Name: Pooja Pathak
Roll No: 14

Practical_8: 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 JavaTM 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.

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 'emp', 'office'


```
hbase(main):046:0> create 'emp','office'
0 row(s) in 0.4470 seconds

=> Hbase::Table - emp
hbase(main):047:0>
```


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):047:0> list

```
hbase(main):047:0> list
TABLE
customer1
emp
2 row(s) in 0.0060 seconds

=> ["customer1", "emp"]
hbase(main):048:0>
```

'DESCRIBE' command to describe the details and configuration of the HBase table. For example, version, compression, blocksize, replication etc. The syntax to describe the table is as follows.

Syntax: describe <'namespace':'table_name'>

hbase(main):047:0>describe 'emp'

```
hbase(main):048:0> describe 'emp'
Table emp is ENABLED
emp
COLUMN FAMILIES DESCRIPTION
{NAME => 'office', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE',
 MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0220 seconds

hbase(main):049:0>
```

'Put' command to insert data to rows and columns on a table. This would be similar to insert statement on RDBMS but, the syntax is completely different.

Syntax: put '<name_space:table_name>', '<row_key>' '<cf:column_name>', '<value>'

In above examples, notice that we have added 2 rows; row key '1' with one column 'office:name' and row key '2' with three columns 'office:name', 'office:gender' and 'office:age'. If you are coming from RDBMS world, you probably would confuse with this. Once you understand how column database works it's not that difficult to get around it.

Also, note that last command from above example actually inserts a new column 'office:age' at row key '2' with '50'

Internally, HBase doesn't do an update but it assigns a column with new timestamp and scan fetches the latest data from columns.

```
hbase(main):049:0> put 'emp', '1' , 'office:name', 'Scott'
0 row(s) in 0.0140 seconds

hbase(main):050:0> put 'emp', '2' , 'office:name', 'Mark'
0 row(s) in 0.0070 seconds

hbase(main):051:0> put 'emp', '2' , 'office:gender', 'M'
0 row(s) in 0.0080 seconds

hbase(main):052:0> put 'emp', '2' , 'office:age', '30'
0 row(s) in 0.0090 seconds

hbase(main):053:0> put 'emp', '2' , 'office:age', '50'
0 row(s) in 0.0090 seconds
```

```
hbase(main):054:0> put 'emp', '3', 'office:salary', '10000'
0 row(s) in 0.0080 seconds

hbase(main):055:0> put 'emp', '3', 'office:name', 'Jeff'
0 row(s) in 0.0060 seconds

hbase(main):056:0> put 'emp', '3', 'office:salary', '20000'
0 row(s) in 0.0110 seconds

hbase(main):057:0> put 'emp', '3', 'office:salary', '30000'
0 row(s) in 0.0070 seconds

hbase(main):058:0> put 'emp', '3', 'office:salary', '40000'
0 row(s) in 0.0070 seconds

hbase(main):059:0> put 'emp','1','office:age','20'
0 row(s) in 0.0080 seconds

hbase(main):060:0> put 'emp','3','office:age','30'
0 row(s) in 0.0070 seconds

hbase(main):061:0> █
```

'Scan' command to get the data from the HBase table. By default, it fetches all data from the table.

Syntax: scan '<name_space:table_name>'

```
hbase(main):061:0> scan 'emp'
ROW                           COLUMN+CELL
 1                            column=office:age, timestamp=1648183680203, value=20
 1                            column=office:name, timestamp=1648183480385, value=Scott
 2                            column=office:age, timestamp=1648183537136, value=50
 2                            column=office:gender, timestamp=1648183500661, value=M
 2                            column=office:name, timestamp=1648183491849, value=Mark
 3                            column=office:age, timestamp=1648183687737, value=30
 3                            column=office:name, timestamp=1648183643747, value=Jeff
 3                            column=office:salary, timestamp=1648183671020, value=40000
3 row(s) in 0.0150 seconds

hbase(main):062:0> █
```

