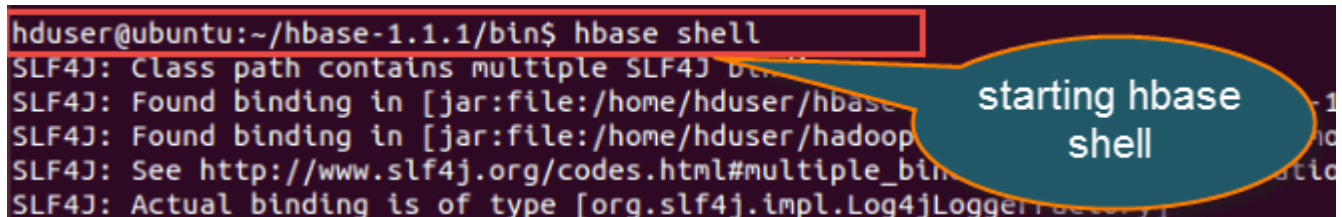


General commands

In Hbase, general commands are categorized into following commands

- **Status**
- **Version**
- **Table_help** (scan, drop, get, put, disable, etc.)
- **Whoami**

To get enter into HBase shell command, first of all, we have to execute the code as mentioned below



```
hduser@ubuntu:~/hbase-1.1.1/bin$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings
SLF4J: Found binding in [jar:file:/home/hduser/hbase-1.1.1/lib/slf4j-log4j12-1.7.12.jar:]
SLF4J: Found binding in [jar:file:/home/hduser/hadoop-2.6.0/lib/slf4j-log4j12-1.7.12.jar:]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

hbase Shell

Once we get to enter into HBase shell, we can execute all shell commands mentioned below. With the help of these commands, we can perform all type of table operations in the HBase shell mode.

Let us look into all of these commands and their usage one by one with an example.

Status

Syntax: status

This command will give details about the system status like a number of servers present in the cluster, active server count, and average load value. You can also pass any particular parameters depending on how detailed status you want to know about the system. The parameters can be **‘summary’, ‘simple’, or ‘detailed’**, the default parameter provided is “summary”.

Below we have shown how you can pass different parameters to the status command.

If we observe the below screen shot, we will get a better idea.

```
hbase(main):001:0> status
1 servers, 1 dead, 7.0000 average load

hbase(main):002:0> status 'simple'
1 live servers
  ubuntu:45056 1441963149129
    requestsPerSecond=0.0, numberOfOnlineRegions=7, usedHeapMB=25, maxIndexSizeMB=0, readRequestsCount=105, writeRequestsCount=7, rootIndexSizeKB=0, master coprocessors=[]
1 dead servers
  ubuntu:33734,1440738704687
Aggregate load: 0, regions: 7

hbase(main):003:0> status 'detail'
1 servers, 1 dead, 7.0000 average load

hbase(main):004:0> status 'detailed'
version 1.1.1
0 regionsInTransition
master coprocessors: []
1 live servers
  ubuntu:45056 1441963149129
    requestsPerSecond=0.0, numberOfOnlineRegions=7, usedHeapMB=27, maxIndexSizeMB=0, readRequestsCount=105, writeRequestsCount=7, rootIndexSizeKB=0, master coprocessors=[]
```

Status command with different forms

```
hbase(main):001:0>status
hbase(main):002:0>status 'simple'
hbase(main):003:0>status 'summary'
hbase(main):004:0> status 'detailed'
```

When we execute this command status, it will give information about number of server's present, dead servers and average load of server, here in screenshot it shows the information like- **1 live server, 1 dead servers, and 7.0000 average load.**

Version

Syntax: version

```
hbase(main):005:0> version
1.1.1, rd0a115a7267f54e01c72c603ec53e91ec418292f, Tue Jun 23 14:44:07 PDT 2015
```

- This command will display the currently used HBase version in command mode
- If you run version command, it will give output as shown above

Table help

Syntax: table_help

```

hbase(main):007:0> table_help
Help for table-reference commands.

You can either create a table via 'create' and then manipulate the table via commands
See the standard help information for how to use each of these commands.

