**HBase Tutorials**

HBase is an open source, distributed database, developed by[Apache](https://www.guru99.com/apache.html)Software foundation.

Initially, it was Google Big Table, afterwards it was re-named as HBase and is primarily written in Java.

HBase can store massive amounts of data from terabytes to petabytes.

**HBase Unique Features**

* HBase is built for low latency operations
* HBase is used extensively for random read and write operations
* HBase stores large amount of data in terms of tables
* Provides linear and modular scalability over cluster environment
* Strictly consistent to read and write operations
* Automatic and configurable sharding of tables
* Automatic failover supports between Region Servers
* Convenient base classes for backing[Hadoop](https://www.guru99.com/bigdata-tutorials.html)MapReduce jobs in HBase tables
* Easy to use[Java](https://www.guru99.com/java-tutorial.html)API for client access
* Block cache and Bloom Filters for real-time queries
* Query predicate pushes down via server side filters.

## Why to Choose HBase?

A table for a popular web application may consist of billions of rows. If we want to search particular row from such huge amount of data, HBase is the ideal choice as query fetch time in less. Most of the online analytics applications uses HBase.

Traditional relational data models fail to meet performance requirements of very big databases. These performance and processing limitations can be overcomed by HBase.

## Importance of NoSQL Databases in Hadoop

In big data analytics, Hadoop plays a vital role in solving typical business problems by managing large data sets and gives best solutions in analytics domain.

In Hadoop ecosystem, each component plays its unique role for the

* Data processing
* Data validation
* Data storing

In terms of storing unstructured, semi-structured data storage as well as retrieval of such data's, relational databases are less useful. Also, fetching results by applying query on huge data sets that are stored in Hadoop storage is a challenging task. NoSQL storage technologies provide the best solution for faster querying on huge data sets.

## Other NoSQL storage type Databases

Some of the NoSQL models present in the market are **Cassandra, MongoDB, and CouchDB**. Each of these models has different ways of storage mechanism.

For example, MongoDB is a document-oriented database from NoSQL family tree. Compared to traditional databases it provides best features in terms of performance, availability and scalability. It is an open source document-oriented database, and it's written in C++.

Cassandra is also a distributed database from open source Apache software which is designed to handle a huge amount of data stored across commodity servers. Cassandra provides high availability with no single point of failure.

## How HBase different from other NoSQL model

HBase storage model is different from other NoSQL models discussed above. This can be stated as follow

* HBase stores data in the form of key/value pairs in a columnar model. In this model, all the columns are grouped together as Column families
* HBase provides flexible data model and low latency access to small amounts of data stored in large data sets
* HBase on top of Hadoop will increase throughput and performance of distributed cluster set up. In turn, it provides faster random reads and writes operations

## Which NoSQL Database to choose?

MongoDB, CouchDB, and Cassandra are of NoSQL type databases that are feature specific and used as per their business needs. Here, we have listed out different NoSQL database as per their use case.

|  |  |  |
| --- | --- | --- |
| **Data Base Type Based on Feature** | **Example of Database** | **Use case (When to Use)** |
| Key/ Value | Redis, MemcacheDB | Caching, Queue-ing, Distributing information |
| Column Oriented | Cassandra, HBase | Scaling, Keeping Unstructured, non-volatile |
| Document Oriented | MongoDB, Couchbase | Nested Information, JavaScript friendly |
| Graph Based | OrientDB, Neo4J | Handling Complex relational information. Modeling and Handling classification. |

## Where is HBase used?

**Telecom Industry**

Problem Statement:

* Storing billions of CDR (Call detailed recording) log records generated by telecom domain
* Providing real-time access to CDR logs and billing information of customers
* Provide cost effective solution comparing to traditional database systems

**Solution:**

HBase is used to store billions of rows of call detailed records. If 20TB of data is added per month to the existing RDBMS database, performance will deteriorate. To handle a large amount of data in this use case, HBase is the best solution. HBase performs fast querying and display records.