This scan's the 'emp' table to return name and age columns from starting row 1 and ending row 3.

```
hbase(main):062:0> scan 'emp',{COLUMNS=>['office:name','office:age'],STARTROW=>'1',STOPROW=>'3'}
ROW                           COLUMN+CELL
 1                            column=office:age, timestamp=1648183680203, value=20
 1                            column=office:name, timestamp=1648183480385, value=Scott
 2                            column=office:age, timestamp=1648183537136, value=50
 2                            column=office:name, timestamp=1648183491849, value=Mark
2 row(s) in 0.0130 seconds

hbase(main):063:0> █
```

```
hbase(main):065:0> scan 'emp'
ROW                           COLUMN+CELL
 1                            column=office:age, timestamp=1648183680203, value=20
 1                            column=office:name, timestamp=1648183480385, value=Scott
 2                            column=office:age, timestamp=1648183537136, value=50
 2                            column=office:gender, timestamp=1648183500661, value=M
 2                            column=office:name, timestamp=1648183491849, value=Mark
 3                            column=office:age, timestamp=1648183687737, value=30
 3                            column=office:name, timestamp=1648183643747, value=Jeff
 3                            column=office:salary, timestamp=1648183671020, value=40000
3 row(s) in 0.0190 seconds

hbase(main):066:0> █
```

```
hbase(main):066:0> get 'emp','2'
COLUMN                        CELL
 office:age                   timestamp=1648183537136, value=50
 office:gender                timestamp=1648183500661, value=M
 office:name                  timestamp=1648183491849, value=Mark
3 row(s) in 0.0100 seconds

hbase(main):067:0> █
```

```
hbase(main):067:0> get 'emp','2',{COLUMNS=>['office:name','office:age']}
COLUMN                        CELL
 office:age                   timestamp=1648183537136, value=50
 office:name                  timestamp=1648183491849, value=Mark
2 row(s) in 0.0040 seconds

hbase(main):068:0> █
```

Use 'disable' to disable a table. Prior to delete a table or change its setting, first, you need to disable the table. The syntax to disable the table is as follows.

Syntax: disable '<namespace>:<table_name>'

```
hbase(main):068:0> disable 'emp'
0 row(s) in 1.1800 seconds

hbase(main):069:0>
```

Use `is_disabled` to check if the table is disabled. When it disabled it returns '`true`'

```
hbase(main):069:0> is_disabled 'emp'
true
0 row(s) in 0.0100 seconds

hbase(main):070:0>
```

Let's check if the table disabled by using `describe`

```
hbase(main):001:0> describe 'emp'
Table emp is DISABLED
emp
COLUMN FAMILIES DESCRIPTION
{NAME => 'office', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICAT
ION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', T
TL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY
=> 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.3830 seconds

hbase(main):002:0>
```

'`enable`' command is used to enable a disabled table. You need to enable a disabled table first to perform any regular commands., The syntax to enable the table is as follows.
Syntax: enable '<namespace>:<table_name>'

Syntax: `enable` '<namespace>:<table_name>'

```
hbase(main):002:0> enable 'emp'
0 row(s) in 0.4350 seconds

hbase(main):003:0>
```

Use `drop` command to delete a table. You should disable a table first before you drop it.

Syntax: drop '<table_name>'

```
hbase(main):003:0> disable 'emp'
0 row(s) in 1.2420 seconds

hbase(main):004:0> drop 'emp'
0 row(s) in 0.2040 seconds

hbase(main):005:0>
```

