Name – Pushpa Yadav

Roll No - 46

Experiment 11 - Schema design using HBase

What is HBase?

HBase is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (HDFS). HBase provides a fault-tolerant way of storing sparse data sets, which are common in many big data use cases. It is well suited for real-time data processing or random read/write access to large volumes of data.

Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java™ much like a typical Apache MapReduce application. HBase does support writing applications in Apache Avro, REST and Thrift.

An HBase system is designed to scale linearly. It comprises a set of standard tables with rows and columns, much like a traditional database. Each table must have an element defined as a primary key, and all access attempts to HBase tables must use this primary key.

Avro, as a component, supports a rich set of primitive data types including: numeric, binary data and strings; and a number of complex types including arrays, maps, enumerations and records. A sort order can also be defined for the data.

HBase relies on ZooKeeper for high-performance coordination. ZooKeeper is built into HBase, but if you're running a production cluster, it's suggested that you have a dedicated ZooKeeper cluster that's integrated with your HBase cluster.

HBase works well with Hive, a query engine for batch processing of big data, to enable fault-tolerant big data applications.

An example of HBase

An HBase column represents an attribute of an object; if the table is storing diagnostic logs from servers in your environment, each row might be a log record, and a typical column could be the timestamp of when the log record was written, or the server name where the record originated.

HBase allows for many attributes to be grouped together into column families, such that the elements of a column family are all stored together. This is different from a row-oriented relational database, where all the columns of a given row are stored together. With HBase you must predefine the table schema and specify the column families. However, new columns can be added to families at any time, making the schema flexible and able to adapt to changing application requirements.

Just as HDFS has a NameNode and slave nodes, and MapReduce has JobTracker and TaskTracker slaves, HBase is built on similar concepts. In HBase a master node manages the cluster and region servers store portions of the tables and perform the work on the data. In the same way HDFS has some enterprise concerns due to the availability of the NameNode HBase is also sensitive to the loss of its master node.

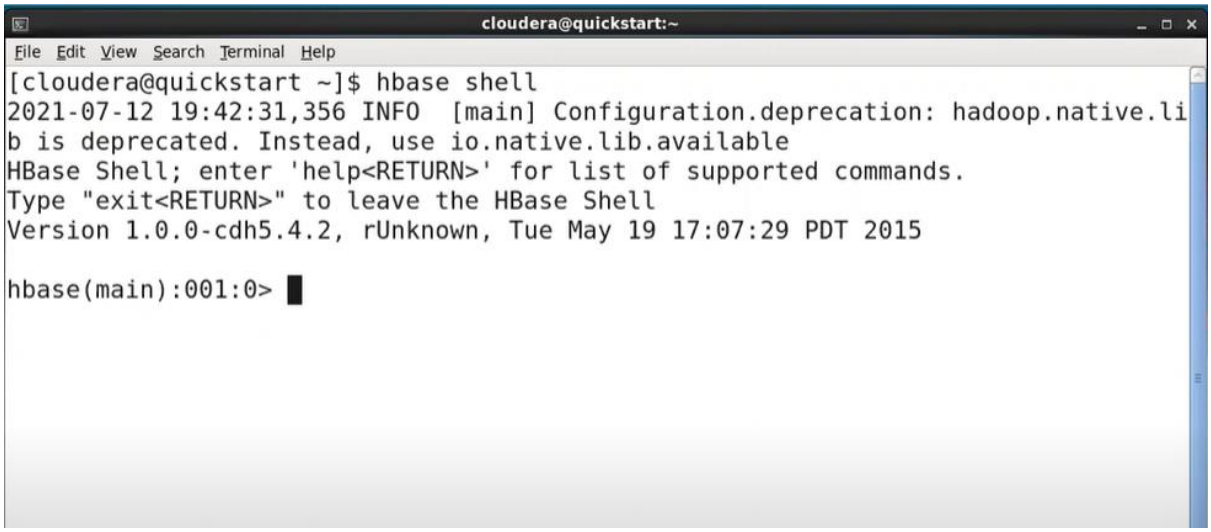
HBase Shell

HBase contains a shell using which you can communicate with HBase. HBase uses the Hadoop File System to store its data. It will have a master server and region servers. The data storage will be in the form of regions (tables). These regions will be split up and stored in region servers.

The master server manages these region servers and all these tasks take place on HDFS. Given below are some of the commands supported by HBase Shell.