**Banking Industry**

Problem Statement:

The Banking industry generates millions of records on a daily basis. In addition to this, banking industry also needs analytics solution that can detect Fraud in money transactions.

Solution:

To store, process and update huge volumes of data and performing analytics, an ideal solution is - HBase integrated with several Hadoop eco system components.

That apart, HBase can be used**-**

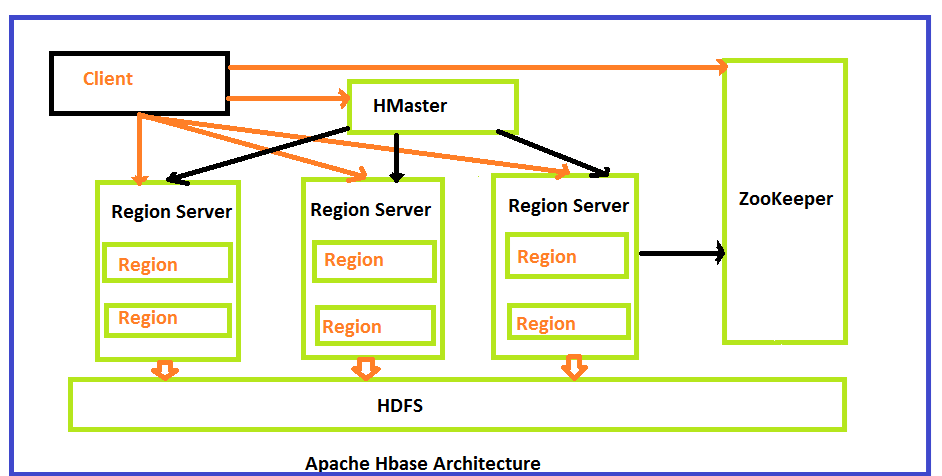
* Whenever there is a need to write heavy applications.
* Performing online log analytics and to generate compliance reports.

**Summary:-**

HBase provides unique features and will solve typical industrial use cases. As a column-oriented storage, it provides fast querying, fetching of results and high amount of data storage.

* Set of tables
* Each table with column families and rows
* Each table must have an element defined as Primary Key.
* Row key acts as a Primary key in HBase.
* Any access to HBase tables uses this Primary Key
* Each column present in HBase denotes attribute corresponding to object

## HBase Architecture and its Important Components

[](https://www.guru99.com/images/HBase/100715_1237_HBaseArchit2.png)

HBase architecture consists mainly of four components

* HMaster
* HRegionserver
* HRegions
* Zookeeper

**HMaster:**

HMaster is the implementation of Master server in HBase architecture. It acts like monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

* Plays a vital role in terms of performance and maintaining nodes in the cluster.
* HMaster provides admin performance and distributes services to different region servers.
* HMaster assigns regions to region servers.
* HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.
* When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.

* Table ( createTable, removeTable, enable, disable)
* ColumnFamily (add Column, modify Column)
* Region (move, assign)

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn check the health status of region servers.

In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

**HRegions Servers:**

When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

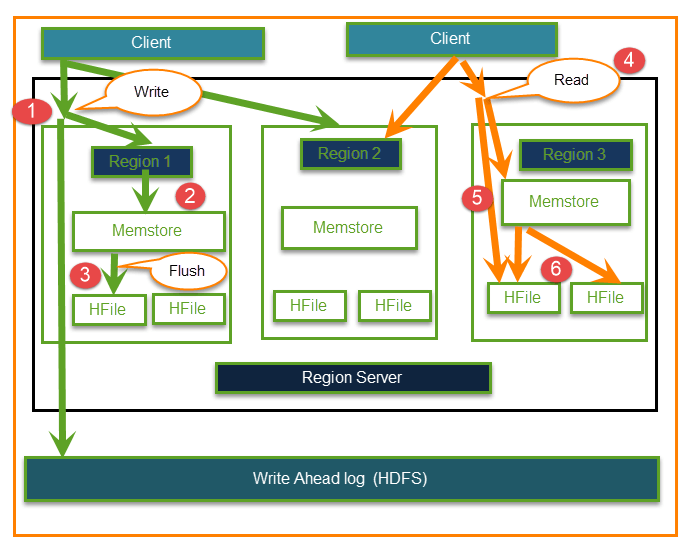
HMaster can get into contact with multiple HRegion servers and performs the following functions.

