Topics

* HBase Overview
* HBase Architecture
* HBase vs HDFS
* Installation
* HBase Shell and Commands
* HBase Query : structure/object manage
* HBase Query: Data manage

HBase Intro

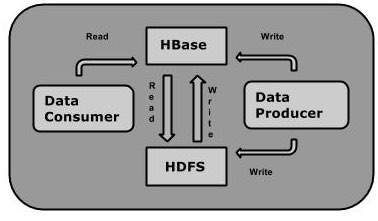
HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.

HBase is a data model that is similar to Google’s big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).

It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



HBase and HDFS

|  |  |
| --- | --- |
| **HDFS** | **HBase** |
| HDFS is a distributed file system suitable for storing large files. | HBase is a database built on top of the HDFS. |
| HDFS does not support fast individual record lookups. | HBase provides fast lookups for larger tables. |
| It provides high latency batch processing; no concept of batch processing. | It provides low latency access to single rows from billions of records (Random access). |
| It provides only sequential access of data. | HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups. |

## Storage Mechanism in HBase

HBase is a **column-oriented database** and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs. A table have multiple column families and each column family can have any number of columns. Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

* Table is a collection of rows.
* Row is a collection of column families.
* Column family is a collection of columns.
* Column is a collection of key value pairs.

Given below is an example schema of table in HBase.

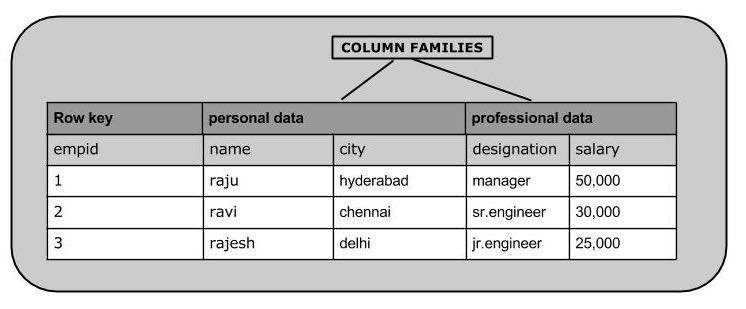
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rowid** | **Column Family** | | | **Column Family** | | | **Column Family** | | | **Column Family** | | |
| **col1** | **col2** | **col3** | **col1** | **col2** | **col3** | **col1** | **col2** | **col3** | **col1** | **col2** | **col3** |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |

## Column Oriented and Row Oriented

Column-oriented databases are those that store data tables as sections of columns of data, rather than as rows of data. Shortly, they will have column families.

|  |  |
| --- | --- |
| **Row-Oriented Database** | **Column-Oriented Database** |
| It is suitable for Online Transaction Process (OLTP). | It is suitable for Online Analytical Processing (OLAP). |
| Such databases are designed for small number of rows and columns. | Column-oriented databases are designed for huge tables. |

The following image shows column families in a column-oriented database:



## HBase and RDBMS

|  |  |
| --- | --- |
| **HBase** | **RDBMS** |
| HBase is schema-less, it doesn't have the concept of fixed columns schema; defines only column families. | An RDBMS is governed by its schema, which describes the whole structure of tables. |
| It is built for wide tables. HBase is horizontally scalable. | It is thin and built for small tables. Hard to scale. |
| No transactions are there in HBase. | RDBMS is transactional. |
| It has de-normalized data. | It will have normalized data. |
| It is good for semi-structured as well as structured data. | It is good for structured data. |

## Features of HBase

* HBase is linearly scalable.
* It has automatic failure support.
* It provides consistent read and writes.
* It integrates with Hadoop, both as a source and a destination.
* It has easy java API for client.
* It provides data replication across clusters.

## Where to Use HBase

* Apache HBase is used to have random, real-time read/write access to Big Data.
* It hosts very large tables on top of clusters of commodity hardware.
* Apache HBase is a non-relational database modeled after Google's Bigtable. Bigtable acts up on Google File System, likewise Apache HBase works on top of Hadoop and HDFS.

