions can be also set up per column family. You can also globally set up a maximum number of versions per column.

Task 2:

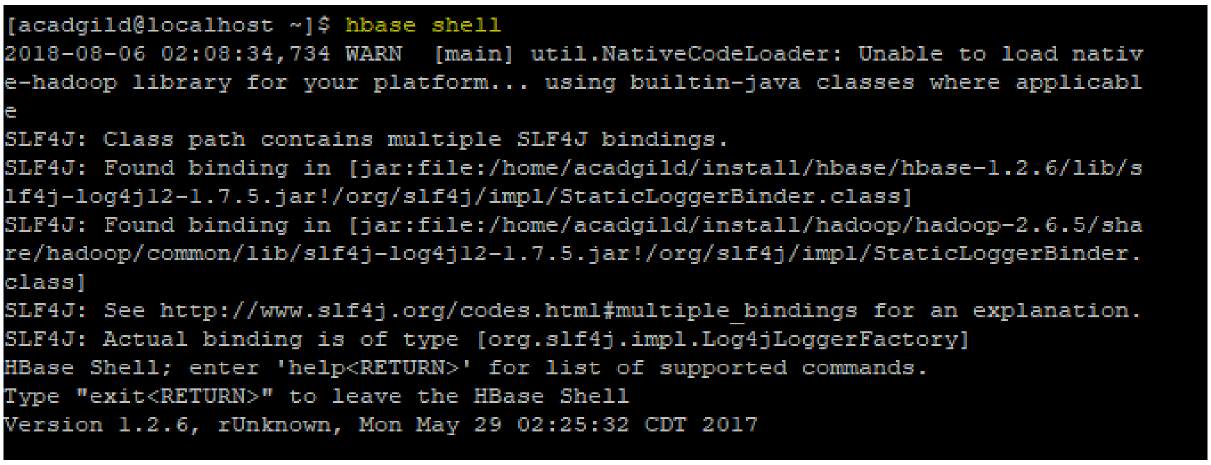
1. **Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.**

   Step 1: Starting zoo keeper using command start-hbase.sh



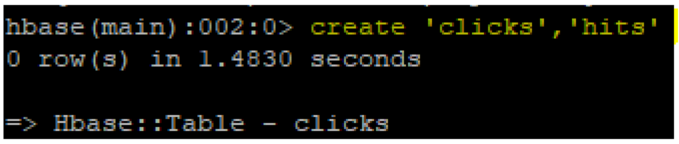
```
[acadgild@localhost ~]$ start-hbase.sh
localhost: starting zookeeper, logging to /home/acadgild/install/hbase/hbase-1.2
.6/logs/hbase-acadgild-zookeeper-localhost.localdomain.out
starting master, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-
acadgild-master-localhost.localdomain.out
starting regionserver, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/
hbase-acadgild-1-regionserver-localhost.localdomain.out
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ jps
3121 DataNode
4146 HQuorumPeer
4470 Jps
3463 ResourceManager
3303 SecondaryNameNode
4347 HRegionServer
3565 NodeManager
4238 HMaster
3023 NameNode
```

Step 2: To get HBase shell using "hbase shell" command

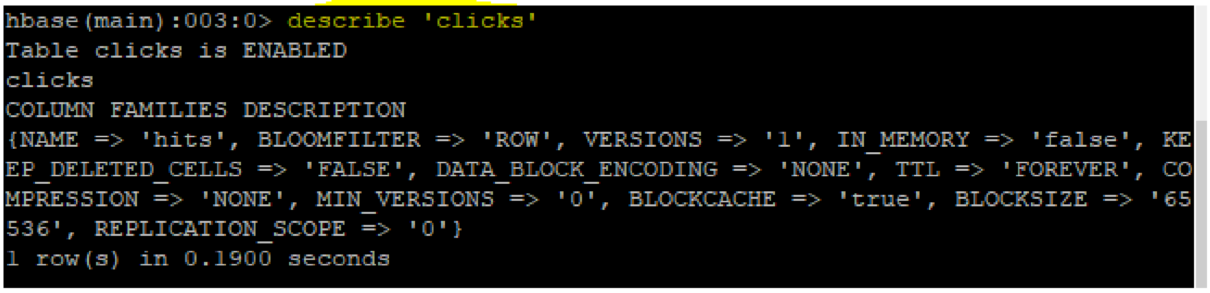


```
[acadgild@localhost ~]$ hbase shell
2018-08-06 02:08:34,734 WARN  [main] util.NativeCodeLoader: Unable to load nativ
e-hadoop library for your platform... using builtin-java classes where applicabl
e
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/s
lf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/sha
re/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]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017
```

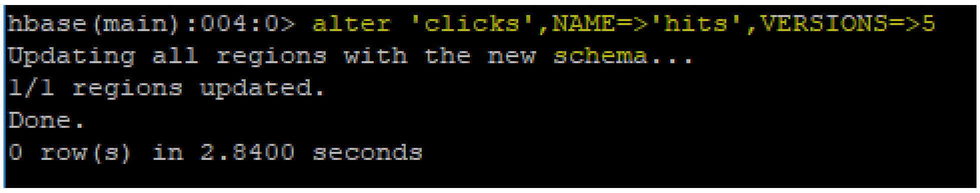
Step 3: **Creating an HBase table named 'clicks' with a column family 'hits'**



```
hbase(main):002:0> create 'clicks','hits'
0 row(s) in 1.4830 seconds

=> Hbase::Table - clicks
```