However, as of 0.96, you can also get a reference to a table, on which you can invoke
For instance, you can get create a table and keep around a reference to it via:

    hbase> t = create 't', 'cf'
Or, if you have already created the table, you can get a reference to it:

    hbase> t = get_table 't'
You can do things like call 'put' on the table:

```

This command guides

- What and how to use table-referenced commands
- It will provide different HBase shell command usages and its syntaxes
- Here in the screen shot above, its shows the syntax to “**create**” and “**get_table**” command with its usage. We can manipulate the table via these commands once the table gets created in HBase.
- It will give table manipulations commands like put, get and all other commands information.

whoami

Syntax:

Syntax: Whoami

```

hbase(main):006:0> whoami
hduser (auth:SIMPLE)
groups: hduser, adm, cdrom, sudo, dip, plugdev, lpadmin, sambashare

```

This command “whoami” is used to return the current HBase user information from the HBase cluster.

It will provide information like

- Groups present in HBase
- The user information, for example in this case “hduser” represent the user name as shown in screen shot

TTL(Time To Live) – Attribute

In HBase, Column families can be set to time values in seconds using TTL. HBase will automatically delete rows once the expiration time is reached. This attribute applies to all versions of a row – even the current version too.

The TTL time encoded in the HBase for the row is specified in UTC. This attribute used with table management commands.

Important differences between TTL handling and Column family TTLs are below

- Cell TTLs are expressed in units of milliseconds instead of seconds.
- A cell TTLs cannot extend the effective lifetime of a cell beyond a Column Family level TTL setting.

Tables Managements commands

These commands will allow programmers to create tables and table schemas with rows and column families.

The following are Table Management commands

- Create
- List
- Describe
- Disable
- Disable_all
- Enable
- Enable_all
- Drop
- Drop_all
- Show_filters
- Alter
- Alter_status

Let us look into various command usage in HBase with an example.

Create

Syntax: create <tablename>, <columnfamilyname>

```
hbase(main):007:0> create 'education', 'guru99'  
0 row(s) in 2.3250 seconds
```

Example:-

```
hbase(main):001:0> create 'education' , 'tushar'  
0 rows(s) in 0.312 seconds  
=>Hbase::Table - education
```

The above example explains how to create a table in HBase with the specified name given according to the dictionary or specifications as per column family. In addition to this we can also pass some table-scope attributes as well into it.

In order to check whether the table 'education' is created or not, we have to use the **“list”** command as mentioned below.

List

Syntax: list

```
hbase(main):008:0> list
TABLE
User
crawldata
edu
education
emp
test
test1
veeru
8 row(s) in 0.0620 seconds
```

- “List” command will display all the tables that are present or created in HBase
- The output showing in above screen shot is currently showing the existing tables in HBase
- Here in this screenshot, it shows that there are total 8 tables present inside HBase
- We can filter output values from tables by passing optional regular expression parameters

Describe

Syntax: describe <table name>

```
hbase(main):010:0> describe 'education'
Table education is ENABLED
education
COLUMN FAMILIES DESCRIPTION
{NAME => 'guru99', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSION => '1', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.1010 seconds
```

```
hbase(main):010:0>describe 'education'
```

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

disable

Syntax: disable <tablename>

