

HBASE LAB ON CLOUDERA *DOCKER CONTAINER*

BY: MANOJ KUMAR-09916

SAMAN SYED-11581

IRTIJA ALAM-19826

Before 1970, the solution for maintenance and data storage related problems was RDBMS. But once big data stepped in, companies found the benefit of processing big data and started using solutions like Hadoop

For storing big data, Hadoop makes use of distributed file system. And to process, it uses MapReduce. Hadoop excels in storing and processing of massive data of various formats consisting of arbitrary, semi-, or maybe unstructured

- ***What is HBase?***

HBase is a column-oriented and open-source distributed database system, in a Hadoop ecosystem. HBase is written in java, initially it was Google Big Table, subsequently it was re-named as HBase and is written mainly in Java. Apache HBase is required for Big Data applications in real - time systems.

HBase can store bulk data from terabytes to petabytes. The tables in HBase are composed of billions of rows and millions of columns. HBase is designed for low latency operations which have some different characteristics compared to conventional relational models.

- ***Key Properties:***

HBase is not a conventional Relational (RDBMS) database. HBase was created to scale through a cluster. certain of HBase's key properties include:

- NoSQL: Distributed and possible to scale. HBase groups rows into regions that define how the table data in a cluster is split over different nodes. Whenever a region gets too big, dividing the load over more servers is automatically split.

- Wide column: HBase stores data in a table-like format with the ability to store billions and millions of rows & columns respectively. Columns can be grouped together in column families, allowing for the physical distribution of row values on various cluster nodes.
- Unstructured or semi-structured data: As with an RDBMS, data stored in HBase does not need to fit into a fixed schema, making it suitable for storing unstructured or semi-structured data.
- Consistent: Unlike other NoSQL databases, such as Cassandra, which are eventually consistent, HBase is architected to have strongly consistent reads and writes. Once a write is done, all queries for that data to be read will return the same value.

• **Why HBase?**

Challenges with Earlier Technologies

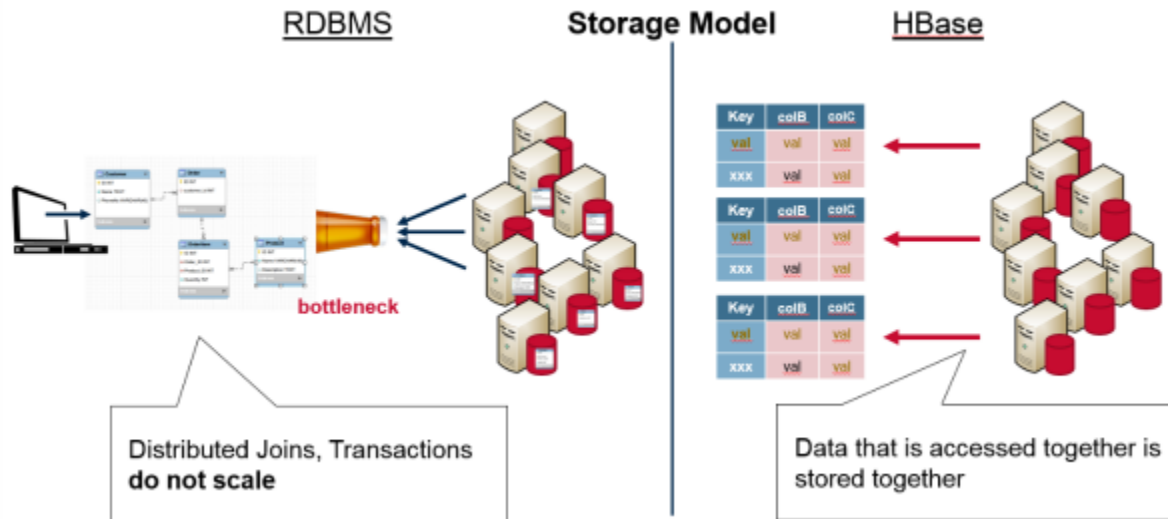
For years, relational databases were the standard, so what's changed? The need to scale out came with more and more info. Relational databases were therefore built for a single node. They were not developed to execute on clusters.

You normalize your schema in a relational database which eliminates redundant data and makes storage efficient. Indexes and queries with joins are used to recoup the data. Indexes slow down ingestion of data with lots of non-sequential disk I/O and joins cause bottlenecks with lots of data on reads. The relational model does not apply horizontally in a cluster.