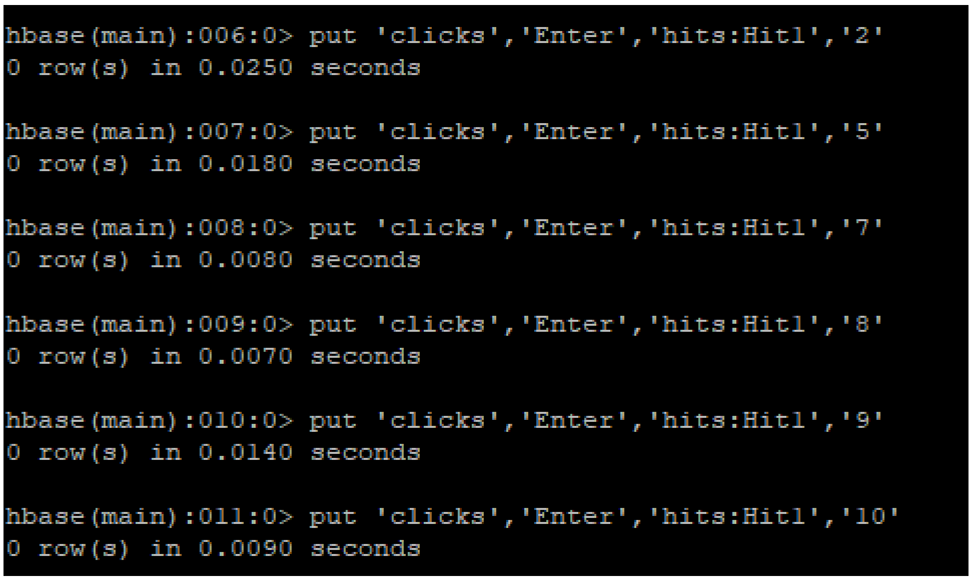


```
hbase(main):003:0> describe 'clicks'
Table clicks is ENABLED
clicks
COLUMN FAMILIES DESCRIPTION
{NAME => 'hits', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KE
EP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', CO
MPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65
536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.1900 seconds
```

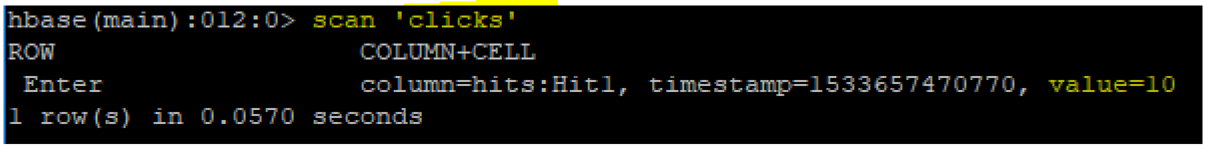


```
hbase(main):004:0> alter 'clicks',NAME=>'hits',VERSIONS=>5
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.8400 seconds
```

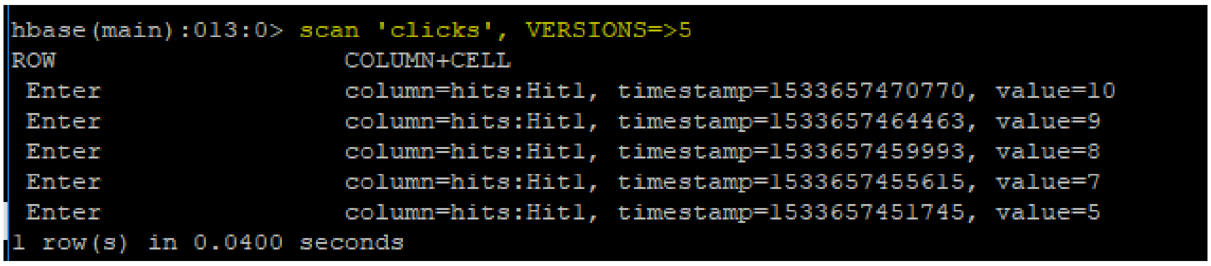
Last 5 values of qualifiers



```
hbase(main):006:0> put 'clicks','Enter','hits:Hit1','2'
0 row(s) in 0.0250 seconds

hbase(main):007:0> put 'clicks','Enter','hits:Hit1','5'
0 row(s) in 0.0180 seconds

hbase(main):008:0> put 'clicks','Enter','hits:Hit1','7'
0 row(s) in 0.0080 seconds

hbase(main):009:0> put 'clicks','Enter','hits:Hit1','8'
0 row(s) in 0.0070 seconds

hbase(main):010:0> put 'clicks','Enter','hits:Hit1','9'
0 row(s) in 0.0140 seconds

hbase(main):011:0> put 'clicks','Enter','hits:Hit1','10'
0 row(s) in 0.0090 seconds
```



```
hbase(main):012:0> scan 'clicks'
ROW                    COLUMN+CELL
 Enter                 column=hits:Hit1, timestamp=1533657470770, value=10
1 row(s) in 0.0570 seconds
```

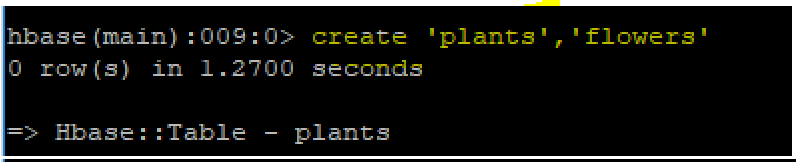


```
hbase(main):013:0> scan 'clicks', VERSIONS=>5
ROW                    COLUMN+CELL
 Enter                 column=hits:Hit1, timestamp=1533657470770, value=10
 Enter                 column=hits:Hit1, timestamp=1533657464463, value=9
 Enter                 column=hits:Hit1, timestamp=1533657459993, value=8
 Enter                 column=hits:Hit1, timestamp=1533657455615, value=7
 Enter                 column=hits:Hit1, timestamp=1533657451745, value=5
1 row(s) in 0.0400 seconds
```