```
hbase(main):011:0> disable 'education'
0 row(s) in 2.3760 seconds
```

```
hbase(main):011:0>disable 'education'
```

- This command will start disabling the named table
- If table needs to be deleted or dropped, it has to disable first

Here, in the above screenshot we are disabling table education

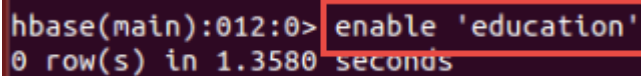
disable_all

Syntax: disable_all<"matching regex">

- This command will disable all the tables matching the given regex.
- The implementation is same as delete command (Except adding regex for matching)
- Once the table gets disabled the user can able to delete the table from HBase
- Before delete or dropping table, it should be disabled first

Enable

Syntax: enable <tablename>

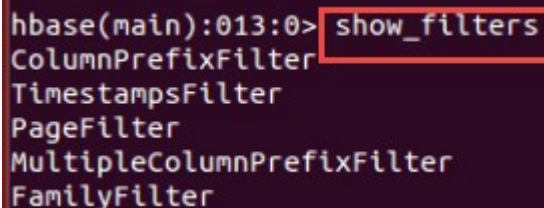
A screenshot of a terminal window showing the HBase shell command 'enable 'education'' being executed. The command is highlighted with a red rectangular box. The output shows '0 row(s) in 1.3580 seconds'.

hbase(main):012:0>enable 'education'

- This command will start enabling the named table
- Whichever table is disabled, to retrieve back to its previous state we use this command
- If a table is disabled in the first instance and not deleted or dropped, and if we want to re-use the disabled table then we have to enable it by using this command.
- Here in the above screenshot we are enabling the table “education.”

show_filters

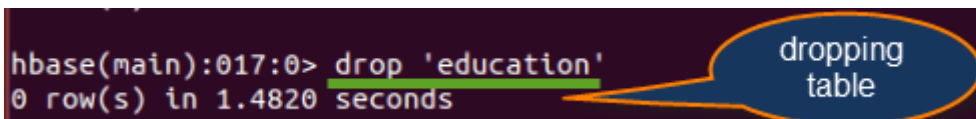
Syntax: show_filters

A screenshot of a terminal window showing the HBase shell command 'show_filters' being executed. The command is highlighted with a red rectangular box. The output lists several filters: ColumnPrefixFilter, TimestampsFilter, PageFilter, MultipleColumnPrefixFilter, and FamilyFilter.

This command displays all the filters present in HBase like ColumnPrefix Filter, TimestampsFilter, PageFilter, FamilyFilter, etc.

drop

Syntax: drop <table name>

A screenshot of a terminal window showing the HBase shell command 'drop 'education'' being executed. The command is highlighted with a green rectangular box. The output shows '0 row(s) in 1.4820 seconds'. A blue speech bubble with an orange border points to the command, containing the text 'dropping table'.

hbase(main):017:0>drop 'education'

We have to observe below points for drop command

- 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 disabled using disable command

- Here in above screenshot we are dropping table “education.”
- Before execution of this command, it is necessary that you disable table “education.”

drop_all

Syntax: `drop_all<"regex">`

- This command will drop all the tables matching the given regex
- Tables have to disable first before executing this command using `disable_all`
- Tables with regex matching expressions are going to drop from HBase

is_enabled

Syntax: `is_enabled 'education'`

This command will verify whether the named table is enabled or not. Usually, there is a little confusion between “enable” and “is_enabled” command action, which we clear here

- Suppose a table is disabled, to use that table we have to enable it by using `enable` command
- `is_enabled` command will check either the table is enabled or not

alter

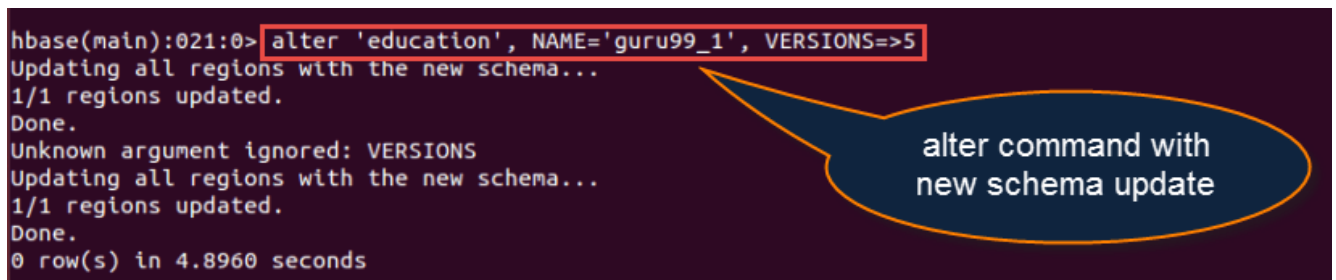
Syntax: `alter <tablename>, NAME=><column familyname>, VERSIONS=>5`

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

Examples:

In these examples, we are going to perform alter command operations on tables and on its columns. We will perform operations like

- Altering single, multiple column family names
 - Deleting column family names from table
 - Several other operations using scope attributes with table
1. To change or add the ‘tushar_1’ column family in table ‘education’ from current value to keep a maximum of 5 cell VERSIONS,
 - “education” is table name created with column name “tushar” previously
 - Here with the help of an alter command we are trying to change the column family schema to tushar_1 from tushar



```
hbase(main):021:0> alter 'education', NAME='guru99_1', VERSIONS=>5
Updating all regions with the new schema...
1/1 regions updated.
Done.
Unknown argument ignored: VERSIONS
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 4.8960 seconds
```

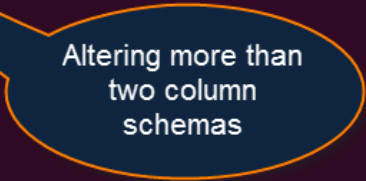
alter command with new schema update


```
hbase> alter 'education', NAME='tushar_1', VERSIONS=>5
```

2. You can also operate the alter command on several column families as well. For example, we will define two new column to our existing table “education”.