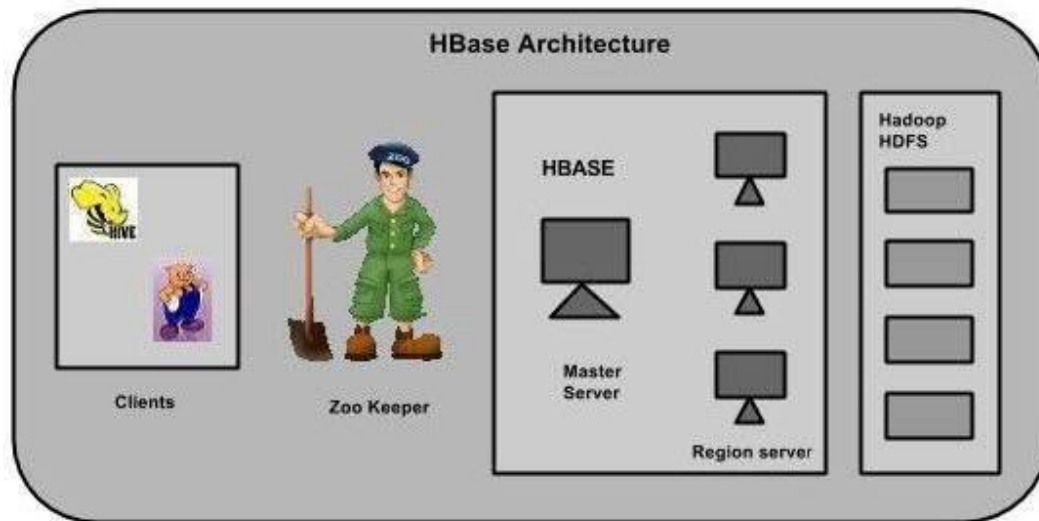
Advantages of HBase

By running on clusters HBase supports large volumes of data. HBase has been designed to scale up; data accessed together are stored together. For running on a cluster, grouping data by row key is essential. In HBase the data is spread automatically in a cluster. Sharding distributes various data through several servers, and the source for a subset of data is each server. Distributed data is accessed together, thereby making scaling quicker.

A table could consist of billions of rows for a successful Web application. If we want to search a specific row from such an enormous amount of data, HBase is the ideal choice because the time is less for query fetch. HBase is used by most online analytics applications.



- ***HBase Shell***



HBase provides a shell to use to interact with HBase. HBase uses the Hadoop File System for the data storage. It has region servers and a master server. The data will be stored in regions i.e. tables format. These regions will be broken down and stored in region servers. Given the requirements, region servers can be added or removed.

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.

General Commands

version - Provides the version of HBase being used.

whoami - Provides information about the user.

table_help - Provides help for table-reference commands.

status - Provides the status of HBase, for example, the number of servers.

MasterServer:

The master server

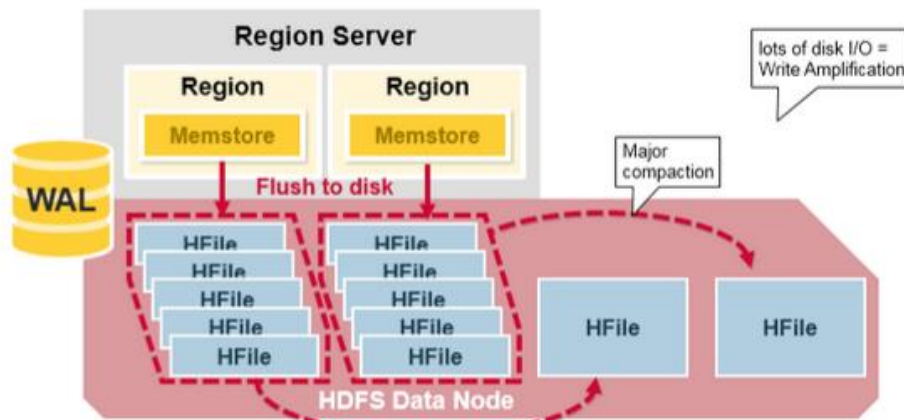
- Assign appropriate regions to region servers and takes Apache ZooKeeper's assistance for this task.
- Handles the regions load balancing across region servers. It unloads the crowded servers and transfers those regions to less crowded servers.
- Maintains cluster status by load balancing negotiations.
- Is responsible for improvements to the schema and other metadata operations like column families & tables creation.

Region server

The region servers have regions that

- Performs communication with the client and handle operations relating to the data.
- Take read and write queries for all the regions that are under it.
- Decide region size by following region size limits.

