Work Integrated Learning Programmes

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Computer Science & Information Systems

Big Data Systems – Lab Sheet

HBASE

Objectives

Students should be able to

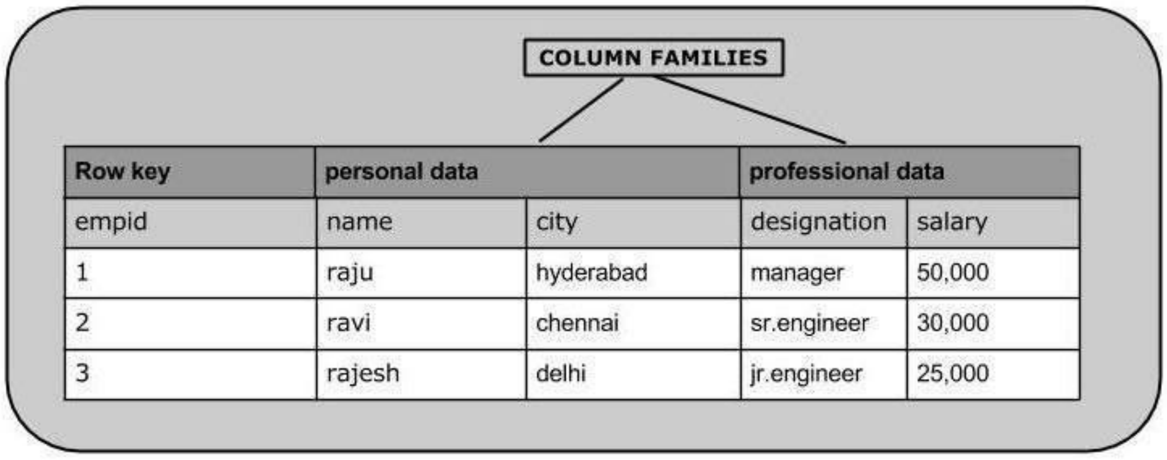
A. Gain understanding about HBASE
B. Store, manipulate and retrieve data using HBASE queries

Introduction to HBASE

HBASE is column oriented non-relational database management system that provides capability to access large files in HDFS. It is a key-value store with no fixed schema unlike RDBMS. It is typically used for analytical queries that access specific columns. It is strongly consistent data store.

Columnar Storage

Data in a row is collection of column families with each column being key-value pair.



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

The advantage of having creating column families is that queries that access information in a particular column family will run faster. For example, queries that access only personal data or only professional data in the above snapshot will run faster.

HBASE QUERIES

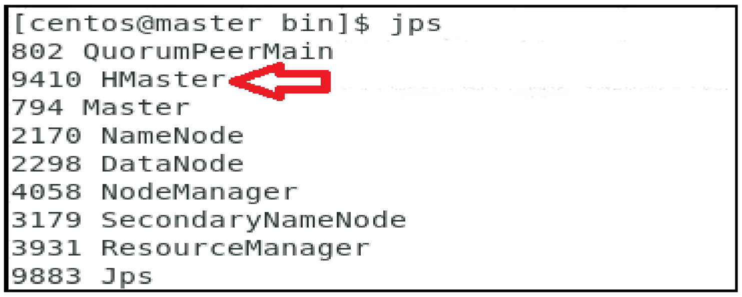
In this section we will discuss various clauses/operators used for HBASE queries.

In order to run HBASE queries, we need to start HMaster as follows.

```
[centos@master~]cd /opt/hbase-2.4.15/bin
[centos@master bin]$ ./start-hbase.sh
```

You can check if HMaster is running by executing jps as follows

```
[centos@master bin]$ jps
```



```
[centos@master bin]$ jps
802 QuorumPeerMain
9410 HMaster
794 Master
2170 NameNode
2298 DataNode
4058 NodeManager
3179 SecondaryNameNode
3931 ResourceManager
9883 Jps
```

In order to access HBASE shell use the following command

```
[centos@master bin]$ hbase shell
```

CREATE

The create command is used to create a table. While creating a table you must specify the table name and the Column Family name. The syntax to create a table in HBase shell is shown below.

Syntax

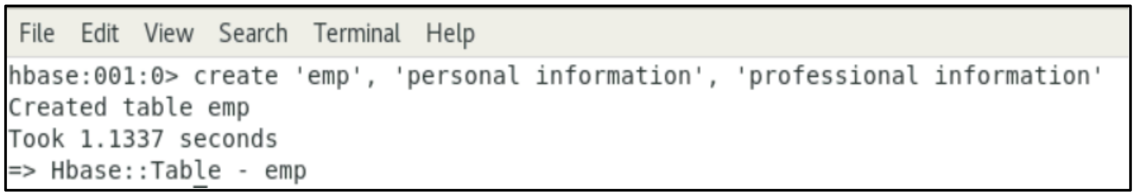
```
hbase> create ‘<table-name>’,‘<column-family-name>’
```

Example

```
hbase> create ‘emp’ ,‘personal information’ , ‘professional
information’
```

The above command will create a table with emp having 2 column families ‘personal information’ and ‘professional information’. You need to specify at least one column family in order to create a table.

Output



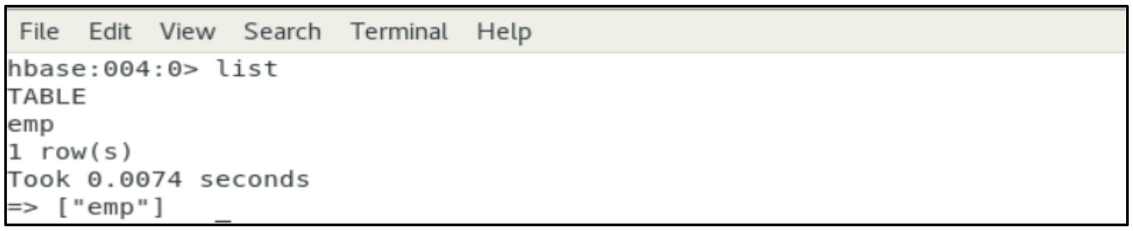
```
hbase:001:0> create 'emp', 'personal information', 'professional information'
Created table emp
Took 1.1337 seconds
=> Hbase::Table - emp
```

LIST TABLES

The list command is used to list all the tables;

```
hbase> list;
```

The above command will list all the tables as follows.



```
hbase:004:0> list
TABLE
emp
1 row(s)
Took 0.0074 seconds
=> ["emp"]
```

DISABLING A TABLE

To delete a table or change its settings, you need to first disable the table using the disable command.The syntax is as follows.