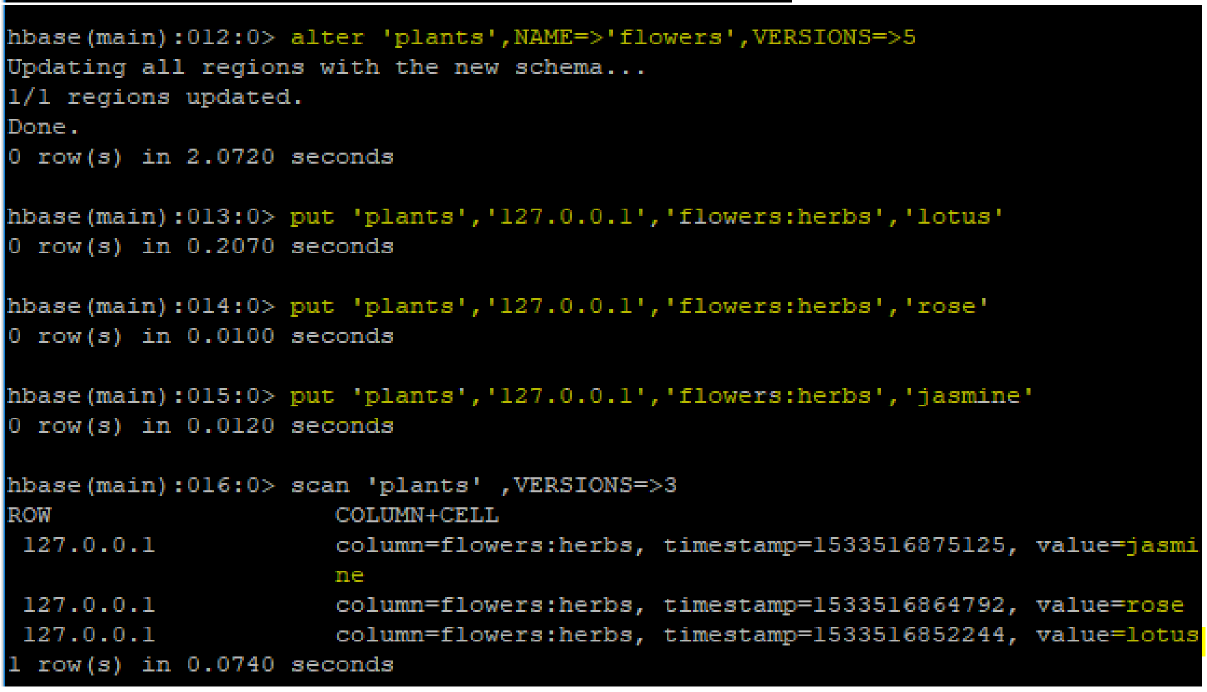
2. Add few records in the table and update some of them. Use IP Address as row-key. Scan the table to view if all the previous versions are getting displayed.

<u>**Comments:**</u> Created a table name plants with column family as flowers and where row-key was IP address.

The scan command shows the previous versions.



```
hbase(main):009:0> create 'plants','flowers'
0 row(s) in 1.2700 seconds

=> Hbase::Table - plants
```



```
hbase(main):012:0> alter 'plants',NAME=>'flowers',VERSIONS=>5
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.0720 seconds

hbase(main):013:0> put 'plants','127.0.0.1','flowers:herbs','lotus'
0 row(s) in 0.2070 seconds

hbase(main):014:0> put 'plants','127.0.0.1','flowers:herbs','rose'
0 row(s) in 0.0100 seconds

hbase(main):015:0> put 'plants','127.0.0.1','flowers:herbs','jasmine'
0 row(s) in 0.0120 seconds

hbase(main):016:0> scan 'plants' ,VERSIONS=>3
ROW                    COLUMN+CELL
 127.0.0.1             column=flowers:herbs, timestamp=1533516875125, value=jasmi
                       ne
 127.0.0.1             column=flowers:herbs, timestamp=1533516864792, value=rose
 127.0.0.1             column=flowers:herbs, timestamp=1533516852244, value=lotus
1 row(s) in 0.0740 seconds
```

<u>Advanced HBase:</u>

<u>Task 1:</u>

Explain the below concepts with an example in brief.

● **<u>Nosql Databases</u>**: NoSQL basically means "not only SQL" or it also means non-relational database. It's an alternate for traditional relational database. They are especially useful for working with large sets of distributed data. of distributed data.