* Hosting and managing regions
* Splitting regions automatically
* Handling read and writes requests
* Communicating with the client directly

**HRegions:**

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

## Data flow in HBase

[](https://www.guru99.com/images/HBase/100715_1237_HBaseArchit3.png)

**Write and Read operations**

The Read and Write operations from Client into Hfile can be shown in below diagram.

**Step 1)** Client wants to write data and in turn first communicates with Regions server and then regions

**Step 2)** Regions contacting memstore for storing associated with the column family

**Step 3)**First data stores into Memstore, where the data is sorted and after that it flushes into HFile. The main reason for using Memstore is to store data in Distributed file system based on Row Key. Memstore will be placed in Region server main memory while HFiles are written into HDFS.

**Step 4)**Client wants to read data from Regions

**Step 5)**In turn Client can have direct access to Mem store, and it can request for data.

**Step 6) Client** approaches HFiles to get the data. The data are fetched and retrieved by the Client.

Memstore holds in-memory modifications to the store. The hierarchy of objects in HBase Regions is as shown from top to bottom in below table.

|  |  |
| --- | --- |
| Table | HBase table present in the HBase cluster |
| Region | HRegions for the presented tables |
| Store | It store per ColumnFamily for each region for the table |
| Memstore | * Memstore for each store for each region for the table * It sorts data before flushing into HFiles * Write and read performance will increase because of sorting |
| StoreFile | StoreFiles for each store for each region for the table |
| Block | Blocks present inside StoreFiles |

**ZooKeeper:**

In Hbase, Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes. If the client wants to communicate with regions, the servers client has to approach ZooKeeper first.

It is an open source project, and it provides so many important services.

Services provided by ZooKeeper

* Maintains Configuration information
* Provides distributed synchronization
* Client Communication establishment with region servers
* Provides ephemeral nodes for which represent different region servers
* Master servers usability of ephemeral nodes for discovering available servers in the cluster
* To track server failure and network partitions

Master and HBase slave nodes ( region servers) registered themselves with ZooKeeper. The client needs access to ZK(zookeeper) quorum configuration to connect with master and region servers.

## Starting HBase Shell

To access the HBase shell, you have to navigate to the HBase home folder.

cd /usr/localhost/

cd Hbase

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

./bin/hbase shell

If you have successfully installed HBase in your system, then it gives you the HBase shell prompt as shown below.

HBase Shell; enter 'help<RETURN>' for list of supported commands.

Type "exit<RETURN>" to leave the HBase Shell

Version 0.94.23, rf42302b28aceaab773b15f234aa8718fff7eea3c, Wed Aug 27

00:54:09 UTC 2014

hbase(main):001:0>

To exit the interactive shell command at any moment, type exit or use <ctrl+c>. 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. First of all, verify the installation and the configuration of HBase in your system using this command as shown below.

hbase(main):001:0> list

When you type this command, it gives you the following output.

hbase(main):001:0> list

TABLE

# General Commands

## status

This command returns the status of the system including the details of the servers running on the system. Its syntax is as follows:

hbase(main):009:0> status

If you execute this command, it returns the following output.

hbase(main):009:0> status

3 servers, 0 dead, 1.3333 average load

## version

This command returns the version of HBase used in your system. Its syntax is as follows:

hbase(main):010:0> version

If you execute this command, it returns the following output.

hbase(main):009:0> version

0.98.8-hadoop2, r6cfc8d064754251365e070a10a82eb169956d5fe, Fri Nov 14

18:26:29 PST 2014

## table\_help

This command guides you what and how to use table-referenced commands. Given below is the syntax to use this command.

hbase(main):02:0> table\_help

When you use this command, it shows help topics for table-related commands. Given below is the partial output of this command.

hbase(main):002:0> table\_help

Help for table-reference commands.