We can start the HBase interactive shell using “hbase shell” command as shown below.

$ hbase shell



Check the shell functioning before proceeding further. Use the list command for this purpose. List is a command used to get the list of all the tables in HBase. It lists all the tables in HBase.

hbase(main):001:0> list

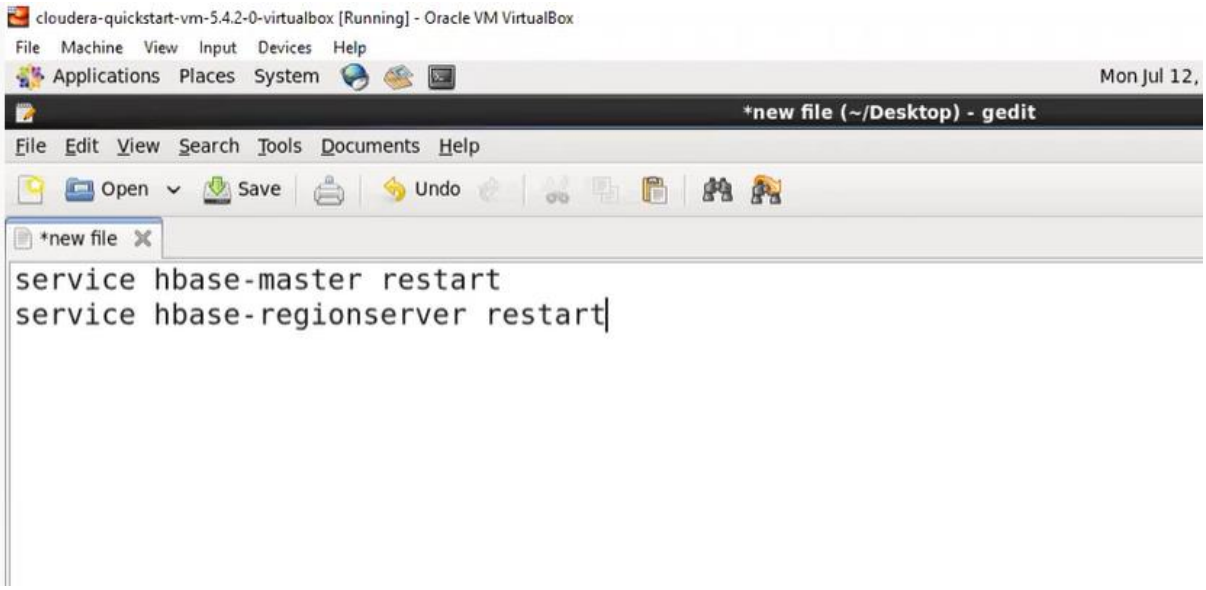
```
[cloudera@quickstart ~]$ hbase shell
2021-07-12 19:42:31,356 INFO  [main] Configuration.deprecation: hadoop.native.li
b is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.0.0-cdh5.4.2, rUnknown, Tue May 19 17:07:29 PDT 2015

hbase(main):001:0> list
TABLE
demo
student
2 row(s) in 0.2440 seconds

=> ["demo", "student"]
hbase(main):002:0>
```

Restart HBase services if this is not running on terminal

$ sudo su – This commands is to become super user

$ service hbase-master restart – This commands is to restart hbase-master services

$ service hbase-regionserver restart – This commands is to restart hbase-regionserver services



Once these commands run successfully then Open the browser and refresh the page and see all the HBase serveries will be restarted.

Schema Design using HBase

The HBase schema design is very different compared to the relation database schema design. Below are some of general concept that should be followed while designing schema in Hbase:

Row key: Each table in HBase table is indexed on row key. Data is sorted lexicographically by this row key. There are no secondary indices available on HBase table.

Automaticity: Avoid designing table that requires atomacity across all rows. All operations on HBase rows are atomic at row level.

Even distribution: Read and write should uniformly distributed across all nodes available in cluster. Design row key in such a way that, related entities should be stored in adjacent rows to increase read efficacy.

HBase Schema Row key, Column family, Column qualifier, individual and Row value Size Limit

1)Creating a Table using HBase Shell

We can create a table using the create command, here you must specify the table name and the Column Family name. The syntax to create a table in HBase shell is shown below.

create '<table name>','<column family>'

>create 'customer', 'address','order'

```
=> ["demo", "student"]
hbase(main):002:0> create 'customer','address','order'
0 row(s) in 0.4860 seconds

=> Hbase::Table - customer
hbase(main):003:0> list
TABLE
customer
demo
student
3 row(s) in 0.0300 seconds

=> ["customer", "demo", "student"]
hbase(main):004:0>
```

2)Put: Inserts a new record into the table with row identified by 'row..'