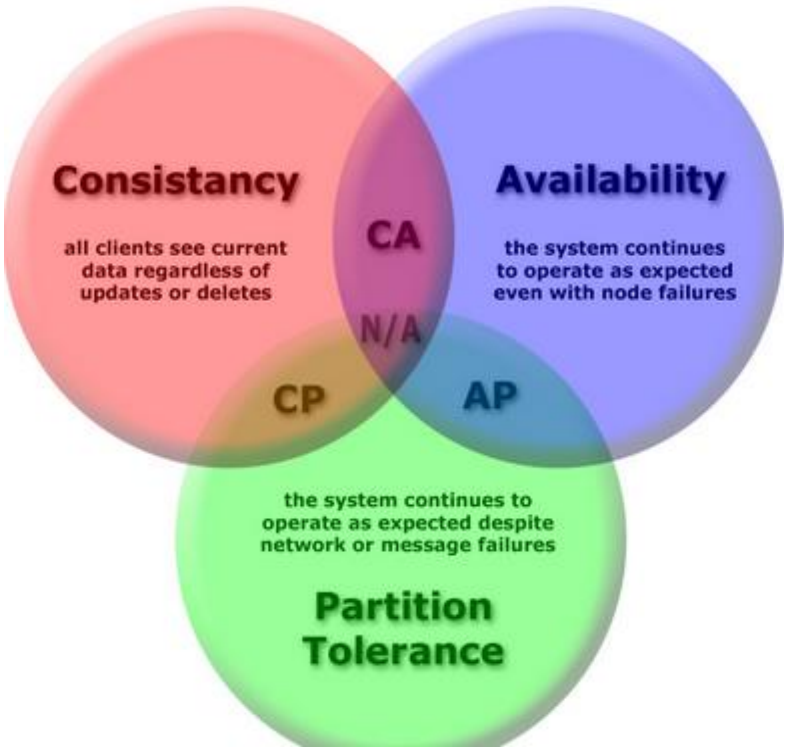
NoSQL databases are purpose built for specific data models and have flexible schemas for building modern applications. They are widely recognized for their ease of development, functionality, and performance at scale. They use a variety of data models, including document, graph, key-value, in-memory, and search.

● **<u>Types of Nosql Databases</u>**: There are 4 basic types of NoSQL databases:

1. **Key-Value Store** – It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
2. **Document-based Store-** It stores documents made up of tagged elements. {Example- CouchDB}
3. **Column-based Store-** Each storage block contains data from only one column, {Example- HBase, Cassandra}
4. **Graph-based**-A network database that uses edges and nodes to represent and store data. {Example- Neo4J}

● **<u>CAP Theorem</u>**: For any distributed system, CAP Theorem reiterates the need to find balance between Consistency, Availability and Partition tolerance.

- Consistency - This means that the data in the database remains consistent after the execution of an operation. For example, after an update operation, all clients see the same data.
- Availability - This means that the system is always on (service guarantee availability), no downtime.
- Partition Tolerance - This means that the system continues to function even if the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.


Consistancy
all clients see current data regardless of updates or deletes
Availability
the system continues to operate as expected even with node failures
CA
N/A
CP
AP
the system continues to operate as expected despite network or message failures
Partition Tolerance

● **HBase 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.

HMaster is the master server of Hbase and it coordinates the HBase cluster. It is responsible for the administrative operations of the cluster. A region server serves a region at the start of the application. During failure of region server, HMaster assign the region to another Region server. HMaster can also assign a region to another region server as part of load balancing.

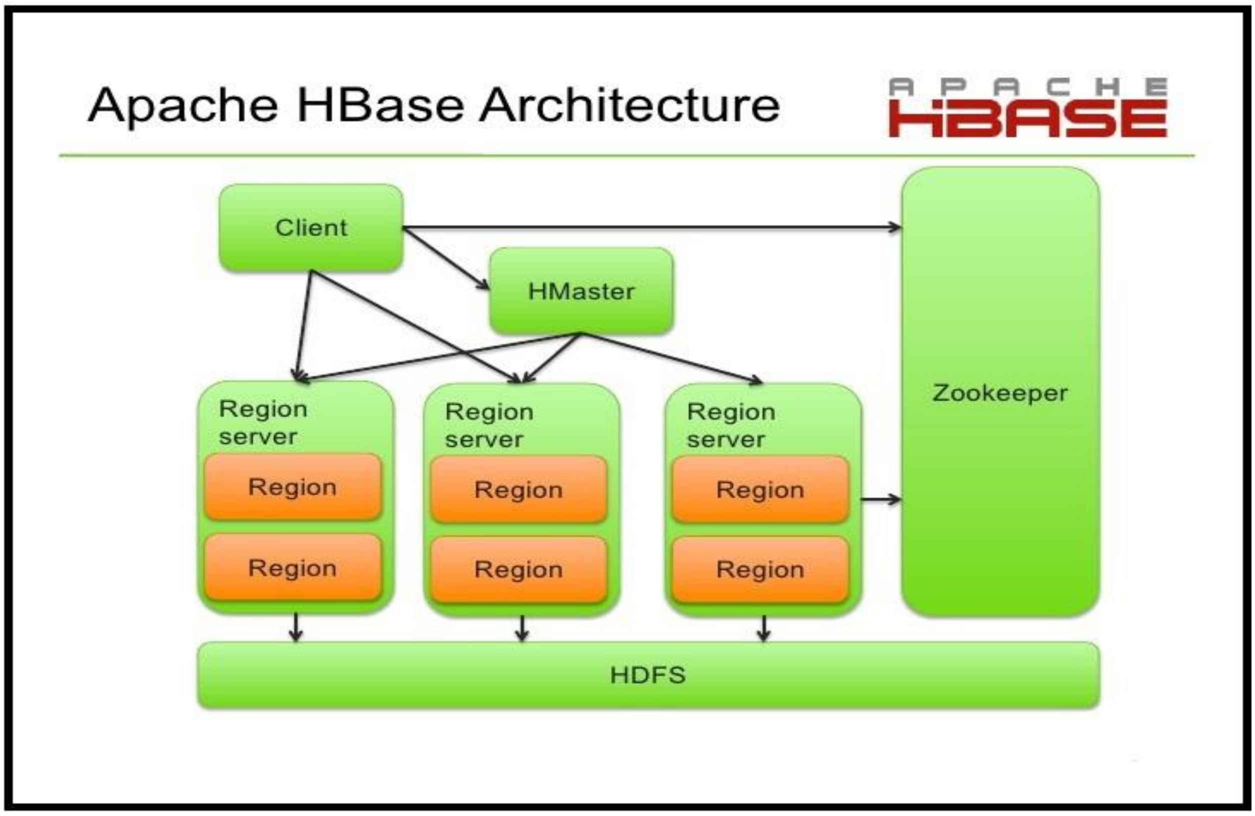
HRegions Servers: It will perform the following functions in communication with HMaster and Zookeeper.