Use `drop_all` command to delete many tables using a regular expression.
hbase(main):041:0> drop_all 'em.*'

```
 hbase(main):041:0> drop_all 'em.*'
```

```
hbase(main):005:0> drop_all 'em.*'

ERROR: wrong number of arguments (0 for 1)

Here is some help for this command:
Drop all of the tables matching the given regex:

hbase> drop_all 't.*'
hbase> drop_all 'ns:t.*'
hbase> drop_all 'ns:.*'


[cloudera@quickstart ~]$
```
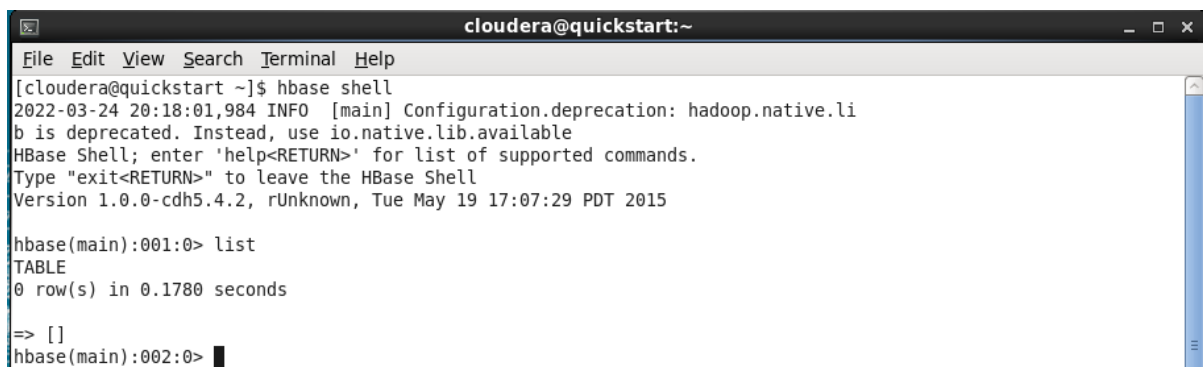
```
cloudera@quickstart:~                                    _ □ ×
File  Edit  View  Search  Terminal  Help
[cloudera@quickstart ~]$ hbase shell
2022-03-24 20:18:01,984 INFO  [main] Configuration.deprecation: hadoop.native.li
b is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<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
0 row(s) in 0.1780 seconds

=> []
hbase(main):002:0>
```

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'

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.1780 seconds

=> []
hbase(main):002:0> create 'customer','address','order'
0 row(s) in 0.6290 seconds

=> Hbase::Table - customer
hbase(main):003:0> list
TABLE
customer
1 row(s) in 0.0100 seconds

=> ["customer"]
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'
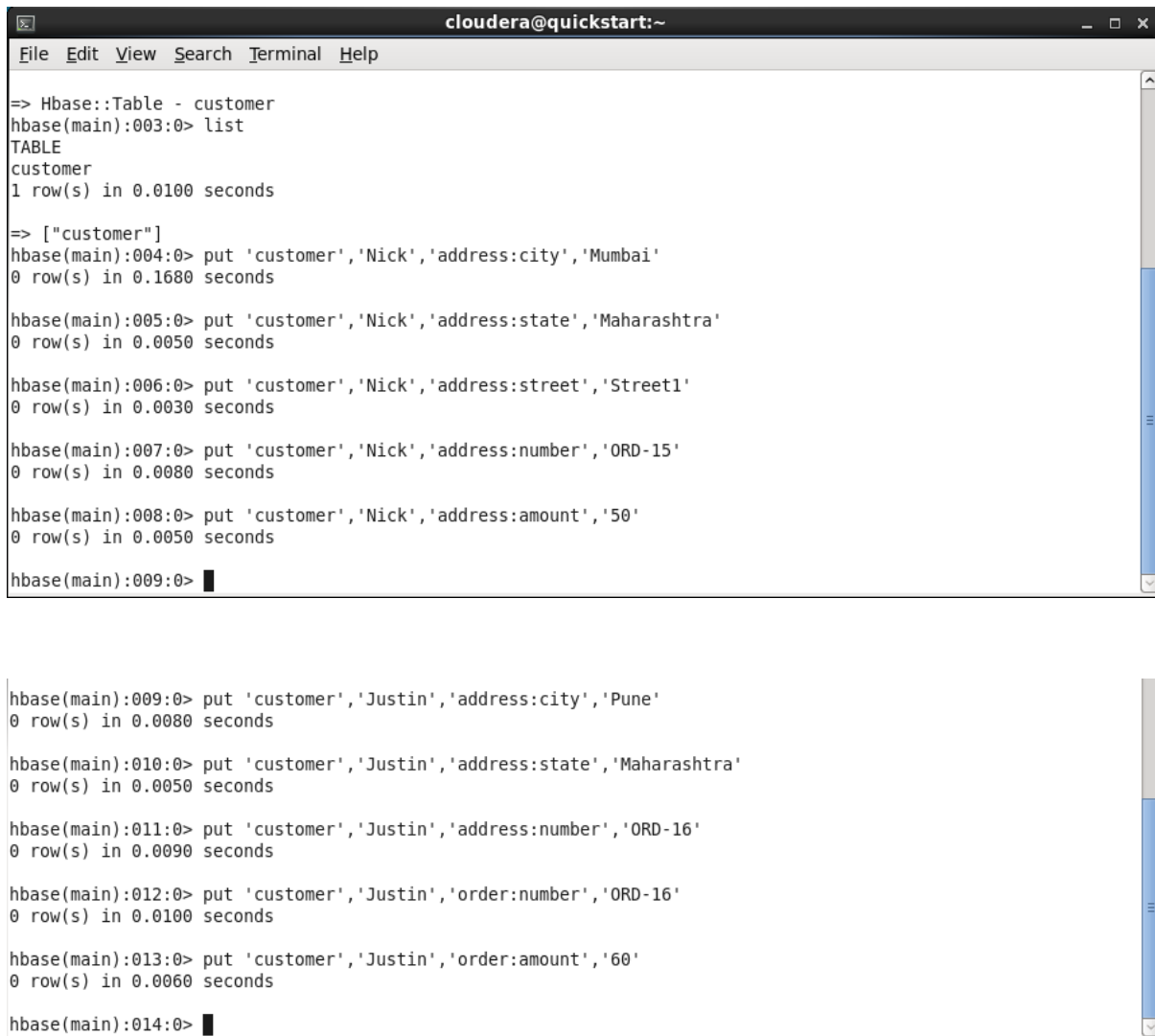
Adding one more record of customer name "Justin". Here name "Justin" is Row key