```
hbase> alter 'edu', 'tushar_1', {NAME => 'tushar_2', IN_MEMORY => true}, {NAME => 'tushar_3', VERSIONS => 5}
```

```
hbase(main):028:0> alter 'edu', 'guru99_1', {NAME=>'guru99_2', IN_MEMORY=>true}, {NAME=>'guru99_3', VERSIONS=>5}
Updating all regions with the new schema...
1/1 regions updated.
Done.
Updating all regions with the new schema...
1/1 regions updated.
Done.
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 5.9090 seconds
```



- We can change more than one column schemas at a time using this command
- tushar_2 and tushar_3 as shown in above screenshot are the two new column names that we have defined for the table education
- We can see the way of using this command in the previous screen shot

3. In this step, we will see how to delete column family from the table. To delete the ‘f1’ column family in table ‘education’.

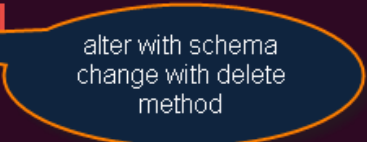
Use one of these commands below,

```
hbase> alter 'education', NAME => 'f1', METHOD => 'delete'
```

```
hbase> alter 'education', 'delete' => 'tushar_1'
```

- In this command, we are trying to delete the column space name tushar_1 that we previously created in the first step

```
hbase(main):001:0> alter 'education', 'delete'=>'guru99_1'
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.4350 seconds
```




4. As shown in the below screen shots, it shows two steps – how to change table scope attribute and how to remove the table scope attribute.

Syntax: alter <'tablename'>, MAX_FILESIZE=>'132545224'

```
hbase(main):002:0> alter 'education', MAX_FILESIZE=>'132545224'
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 1.9410 seconds
```

1



```
hbase(main):003:0> alter 'education', METHOD => 'table_att_unset', NAME => 'MAX_FILESIZE'
```

2

Step 1) You can change table-scope attributes like MAX_FILESIZE, READONLY, MEMSTORE_FLUSH_SIZE, DEFERRED_LOG_FLUSH, etc. These can be put at the end; for example, to change the max size of a region to 128MB or any other memory value we use this command.

Usage:

- We can use MAX_FILESIZE with the table as scope attribute as above
- The number represent in MAX_FILESIZE is in term of memory in bytes

NOTE: MAX_FILESIZE Attribute Table scope will be determined by some attributes present in the HBase. MAX_FILESIZE also come under table scope attributes.

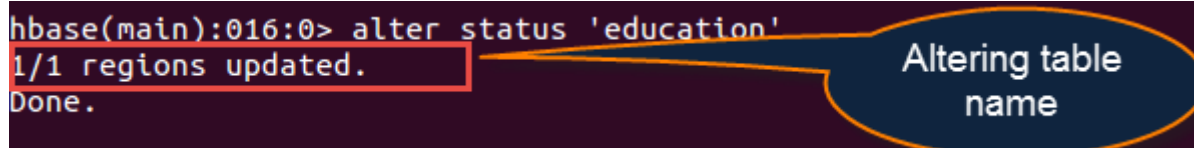
Step 2) You can also remove a table-scope attribute using table_att_unset method. If you see the command

