Assignment 1

1. What is NoSQL database?

NoSQL means ‘not SQL’ or ‘not only SQL’. Relational database systems which use SQL are implemented on concepts such as tables or schemas, rows and columns in an organized manner. The problem which such system is that it fails to offer performance, scalability as well as flexibility when input data has no proper structure and size of data is enormous.
NoSQL databases are introduced to overcome these problems. They are specifically designed for ingesting unstructured data and retrieving the same in faster manner than relational database systems.
Types of NoSQL databases:

- Wide-column stores: These type of databases group columns of related data together. They classify data of tables as columns rather than rows which is the case in RDBMS. Wide-column stores are mainly designed query and retrieve data from large volume of data quicker than traditional database systems. Few examples of such databases are Google’s BigTable, HBase and Cassandra.

- Key-value stores: These kind of stores offer a simple and easy to use data model that associates a distinct key with a value. Since it is quite simple, it gives high performance and scalability when data size grows larger. Redis, Berkeley DB, Aerospike, MemcacheDB and Riak are few examples of key-value databases.

- Document databases: Document databases usually store semi-structured data such as JSON (Java Script Object Notation), XML (eXtended Markup Language) and BSON (Binary JSON). Document stores are typically used in content management and mobile application data handling. Examples include MongoDB, DocumentDB, CouchDB and MarkLogic.

- Graph databases: Graph databases organize data as nodes, similar to rows in relational databases, and edges represents connections between nodes. Graph databases can evolve over time and usage and users can visualize data in more easier and interactive manner. Some examples of such databases are Neo4j, Titan and IBM Graph.

2. How does data get stored in NoSQL database?

There are many kinds of NoSQL databases such as key-value stores, document stores, wide-column stores and graph databases. Each of them uses different method of storing data. Let's explore one by one with a simple example.

- Key-value stores: As the name implies, key-value databases store values in a pair consisting of a key and its corresponding value. Here, each key of a specific table or entity should be unique so as to distinguish between all the other keys that exist in that table; values can be repetitive or distinct based on user data.

  Example:
  Cricket Match Score Card:
  Key Value

  R Sharma 76
  KL Rahul 38
  Virat Kohli 84
  M S Dhoni 24
  H Pandya 35

- Document databases: Document stores contain data in the form of JSON, XML or BSON, etc. Let's take an example of MongoDB which stores data in the form of JSON.
  Students Test Scores:

```
{
    "name": "Reena",
    "class": 09,
    "subjects": ["English", "Science", "Mathematics"],
    "marks": [80, 85, 90]
}
```

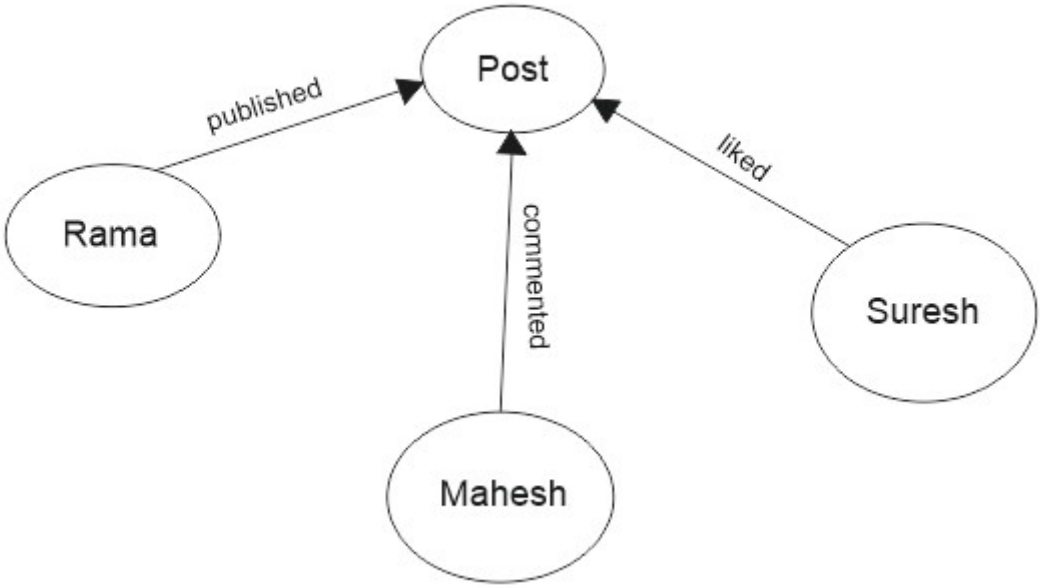
  In the above example, the fields named 'subjects' and 'marks' hold a collection of values of the same type.