Syntax

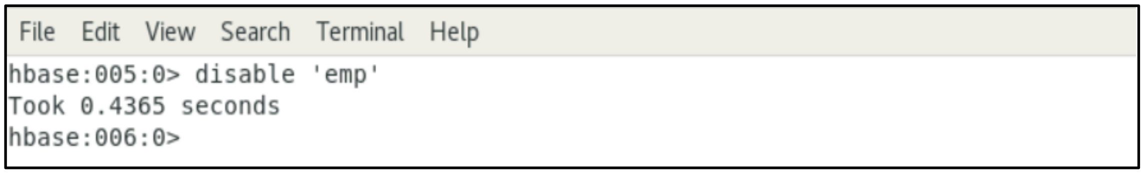
```
hbase> disable ‘<table-name>’
```

Example

```
hbase> disable ‘emp’
```

The above command will disable the table ‘emp’.

Output



```
hbase:005:0> disable 'emp'
Took 0.4365 seconds
hbase:006:0>
```

DISABLE_ALL

This command is used to disable all the tables matching the given regex. The syntax for disable_all command is given below.

Syntax

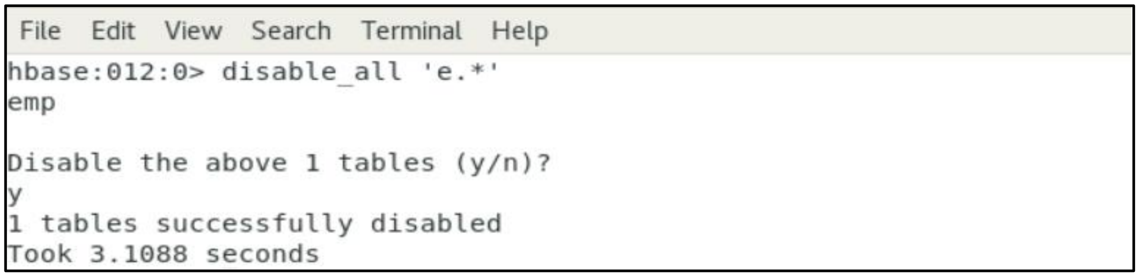
```
hbase>disable_all ‘<regex>’
```

Example

```
hbase>disable_all ‘e.*’
```

The above command will disable all the tables starting with e.

Output



```
hbase:012:0> disable_all 'e.*'
emp

Disable the above 1 tables (y/n)?
y
1 tables successfully disabled
Took 3.1088 seconds
```

ENABLING A TABLE

You need to enable a disabled table before you can use it. The syntax is as follows.

Syntax

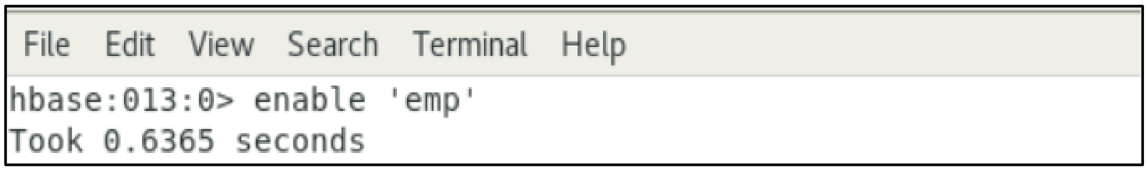
```
hbase> enable '<table-name>'
```

Example

```
hbase> enable 'emp'
```

The above command will enable the table 'emp'.

Output



```
hbase:013:0> enable 'emp'
Took 0.6365 seconds
```

DESCRIBE

The DESCRIBE command returns the description of the table. The syntax is as follows

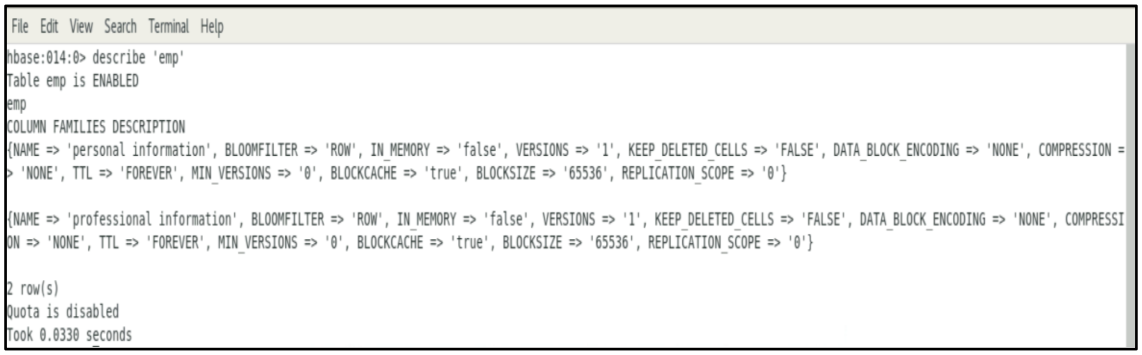
Syntax

```
hbase> describe '<table-name>'
```

Example

```
hbase> describe 'emp'
```

Output



```
hbase:014:0> describe 'emp'
Table emp is ENABLED
emp
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal information', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION =
> 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

{NAME => 'professional information', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSI
ON => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

2 row(s)
Quota is disabled
Took 0.0330 seconds
```

ALTER

The ALTER command can be used to make changes to existing table. Using this command, you can change the maximum number of cells of a column family, set and delete table scope operators, and delete a column family from a table.