```
alter 'education', METHOD => 'table_att_unset', NAME => 'MAX_FILESIZE'
```

- The above screen shot shows altered table name with scope attributes
- Method table_att_unset is used to unset attributes present in the table
- The second instance we are unsetting attribute MAX_FILESIZE
- After execution of the command, it will simply unset MAX_FILESIZE attribute from "education" table.

alter_status

Syntax: alter_status 'education'



- Through this command, you can get the status of the alter command
- Which indicates the number of regions of the table that have received the updated schema pass table name
- Here in above screen shot it shows 1/1 regions updated. It means that it has updated one region. After that if it successful it will display comment done.

Data manipulation commands

These commands will work on the table related to data manipulations such as putting data into a table, retrieving data from a table and deleting schema, etc.

The commands come under these are

- Count
- Put
- Get
- Delete
- Delete all
- Truncate

- Scan

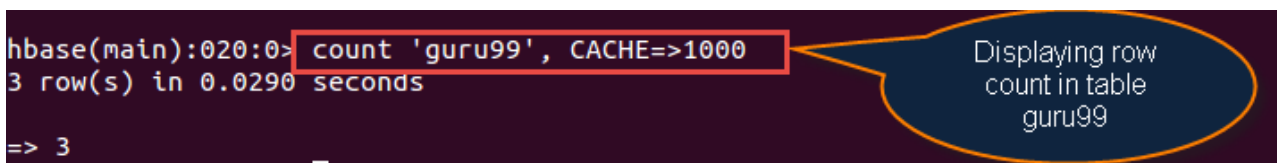
Let look into these commands usage with an example.

Count

Syntax: count <'tablename'>, CACHE =>1000

- The command will retrieve the count of a number of rows in a table. The value returned by this one is the number of rows.
- Current count is shown per every 1000 rows by default.
- Count interval may be optionally specified.
- Default cache size is 10 rows.
- Count command will work fast when it is configured with right Cache.

Example:



```
hbase(main):020:0> count 'guru99', CACHE=>1000
3 row(s) in 0.0290 seconds
=> 3
```

Displaying row count in table guru99

```
hbase> count 'tushar', CACHE=>1000
```

This example count fetches 1000 rows at a time from “tushar” table.

We can make cache to some lower value if the table consists of more rows.

But by default it will fetch one row at a time.

```
hbase>count 'tushar', INTERVAL => 100000
hbase> count 'tushar', INTERVAL =>10, CACHE=> 1000
```

If suppose if the table “tushar” having some table reference like say g.

We can run the count command on table reference also like below

```
hbase>g.count INTERVAL=>100000
hbase>g.count INTERVAL=>10, CACHE=>1000
```

Put

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

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.

Example:

- Here we are placing values into table “tushar” under row r1 and column c1

```
hbase> put 'tushar', 'r1', 'c1', 'value', 10
```

- We have placed three values, 10,15 and 30 in table “tushar” as shown in screenshot below

```
=> Hbase::Table - guru99
hbase(main):004:0> put 'guru99','r1','c1','value',10
0 row(s) in 0.1320 seconds

hbase(main):005:0> put 'guru99','r1','c1','value',15
0 row(s) in 0.0190 seconds

hbase(main):006:0> put 'guru99','r1','c1','value',30
0 row(s) in 0.0080 seconds
```

- Suppose if the table “tushar” having some table reference like say g. We can also run the command on table reference also like

```
hbase> g.put 'tushar', 'r1', 'c1', 'value', 10
```

- The output will be as shown in the above screen shot after placing values into “tushar”.

To check whether the input value is correctly inserted into the table, we use “scan” command. In the below screen shot, we can see the values are inserted correctly

```
hbase(main):008:0> scan 'guru99',{RAW=>true,VERSIONS=>1000}
ROW                                COLUMN+CELL
 r1                                column=c1:, timestamp=30, value=value
 r1                                column=c1:, timestamp=15, value=value
 r1                                column=c1:, timestamp=10, value=value
1 row(s) in 0.0340 seconds
```

Code Snippet: For Practice

```
create 'tushar', {NAME=>'Edu', VERSIONS=>213423443}
put 'tushar', 'r1', 'Edu:c1', 'value', 10
put 'tushar', 'r1', 'Edu:c1', 'value', 15
put 'tushar', 'r1', 'Edu:c1', 'value', 30
```

From the code snippet, we are doing these things

- Here we are creating a table named ‘tushar’ with the column name as “Edu.”
- By using “put” command, we are placing values into row name r1 in column “Edu” into table “tushar.”

Get

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

Here <Additional Parameters> include TIMERANGE, TIMESTAMP, VERSIONS and FILTERS.

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.