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'

```
cloudera@quickstart:~                                          _ □ ×

File  Edit  View  Search  Terminal  Help

=> Hbase::Table - customer
hbase(main):003:0> list
TABLE
customer
1 row(s) in 0.0100 seconds

=> ["customer"]
hbase(main):004:0> put 'customer','Nick','address:city','Mumbai'
0 row(s) in 0.1680 seconds

hbase(main):005:0> put 'customer','Nick','address:state','Maharashtra'
0 row(s) in 0.0050 seconds

hbase(main):006:0> put 'customer','Nick','address:street','Street1'
0 row(s) in 0.0030 seconds

hbase(main):007:0> put 'customer','Nick','address:number','ORD-15'
0 row(s) in 0.0080 seconds

hbase(main):008:0> put 'customer','Nick','address:amount','50'
0 row(s) in 0.0050 seconds

hbase(main):009:0> ▮
```

```
hbase(main):009:0> put 'customer','Justin','address:city','Pune'
0 row(s) in 0.0080 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','address:number','ORD-16'
0 row(s) in 0.0090 seconds

hbase(main):012:0> put 'customer','Justin','order:number','ORD-16'
0 row(s) in 0.0100 seconds

hbase(main):013:0> put 'customer','Justin','order:amount','60'
0 row(s) in 0.0060 seconds

hbase(main):014:0> ▮
```

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):014:0> get 'customer','Nick'
COLUMN                    CELL
 address:amount           timestamp=1648179300824, value=50
 address:city             timestamp=1648179065272, value=Mumbai
 address:number           timestamp=1648179263086, value=ORD-15
 address:state            timestamp=1648179093320, value=Maharashtra
 address:street           timestamp=1648179136727, value=Street1
5 row(s) in 0.0240 seconds

hbase(main):015:0>
```

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):015:0> get 'customer','Nick','address'
COLUMN                    CELL
 address:amount           timestamp=1648179300824, value=50
 address:city             timestamp=1648179065272, value=Mumbai
 address:number           timestamp=1648179263086, value=ORD-15
 address:state            timestamp=1648179093320, value=Maharashtra
 address:street           timestamp=1648179136727, value=Street1
5 row(s) in 0.0100 seconds

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

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):016:0> get 'customer','Nick','address:city'
COLUMN                    CELL
 address:city             timestamp=1648179065272, value=Mumbai
1 row(s) in 0.0050 seconds

hbase(main):017:0>
```