The following diagram shows the look of region server and how it contains regions and stores them:



The store contains HFiles and Memory store. Memstore's very much like a memory cache. Everything that is entered into the HBase is initially stored here. The data is subsequently transferred and stored as blocks in Hfiles and the memstore is flushed.

- **Working**

Step 1: Start the cloudera container:

```
[root@quickstart /]# sudo docker run --hostname=quickstart.cloudera --privileged=true -t -v /home/saman/Desktop/dataset:/user/cloudera/shared i -p 8889:8888 -p 7180:7181 cloudera/quickstart /usr/bin/docker-quickstart
```

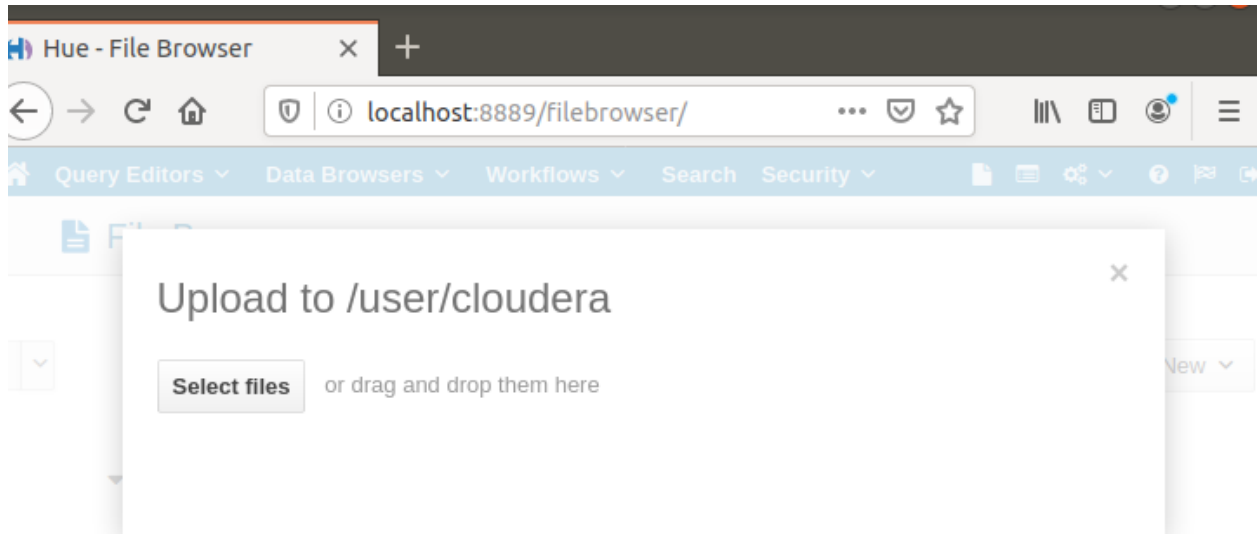
Start HMAster and regionserver service (in case they have not started)

```
[root@quickstart /]# service hbase-regionserver start
```