## Applications of HBase

* It is used whenever there is a need to write heavy applications.
* HBase is used whenever we need to provide fast random access to available data.
* Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

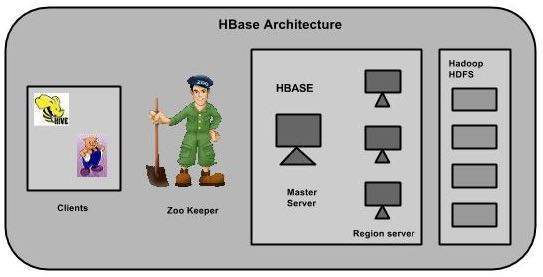
## HBase History

|  |  |
| --- | --- |
| **Year** | **Event** |
| Nov 2006 | Google released the paper on BigTable. |
| Feb 2007 | Initial HBase prototype was created as a Hadoop contribution. |
| Oct 2007 | The first usable HBase along with Hadoop 0.15.0 was released. |
| Jan 2008 | HBase became the sub project of Hadoop. |
| Oct 2008 | HBase 0.18.1 was released. |
| Jan 2009 | HBase 0.19.0 was released. |
| Sept 2009 | HBase 0.20.0 was released. |
| May 2010 | HBase became Apache top-level project. |

# **Architecture**

tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.

**Note:** The term ‘store’ is used for regions to explain the storage structure.



HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

MasterServer

The master server -

* Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
* Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
* Maintains the state of the cluster by negotiating the load balancing.
* Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

Regions are nothing but tables that are split up and spread across the region servers.

Region server

The region servers have regions that -

* Communicate with the client and handle data-related operations.
* Handle read and write requests for all the regions under it.
* Decide the size of the region by following the region size thresholds.

When we take a deeper look into the region server, it contain regions and stores as shown below:



The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

Zookeeper

* Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
* Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
* In addition to availability, the nodes are also used to track server failures or network partitions.
* Clients communicate with region servers via zookeeper.
* In pseudo and standalone modes, HBase itself will take care of zookeeper.

Installing HBase

We can install HBase in any of the three modes: Standalone mode, Pseudo Distributed mode, and Fully Distributed mode.

Installing HBase in Standalone Mode

$wget http://www.interior-dsgn.com/apache/hbase/stable/hbase-0.98.8-

hadoop2-bin.tar.gz

$tar -zxvf hbase-0.98.8-hadoop2-bin.tar.gz

Shift to super user mode and move the HBase folder to /usr/local as shown below.

$su

$password: enter your password here

mv hbase-0.99.1/\* Hbase/

### Configuring HBase in Standalone Mode

Before proceeding with HBase, you have to edit the following files and configure HBase.

### hbase-env.sh

Set the java Home for HBase and open **hbase-env.sh** file from the conf folder. Edit JAVA\_HOME environment variable and change the existing path to your current JAVA\_HOME variable as shown below.

cd /usr/local/Hbase/conf

gedit hbase-env.sh

This will open the env.sh file of HBase. Now replace the existing **JAVA\_HOME**value with your current value as shown below.

export JAVA\_HOME=/usr/lib/jvm/java-1.7.0

### hbase-site.xml

This is the main configuration file of HBase. Set the data directory to an appropriate location by opening the HBase home folder in /usr/local/HBase. Inside the conf folder, you will find several files, open the **hbase-site.xml** file as shown below.

#cd /usr/local/HBase/

#cd conf

# gedit hbase-site.xml

Inside the **hbase-site.xml** file, you will find the <configuration> and </configuration> tags. Within them, set the HBase directory under the property key with the name “hbase.rootdir” as shown below.

<configuration>

//Here you have to set the path where you want HBase to store its files.