- Wide-column stores: These type databases are vertically scalable i.e. they define data in terms of column families. A column is a basic unit of wide-column database and a column family is a collection of columns and we can define any number of column families on a table. Each row is distinguished by a unique field termed as row key or row id. Let's see an example:

ROW ID	WEBSITE		
1	Protocol	Domain	Subdomain
	https://	google.com	www

- Graph databases: In this type data stores, all data of a specific row will be represented as a single node and this node can be connected to another by specifying a relationship between them which is termed as edge. This is mostly useful in depicting social network related data using databases such as Ne04j or Titan. Let's see an example:

3.



What is a column family in HBase?

A column is the fundamental unit of HBase, which is a collection of key value pairs. A column family is a grouping of data in a row. Each column family contains one or more columns and these columns are saved in a low level storage file called HFile. A column family appears as a table. However, it has a larger scope than just a table. The general use case of this concept is to store data that belongs to different departments of an organization.

Here are the two relations which we typically use in relational database systems:

Table:Employee

Employee ID	Employee Name	Employee City	Employee Sex	Department ID
1	Suresh	Bengaluru	M	4
2	Prateek	Pune	M	2
3	Prerana	Chennai	F	3
4	Sathwik	Cochin	M	1

Table:Department

Department ID	Department Name	Department Description
1	HR	HR Function
2	IT	Development
3	Payroll	Payment

To retrieve data from employees data from both tables, an SQL query need to look up from both and it returns results quickly for such small amount of data. If these tables have around millions to billions of rows, it will take lot of time. This problem can be overcome by introducing column families to group all the data under one table as shown below:

CF1: Employee					CF2: Department		
Row key	Emp id	Emp Name	Emp City	Emp sex	Dept id	Dept name	Dept Desc
1	151	Suresh	Bengaluru	M	4	HR	HR Function
2	152	Prateek	Pune	M	2	IT	Development
3	153	Prerana	Chennai	F	3	Payroll	Payment
4	154	Sathwik	Cochin	M	1	HR	HR Function

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

There is no limit on the number of columns in an HBase table. However, we need to consider few things:

- Row level locking: when a user performs an operation on a row, it will be locked by the region server until it applies the mutation. An advantage of such behavior is that user can atomically operate on several columns. Many users in parallel can see or perform a complete update on a row or won't be able to make an update at all. There is no partial update as such. A drawback of this approach is that the throughput of write operation is limited within a single row.
- Data distribution across regions: The basic unit of data distribution and load balancing in HBase is a region. However, a row will never be split across regions. A row will always be served by a single server. If you still want to split data across several rows, you can force two rows to get split and distributed between two hosts.
- Known bugs: In earlier versions of HBase there were some bugs which caused an entire row to get loaded or deserialized into RAM. Hence, if a row is too large (more than 1GB), it may cause serious performance issues or out of memory errors, etc. Even though the Region Server loads only necessary columns, this is something users must be aware of when scaling up a row vertically.

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

An essential property that needs to be defined at the time of table creation is the column family. A column family can contain one or more columns. Since we are dealing with schema-less database, it doesn't really force the user to define the column structure while creating a table. Columns can be added dynamically at the time data insertion based on the dataset available. HBase is designed to optimize storing of columns altogether on disk, offering more efficient storage since columns that are not present will not up any space unlike relational databases where null values will get stored.

All column family members are stored together physically on the file system. Since storage specifications and performance tunings are done at the column family level, only the information regarding column families is necessary to create table.

The main reason why column families are part of a table's schema is that they impact the way data is stored, both on disk and in memory. Every column family has its own group of HFiles and its own set of data structures in memory of RegionServer. Data for each column will get stored on these files and user can change the number of columns he/she can add on insertion of data for every row. Since the column structure gets changed more often than not, the columns are not added at the time of table creation in Hbase.

6. How does data get managed in Hbase?

The lack of schema enforcement in HBase offers relaxation on the requirements of consistency enforced in relational databases. HBase provides a strong read consistency model that performs exceptionally well while scaling read operations but it does not scale on writes.