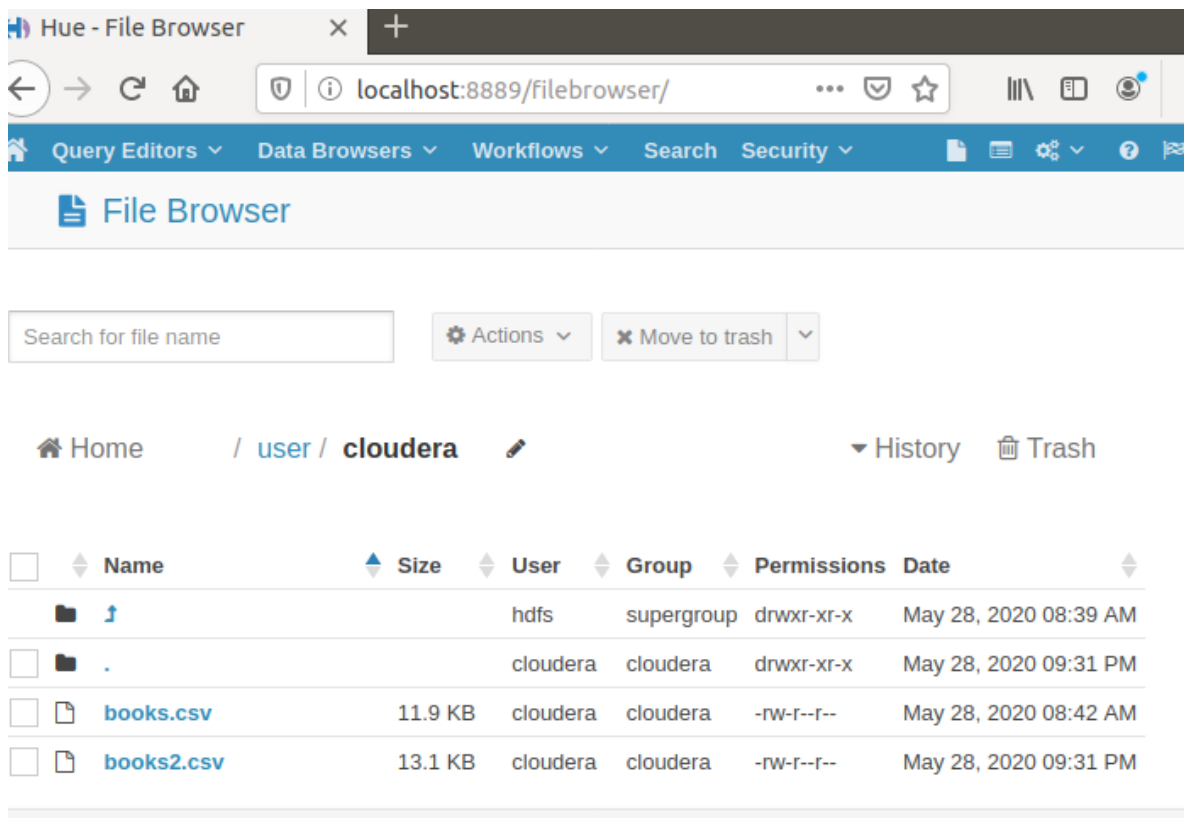
```
[root@quickstart /]# service hbase-master start
```

To import the file in hue:

File Browse -> Upload file ->select file



Upload file



START :HBASE SHELL

```
[root@quickstart ~]# hbase shell
```

LOCATION OF FILE: find the location of file /user/cloudera

```
[root@quickstart ~]# hdfs dfs -ls /
Found 5 items
drwxrwxrwx   - hdfs  supergroup          0 2016-04-06 02:26 /benchmarks
drwxr-xr-x   - hbase supergroup          0 2020-05-29 13:24 /hbase
drwxrwxrwt   - hdfs  supergroup          0 2020-05-29 08:48 /tmp
drwxr-xr-x   - hdfs  supergroup          0 2020-05-28 15:39 /user
drwxr-xr-x   - hdfs  supergroup          0 2016-04-06 02:27 /var
```

Command to import file from hdfs to hbase:

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=';' -
Dimporttsv.columns=HBASE_ROW_KEY,books:id,books:Title,books:Author,books:Genre,books:Height,books:publisher booklibrary /user/hdfs/books.csv
```

```
[root@quickstart ~]# hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=';' -Dimporttsv.columns=HBASE_ROW_KEY,books:id,books:Title,books:Author,books:Genre,books:Height,books:publisher booklibrary /user/hdfs/books.csv
```

Command :**Create**

Different method to create table

```
> true
hbase> create 't1', {NAME => 'f1', CONFIGURATION => {'hbase.hstore.blockingStoreFiles' => '10'}}

table configuration options can be put at the end.
examples:

hbase> create 'ns1:t1', 'f1', SPLITS => ['10', '20', '30', '40']
hbase> create 't1', 'f1', SPLITS => ['10', '20', '30', '40']
hbase> create 't1', 'f1', SPLITS_FILE => 'splits.txt', OWNER => 'johndoe'
hbase> create 't1', {NAME => 'f1', VERSIONS => 5}, METADATA => { 'mykey' => 'myvalue' }
hbase> # Optionally pre-split the table into NUMREGIONS, using
hbase> # SPLITALGO ("HexStringSplit", "UniformSplit" or classname)
hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}
hbase> create 't1', 'f1', {NUMREGIONS => 15, SPLITALGO => 'HexStringSplit', REGION_REPLICATION => 2, CONFIGURATION => {'hbase.hregion.scan.loadColumnFamilyOnDemand' => 'true'}}
hbase> create 't1', {NAME => 'f1', DFS_REPLICATION => 1}

you can also keep around a reference to the created table:

hbase> t1 = create 't1', 'f1'

which gives you a reference to the table named 't1', on which you can then call all methods.
```

List: all the tables in database