You can either create a table via 'create' and then manipulate the table

via commands like 'put', 'get', etc.

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 commands.

For instance, you can get create a table and keep around a reference to

it via:

hbase> t = create 't', 'cf'…...

## whoami

This command returns the user details of HBase. If you execute this command, returns the current HBase user as shown below.

hbase(main):008:0> whoami

hadoop (auth:SIMPLE)

groups: hadoop

## Creating a Table using HBase Shell

You 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>’

### Example

Given below is a sample schema of a table named emp. It has two column families: “personal data” and “professional data”.

|  |  |  |
| --- | --- | --- |
| **Row key** | **personal data** | **professional data** |
|  |  |  |
|  |  |  |

You can create this table in HBase shell as shown below.

hbase(main):002:0> create 'emp', 'personal data', 'professional data'

And it will give you the following output.

0 row(s) in 1.1300 seconds

=> Hbase::Table - emp

### Verification

You can verify whether the table is created using the **list** command as shown below. Here you can observe the created emp table.

hbase(main):002:0> list

TABLE

emp

2 row(s) in 0.0340 seconds

## Listing a Table using HBase Shell

list is the command that is used to list all the tables in HBase. Given below is the syntax of the list command.

hbase(main):001:0 > list

When you type this command and execute in HBase prompt, it will display the list of all the tables in HBase as shown below.

hbase(main):001:0> list

TABLE

emp

# Disabling a Table

## Disabling a Table using HBase Shell

To delete a table or change its settings, you need to first disable the table using the disable command. You can re-enable it using the enable command.

Given below is the syntax to disable a table:

disable ‘emp’

### Example

Given below is an example that shows how to disable a table.

hbase(main):025:0> disable 'emp'

0 row(s) in 1.2760 seconds

### Verification

After disabling the table, you can still sense its existence through **list** and **exists** commands. You cannot scan it. It will give you the following error.

hbase(main):028:0> scan 'emp'

ROW COLUMN + CELL

ERROR: emp is disabled.

### is\_disabled

This command is used to find whether a table is disabled. Its syntax is as follows.

hbase> is\_disabled 'table name'

The following example verifies whether the table named emp is disabled. If it is disabled, it will return true and if not, it will return false.

hbase(main):031:0> is\_disabled 'emp'

true

0 row(s) in 0.0440 seconds

### disable\_all

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

hbase> disable\_all 'r.\*'

Suppose there are 5 tables in HBase, namely raja, rajani, rajendra, rajesh, and raju. The following code will disable all the tables starting with **raj.**

hbase(main):002:07> disable\_all 'raj.\*'

raja

rajani

rajendra

rajesh

raju

Disable the above 5 tables (y/n)?

y

5 tables successfully disabled

## Enabling a Table using HBase Shell

Syntax to enable a table:

enable ‘emp’

### Example

Given below is an example to enable a table.

hbase(main):005:0> enable 'emp'

0 row(s) in 0.4580 seconds

### Verification

After enabling the table, scan it. If you can see the schema, your table is successfully enabled.

hbase(main):006:0> scan 'emp'

## is\_enabled

This command is used to find whether a table is enabled. Its syntax is as follows:

hbase> is\_enabled 'table name'

The following code verifies whether the table named **emp** is enabled. If it is enabled, it will return true and if not, it will return false.

hbase(main):031:0> is\_enabled 'emp'

true

0 row(s) in 0.0440 seconds

# Describe & Alter

## describe

This command returns the description of the table. Its syntax is as follows:

hbase> describe 'table name'

## alter

Alter is the command used to make changes to an 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.

### Changing the Maximum Number of Cells of a Column Family

Given below is the syntax to change the maximum number of cells of a column family.

hbase> alter 't1', NAME ⇒ 'f1', VERSIONS ⇒ 5

In the following example, the maximum number of cells is set to 5.

hbase(main):003:0> alter 'emp', NAME ⇒ 'personal data', VERSIONS ⇒ 5

### able Scope Operators

Using alter, you can set and remove table scope operators such as MAX\_FILESIZE, READONLY, MEMSTORE\_FLUSHSIZE, DEFERRED\_LOG\_FLUSH, etc.