This command is used for following things

- It will put a cell 'value' at defined or specified table or row or column.
- It will optionally coordinate time stamp.

Syntax: put <'tablename'>,<'rowname'>,<'columnvalue'>,<'value'>

Example – with the help of put commands we have inserted new records in "customer" table for address and order family. Here name "Nick" is Row key

put 'customer','Nick','address:city','Mumbai'

put 'customer','Nick','address:state','Maharashtra'

put 'customer','Nick','address:street','Steet1'

put 'customer','Nick','order:number','ORD-15'

put 'customer','Nick','order:amount','50'

```
=> ["customer", "demo", "student"]
hbase(main):004:0> put 'customer','Nick','address:city','Mumbai'
0 row(s) in 0.2570 seconds

hbase(main):005:0> put 'customer','Nick','address:state','Maharashtra'
0 row(s) in 0.0100 seconds

hbase(main):006:0> put 'customer','Nick','address:street','Street1'
0 row(s) in 0.0160 seconds

hbase(main):007:0> put 'customer','Nick','order:number','ORD-15'
0 row(s) in 0.0470 seconds

hbase(main):008:0> put 'customer','Nick','order:amount','50'
0 row(s) in 0.0250 seconds
```

Adding one more record of customer name “Justin”. Here name “Justin” is Row key

put 'customer','Justin','address:city','Pune'

put 'customer','Justin,'address:state','Maharashtra'

put 'customer','Justin','order:number','ORD-16'

put 'customer','Justin','order:amount','60'

```
hbase(main):007:0> put 'customer','Nick','order:number','ORD-15'
0 row(s) in 0.0470 seconds

hbase(main):008:0> put 'customer','Nick','order:amount','50'
0 row(s) in 0.0250 seconds

hbase(main):009:0> put 'customer','Justin','address:city','Pune'
0 row(s) in 0.0120 seconds

hbase(main):010:0> put 'customer','Justin','address:state','Maharashtra'
0 row(s) in 0.0050 seconds

hbase(main):011:0> put 'customer','Justin','order:number','ORD-16'
0 row(s) in 0.0120 seconds

hbase(main):012:0> put 'customer','Justin','order:amount','60'
0 row(s) in 0.0060 seconds
```

3)Get: Returns the records matching the row identifier provided in the table

By using this command, you will get a row or cell contents present in the table. In addition to that you can also add additional parameters to it like TIMESTAMP, TIMERANGE,VERSIONS, FILTERS, etc. to get a particular row or cell content.

Syntax: get <'tablename'>, <'rowname'>, {< Additional parameters>}

a) **get 'customer', 'Nick'**

By executing above commands we are getting all the content of customer “Nick” as shown in below screenshot.

```
hbase(main):013:0> get 'customer','Nick'
COLUMN                    CELL
 address:city             timestamp=1626145052337, value=Mumbai
 address:state            timestamp=1626145082811, value=Maharashtra
 address:street           timestamp=1626145105993, value=Street1
 order:amount             timestamp=1626145209102, value=50
 order:number             timestamp=1626145189243, value=ORD-15
5 row(s) in 0.0340 seconds
```

b) **Additional parameters to get only address details**

get 'customer', 'Nick' , 'address'

By executing above commands we are getting all the content of family address for row key “Nick” from “customer” table as per screenshot below.

```
hbase(main):014:0> get 'customer','Nick','address'
COLUMN                    CELL
 address:city             timestamp=1626145052337, value=Mumbai
 address:state            timestamp=1626145082811, value=Maharashtra
 address:street           timestamp=1626145105993, value=Street1
3 row(s) in 0.0210 seconds
```

c) **Additional parameters to get only city details**

get 'customer', 'Nick' , 'address:city'

By executing above commands we are getting all the content of family address city for row key “Nick” from “customer” table as per screenshot below.

```
hbase(main):015:0> get 'customer','Nick','address:city'
COLUMN                    CELL
 address:city             timestamp=1626145052337, value=Mumbai
1 row(s) in 0.0170 seconds

hbase(main):016:0>
```

4)Scan :The scan command is used to view the data in HTable.