- Hosting and managing regions.
- Splitting regions automatically.
- Handling read and writes requests.
- Communicating with clients directly.

HRegions: For each column family, HRegions maintain a store. Main components of HRegions are

- Memstore - Holds in-memory modifications to the store
- Hfile

BASIC ARCHITECTURE OF HBASE



● <u>HBase vs RDBMS</u>

HBASE	RDBMS
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.

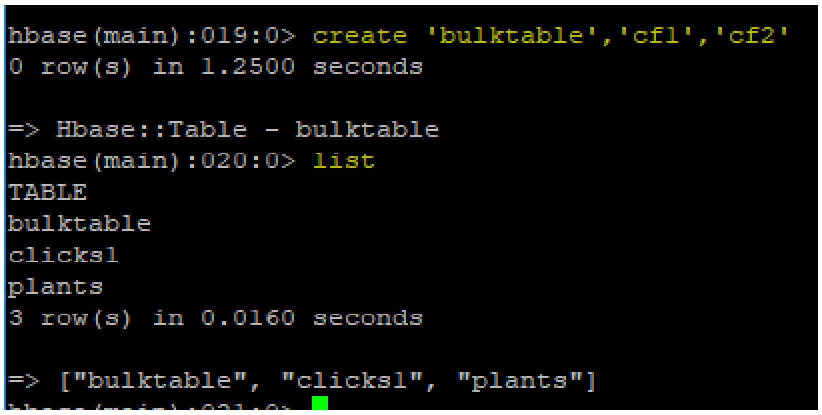
Well suited for OLAP systems.	Well suited for OLTP systems.
Read only relevant data from database.	To retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required.
Structured and semi structure data can be stored and processed using Hbase.	Structured data can be stored and processed using an RDBMS.
Enables aggregation over many rows and columns.	Aggregation is an expensive operation.

Big Data- Hadoop – Assignment 4

Task 2:

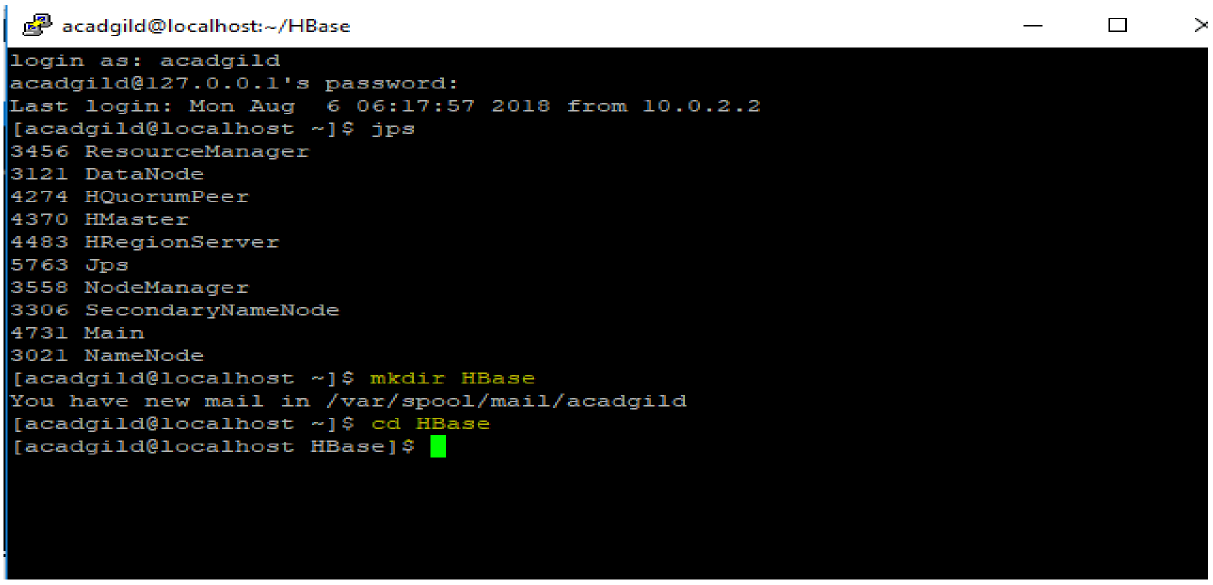
Execute blog present in below link

Step 1: To create table along with 2 column family.
Create 'bulktable', 'cf1', 'cf2'



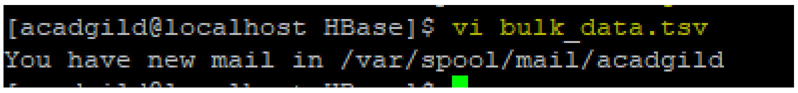
```
hbase(main):019:0> create 'bulktable','cf1','cf2'
0 row(s) in 1.2500 seconds

=> Hbase::Table - bulktable
hbase(main):020:0> list
TABLE
bulktable
clicks1
plants
3 row(s) in 0.0160 seconds

=> ["bulktable", "clicks1", "plants"]
```