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. o

These are TIMERANGE, FILTER, TIMESTAMP, LIMIT, MAXLENGTH,
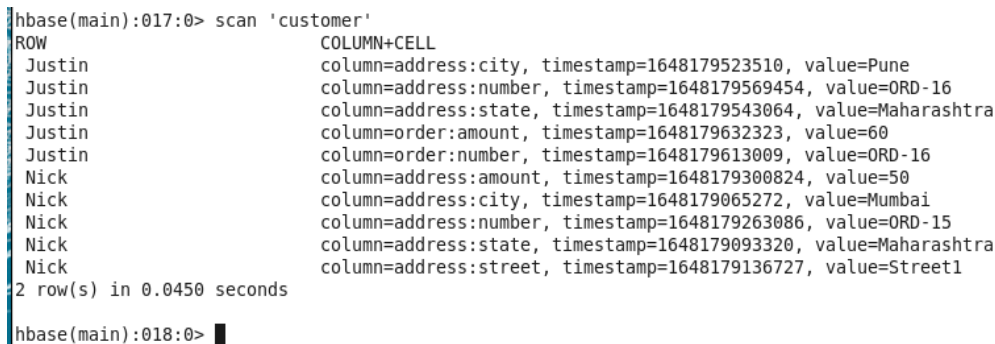
COLUMNS, CACHE, STARTROW and STOPROW.

Its syntax is as follows:

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

<u>scan 'customer'</u>

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):017:0> scan 'customer'
ROW                      COLUMN+CELL
 Justin                  column=address:city, timestamp=1648179523510, value=Pune
 Justin                  column=address:number, timestamp=1648179569454, value=ORD-16
 Justin                  column=address:state, timestamp=1648179543064, value=Maharashtra
 Justin                  column=order:amount, timestamp=1648179632323, value=60
 Justin                  column=order:number, timestamp=1648179613009, value=ORD-16
 Nick                    column=address:amount, timestamp=1648179300824, value=50
 Nick                    column=address:city, timestamp=1648179065272, value=Mumbai
 Nick                    column=address:number, timestamp=1648179263086, value=ORD-15
 Nick                    column=address:state, timestamp=1648179093320, value=Maharashtra
 Nick                    column=address:street, timestamp=1648179136727, value=Street1
2 row(s) in 0.0450 seconds

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

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):018:0> delete 'customer','Nick','address:street'
0 row(s) in 0.0520 seconds
```

```
hbase(main):019:0> scan 'customer'
ROW                    COLUMN+CELL
 Justin                column=address:city, timestamp=1648179523510, value=Pune
 Justin                column=address:number, timestamp=1648179569454, value=ORD-16
 Justin                column=address:state, timestamp=1648179543064, value=Maharashtra
 Justin                column=order:amount, timestamp=1648179632323, value=60
 Justin                column=order:number, timestamp=1648179613009, value=ORD-16
 Nick                  column=address:amount, timestamp=1648179300824, value=50
 Nick                  column=address:city, timestamp=1648179065272, value=Mumbai
 Nick                  column=address:number, timestamp=1648179263086, value=ORD-15
 Nick                  column=address:state, timestamp=1648179093320, value=Maharashtra
2 row(s) in 0.0230 seconds

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

```
hbase(main):020:0> delete 'customer','Nick','address:street'
0 row(s) in 0.0100 seconds
```

```
hbase(main):024:0> scan 'customer'
ROW                    COLUMN+CELL
 Justin                column=order:amount, timestamp=1648179632323, value=60
 Justin                column=order:number, timestamp=1648179613009, value=ORD-16
1 row(s) in 0.0090 seconds

hbase(main):025:0>
```

```
hbase(main):026:0> desc 'customer'
Table customer is ENABLED
customer
COLUMN FAMILIES DESCRIPTION
{NAME => 'order', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1
', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '6
5536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0180 seconds

hbase(main):027:0>
```

```
hbase(main):028:0> create 'customer1',{NAME=>'address',VERSIONS=>3}
0 row(s) in 0.1730 seconds

=> Hbase::Table - customer1
hbase(main):029:0>
```

```
hbase(main):029:0> count 'customer'
1 row(s) in 0.0250 seconds

=> 1
hbase(main):030:0>
```