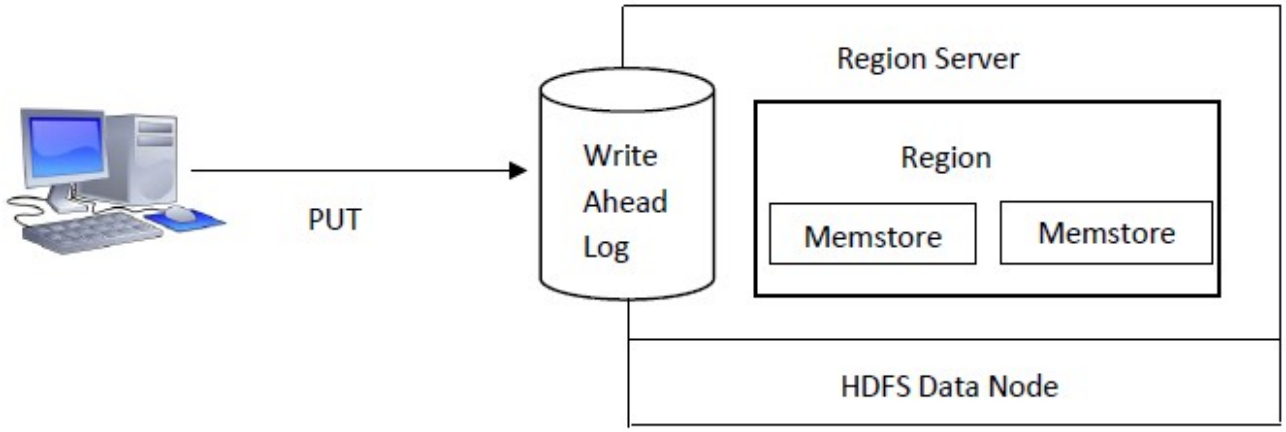
The main features that make HBase an excellent data management platform are usability, speed and fault tolerance. Usability feature is offered by the data model that is flexible enough to allow many uses with a simple Java API. Speed is provided by the near real-time lookups server side processing and in memory caching. Fault tolerance is offered by strong read consistency in row level operations and replication, automatic fail-over and load balanced and automatically sharded tables.

HBase can be installed in stand alone mode on local file system. But this environment does not guarantee durability. Edit logs will get lost in case daemons are not started and stopped properly. As an alternative, HBase can also be run on a single node or multi-node cluster and can make use of HDFS.

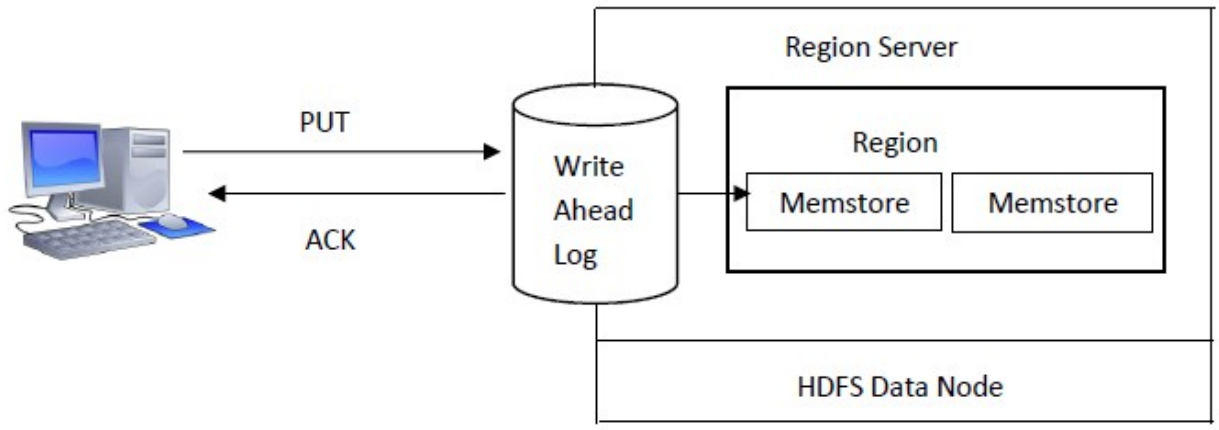
HBase has important feature of defining a column family and then adding as many columns as required within that column family, recognized by the column qualifier. HBase is designed to optimize storing of columns altogether on disk, offering more efficient storage since columns that are not present will not up any space unlike relational databases where null values will get stored.