### Setting Read Only

Below given is the syntax to make a table read only.

hbase>alter 't1', READONLY(option)

In the following example, we have made the **emp** table read only.

hbase(main):006:0> alter 'emp', READONLY

### Removing Table Scope Operators

We can also remove the table scope operators. Given below is the syntax to remove ‘MAX\_FILESIZE’ from emp table.

hbase> alter 't1', METHOD ⇒ 'table\_att\_unset', NAME ⇒ 'MAX\_FILESIZE'

### Deleting a Column Family

Using alter, you can also delete a column family. Given below is the syntax to delete a column family using alter.

hbase> alter ‘ table name ’, ‘delete’ ⇒ ‘ column family ’

Given below is an example to delete a column family from the ‘emp’ table.

Assume there is a table named employee in HBase. It contains the following data:

hbase(main):006:0> scan 'employee'

# HBase - Exists

## Existence of Table using HBase Shell

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

hbase(main):024:0> exists 'emp'

# Drop a Table

## Dropping a Table using HBase Shell

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

hbase(main):018:0> disable 'emp'

0 row(s) in 1.4580 seconds

hbase(main):019:0> drop 'emp'

0 row(s) in 0.3060 seconds

Verify whether the table is deleted using the exists command.

hbase(main):020:07gt; exists 'emp'

Table emp does not exist

0 row(s) in 0.0730 seconds

## drop\_all

This command is used to drop the tables matching the “regex” given in the command. Its syntax is as follows:

hbase> drop\_all ‘t.\*’

**Note:** Before dropping a table, you must disable it.

### Example

Assume there are tables named raja, rajani, rajendra, rajesh, and raju.

hbase(main):017:0> list

TABLE

raja

rajani

rajendra

rajesh

raju

9 row(s) in 0.0270 seconds

All these tables start with the letters **raj**. First of all, let us disable all these tables using the **disable\_all** command as shown below.

hbase(main):002:0> disable\_all 'raj.\*'

raja

rajani

rajendra

rajesh

raju

Disable the above 5 tables (y/n)?

y

5 tables successfully disabled

Now you can delete all of them using the **drop\_all** command as given below.

hbase(main):018:0> drop\_all 'raj.\*'

## exit

You exit the shell by typing the **exit** command.

hbase(main):021:0> exit

## Stopping HBase

To stop HBase, browse to the HBase home folder and type the following command.

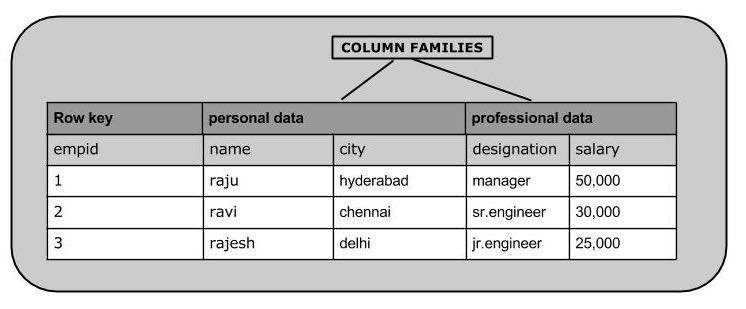
./bin/stop-hbase.sh

## Inserting Data using HBase Shell

This chapter demonstrates how to create data in an HBase table. To create data in an HBase table, the following commands and methods are used:

* **put** command,
* **add()** method of **Put** class, and
* **put()** method of **HTable** class.

As an example, we are going to create the following table in HBase.