Step 2: Make a directory for Hbase in the local drive

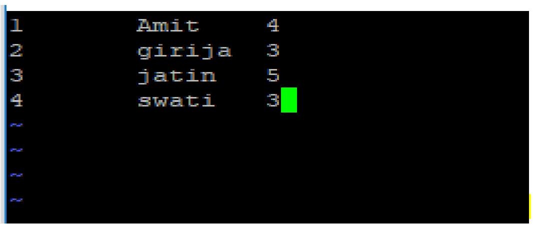


```
acadgild@localhost:~/HBase
login as: acadgild
acadgild@127.0.0.1's password:
Last login: Mon Aug  6 06:17:57 2018 from 10.0.2.2
[acadgild@localhost ~]$ jps
3456 ResourceManager
3121 DataNode
4274 HQuorumPeer
4370 HMaster
4483 HRegionServer
5763 Jps
3558 NodeManager
3306 SecondaryNameNode
4731 Main
3021 NameNode
[acadgild@localhost ~]$ mkdir HBase
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ cd HBase
[acadgild@localhost HBase]$
```

Step 3: Create a file inside the HBase directory named bulk_data.tsv with tab separated data inside using below command in terminal.



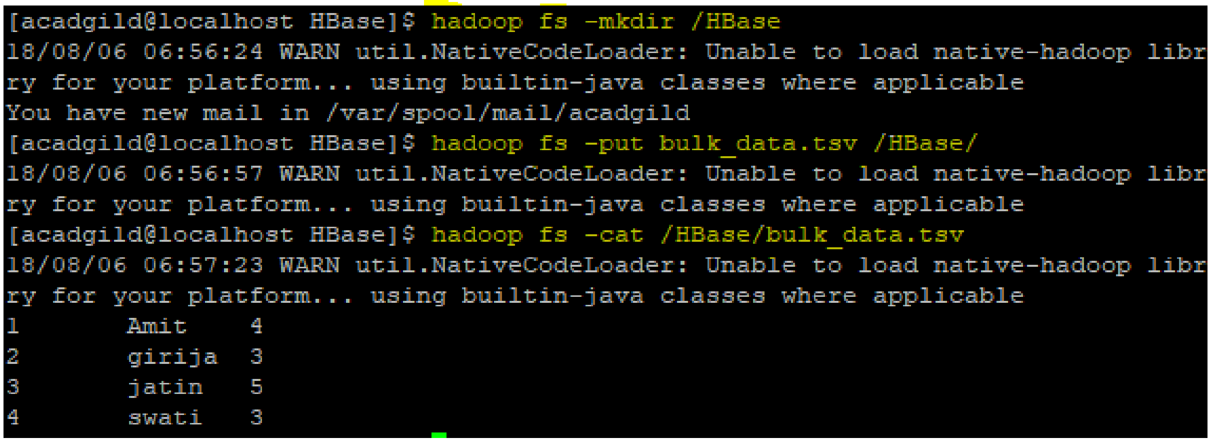
```
[acadgild@localhost HBase]$ vi bulk_data.tsv
You have new mail in /var/spool/mail/acadgild
```



```
1       Amit    4
2       girija  3
3       jatin   5
4       swati   3
~
~
~
~
```