Use of alter to change maximum number of cells of a column family

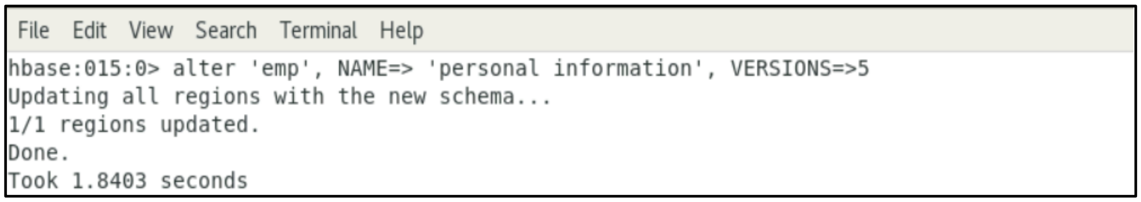
Syntax

```
hbase> alter ‘<table-name>’, NAME=> ‘<cf-name>’, VERSIONS => 5
```

Example

```
hbase> alter ‘emp’, NAME=> ‘personal information’, VERSIONS =>
5
```

Output



```
hbase:015:0> alter 'emp', NAME=> 'personal information', VERSIONS=>5
Updating all regions with the new schema...
1/1 regions updated.
Done.
Took 1.8403 seconds
```

Use of alter to set table scope operators

Using alter, you can set table scope operators such as MAX_FILESIZE, READONLY, MEMSTORE_FLUSHSIZE, DEFERRED_LOG_FLUSH, etc.

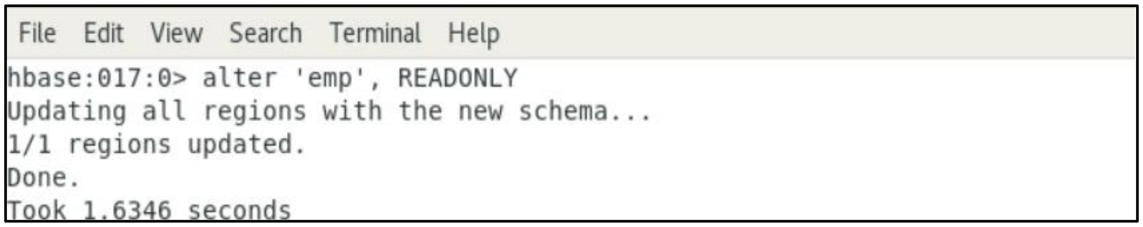
Syntax

```
hbase> alter ‘<table-name>’, <table-scope-operator>
```

Example

```
hbase> alter ‘emp’, READONLY
```

Output



```
hbase:017:0> alter 'emp', READONLY
Updating all regions with the new schema...
1/1 regions updated.
Done.
Took 1.6346 seconds
```

Use of alter to remove table scope operators

Using alter, you can also remove table scope operators such as MAX_FILESIZE, READONLY, MEMSTORE_FLUSHSIZE, DEFERRED_LOG_FLUSH, etc.

Syntax

```
hbase> alter '<table-name>', <table-scope-operator>
```

Example

```
hbase> alter 'emp', METHOD => 'table_att_unset',
NAME=>READONLY
```

Use of alter to remove a column family

Using alter, you can also delete a column family.

Syntax

```
hbase> alter '<table-name>', delete=> 'cf-name'
```

Example

```
hbase> alter 'emp', delete=> 'professional information'
```

EXISTS

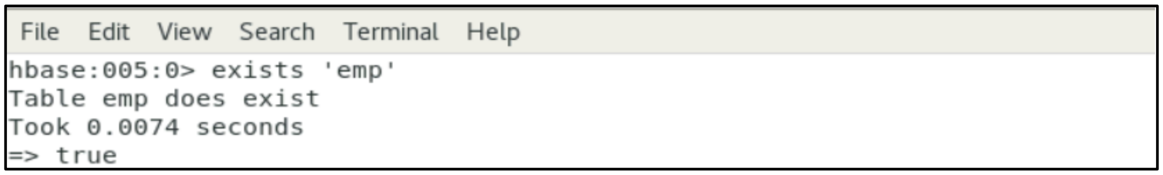
You can verify the existence of a table using the exists command. The following example shows how to use this command.

Syntax

```
hbase> exists '<table-name>'
```

Example

```
hbase> exists 'emp'
```



```
hbase:005:0> exists 'emp'
Table emp does exist
Took 0.0074 seconds
=> true
```

DROP A TABLE

Using the drop command, you can delete a table. Before dropping a table, you have to disable it.

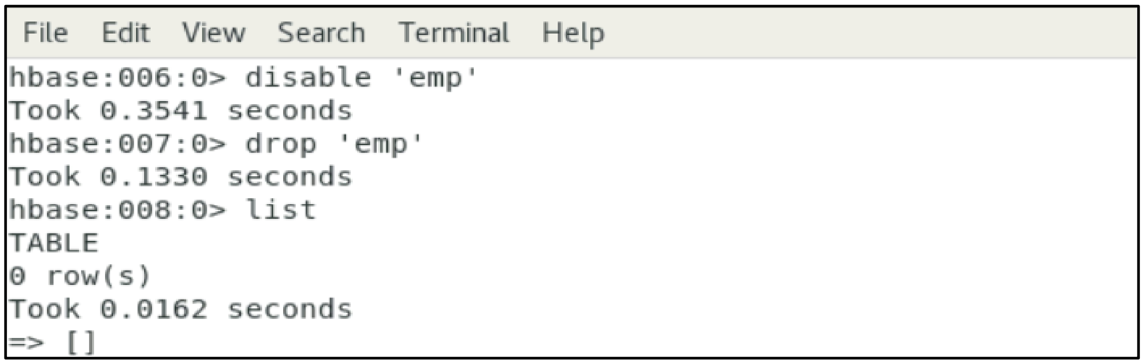
Syntax

```
hbase> disable '<table-name>'
hbase> drop '<table-name>'
```

Example

```
hbase> disable 'emp'
hbase> drop 'emp'
```

Output



```
hbase:006:0> disable 'emp'
Took 0.3541 seconds
hbase:007:0> drop 'emp'
Took 0.1330 seconds
hbase:008:0> list
TABLE
0 row(s)
Took 0.0162 seconds
=> []
```