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

Data insertion into HBase are explained in steps below:
Step 1: When a client submits a Put request, the data will be written to Write Ahead Log (WAL). Edits are added to the end of WAL file which will get stored on disk. The WAL is used to restore data that is not persisted in case of server crashes. Each incoming record is written to WAL file for durability. It stores the log information of Put requests and the data updates are appended sequentially.



Step 2: Once the data is written to the WAL, it is placed in the Memstore. Finally, an acknowledgment is sent from put request back to the client.



HBase MemStore: The MemStore stores updates in memory as sorted key values, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.
HBase HFile: Data is stored in an HFile which contains sorted key/values. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.

Task 2:

Let's create a table in HBase named 'clicks' with column family 'hits'. We can make use of 'VERSIONS' option for it to support retrieval of last 5 versions of data related 'hits' column family. Here is the command to do the same on HBase shell:
hbase> create 'clicks', {NAME=>'hits', VERSIONS=>5}

```
hbase(main):003:0> create 'clicks', {NAME=>'hits', VERSIONS=>5}
0 row(s) in 1.0630 seconds

=> Hbase::Table - clicks
hbase(main):004:0> list
TABLE
clicks
1 row(s) in 0.0280 seconds

=> ["clicks"]
hbase(main):005:0> scan 'clicks'
ROW                             COLUMN+CELL
0 row(s) in 0.1380 seconds

hbase(main):006:0>
```

Step 1: Let's insert few rows into this table using 'put' command as shown below:
hbase> put 'clicks','192.168.124.1','hits:hits_count','20'
hbase> put 'clicks','192.168.124.1','hits:website','www.acadgild.com'
hbase> put 'clicks','192.168.124.2','hits:hits_count','30'
hbase> put 'clicks','192.168.124.2','hits:website','www.google.com'
hbase> put 'clicks','192.168.124.3','hits:hits_count','25'
hbase> put 'clicks','192.168.124.3','hits:website','www.facebook.com'
hbase> put 'clicks','192.168.124.4','hits:hits_count','40'
hbase> put 'clicks','192.168.124.4','hits:website','www.acadgild.com'

In the above commands, I have added two columns such as 'hits_count' and 'website' for column family 'hits'. An IP address is used as row key as given in the problem statement. Now we can see 4 rows getting inserted into 'clicks' table by executing scan command:
hbase> scan 'clicks'

```
hbase(main):026:0> put 'clicks','192.168.124.1','hits:hits_count','20'
0 row(s) in 0.0340 seconds

hbase(main):027:0> put 'clicks','192.168.124.1','hits:website','www.acadgild.com'
0 row(s) in 0.0160 seconds

hbase(main):028:0>
hbase(main):029:0* put 'clicks','192.168.124.2','hits:hits_count','30'
0 row(s) in 0.0140 seconds

hbase(main):030:0> put 'clicks','192.168.124.2','hits:website','www.google.com'
0 row(s) in 0.0160 seconds

hbase(main):031:0>
hbase(main):032:0* put 'clicks','192.168.124.3','hits:hits_count','25'
0 row(s) in 0.0100 seconds

hbase(main):033:0> put 'clicks','192.168.124.3','hits:website','www.facebook.com'
0 row(s) in 0.0210 seconds

hbase(main):034:0>
hbase(main):035:0* put 'clicks','192.168.124.4','hits:hits_count','40'
0 row(s) in 0.0130 seconds

hbase(main):036:0> put 'clicks','192.168.124.4','hits:website','www.acadgild.com'
0 row(s) in 0.0180 seconds

hbase(main):037:0> scan 'clicks'
ROW                             COLUMN+CELL
 192.168.124.1                  column=hits:hits_count, timestamp=1514265531592, value=20
 192.168.124.1                  column=hits:website, timestamp=1514265531659, value=www.acadgild.com
 192.168.124.2                  column=hits:hits_count, timestamp=1514265531750, value=30
 192.168.124.2                  column=hits:website, timestamp=1514265531808, value=www.google.com
 192.168.124.3                  column=hits:hits_count, timestamp=1514265531882, value=25
 192.168.124.3                  column=hits:website, timestamp=1514265531958, value=www.facebook.com
 192.168.124.4                  column=hits:hits_count, timestamp=1514265532091, value=40
 192.168.124.4                  column=hits:website, timestamp=1514265534298, value=www.acadgild.com
4 row(s) in 0.1210 seconds
```