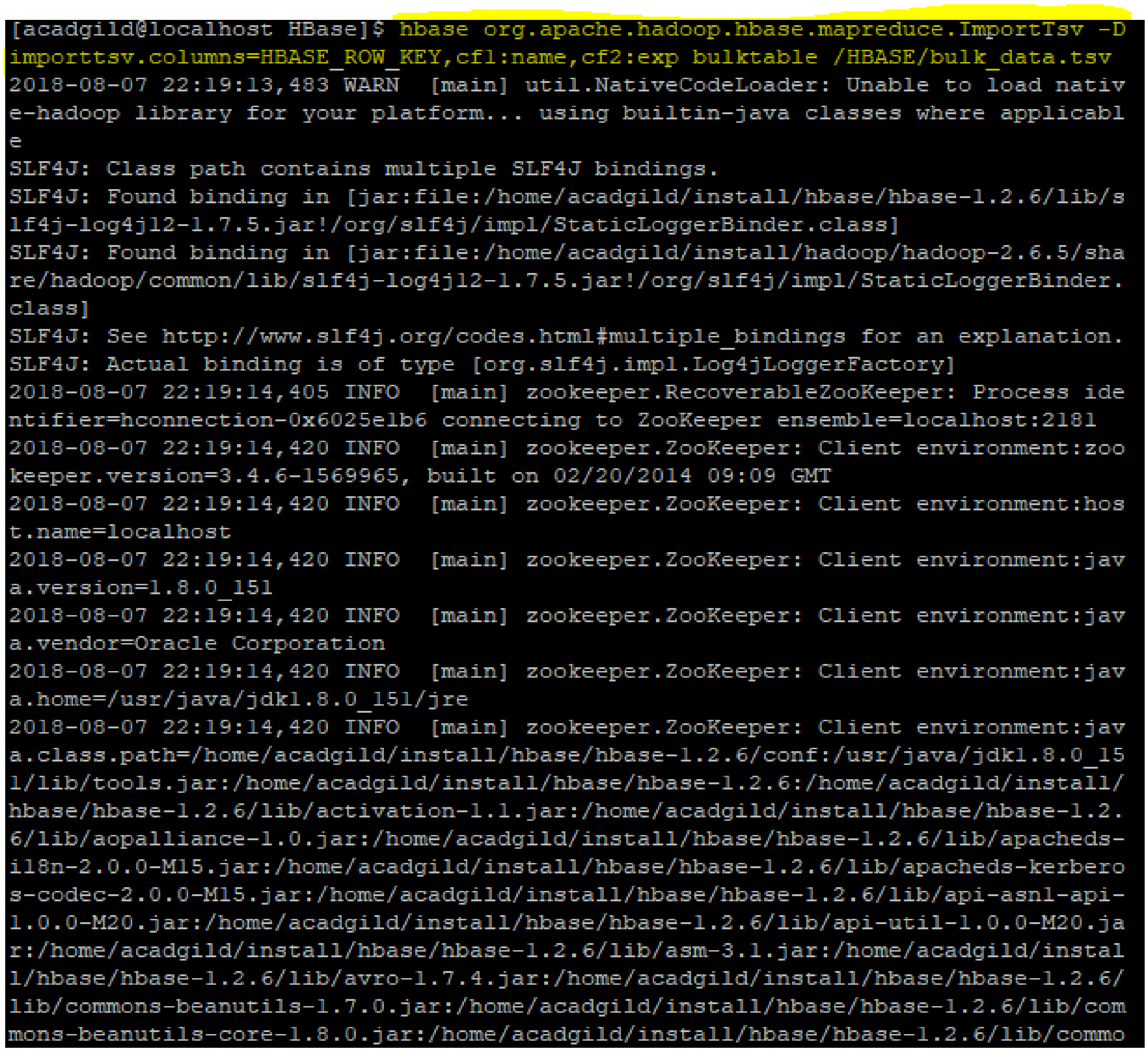
Big Data- Hadoop – Assignment 4

Step 4: Copying the data inside HDFS

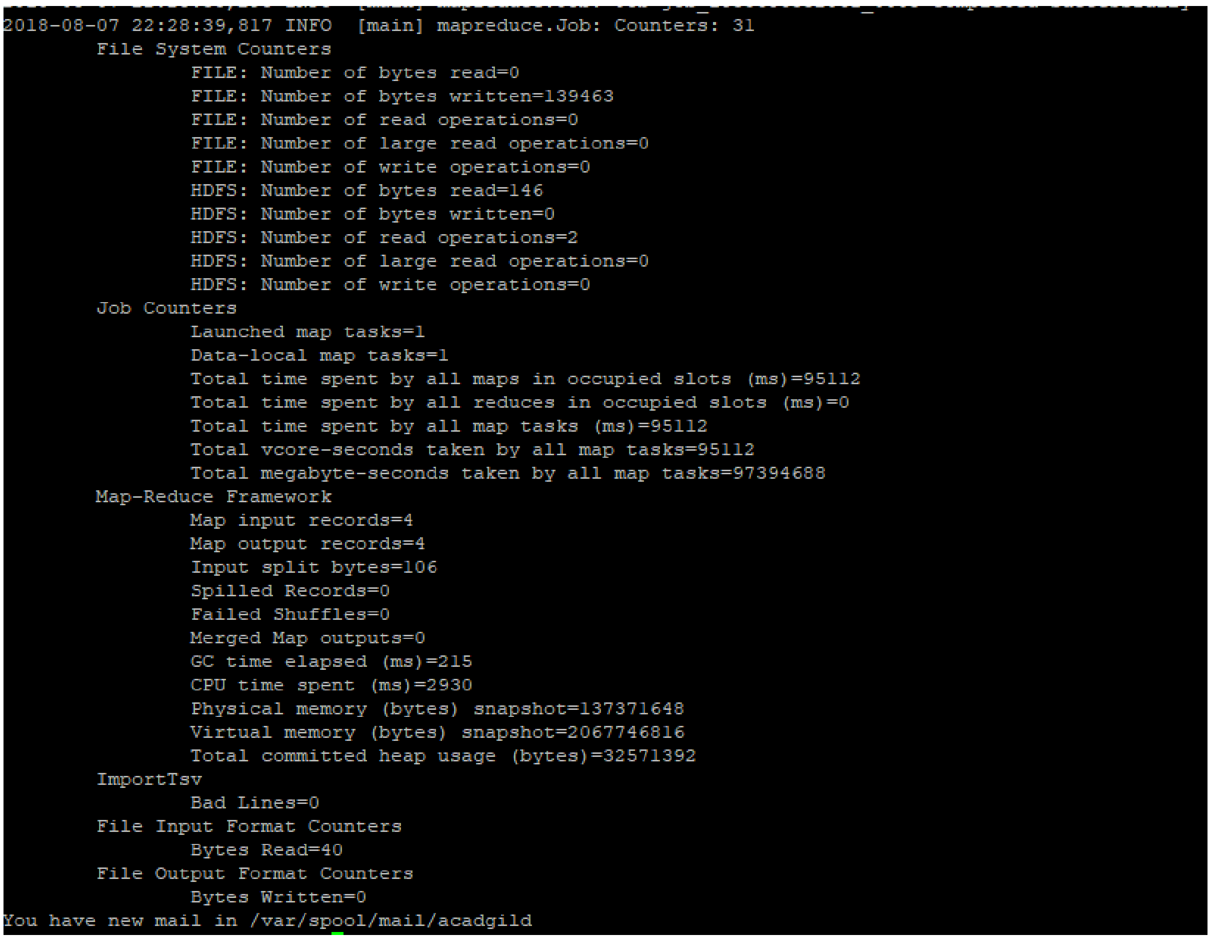


```
[acadgild@localhost HBase]$ hadoop fs -mkdir /HBase
18/08/06 06:56:24 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ry for your platform... using builtin-java classes where applicable
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost HBase]$ hadoop fs -put bulk_data.tsv /HBase/
18/08/06 06:56:57 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ry for your platform... using builtin-java classes where applicable
[acadgild@localhost HBase]$ hadoop fs -cat /HBase/bulk_data.tsv
18/08/06 06:57:23 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ry for your platform... using builtin-java classes where applicable
1       Amit    4
2       girija  3
3       jatin   5
4       swati   3
```

Step 5: hbase org.apache.hadoop.hbase.mapreduce.ImportTsv –Dimporttsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /hbase/bulk_data.tsv the command which takes data from HDFS and loads into HBase.