Using the scan command, you can get the table data.

- This command scans entire table and displays the table contents.
- We can pass several optional specifications to this scan command to get more information about the tables present in the system.
- Scanner specifications may include one or more of the following attributes.
    - These are TIMERANGE, FILTER, TIMESTAMP, LIMIT, MAXLENGTH, COLUMNS, CACHE, STARTROW and STOPROW.

Its syntax is as follows:

Syntax: scan <'tablename'>, {Optional parameters}

scan 'customer'

When we execute above commands in HBase then we will be getting all the table "customer" contents along with additional parameters like timestamp as show in below screenshot.

```
hbase(main):016:0> scan 'customer'
ROW                    COLUMN+CELL
 Justin                column=address:city, timestamp=1626145251112, value=Pune
 Justin                column=address:state, timestamp=1626145269253, value=Maharashtra
 Justin                column=order:amount, timestamp=1626145318071, value=60
 Justin                column=order:number, timestamp=1626145301942, value=ORD-16
 Nick                  column=address:city, timestamp=1626145052337, value=Mumbai
 Nick                  column=address:state, timestamp=1626145082811, value=Maharashtra
 Nick                  column=address:street, timestamp=1626145105993, value=Street1
 Nick                  column=order:amount, timestamp=1626145209102, value=50
 Nick                  column=order:number, timestamp=1626145189243, value=ORD-15
2 row(s) in 0.1380 seconds
```

5)Delete -Using the delete command, you can delete a specific cell in a table.

- This command will delete cell value at defined table of row or column.
- Delete must and should match the deleted cells coordinates exactly.
- When scanning, delete cell suppresses older versions of values.

The syntax of delete command is as follows:

Syntax:delete <'tablename'>,<'row name'>,<'column name'>

delete 'customer', 'Nick' ,'address:street'

The above command below delete street from address family for row key "Nick" from "customer" table.

```
hbase(main):017:0> delete 'customer','Nick','address:street'
0 row(s) in 0.0220 seconds
```

Use scan command to see if street is deleted for customer "Nick". As we can see "street" information is deleted from customer "Nick" from below screenshot.

```
hbase(main):018:0> scan 'customer'
ROW                     COLUMN+CELL
 Justin                 column=address:city, timestamp=1626145251112, value=Pune
 Justin                 column=address:state, timestamp=1626145269253, value=Maharashtra
 Justin                 column=order:amount, timestamp=1626145318071, value=60
 Justin                 column=order:number, timestamp=1626145301942, value=ORD-16
 Nick                   column=address:city, timestamp=1626145052337, value=Mumbai
 Nick                   column=address:state, timestamp=1626145082811, value=Maharashtra
 Nick                   column=order:amount, timestamp=1626145209102, value=50
 Nick                   column=order:number, timestamp=1626145189243, value=ORD-15
2 row(s) in 0.0330 seconds
```

6)Alter - This command alters the column family schema. To understand what exactly it does, we have explained it here with an example.

Alter commands are useful for below cases -

- Altering single, multiple column family names
- Deleting column family names from table
- Several other operations using scope attributes with table

Syntax: alter <tablename>, NAME=><column familyname>, VERSIONS=>5

We can delete specific column family by using alter commands

alter 'customer', 'delete' => 'address'

```
hbase(main):019:0> alter 'customer','delete' => 'address'
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.2310 seconds

hbase(main):020:0>
```

After deleting “address” family from “customer” table. Let’s again check customer table using “scan” commands as follow

>scan ‘customer’

As you can see from below screenshot we only now have order:amount and order:number.

```
hbase(main):020:0> scan 'customer'
ROW                     COLUMN+CELL
 Justin                 column=order:amount, timestamp=1626145318071, value=60
 Justin                 column=order:number, timestamp=1626145301942, value=ORD-16
 Nick                   column=order:amount, timestamp=1626145209102, value=50
 Nick                   column=order:number, timestamp=1626145189243, value=ORD-15
2 row(s) in 0.0230 seconds
```

7)Describe - This command describes the named table.

- It will give more information about column families present in the mentioned table
- In our case, it gives the description about table "customer."

- It will give information about table name with column families, associated filters, versions and some more details.