DROP ALL

The drop_allcommand is used to drop the tables matching the "regex" given in the command. Before dropping a table, you must disable it. Its syntax is as follows:

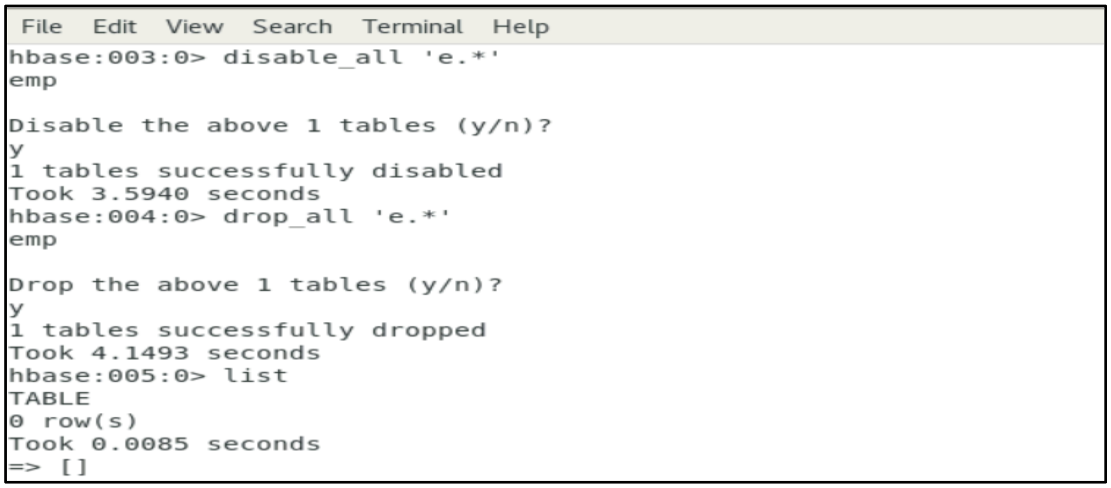
Syntax

```
hbase>disable_all '<regex>'
hbase>drop_all '<regex>'
```

Example

```
hbase>disable_all 'e.*'
hbase>drop_all 'e.*'
```

The above command will first disable all the tables starting with e and then drop all those tables.



```
hbase:003:0> disable_all 'e.*'
emp

Disable the above 1 tables (y/n)?
y
1 tables successfully disabled
Took 3.5940 seconds
hbase:004:0> drop_all 'e.*'
emp

Drop the above 1 tables (y/n)?
y
1 tables successfully dropped
Took 4.1493 seconds
hbase:005:0> list
TABLE
0 row(s)
Took 0.0085 seconds
=> []
```

INSERTING DATA INTO TABLES

Using **put** command, you can insert rows into a table. Its syntax is as follows:

Syntax

```
hbase> put '<tab-name>', 'row-num', '<colfamily:colname>',
'<value>'
```

Example

```
hbase> put 'emp','1', 'personal information:name', 'raju'
```

SCAN

The scan command is used to view the data in HBASE table.

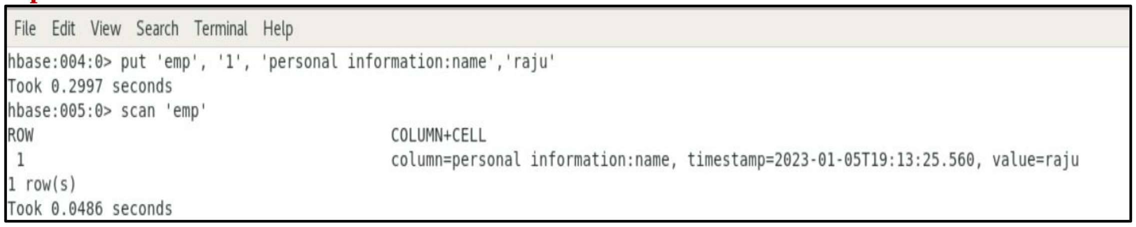
Syntax

```
hbase> scan '<table-name>'
```

Example

```
hbase> scan 'emp'
```

Output



```
hbase:004:0> put 'emp', '1', 'personal information:name','raju'
Took 0.2997 seconds
hbase:005:0> scan 'emp'
ROW                                        COLUMN+CELL
 1                                         column=personal information:name, timestamp=2023-01-05T19:13:25.560, value=raju
1 row(s)
Took 0.0486 seconds
```

UPDATING DATA IN TABLES

Using **put** command, you can also update data stored in a table. Its syntax is as follows:

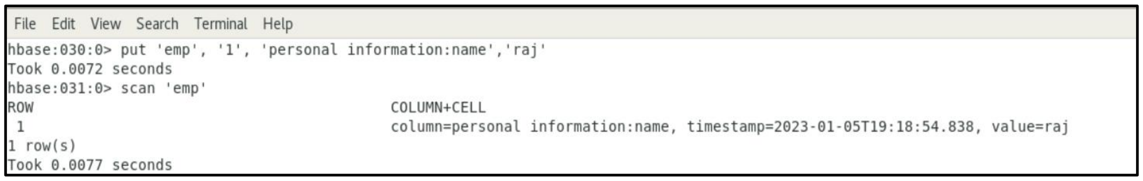
Syntax

```
hbase> put '<tab-name>', 'row-num', '<colfamily:colname>',
'<new-value>'
```

Example

```
hbase> put 'emp' '1' 'personal information:name', 'raj'
```

Output



```
hbase:030:0> put 'emp', '1', 'personal information:name','raj'
Took 0.0072 seconds
hbase:031:0> scan 'emp'
ROW                                          COLUMN+CELL
 1                                           column=personal information:name, timestamp=2023-01-05T19:18:54.838, value=raj
1 row(s)
Took 0.0077 seconds
```

READING DATA FROM TABLES

Using **get** command, you can read the data from a table. Its syntax is as follows:

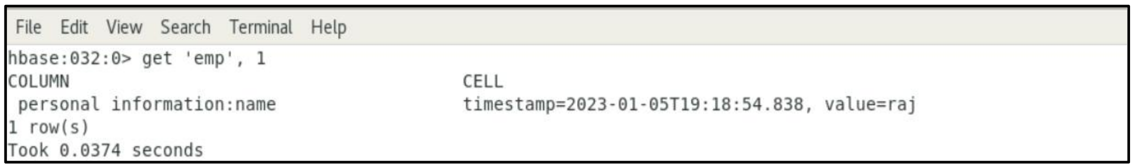
Syntax: Reading a specific row

```
hbase> get '<tab-name>', 'row-num'
```

Example

```
hbase> get 'emp', '1'
```

Output



```
hbase:032:0> get 'emp', 1
COLUMN                                  CELL
 personal information:name              timestamp=2023-01-05T19:18:54.838, value=raj
1 row(s)
Took 0.0374 seconds
```

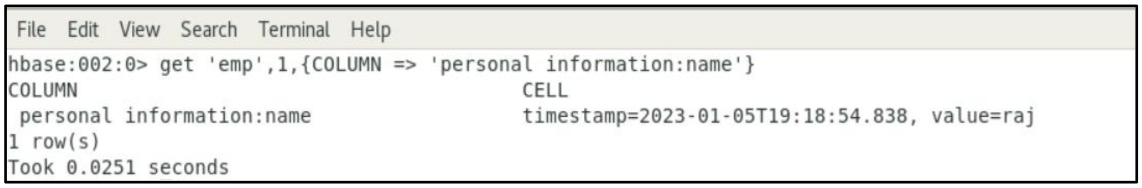
Syntax: Reading a specific column

```
hbase> get ‘<tab-name>’,‘row-num’ {COLUMN=> ‘cf-name:col-name’}
```

Example

```
hbase> get ‘emp’ ‘1’ {COLUMN => ‘personalinformation:name’}
```

Output



```
hbase:002:0> get 'emp',1,{COLUMN => 'personal information:name'}
COLUMN                                      CELL
 personal information:name                  timestamp=2023-01-05T19:18:54.838, value=raj
1 row(s)
Took 0.0251 seconds
```

DELETE DATA FROM TABLES

Using **delete** command, you can delete specific cells in a table. Its syntax is as follows:

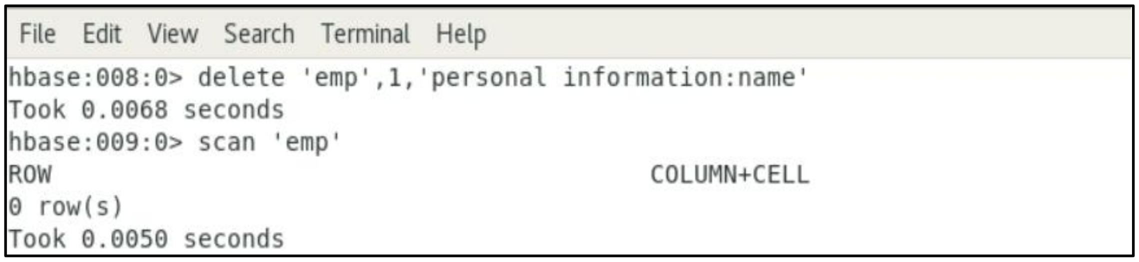
Syntax:

```
hbase> delete ‘<tab-name>’, ‘row-num’, ‘<col-name>’, ‘<time-stamp>’
```

Example

```
hbase>delete,‘emp’ 1, ‘personal information:name’
```

Output



```
hbase:008:0> delete 'emp',1,'personal information:name'
Took 0.0068 seconds
hbase:009:0> scan 'emp'
ROW                                    COLUMN+CELL
0 row(s)
Took 0.0050 seconds
```

Using the “deleteall” command, you can delete all the cells in a row.

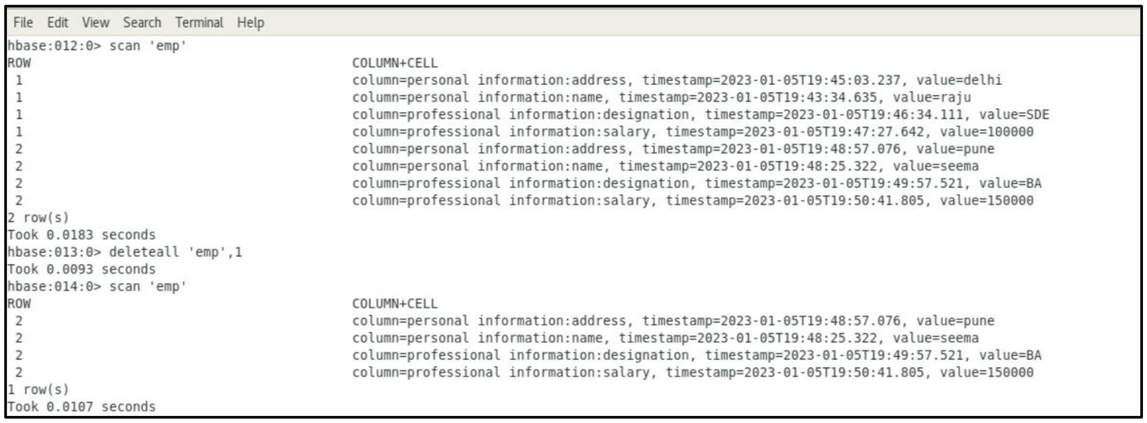
Syntax:

```
hbase>deleteall '<tab-name>','row-num'
```

Example

```
hbase>deleteall 'emp' '1'
```

Output



```
hbase:012:0> scan 'emp'
ROW                                           COLUMN+CELL
 1                                            column=personal information:address, timestamp=2023-01-05T19:45:03.237, value=delhi
 1                                            column=personal information:name, timestamp=2023-01-05T19:43:34.635, value=raju
 1                                            column=professional information:designation, timestamp=2023-01-05T19:46:34.111, value=SDE
 1                                            column=professional information:salary, timestamp=2023-01-05T19:47:27.642, value=100000
 2                                            column=personal information:address, timestamp=2023-01-05T19:48:57.076, value=pune
 2                                            column=personal information:name, timestamp=2023-01-05T19:48:25.322, value=seema
 2                                            column=professional information:designation, timestamp=2023-01-05T19:49:57.521, value=BA
 2                                            column=professional information:salary, timestamp=2023-01-05T19:50:41.805, value=150000
2 row(s)
Took 0.0183 seconds
hbase:013:0> deleteall 'emp',1
Took 0.0093 seconds
hbase:014:0> scan 'emp'
ROW                                           COLUMN+CELL
 2                                            column=personal information:address, timestamp=2023-01-05T19:48:57.076, value=pune
 2                                            column=personal information:name, timestamp=2023-01-05T19:48:25.322, value=seema
 2                                            column=professional information:designation, timestamp=2023-01-05T19:49:57.521, value=BA
 2                                            column=professional information:salary, timestamp=2023-01-05T19:50:41.805, value=150000
1 row(s)
Took 0.0107 seconds
```

COUNT

The count command is used to count number of rows in HBASE table.