```
hbase(main):010:0> get 'guru99','r1','c1'
COLUMN                                CELL
 c1:                                timestamp=30, value=value
1 row(s) in 0.0440 seconds
```

getting records from 'guru99' using get

Examples:-

```
hbase> get 'tushar', 'r1', {COLUMN => 'c1'}
```

For table “tushar” row r1 and column c1 values will display using this command as shown in the above screen shot

```
hbase> get 'tushar', 'r1'
```

For table “tushar” row r1 values will be displayed using this command

```
hbase> get 'tushar', 'r1', {TIMERANGE => [ts1, ts2]}
```

For table “tushar” row 1 values in the time range ts1 and ts2 will be displayed using this command

```
hbase> get 'tushar', 'r1', {COLUMN => ['c1', 'c2', 'c3']}
```

For table “tushar” row r1 and column families’ c1, c2, c3 values will be displayed using this command

Delete

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

- 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 screenshot shows a terminal window with the following text:

```
hbase(main):020:0> delete 'guru99', 'r1', 'c1'
0 row(s) in 0.0680 seconds
hbase(main):021:0> scan 'guru99'
```

Annotations on the screenshot:

- A blue oval points to the `delete` command with the text "Deleting r1 from column c1".
- A red box highlights the `scan` command and its output.
- A blue oval points to the output of the `scan` command with the text "schema after deleting r1".

The output of the `scan` command is:

```
ROW          COLUMN+CELL
 r2          column=c1:, timestamp=15, value=value
 r3          column=c1:, timestamp=15, value=value
2 row(s) in 0.0760 seconds
```

Example:

```
hbase(main):020:0> delete 'tushar', 'r1', 'c1'.
```

- The above execution will delete row r1 from column family c1 in table “tushar.”
- Suppose if the table “tushar” having some table reference like say g.
- We can run the command on table reference also like **hbase> g.delete** ‘tushar’, ‘r1’, ‘c1’.

deleteall

Syntax: deleteall <'tablename'>, <'rowname'>

The screenshot shows a terminal window with the following text:

```
hbase(main):022:0> deleteall 'guru99', 'c1'
0 row(s) in 0.0220 seconds
```

- This Command will delete all cells in a given row.
- We can define optionally column names and time stamp to the syntax.

Example:-

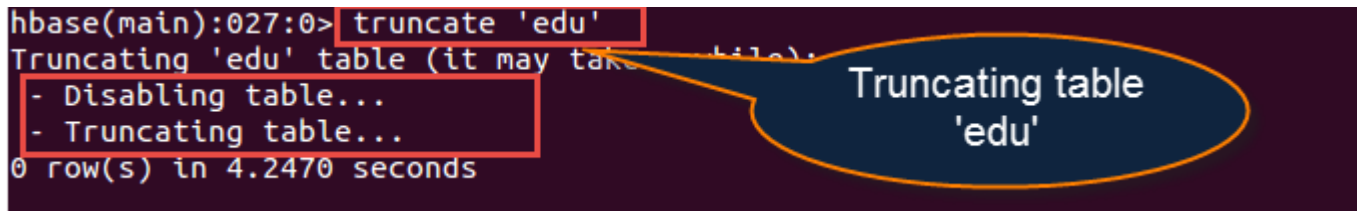
```
hbase> deleteall 'tushar', 'r1', 'c1'
```

This will delete all the rows and columns present in the table. Optionally we can mention column names in that.

Truncate

Syntax: `truncate <tablename>`

```
hbase(main):027:0> truncate 'edu'
Truncating 'edu' table (it may take a while)...
- Disabling table...
- Truncating table...
0 row(s) in 4.2470 seconds
```

A screenshot of a terminal window showing the execution of the 'truncate' command in HBase. The command 'truncate 'edu'' is entered at the prompt. The output shows the process of truncating the table, including disabling it and then truncating it. A red box highlights the command and the first two lines of output. A blue speech bubble points to the output line 'Truncating table 'edu'', containing the text 'Truncating table 'edu''.

After truncate of an hbase table, the schema will present but not the records. This command performs 3 functions; those are listed below

- Disables table if it already presents
- Drops table if it already presents
- Recreates the mentioned table

Scan

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

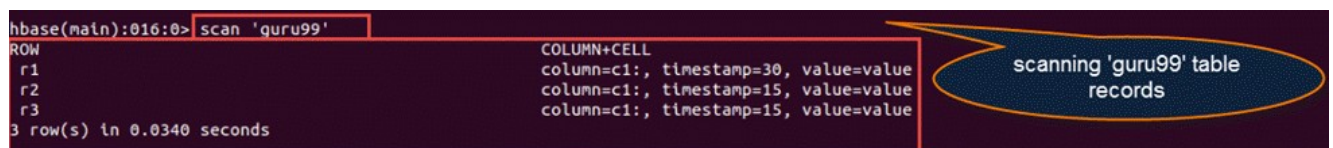
This command scans entire table and displays the table contents.

- 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.