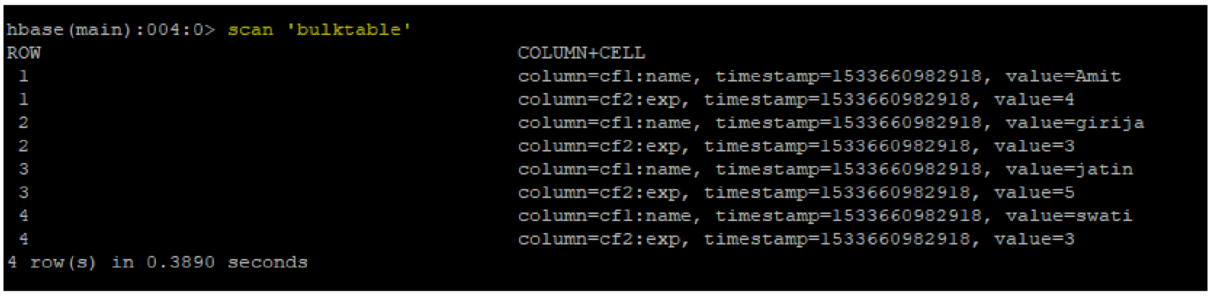


```
[acadgild@localhost HBase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -D
importtsv.columns=HBASE_ROW_KEY,cf1:name,cf2:exp bulktable /HBASE/bulk_data.tsv
2018-08-07 22:19:13,483 WARN  [main] util.NativeCodeLoader: Unable to load nativ
e-hadoop library for your platform... using builtin-java classes where applicabl
e
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/s
lf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/sha
re/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]
2018-08-07 22:19:14,405 INFO  [main] zookeeper.RecoverableZooKeeper: Process ide
ntifier=hconnection-0x6025e1b6 connecting to ZooKeeper ensemble=localhost:2181
2018-08-07 22:19:14,420 INFO  [main] zookeeper.ZooKeeper: Client environment:zoo
keeper.version=3.4.6-1569965, built on 02/20/2014 09:09 GMT
2018-08-07 22:19:14,420 INFO  [main] zookeeper.ZooKeeper: Client environment:hos
t.name=localhost
2018-08-07 22:19:14,420 INFO  [main] zookeeper.ZooKeeper: Client environment:jav
a.version=1.8.0_151
2018-08-07 22:19:14,420 INFO  [main] zookeeper.ZooKeeper: Client environment:jav
a.vendor=Oracle Corporation
2018-08-07 22:19:14,420 INFO  [main] zookeeper.ZooKeeper: Client environment:jav
a.home=/usr/java/jdk1.8.0_151/jre
2018-08-07 22:19:14,420 INFO  [main] zookeeper.ZooKeeper: Client environment:jav
a.class.path=/home/acadgild/install/hbase/hbase-1.2.6/conf:/usr/java/jdk1.8.0_15
1/lib/tools.jar:/home/acadgild/install/hbase/hbase-1.2.6:/home/acadgild/install/
hbase/hbase-1.2.6/lib/activation-1.1.jar:/home/acadgild/install/hbase/hbase-1.2.
6/lib/aopalliance-1.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-
i18n-2.0.0-M15.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/apacheds-kerbero
s-codec-2.0.0-M15.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/api-asn1-api-
1.0.0-M20.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/api-util-1.0.0-M20.ja
r:/home/acadgild/install/hbase/hbase-1.2.6/lib/asm-3.1.jar:/home/acadgild/instal
l/hbase/hbase-1.2.6/lib/avro-1.7.4.jar:/home/acadgild/install/hbase/hbase-1.2.6/
lib/commons-beanutils-1.7.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/com
mons-beanutils-core-1.8.0.jar:/home/acadgild/install/hbase/hbase-1.2.6/lib/commo
```



```
2018-08-07 22:28:39,817 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)=95112
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=95112
                Total vcore-seconds taken by all map tasks=95112
                Total megabyte-seconds taken by all map tasks=97394688
        Map-Reduce Framework
                Map input records=4
                Map output records=4
                Input split bytes=106
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=215
                CPU time spent (ms)=2930
                Physical memory (bytes) snapshot=137371648
                Virtual memory (bytes) snapshot=2067746816
                Total committed heap usage (bytes)=32571392
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=40
        File Output Format Counters
                Bytes Written=0
You have new mail in /var/spool/mail/acadgild
```

OUTPUT: check whether we actually got the data inside HBase



```
hbase(main):004:0> scan 'bulktable'
ROW                                        COLUMN+CELL
 1                                         column=cf1:name, timestamp=1533660982918, value=Amit
 1                                         column=cf2:exp, timestamp=1533660982918, value=4
 2                                         column=cf1:name, timestamp=1533660982918, value=girija
 2                                         column=cf2:exp, timestamp=1533660982918, value=3
 3                                         column=cf1:name, timestamp=1533660982918, value=jatin
 3                                         column=cf2:exp, timestamp=1533660982918, value=5
 4                                         column=cf1:name, timestamp=1533660982918, value=swati
 4                                         column=cf2:exp, timestamp=1533660982918, value=3
4 row(s) in 0.3890 seconds
```