Syntax:describe <table name>

desc 'customer'

```
hbase(main):021:0> desc 'customer'
Table customer is ENABLED
customer
COLUMN FAMILIES DESCRIPTION
{NAME => 'order', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIO
NS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
 BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0310 seconds
```

8)Versions –

A {row, column, version} tuple exactly specifies a cell in HBase. In the Apache HBase you can have many cells where row and columns are same but differs only in version values. A version is a timestamp values is written alongside each value. By default, the timestamp values represent the time on the RegionServer when the data was written, but you can change the default HBase setting and specify a different timestamp value when you put data into the cell.

In HBase, rows and column keys are expressed as bytes, the version is specified using a long integer. The HBase version dimension is stored in decreasing order, so that when reading from a store file, the most recent values are found first.

create 'customer1', {NAME => 'address', VERSIONS => 3}

With the help of above commands we are creating 3 versions

```
hbase(main):004:0> create 'customer1', {NAME => 'address', VERSIONS => 3}
0 row(s) in 0.4240 seconds

=> Hbase::Table - customer1
hbase(main):005:0> lis
```

Verifying if “customer1” is created after executing above commands with the help of list commands as shown in screenshot below.

```
hbase(main):005:0> list
TABLE
customer
customer1
demo
student
4 row(s) in 0.0120 seconds

=> ["customer", "customer1", "demo", "student"]
hbase(main):006:0> count 'customer1'
0 row(s) in 0.0330 seconds

=> 0
```

9)Count - You can count the number of rows of a table using the count command. Its syntax is as follows:

count 'customer'

```
=> 0
hbase(main):007:0> count 'customer'
2 row(s) in 0.0080 seconds

=> 2
hbase(main):008:0>
```

10)Alter -Update the version number for already existing columns family

alter 'customer' , NAME => 'address', VERSIONS =>5

With the help of above commands we are altering version number of “customer” table to “5” versions for address family. See screenshot below.

```
hbase(main):008:0> alter 'customer', NAME => 'address', VERSIONS => 5
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.2320 seconds

hbase(main):009:0>
```

Again using describe command to see version is updated

>desc ‘customer’

As you can see on below screenshot version is now 5.

```
hbase(main):009:0> desc 'customer'
Table customer is ENABLED
customer
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', COMP
RESSION => 'NONE', VERSIONS => '5', TTL => 'FOREVER', MIN_VERSIONS => '0', KEEP_DELETED_CELLS => 'FALSE
', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'order', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIO
NS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
 BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
2 row(s) in 0.0470 seconds

hbase(main):010:0>
```

Below are example of adding version of address:city

Currently below is the data in ‘customer’. We will be adding more information in it one by one.

>scan 'customer'

```
hbase(main):010:0> scan 'customer'
ROW                     COLUMN+CELL
 Justin                 column=address:city, timestamp=1626145251112, value=Pune
 Justin                 column=address:state, timestamp=1626145269253, value=Maharashtra
 Justin                 column=order:amount, timestamp=1626145318071, value=60
 Justin                 column=order:number, timestamp=1626145301942, value=ORD-16
 Nick                   column=address:city, timestamp=1626145052337, value=Mumbai
 Nick                   column=address:state, timestamp=1626145082811, value=Maharashtra
 Nick                   column=order:amount, timestamp=1626145209102, value=50
 Nick                   column=order:number, timestamp=1626145189243, value=ORD-15
2 row(s) in 0.0420 seconds

hbase(main):011:0>
```

put 'customer', 'Nick' , 'address:city', 'Pune'

```
hbase(main):011:0> put 'customer', 'Nick','address:city','Pune'
0 row(s) in 0.1000 seconds
```

put 'customer', 'Nick' , 'address:city', 'Bangalore'

```
hbase(main):012:0> put 'customer', 'Nick','address:city','Bangalore'
0 row(s) in 0.0050 seconds
```

put 'customer', 'Nick' , 'address:city', 'Delhi'

```
hbase(main):013:0> put 'customer', 'Nick','address:city','Delhi'
0 row(s) in 0.0080 seconds
```

Now we have 4 values for customer Nick for city

>scan 'customer', {COLUMN=> 'address:city', VERSIONS => 2 }

After executing above commands its giving us 2 version of address:city records as shown in below screenshot.

```
hbase(main):015:0> scan 'customer',{COLUMN => 'address:city',VERSIONS => 2}
ROW                     COLUMN+CELL
 Justin                 column=address:city, timestamp=1626145251112, value=Pune
 Nick                   column=address:city, timestamp=1626146374063, value=Delhi
 Nick                   column=address:city, timestamp=1626146364276, value=Bangalore
2 row(s) in 0.0210 seconds

hbase(main):016:0>
```