Step 2: Update few records and see the changes by scanning the whole table.
Let's try to update the record with row key: '192.168.124.3'.
hbase> put 'clicks','192.168.124.3','hits:hits_count','20'
hbase> put 'clicks','192.168.124.3','hits:website','www.linkedin.com'
hbase> put 'clicks','192.168.124.3','hits:hits_count','25'
hbase> put 'clicks','192.168.124.3','hits:website','www.acadgild.com'
hbase> put 'clicks','192.168.124.3','hits:hits_count','40'
hbase> put 'clicks','192.168.124.3','hits:website','www.google.com' hbase> put 'clicks','192.168.124.3','hits:hits_count','30'
hbase> put 'clicks','192.168.124.3','hits:website','www.cloudera.com'
We can notice from these commands that the record with row key '192.168.124.3' has got updated four times and now it has five versions in total including the one which was been inserted in the beginning. Let's scan the table to see which value this record holds:
hbase> scan 'clicks'

```
hbase(main):038:0> put 'clicks','192.168.124.3','hits:hits_count','20'
0 row(s) in 0.0290 seconds

hbase(main):039:0> put 'clicks','192.168.124.3','hits:website','www.linkedin.com'
0 row(s) in 0.0150 seconds

hbase(main):040:0> put 'clicks','192.168.124.3','hits:hits_count','25'
0 row(s) in 0.0070 seconds

hbase(main):041:0> put 'clicks','192.168.124.3','hits:website','www.acadgild.com'
0 row(s) in 0.0080 seconds

hbase(main):042:0> put 'clicks','192.168.124.3','hits:hits_count','40'
0 row(s) in 0.0100 seconds

hbase(main):043:0> put 'clicks','192.168.124.3','hits:website','www.google.com'
0 row(s) in 0.0070 seconds

hbase(main):044:0> put 'clicks','192.168.124.3','hits:hits_count','30'
0 row(s) in 0.0040 seconds

hbase(main):045:0> put 'clicks','192.168.124.3','hits:website','www.cloudera.com'
0 row(s) in 0.0160 seconds

hbase(main):046:0> scan 'clicks'
ROW                           COLUMN+CELL
 192.168.124.1                column=hits:hits_count, timestamp=1514265531592, value=20
 192.168.124.1                column=hits:website, timestamp=1514265531659, value=www.acadgild.com
 192.168.124.2                column=hits:hits_count, timestamp=1514265531750, value=30
 192.168.124.2                column=hits:website, timestamp=1514265531808, value=www.google.com
 192.168.124.3                column=hits:hits_count, timestamp=1514266344956, value=30
 192.168.124.3                column=hits:website, timestamp=1514266350313, value=www.cloudera.com
 192.168.124.4                column=hits:hits_count, timestamp=1514265532091, value=40
 192.168.124.4                column=hits:website, timestamp=1514265534298, value=www.acadgild.com
4 row(s) in 0.0980 seconds

hbase(main):047:0>
```

As we can see from the screenshot above, the record holds only the latest updated value. Now if we want to see all five versions data for this column family, we can get it by following command:

```
hbase(main):047:0> scan 'clicks',{COLUMN=>'hits',VERSIONS=>5}
ROW                           COLUMN+CELL
 192.168.124.1                column=hits:hits_count, timestamp=1514265531592, value=20
 192.168.124.1                column=hits:website, timestamp=1514265531659, value=www.acadgild.com
 192.168.124.2                column=hits:hits_count, timestamp=1514265531750, value=30
 192.168.124.2                column=hits:website, timestamp=1514265531808, value=www.google.com
 192.168.124.3                column=hits:hits_count, timestamp=1514266344956, value=30
 192.168.124.3                column=hits:hits_count, timestamp=1514266344843, value=40
 192.168.124.3                column=hits:hits_count, timestamp=1514266344735, value=25
 192.168.124.3                column=hits:hits_count, timestamp=1514266344641, value=20
 192.168.124.3                column=hits:hits_count, timestamp=1514265531882, value=25
 192.168.124.3                column=hits:website, timestamp=1514266350313, value=www.cloudera.com
 192.168.124.3                column=hits:website, timestamp=1514266344910, value=www.google.com
 192.168.124.3                column=hits:website, timestamp=1514266344779, value=www.acadgild.com
 192.168.124.3                column=hits:website, timestamp=1514266344702, value=www.linkedin.com
 192.168.124.3                column=hits:website, timestamp=1514265531958, value=www.facebook.com
 192.168.124.4                column=hits:hits_count, timestamp=1514265532091, value=40
 192.168.124.4                column=hits:website, timestamp=1514265534298, value=www.acadgild.com
4 row(s) in 0.1450 seconds
```