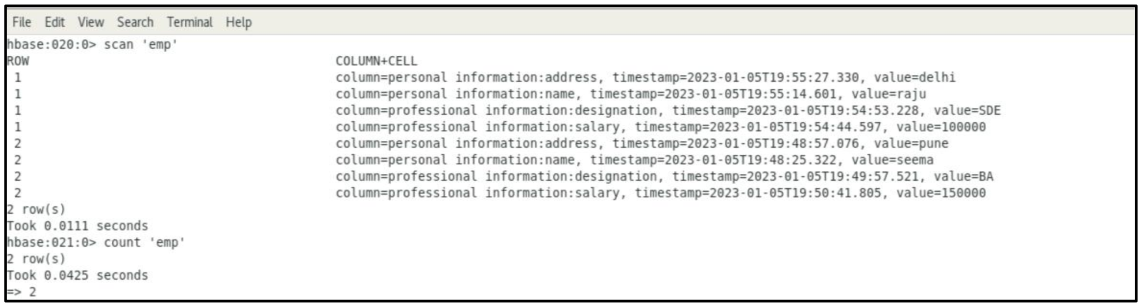
Syntax

```
hbase> count '<table-name>'
```

Example

```
hbase> count 'emp'
```

Output



```
hbase:020:0> scan 'emp'
ROW                                           COLUMN+CELL
 1                                            column=personal information:address, timestamp=2023-01-05T19:55:27.330, value=delhi
 1                                            column=personal information:name, timestamp=2023-01-05T19:55:14.601, value=raju
 1                                            column=professional information:designation, timestamp=2023-01-05T19:54:53.228, value=SDE
 1                                            column=professional information:salary, timestamp=2023-01-05T19:54:44.597, value=100000
 2                                            column=personal information:address, timestamp=2023-01-05T19:48:57.076, value=pune
 2                                            column=personal information:name, timestamp=2023-01-05T19:48:25.322, value=seema
 2                                            column=professional information:designation, timestamp=2023-01-05T19:49:57.521, value=BA
 2                                            column=professional information:salary, timestamp=2023-01-05T19:50:41.805, value=150000
2 row(s)
Took 0.0111 seconds
hbase:021:0> count 'emp'
2 row(s)
Took 0.0425 seconds
=> 2
```

TRUNCATE

The truncate command disables drop and recreates a table

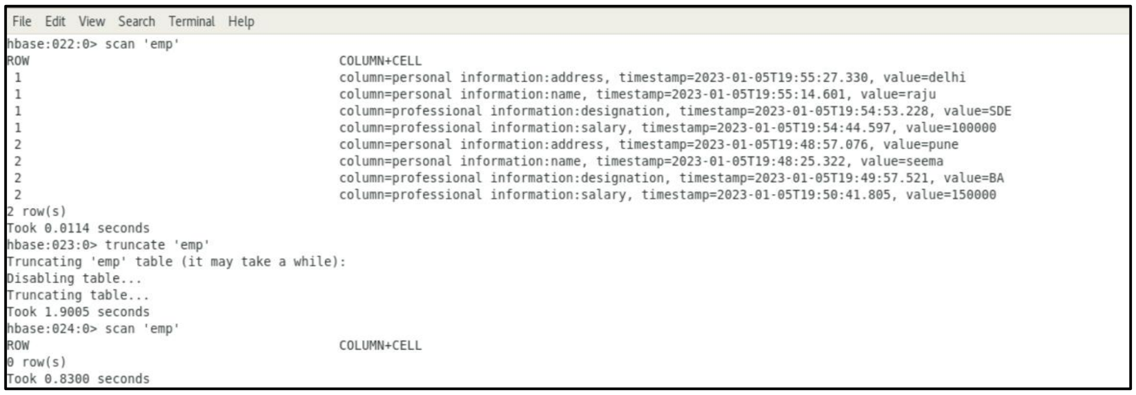
Syntax

```
hbase> truncate ‘<table-name>’
```

Example

```
hbase> truncate ‘emp’
```

Output



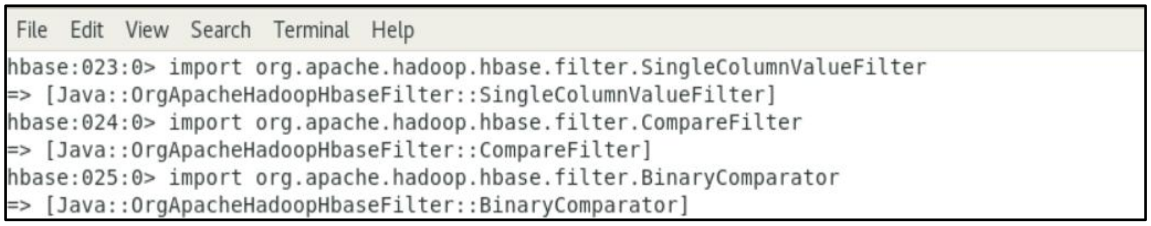
```
hbase:022:0> scan 'emp'
ROW                                          COLUMN+CELL
 1                                           column=personal information:address, timestamp=2023-01-05T19:55:27.330, value=delhi
 1                                           column=personal information:name, timestamp=2023-01-05T19:55:14.601, value=raju
 1                                           column=professional information:designation, timestamp=2023-01-05T19:54:53.228, value=SDE
 1                                           column=professional information:salary, timestamp=2023-01-05T19:54:44.597, value=100000
 2                                           column=personal information:address, timestamp=2023-01-05T19:48:57.076, value=pune
 2                                           column=personal information:name, timestamp=2023-01-05T19:48:25.322, value=seema
 2                                           column=professional information:designation, timestamp=2023-01-05T19:49:57.521, value=BA
 2                                           column=professional information:salary, timestamp=2023-01-05T19:50:41.805, value=150000
2 row(s)
Took 0.0114 seconds
hbase:023:0> truncate 'emp'
Truncating 'emp' table (it may take a while):
Disabling table...
Truncating table...
Took 1.9005 seconds
hbase:024:0> scan 'emp'
ROW                                          COLUMN+CELL
0 row(s)
Took 0.8300 seconds
```