```
hbase(main):005:0> list 'course'
TABLE
course
1 row(s) in 0.0520 seconds

=> ["course"]
hbase(main):006:0> scan 'course'
ROW          COLUMN+CELL
 name        column=science:, timestamp=1590519071160, value=physics
1 row(s) in 0.3050 seconds

hbase(main):007:0> get 'course' , 'name'
COLUMN      CELL
 science:    timestamp=1590519071160, value=physics
1 row(s) in 0.0810 seconds

hbase(main):008:0> describe 'course'
Table course is ENABLED
course
COLUMN FAMILIES DESCRIPTION
{NAME => 'science', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLIC
ATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0'
, TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEM
ORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.2360 seconds

hbase(main):009:0> █
```

Create table command

Course: table name

Science: column family name

Put command: to insert data

Name: column name

Chemistry: row

```

hbase(main):001:0> list
TABLE
0 row(s) in 10.0940 seconds

=> []
hbase(main):002:0> create 'course','science'
0 row(s) in 2.8440 seconds

=> Hbase::Table - course
hbase(main):003:0> put 'course','name','science','chemistry'
0 row(s) in 1.0100 seconds

hbase(main):004:0> put 'course','name','science','physics'
0 row(s) in 0.0180 seconds

hbase(main):005:0> 

```

Delete the row

```

=> ["booklibrary", "booksdata", "booksinfo", "bookslibrary"]
hbase(main):002:0> delete 'booklibrary', '55', 'books'
0 row(s) in 0.7290 seconds

```

Describe the table

```

hbase(main):003:0> describe 'bookslibrary'
Table bookslibrary is ENABLED
bookslibrary
COLUMN FAMILIES DESCRIPTION
{NAME => 'books', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICAT
ION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMOR
Y => 'false', BLOCKCACHE => 'true'}
1 row(s) in 1.1390 seconds

```

```
hbase(main):004:0>
```

Activate Windows
Go to PC settings to activate Windows

Scan: view all table

```

e
153 row(s) in 6.8810 seconds

hbase(main):002:0> scan 'bookslibrary'

```

```

"Prince" column=books:Author, timestamp=1590724257070, value=Machia
velli
"Prince" column=books:Genre, timestamp=1590724257070, value=philoso
phy
"Prince" column=books:Height, timestamp=1590724257070, value=173
"Prince" column=books:Title, timestamp=1590724257070, value="The"
"Prince" column=books:publisher, timestamp=1590724257070, value=
"Social Choice & We column=books:Author, timestamp=1590724257070, value=Variou
lfare s
"Social Choice & We column=books:Genre, timestamp=1590724257070, value=economi
lfare cs
"Social Choice & We column=books:Height, timestamp=1590724257070, value=235
lfare
"Social Choice & We column=books:Title, timestamp=1590724257070, value= Vol 39
lfare No. 1"
"Social Choice & We column=books:publisher, timestamp=1590724257070, value=Spr
lfare inger
"World's Great Thin column=books:Author, timestamp=1590724257070, value=
kers
"World's Great Thin column=books:Genre, timestamp=1590724257070, value=philoso
kers phy
"World's Great Thin column=books:Height, timestamp=1590724257070, value=189
kers
"World's Great Thin column=books:Title, timestamp=1590724257070, value="The"
kers
"World's Great Thin column=books:publisher, timestamp=1590724257070, value=
kers
"World's Greatest S column=books:Author, timestamp=1590724257070, value=

```

STARTROW & STOPROW

Specify a `startrow` or `stoprow` or both. Neither `startrow` nor `stoprow` need to exist. Because HBase sorts rows lexicographically, it will return the first row after `startrow` would have occurred, and will stop returning rows after `stoprow` would have occurred. The goal is to reduce IO and network.

```

hbase(main):008:0> scan 'booklibrary',{COLUMN=>['books:Title'], STARTROW => '1'
, STOPROW => '20'}
ROW COLUMN+CELL
1 column=books:Title, timestamp=1590730137933, value="Goswami
10 column=books:Title, timestamp=1590730137933, value="Konnikova

```

Describe the table

Disable the table: `disable 'tablename'`

Enable the table: `enable 'tablename'`