```
scan 'tushar'
```

The output as below shown in screen shot

```
hbase(main):016:0> scan 'guru99'
ROW          COLUMN+CELL
r1           column=c1:, timestamp=30, value=value
r2           column=c1:, timestamp=15, value=value
r3           column=c1:, timestamp=15, value=value
3 row(s) in 0.0340 seconds
```

A screenshot of a terminal window showing the execution of the 'scan' command in HBase. The command 'scan 'guru99'' is entered at the prompt. The output shows the scanning of the table, including the row keys and the column values. A red box highlights the command and the first three lines of output. A blue speech bubble points to the output line 'scanning 'guru99' table records', containing the text 'scanning 'guru99' table records'.

In the above screen shot

- It shows “tushar” table with column name and values
- It consists of three row values r1, r2, r3 for single column value c1
- It displays the values associated with rows

Examples:-

The different usages of scan command

Command

Usage

```
scan '.META.', {COLUMNS =>
'info:regioninfo'}
```

It display all the meta data information related to columns that are present in the tables in HBase

```
scan 'tushar', {COLUMNS => ['c1',
'c2'], LIMIT => 10, STARTROW =>
'xyz'}
```

It display contents of table tushar with their column families c1 and c2 limiting the values to 10

```
scan 'tushar', {COLUMNS => 'c1',
TIMERANGE => [1303668804,
1303668904]}
```

It display contents of tushar with its column name c1 with the values present in between the mentioned time range attribute value

```
scan 'tushar', {RAW => true,
VERSIONS =>10}
```

In this command RAW=> true provides advanced feature like to display all the cell values present in the table tushar

Code Example:

First create table and place values into table

```
create 'tushar', {NAME=>'e', VERSIONS=>2147483647}
put 'tushar', 'r1', 'e:c1', 'value', 10
put 'tushar', 'r1', 'e:c1', 'value', 12
put 'tushar', 'r1', 'e:c1', 'value', 14
delete 'tushar', 'r1', 'e:c1', 11
```

Input Screenshot:

```
hbase(main):009:0> create 'guru99', {NAME=>'e', VERSIONS=>6}
0 row(s) in 1.3790 seconds

=> Hbase::Table - guru99
hbase(main):010:0> put 'guru99', 'r1', 'e:c1', 'value', 10
0 row(s) in 0.0880 seconds

hbase(main):011:0> put 'guru99', 'r1', 'e:c1', 'value', 12
0 row(s) in 0.0090 seconds

hbase(main):012:0> put 'guru99', 'r1', 'e:c1', 'value', 14
0 row(s) in 0.0110 seconds

hbase(main):013:0> delete 'guru99', 'r1', 'e:c1', 11
0 row(s) in 0.0270 seconds
```

Creating Table guru99 and placing values in guru99

If we run scan command

Query: scan 'tushar', {RAW=>true, VERSIONS=>1000}

It will display output shown in below.

Output screen shot:

```
hbase(main):014:0> scan 'guru99'
ROW          COLUMN+CELL
 r1          column=e:c1, timestamp=14, value=value
1 row(s) in 0.0520 seconds

hbase(main):015:0> scan 'guru99', {RAW=>true, VERSIONS=>1000}
ROW          COLUMN+CELL
 r1          column=e:c1, timestamp=14, value=value
 r1          column=e:c1, timestamp=12, value=value
 r1          column=e:c1, timestamp=11, type=DeleteColumn
 r1          column=e:c1, timestamp=10, value=value
1 row(s) in 0.0640 seconds
```

Output Of
Code Snippet

The output shown in above screen shot gives the following information

- Scanning tushar table with attributes RAW=>true, VERSIONS=>1000
- Displaying rows with column families and values
- In the third row, the values displayed shows deleted value present in the column
- The output displayed by it is random; it cannot be same order as the values that we inserted in the table