hbase> scan 'clicks',{COLUMN=>'hits',VERSIONS=>5} Now let's try to update the same record again and then fetch all versions of data from the same column family:
hbase> put 'clicks','192.168.124.3','hits:hits_count','40'
hbase> put 'clicks','192.168.124.3','hits:website','www.acadgild.com'
hbase> scan 'clicks'

```
hbase(main):048:0> put 'clicks','192.168.124.3','hits:hits_count','40'
0 row(s) in 0.0200 seconds

hbase(main):049:0> put 'clicks','192.168.124.3','hits:website','www.acadgild.com'
0 row(s) in 0.0320 seconds

hbase(main):050:0> scan 'clicks'
ROW                             COLUMN+CELL
 192.168.124.1                  column=hits:hits_count, timestamp=1514265531592, value=20
 192.168.124.1                  column=hits:website, timestamp=1514265531659, value=www.acadgild.com
 192.168.124.2                  column=hits:hits_count, timestamp=1514265531750, value=30
 192.168.124.2                  column=hits:website, timestamp=1514265531808, value=www.google.com
 192.168.124.3                  column=hits:hits_count, timestamp=1514266647764, value=40
 192.168.124.3                  column=hits:website, timestamp=1514266649192, value=www.acadgild.com
 192.168.124.4                  column=hits:hits_count, timestamp=1514265532091, value=40
 192.168.124.4                  column=hits:website, timestamp=1514265534298, value=www.acadgild.com
4 row(s) in 0.1090 seconds
```

The record with row key '192.168.124.3' has been updated with values for columns from latest put commands. Let's try to get all six versions for this column family:
hbase> scan 'clicks',{COLUMN=>'hits',VERSIONS=>6}

```
hbase(main):051:0> scan 'clicks',{COLUMN=>'hits',VERSIONS=>6}
ROW                             COLUMN+CELL
 192.168.124.1                  column=hits:hits_count, timestamp=1514265531592, value=20
 192.168.124.1                  column=hits:website, timestamp=1514265531659, value=www.acadgild.com
 192.168.124.2                  column=hits:hits_count, timestamp=1514265531750, value=30
 192.168.124.2                  column=hits:website, timestamp=1514265531808, value=www.google.com
 192.168.124.3                  column=hits:hits_count, timestamp=1514266647764, value=40
 192.168.124.3                  column=hits:hits_count, timestamp=1514266344956, value=30
 192.168.124.3                  column=hits:hits_count, timestamp=1514266344843, value=40
 192.168.124.3                  column=hits:hits_count, timestamp=1514266344735, value=25
 192.168.124.3                  column=hits:hits_count, timestamp=1514266344641, value=20
 192.168.124.3                  column=hits:website, timestamp=1514266649192, value=www.acadgild.com
 192.168.124.3                  column=hits:website, timestamp=1514266350313, value=www.cloudera.com
 192.168.124.3                  column=hits:website, timestamp=1514266344910, value=www.google.com
 192.168.124.3                  column=hits:website, timestamp=1514266344779, value=www.acadgild.com
 192.168.124.3                  column=hits:website, timestamp=1514266344702, value=www.linkedin.com
 192.168.124.4                  column=hits:hits_count, timestamp=1514265532091, value=40
 192.168.124.4                  column=hits:website, timestamp=1514265534298, value=www.acadgild.com
4 row(s) in 0.0860 seconds
```

As we can see in the above screenshot, it displays only the last five data versions for columns of given column family and I have lost the initial version of it. Because I had mentioned the maximum number of versions this column family can hold as five while creating the table.