```

hbase(main):005:0> describe 'booksinfo'
Table booksinfo is ENABLED
booksinfo
COLUMN FAMILIES DESCRIPTION
{NAME => 'info', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATI
ON_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 1.1440 seconds

hbase(main):006:0> disabled 'booksinfo'
NoMethodError: undefined method `disabled' for #<Object:0xa0e2803>

hbase(main):007:0> disable 'booksinfo'
0 row(s) in 2.8190 seconds

hbase(main):008:0> describe 'booksinfo'
Table booksinfo is DISABLED
booksinfo
COLUMN FAMILIES DESCRIPTION
{NAME => 'info', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATI
ON_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 0.1740 seconds

hbase(main):009:0> enable 'booksinfo'

```

Activate Windows
Go to PC settings to activate Windows.

For dropping the table you first have to disable the table

Then drop the table

```

hbase(main):008:0> describe 'course'
Table course is ENABLED
course
COLUMN FAMILIES DESCRIPTION
{NAME => 'science', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLIC
ATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0'
, TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEM
ORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.2360 seconds

hbase(main):009:0> drop 'course'

ERROR: Table course is enabled. Disable it first.

Here is some help for this command:
Drop the named table. Table must first be disabled:
  hbase> drop 't1'
  hbase> drop 'ns1:t1'

hbase(main):010:0> disable 'course'
0 row(s) in 2.8240 seconds

hbase(main):011:0> drop 'course'
0 row(s) in 2.8380 seconds

hbase(main):012:0>

```

Get the table Values by rowed: get 'book', '100'

```
hbase(main):001:0> get 'booklibrary','10'
COLUMN                                CELL
books:Author                          timestamp=1590730137933, value= Maria"
books:Genre                           timestamp=1590730137933, value=psychology
books:Height                          timestamp=1590730137933, value=240
books>Title                           timestamp=1590730137933, value="Konnikova
books:id                              timestamp=1590730137933, value=How to Think Like Sherlock
                                      Holmes
books:publisher                       timestamp=1590730137933, value=Penguin
5 row(s) in 1.2280 seconds
```

hbase(main):002:0> Activate Window

Columnprefixfilter: This filter takes one argument as column prefix. It returns solely those key-values present in the very column that starts with the specified column prefix. The column prefix should be the form qualifier.

Filters out by prefix Titlr

```
hbase(main):016:0> scan 'booklibrary' , {FILTER => "ColumnPrefixFilter('Title')"}
ROW      COLUMN+CELL
1         column=books:Title, timestamp=1590730137933, value="Goswami
10        column=books:Title, timestamp=1590730137933, value="Konnikova
100       column=books:Title, timestamp=1590730137933, value="Gordon
101       column=books:Title, timestamp=1590730137933, value="Orwell
102       column=books:Title, timestamp=1590730137933, value="Sen
103       column=books:Title, timestamp=1590730137933, value="Dalrymple
104       column=books:Title, timestamp=1590730137933, value= The"
105       column=books:Title, timestamp=1590730137933, value="Iacoca
106       column=books:Title, timestamp=1590730137933, value="K..."
```

SingleColumnValueFilter: This filter as an argument takes a column family, a qualifier, a compare operator and a comparator. So, if the specified column isn't found, all the columns of that row are going to be emitted. And ,If the column is found and also the comparison with the comparator returns true, all the columns of the row are going to be emitted. If the condition fails, the row won't be emitted.

This query finds out where the author name BBC exists

```
hbase(main):030:0> scan 'booklibrary', { FILTER => SingleColumnValueFilter.new(
Bytes.toBytes('books'),Bytes.toBytes('Author'),CompareFilter::CompareOp.valueOf
('EQUAL'),BinaryComparator.new(Bytes.toBytes('BBC')))}
ROW COLUMN+CELL
54 column=books:Author, timestamp=1590730137933, value=BBC
54 column=books:Genre, timestamp=1590730137933, value=nonfiction
54 column=books:Height, timestamp=1590730137933, value=178
54 column=books:Title, timestamp=1590730137933, value=The"
54 column=books:id, timestamp=1590730137933, value="Complete
Mastermind
54 column=books:publisher, timestamp=1590730137933, value=BBC
1 row(s) in 0.1890 seconds
```

Activate Windows

ValueFilter:

This filter takes a compare operator and a comparator. It compares every value with the comparator using the compare operator and if the comparison returns true, it returns that key-value.

Present all data where Genre is comic