>scan 'customer', {COLUMN=> 'address:city', VERSIONS => 3}

>scan 'customer', {COLUMN=> 'address:city', VERSIONS => 4}

After executing above commands its giving us 3 and 4 version of address:city records as shown in below screenshot.

```
hbase(main):016:0> scan 'customer',{COLUMN => 'address:city',VERSIONS => 3}
ROW                     COLUMN+CELL
 Justin                 column=address:city, timestamp=1626145251112, value=Pune
 Nick                   column=address:city, timestamp=1626146374063, value=Delhi
 Nick                   column=address:city, timestamp=1626146364276, value=Bangalore
 Nick                   column=address:city, timestamp=1626146353695, value=Pune
2 row(s) in 0.0280 seconds

hbase(main):017:0> scan 'customer',{COLUMN => 'address:city',VERSIONS => 4}
ROW                     COLUMN+CELL
 Justin                 column=address:city, timestamp=1626145251112, value=Pune
 Nick                   column=address:city, timestamp=1626146374063, value=Delhi
 Nick                   column=address:city, timestamp=1626146364276, value=Bangalore
 Nick                   column=address:city, timestamp=1626146353695, value=Pune
 Nick                   column=address:city, timestamp=1626145052337, value=Mumbai
2 row(s) in 0.0390 seconds

hbase(main):018:0>
```

>scan 'customer', {COLUMN=> 'address:city', VERSIONS => 5}

After executing above commands its giving us 4 version of address:city records as shown in below screenshot because as of bow we have only 4 version of address:city in our "customer" table.

```
hbase(main):018:0> scan 'customer',{COLUMN => 'address:city',VERSIONS => 5}
ROW                      COLUMN+CELL
 Justin                  column=address:city, timestamp=1626145251112, value=Pune
 Nick                    column=address:city, timestamp=1626146374063, value=Delhi
 Nick                    column=address:city, timestamp=1626146364276, value=Bangalore
 Nick                    column=address:city, timestamp=1626146353695, value=Pune
 Nick                    column=address:city, timestamp=1626145052337, value=Mumbai
2 row(s) in 0.0200 seconds

hbase(main):019:0>
```

scan 'customer', {VERSIONS => 5 } – This for all columns not for any specific column

```
hbase(main):019:0> scan 'customer',{VERSIONS => 5}
ROW                      COLUMN+CELL
 Justin                  column=address:city, timestamp=1626145251112, value=Pune
 Justin                  column=address:state, timestamp=1626145269253, value=Maharashtra
 Justin                  column=order:amount, timestamp=1626145318071, value=60
 Justin                  column=order:number, timestamp=1626145301942, value=ORD-16
 Nick                    column=address:city, timestamp=1626146374063, value=Delhi
 Nick                    column=address:city, timestamp=1626146364276, value=Bangalore
 Nick                    column=address:city, timestamp=1626146353695, value=Pune
 Nick                    column=address:city, timestamp=1626145052337, value=Mumbai
 Nick                    column=address:state, timestamp=1626145082811, value=Maharashtra
 Nick                    column=order:amount, timestamp=1626145209102, value=50
 Nick                    column=order:number, timestamp=1626145189243, value=ORD-15
2 row(s) in 0.0640 seconds

hbase(main):020:0>
```

11)Disable -This command will start disabling the named table

If table needs to be deleted or dropped, it has to disable first

Syntax: disable <tablename>

disable 'customer'

```
hbase(main):020:0> disable 'customer'
0 row(s) in 1.2660 seconds
```

12)Drop– It drops a table from HBase. Drop means complete deletion of table. For this first disable the table then drop it.

- To delete the table present in HBase, first we have to disable it
- To drop the table present in HBase, first we have to disable it
- So either table to drop or delete first the table should be disable using disable command

Syntax:drop <table name>

drop 'customer'

Once above commands finish its execution “customer” table will be dropped from the system. See screenshot below.

```
hbase(main):021:0> drop 'customer'
0 row(s) in 0.2250 seconds

hbase(main):022:0> list
TABLE
customer1
demo
student
3 row(s) in 0.0150 seconds

=> ["customer1", "demo", "student"]
hbase(main):023:0>
```