SingleColumnValueFilter

In order to filter the rows on the HBase shell using Scan, you need to import the org.apache.hadoop.hbase.filter.SingleColumnValueFilter class along with some other class explained below

Syntax

```
hbase>import org.apache.hadoop.hbase.filter.SingleColumnValueFilter
hbase> import org.apache.hadoop.hbase.filter.CompareFilter
hbase> import org.apache.hadoop.hbase.filter.BinaryComparator
```

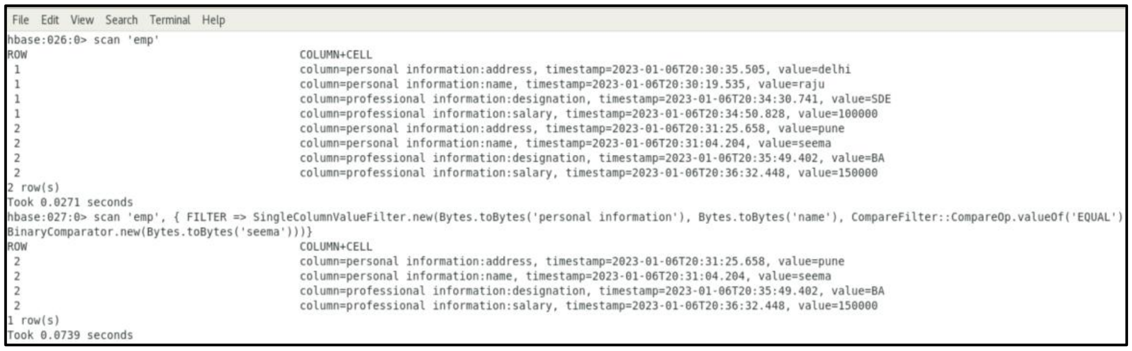


```
hbase:023:0> import org.apache.hadoop.hbase.filter.SingleColumnValueFilter
=> [Java::OrgApacheHadoopHbaseFilter::SingleColumnValueFilter]
hbase:024:0> import org.apache.hadoop.hbase.filter.CompareFilter
=> [Java::OrgApacheHadoopHbaseFilter::CompareFilter]
hbase:025:0> import org.apache.hadoop.hbase.filter.BinaryComparator
=> [Java::OrgApacheHadoopHbaseFilter::BinaryComparator]
```

Example-1: Comparison with name

```
hbase>scan 'emp', { FILTER
=>SingleColumnValueFilter.new(Bytes.toBytes('personal information'),
Bytes.toBytes('name'),
CompareFilter::CompareOp.valueOf('EQUAL'),BinaryComparator.new(Bytes
.toBytes('seema')))}
```

The above query will return the details corresponding to name ‘seema’ as follows

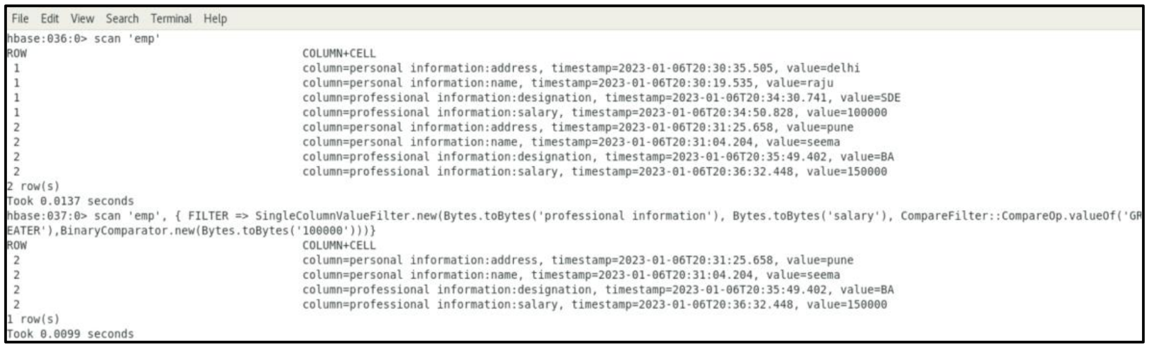


```
hbase:026:0> scan 'emp'
ROW                                        COLUMN+CELL
 1                                         column=personal information:address, timestamp=2023-01-06T20:30:35.505, value=delhi
 1                                         column=personal information:name, timestamp=2023-01-06T20:30:19.535, value=raju
 1                                         column=professional information:designation, timestamp=2023-01-06T20:34:30.741, value=SDE
 1                                         column=professional information:salary, timestamp=2023-01-06T20:34:50.828, value=100000
 2                                         column=personal information:address, timestamp=2023-01-06T20:31:25.658, value=pune
 2                                         column=personal information:name, timestamp=2023-01-06T20:31:04.204, value=seema
 2                                         column=professional information:designation, timestamp=2023-01-06T20:35:49.402, value=BA
 2                                         column=professional information:salary, timestamp=2023-01-06T20:36:32.448, value=150000
2 row(s)
Took 0.0271 seconds
hbase:027:0> scan 'emp', { FILTER => SingleColumnValueFilter.new(Bytes.toBytes('personal information'), Bytes.toBytes('name'), CompareFilter::CompareOp.valueOf('EQUAL')
BinaryComparator.new(Bytes.toBytes('seema')))}
ROW                                        COLUMN+CELL
 2                                         column=personal information:address, timestamp=2023-01-06T20:31:25.658, value=pune
 2                                         column=personal information:name, timestamp=2023-01-06T20:31:04.204, value=seema
 2                                         column=professional information:designation, timestamp=2023-01-06T20:35:49.402, value=BA
 2                                         column=professional information:salary, timestamp=2023-01-06T20:36:32.448, value=150000
1 row(s)
Took 0.0739 seconds
```