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):030:0> 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 2.1500 seconds

hbase(main):031:0>
```

```
hbase(main):031:0> desc 'customer'
Table customer is ENABLED
customer
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIO
NS => '5', TTL => 'FOREVER', MIN_VERSIONS => '0', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLO
CKCACHE => 'true'}
{NAME => 'order', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSION =>
 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCK
CACHE => 'true'}
2 row(s) in 0.0330 seconds
```

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):032:0> scan 'customer'
ROW                       COLUMN+CELL
 Justin                   column=address:city, timestamp=1648179523510, value=Pune
 Justin                   column=address:number, timestamp=1648179569454, value=ORD-16
 Justin                   column=address:state, timestamp=1648179543064, value=Maharashtra
 Justin                   column=order:amount, timestamp=1648179632323, value=60
 Justin                   column=order:number, timestamp=1648179613009, value=ORD-16
 Nick                     column=address:amount, timestamp=1648179300824, value=50
 Nick                     column=address:city, timestamp=1648179065272, value=Mumbai
 Nick                     column=address:number, timestamp=1648179263086, value=ORD-15
 Nick                     column=address:state, timestamp=1648179093320, value=Maharashtra
2 row(s) in 0.0190 seconds

hbase(main):033:0>
```

```
hbase(main):033:0> put 'customer', 'Nick' , 'address:city', 'Pune'
0 row(s) in 0.0100 seconds
```

```
hbase(main):039:0> scan 'customer', {COLUMN=> 'address:city', VERSIONS => 2 }
ROW                        COLUMN+CELL
 Justin                    column=address:city, timestamp=1648179523510, value=Pune
 Nick                      column=address:city, timestamp=1648181738204, value=Delhi
 Nick                      column=address:city, timestamp=1648181722715, value=Bangalore
2 row(s) in 0.0150 seconds
```

```
hbase(main):040:0> scan 'customer', {COLUMN=> 'address:city', VERSIONS => 3}
ROW                        COLUMN+CELL
 Justin                    column=address:city, timestamp=1648179523510, value=Pune
 Nick                      column=address:city, timestamp=1648181738204, value=Delhi
 Nick                      column=address:city, timestamp=1648181722715, value=Bangalore
 Nick                      column=address:city, timestamp=1648181618375, value=Pune
2 row(s) in 0.0250 seconds

hbase(main):041:0>
```

```
hbase(main):041:0> scan 'customer', {COLUMN=> 'address:city', VERSIONS => 4}
ROW                        COLUMN+CELL
 Justin                    column=address:city, timestamp=1648179523510, value=Pune
 Nick                      column=address:city, timestamp=1648181738204, value=Delhi
 Nick                      column=address:city, timestamp=1648181722715, value=Bangalore
 Nick                      column=address:city, timestamp=1648181618375, value=Pune
 Nick                      column=address:city, timestamp=1648179065272, value=Mumbai
2 row(s) in 0.0100 seconds
```

```
hbase(main):042:0> scan 'customer', {COLUMN=> 'address:city', VERSIONS => 5}
ROW                        COLUMN+CELL
 Justin                    column=address:city, timestamp=1648179523510, value=Pune
 Nick                      column=address:city, timestamp=1648181738204, value=Delhi
 Nick                      column=address:city, timestamp=1648181722715, value=Bangalore
 Nick                      column=address:city, timestamp=1648181618375, value=Pune
 Nick                      column=address:city, timestamp=1648179065272, value=Mumbai
2 row(s) in 0.0120 seconds
```

```
hbase(main):043:0> scan 'customer', {VERSIONS => 5 }
ROW                        COLUMN+CELL
 Justin                    column=address:city, timestamp=1648179523510, value=Pune
 Justin                    column=address:number, timestamp=1648179569454, value=ORD-16
 Justin                    column=address:state, timestamp=1648179543064, value=Maharashtra
 Justin                    column=order:amount, timestamp=1648179632323, value=60
 Justin                    column=order:number, timestamp=1648179613009, value=ORD-16
 Nick                      column=address:amount, timestamp=1648179300824, value=50
 Nick                      column=address:city, timestamp=1648181738204, value=Delhi
 Nick                      column=address:city, timestamp=1648181722715, value=Bangalore
 Nick                      column=address:city, timestamp=1648181618375, value=Pune
 Nick                      column=address:city, timestamp=1648179065272, value=Mumbai
 Nick                      column=address:number, timestamp=1648179263086, value=ORD-15
 Nick                      column=address:state, timestamp=1648179093320, value=Maharashtra
2 row(s) in 0.0170 seconds
```

```
hbase(main):044:0> disable 'customer'
0 row(s) in 1.2560 seconds

hbase(main):045:0>
```

```
hbase(main):045:0> drop 'customer'
0 row(s) in 0.3200 seconds

hbase(main):046:0> █
```