```
hbase(main):011:0> scan 'booklibrary',{FILTER => "ValueFilter (='binaryprefix:
comic')"}
ROW COLUMN+CELL
139 column=books:Author, timestamp=1590730137933, value=comic
140 column=books:Genre, timestamp=1590730137933, value=comic
141 column=books:Author, timestamp=1590730137933, value=comic
142 column=books:Author, timestamp=1590730137933, value=comic
143 column=books:Author, timestamp=1590730137933, value=comic
145 column=books:Author, timestamp=1590730137933, value=comic
146 column=books:Author, timestamp=1590730137933, value=comic
147 column=books:Author, timestamp=1590730137933, value=comic
148 column=books:Author, timestamp=1590730137933, value=comic
149 column=books:Genre, timestamp=1590730137933, value=comic
150 column=books:Author, timestamp=1590730137933, value=comic
151 column=books:Author, timestamp=1590730137933, value=comic
205 column=books:Author, timestamp=1590730137933, value=comic
13 row(s) in 0.7150 seconds
```

Activate Windows

SingleColumnFilter: finds the data where height is greater than 250

```
her
id      column=books:Title, timestamp=1590730137933, value=Author
id      column=books:id, timestamp=1590730137933, value=Title
28 row(s) in 1.3030 seconds

hbase(main):012:0> scan 'booklibrary', { FILTER => SingleColumnValueFilter.new(
Bytes.toBytes('books'),Bytes.toBytes('Height'),CompareFilter::CompareOp.valueOf(
('GREATER'),BinaryComparator.new(Bytes.toBytes(new long (250)))))}
(hbase):12 warning: don't put space before argument parentheses
```

Count: No of rows in table

```
hbase(main):040:0> count 'booklibrary', CACHE =>1000
154 row(s) in 0.1440 seconds

=> 154
hbase(main):041:0>
```

LIMIT: Display first three rows

```
hbase(main):010:0> scan 'booklibrary' ,{COLUMNS=>'books:Title',LIMIT => 3}
ROW      COLUMN+CELL
1        column=books:Title, timestamp=1590730137933, value="Goswami"
10       column=books:Title, timestamp=1590730137933, value="Konnikova"
100      column=books:Title, timestamp=1590730137933, value="Gordon"
3 row(s) in 0.3040 seconds

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

Where the Genre comic is

```
hbase(main):029:0> scan 'booklibrary', { FILTER => SingleColumnValueFilter.new(
Bytes.toBytes('books'),Bytes.toBytes('Genre'),CompareFilter::CompareOp.valueOf(
('EQUAL'),BinaryComparator.new(Bytes.toBytes('comic')))}
ROW      COLUMN+CELL
140      column=books:Author, timestamp=1590730137933, value=
140      column=books:Genre, timestamp=1590730137933, value=comic
140      column=books:Height, timestamp=1590730137933, value=283
140      column=books:Title, timestamp=1590730137933, value= The"
140      column=books:id, timestamp=1590730137933, value="Killing Joke
140      column=books:publisher, timestamp=1590730137933, value=nil
149      column=books:Author, timestamp=1590730137933, value=
149      column=books:Genre, timestamp=1590730137933, value=comic
149      column=books:Height, timestamp=1590730137933, value=258
149      column=books:Title, timestamp=1590730137933, value= The"
149      column=books:id, timestamp=1590730137933, value="Death of Superman
149      column=books:publisher, timestamp=1590730137933, value=nil
2 row(s) in 0.2530 seconds
```

Where prefix 10 in row

```
hbase(main):016:0> scan 'booklibrary', {COLUMN=>'books:Height',ROWPREFIXFILTER
=> '10'}
ROW          COLUMN+CELL
 10          column=books:Height, timestamp=1590730137933, value=240
 100         column=books:Height, timestamp=1590730137933, value=179
 101         column=books:Height, timestamp=1590730137933, value=179
 102         column=books:Height, timestamp=1590730137933, value=219
 103         column=books:Height, timestamp=1590730137933, value=197
 104         column=books:Height, timestamp=1590730137933, value=217
 105         column=books:Height, timestamp=1590730137933, value=175
 108         column=books:Height, timestamp=1590730137933, value=160
 109         column=books:Height, timestamp=1590730137933, value=172
9 row(s) in 0.0870 seconds
```

TRUNCATE: Truncate table

```
hbase(main):005:0> create 'testdemo','test'
0 row(s) in 2.7850 seconds

=> Hbase::Table - testdemo
hbase(main):006:0> truncate 'testdemo'
Truncating 'testdemo' table (it may take a while):
- Disabling table...
- Truncating table...
0 row(s) in 5.9500 seconds
```