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

put ’<table name>’,’row1’,’<colfamily:colname>’,’<value>’

### Inserting the First Row

Let us insert the first row values into the emp table as shown below.

hbase(main):005:0> put 'emp','1','personal data:name','raju'

0 row(s) in 0.6600 seconds

hbase(main):006:0> put 'emp','1','personal data:city','hyderabad'

0 row(s) in 0.0410 seconds

hbase(main):007:0> put 'emp','1','professional

data:designation','manager'

0 row(s) in 0.0240 seconds

hbase(main):007:0> put 'emp','1','professional data:salary','50000'

0 row(s) in 0.0240 seconds

Insert the remaining rows using the put command in the same way. If you insert the whole table, you will get the following output.

hbase(main):022:0> scan 'emp'

## Updating Data using HBase Shell

You can update an existing cell value using the **put** command. To do so, just follow the same syntax and mention your new value as shown below.

put ‘table name’,’row ’,'Column family:column name',’new value’

The newly given value replaces the existing value, updating the row.

### Example

Suppose there is a table in HBase called **emp** with the following data.

hbase(main):003:0> scan 'emp'

ROW COLUMN + CELL

row1 column = personal:name, timestamp = 1418051555, value = raju

row1 column = personal:city, timestamp = 1418275907, value = Hyderabad

row1 column = professional:designation, timestamp = 14180555,value = manager

row1 column = professional:salary, timestamp = 1418035791555,value = 50000

1 row(s) in 0.0100 seconds

The following command will update the city value of the employee named ‘Raju’ to Delhi.

hbase(main):002:0> put 'emp','row1','personal:city','Delhi'

0 row(s) in 0.0400 seconds

The updated table looks as follows where you can observe the city of Raju has been changed to ‘Delhi’.

hbase(main):003:0> scan 'emp'

## Reading Data using HBase Shell

The **get** command and the **get()** method of **HTable** class are used to read data from a table in HBase. Using **get** command, you can get a single row of data at a time. Its syntax is as follows:

get ’<table name>’,’row1’

### Example

The following example shows how to use the get command. Let us scan the first row of the **emp** table.

hbase(main):012:0> get 'emp', '1'

## Reading a Specific Column

Given below is the syntax to read a specific column using the **get** method.

hbase> get 'table name', ‘rowid’, {COLUMN ⇒ ‘column family:column name ’}

### Example

Given below is the example to read a specific column in HBase table.

hbase(main):015:0> get 'emp', 'row1', {COLUMN ⇒ 'personal:name'}

## Deleting a Specific Cell in a Table

Using the **delete** command, you can delete a specific cell in a table. The syntax of **delete** command is as follows:

delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’

### Example

Here is an example to delete a specific cell. Here we are deleting the salary.

hbase(main):006:0> delete 'emp', '1', 'personal data:city',

1417521848375

0 row(s) in 0.0060 seconds

## Deleting All Cells in a Table

Using the “deleteall” command, you can delete all the cells in a row. Given below is the syntax of deleteall command.

deleteall ‘<table name>’, ‘<row>’,

### Example

Here is an example of “deleteall” command, where we are deleting all the cells of row1 of emp table.

hbase(main):007:0> deleteall 'emp','1'

0 row(s) in 0.0240 seconds

Verify the table using the **scan** command. A snapshot of the table after deleting the table is given below.

hbase(main):022:0> scan 'emp'

## count

You can count the number of rows of a table using the **count** command. Its syntax is as follows:

count ‘<table name>’

After deleting the first row, emp table will have two rows. Verify it as shown below.

hbase(main):023:0> count 'emp'

2 row(s) in 0.090 seconds

⇒ 2

## truncate

This command disables drops and recreates a table. The syntax of **truncate**is as follows:

hbase> truncate 'table name'

### Example

Given below is the example of truncate command. Here we have truncated the **emp** table.

hbase(main):011:0> truncate 'emp'

Truncating 'one' table (it may take a while):

- Disabling table...

- Truncating table...

0 row(s) in 1.5950 seconds

After truncating the table, use the scan command to verify. You will get a table with zero rows.

hbase(main):017:0> scan ‘emp’

ROW COLUMN + CELL

0 row(s) in 0.3110 seconds