Example-2: Comparison with salary

```
hbase>scan 'emp', { FILTER
=>SingleColumnValueFilter.new(Bytes.toBytes('professional
information'), Bytes.toBytes('salary'),
CompareFilter::CompareOp.valueOf('GREATER'),BinaryComparator.new(Byt
es.toBytes('100000')))}
```

The above query will return the rows where salary>100000 as follows

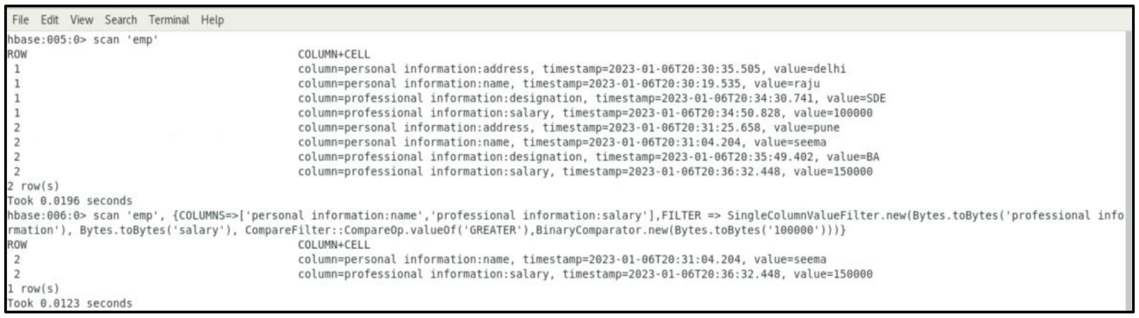


```
hbase:036:0> scan 'emp'
ROW                                        COLUMN+CELL
 1                                         column=personal information:address, timestamp=2023-01-06T20:30:35.505, value=delhi
 1                                         column=personal information:name, timestamp=2023-01-06T20:30:19.535, value=raju
 1                                         column=professional information:designation, timestamp=2023-01-06T20:34:30.741, value=SDE
 1                                         column=professional information:salary, timestamp=2023-01-06T20:34:50.828, value=100000
 2                                         column=personal information:address, timestamp=2023-01-06T20:31:25.658, value=pune
 2                                         column=personal information:name, timestamp=2023-01-06T20:31:04.204, value=seema
 2                                         column=professional information:designation, timestamp=2023-01-06T20:35:49.402, value=BA
 2                                         column=professional information:salary, timestamp=2023-01-06T20:36:32.448, value=150000
2 row(s)
Took 0.0137 seconds
hbase:037:0> scan 'emp', { FILTER => SingleColumnValueFilter.new(Bytes.toBytes('professional information'), Bytes.toBytes('salary'), CompareFilter::CompareOp.valueOf('GR
EATER'),BinaryComparator.new(Bytes.toBytes('100000')))}
ROW                                        COLUMN+CELL
 2                                         column=personal information:address, timestamp=2023-01-06T20:31:25.658, value=pune
 2                                         column=personal information:name, timestamp=2023-01-06T20:31:04.204, value=seema
 2                                         column=professional information:designation, timestamp=2023-01-06T20:35:49.402, value=BA
 2                                         column=professional information:salary, timestamp=2023-01-06T20:36:32.448, value=150000
1 row(s)
Took 0.0099 seconds
```

Example-3: Displaying selected columns based on filtering condition

```
hbase>scan 'emp', {COLUMNS=>['personal
information:name','professional information:salary'],FILTER
=>SingleColumnValueFilter.new(Bytes.toBytes('professional
information'), Bytes.toBytes('salary'),
CompareFilter::CompareOp.valueOf('GREATER'),BinaryComparator.new(Byt
es.toBytes('100000')))}
```

The above query will return the name and salary where salary>100000



```
hbase:005:0> scan 'emp'
ROW                                  COLUMN+CELL
 1                                   column=personal information:address, timestamp=2023-01-06T20:30:35.505, value=delhi
 1                                   column=personal information:name, timestamp=2023-01-06T20:30:19.535, value=raju
 1                                   column=professional information:designation, timestamp=2023-01-06T20:34:30.741, value=SDE
 1                                   column=professional information:salary, timestamp=2023-01-06T20:34:50.828, value=100000
 2                                   column=personal information:address, timestamp=2023-01-06T20:31:25.658, value=pune
 2                                   column=personal information:name, timestamp=2023-01-06T20:31:04.204, value=seema
 2                                   column=professional information:designation, timestamp=2023-01-06T20:35:49.402, value=BA
 2                                   column=professional information:salary, timestamp=2023-01-06T20:36:32.448, value=150000
2 row(s)
Took 0.0196 seconds
hbase:006:0> scan 'emp', {COLUMNS=>['personal information:name','professional information:salary'],FILTER => SingleColumnValueFilter.new(Bytes.toBytes('professional info
rmation'), Bytes.toBytes('salary'), CompareFilter::CompareOp.valueOf('GREATER'),BinaryComparator.new(Bytes.toBytes('100000')))}
ROW                                  COLUMN+CELL
 2                                   column=personal information:name, timestamp=2023-01-06T20:31:04.204, value=seema
 2                                   column=professional information:salary, timestamp=2023-01-06T20:36:32.448, value=150000
1 row(s)
Took 0.0123 seconds
```

Outputs/Results

- Students should be able to appreciate the usage of HBASE.
- Students should be able to appreciate column families of HBASE
- Students should be able to execute queries using various options available

Observations

Students should carefully observe the syntax of HBASE queries and the output

References

- Tutorial point
- Spark By Examples