Update command

```
hbase(main):001:0> put 'booklibrary','55','books:Height','200'
0 row(s) in 1.1510 seconds

hbase(main):002:0> put 'booklibrary','55','books:Height','200'[root@quickstart
/]#
[root@quickstart /]# hbase shell
2020-05-29 07:08:21,155 INFO [main] Configuration.deprecation: hadoop.native.l
ib 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 1.2.0-cdh5.7.0, rUnknown, Wed Mar 23 11:39:14 PDT 2016

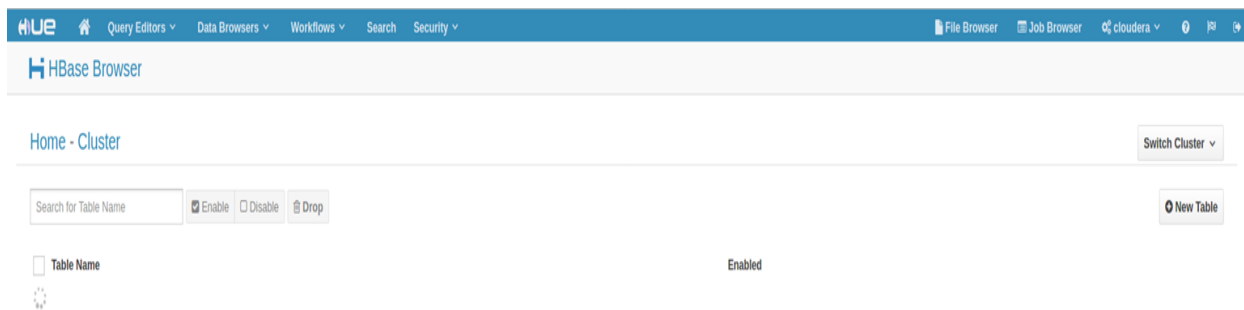
hbase(main):001:0> get 'booklibrary','55'
COLUMN      CELL
books:Author timestamp=1590730137933, value= Bob"
books:Genre  timestamp=1590730137933, value=nonfiction
books:Height timestamp=1590736049760, value=200
books:Title  timestamp=1590730137933, value="Dylan
books:id     timestamp=1590730137933, value=Dylan on Dylan
books:publisher timestamp=1590730137933, value=Random House
6 row(s) in 1.2570 seconds
```

INCR: increment the value by 10

```
> ["booklibrary", "booksdata", "booksinfo", "booklibrary", "testdemo"]
hbase(main):002:0> incr 'booklibrary','25','books:Height',10
COUNTER VALUE = 10
0 row(s) in 0.7660 seconds

hbase(main):003:0> get 'booklibrary','25'
COLUMN CELL
```

HBase Browser Demo



Creating New Tables:

Create New Table

Table Name:

Column Families:

×

Author

+

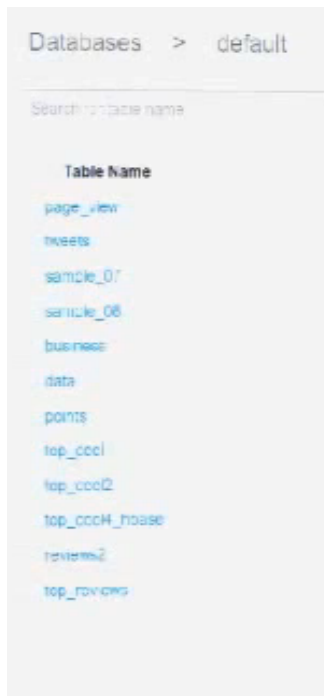
+

+

Cancel

Submit

Table Names:



Files in Hue

The screenshot shows the Hue file browser interface. At the top, there is a breadcrumb 'Home' and a user status 'user ro'. Below this is a table listing files and folders. The table has columns for Type, Name, Size, User, Group, and Permissions. The files listed are: .Trash, gethue.pdf (260.5 KB), hue_ellie_transparent.png (6.0 KB), oozie-oozi, yelp_academic_dataset_review.json (206.0 MB), and yelp_reviews.csv.

Type	Name	Size	User	Group	Permissions
Folder	..		ro	ro	drwxrwxrwx
Folder	..		hdfs	supergroup	drwxr-xr-x
Folder	.Trash		ro	ro	drwxr-xr-x
File	gethue.pdf	260.5 KB	ro	ro	-rw-r--r--
File	hue_ellie_transparent.png	6.0 KB	ro	ro	-rw-r--r--
File	oozie-oozi		ro	ro	drwxrwxrwx
File	yelp_academic_dataset_review.json	206.0 MB	ro	ro	-rw-r--r--
File	yelp_reviews.csv		ro	ro	drwxrwxrwx

Clean Review in Hue