<property>

<name>hbase.rootdir</name>

<value>file:/home/hadoop/HBase/HFiles</value>

</property>

//Here you have to set the path where you want HBase to store its built in zookeeper files.

<property>

<name>hbase.zookeeper.property.dataDir</name>

<value>/home/hadoop/zookeeper</value>

</property>

</configuration>

With this, the HBase installation and configuration part is successfully complete. We can start HBase by using **start-hbase.sh** script provided in the bin folder of HBase. For that, open HBase Home Folder and run HBase start script as shown below.

$cd /usr/local/HBase/bin

$./start-hbase.sh

If everything goes well, when you try to run HBase start script, it will prompt you a message saying that HBase has started.

starting master, logging to /usr/local/HBase/bin/../logs/hbase-tpmaster-localhost.localdomain.out

### Installing HBase in Pseudo-Distributed Mode

Let us now check how HBase is installed in pseudo-distributed mode.

#### CONFIGURING HBASE

Before proceeding with HBase, configure Hadoop and HDFS on your local system or on a remote system and make sure they are running. Stop HBase if it is running.

**hbase-site.xml**

Edit hbase-site.xml file to add the following properties.

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

It will mention in which mode HBase should be run. In the same file from the local file system, change the hbase.rootdir, your HDFS instance address, using the hdfs://// URI syntax. We are running HDFS on the localhost at port 8030.

<property>

<name>hbase.rootdir</name>

<value>hdfs://localhost:8030/hbase</value>

</property>

### Starting HBase

After configuration is over, browse to HBase home folder and start HBase using the following command.

$cd /usr/local/HBase

$bin/start-hbase.sh

**Note:** Before starting HBase, make sure Hadoop is running.

### Checking the HBase Directory in HDFS

HBase creates its directory in HDFS. To see the created directory, browse to Hadoop bin and type the following command.

$ ./bin/hadoop fs -ls /hbase

If everything goes well, it will give you the following output.

Found 7 items

drwxr-xr-x - hbase users 0 2014-06-25 18:58 /hbase/.tmp

drwxr-xr-x - hbase users 0 2014-06-25 21:49 /hbase/WALs

drwxr-xr-x - hbase users 0 2014-06-25 18:48 /hbase/corrupt

drwxr-xr-x - hbase users 0 2014-06-25 18:58 /hbase/data

-rw-r--r-- 3 hbase users 42 2014-06-25 18:41 /hbase/hbase.id

-rw-r--r-- 3 hbase users 7 2014-06-25 18:41 /hbase/hbase.version

drwxr-xr-x - hbase users 0 2014-06-25 21:49 /hbase/oldWALs

## Starting and Stopping a Master

Using the “local-master-backup.sh” you can start up to 10 servers. Open the home folder of HBase, master and execute the following command to start it.

$ ./bin/local-master-backup.sh 2 4

To kill a backup master, you need its process id, which will be stored in a file named **“/tmp/hbase-USER-X-master.pid.”** you can kill the backup master using the following command.

$ cat /tmp/hbase-user-1-master.pid |xargs kill -9

## Starting and Stopping RegionServers

You can run multiple region servers from a single system using the following command.

$ .bin/local-regionservers.sh start 2 3

To stop a region server, use the following command.

$ .bin/local-regionservers.sh stop 3

## Starting HBaseShell

After Installing HBase successfully, you can start HBase Shell. Below given are the sequence of steps that are to be followed to start the HBase shell. Open the terminal, and login as super user.

### Start Hadoop File System

Browse through Hadoop home sbin folder and start Hadoop file system as shown below.

$cd $HADOOP\_HOME/sbin

$start-all.sh

### Start HBase

Browse through the HBase root directory bin folder and start HBase.

$cd /usr/local/HBase

$./bin/start-hbase.sh

### Start HBase Master Server

This will be the same directory. Start it as shown below.

$./bin/local-master-backup.sh start 2 (number signifies specific

server.)

### Start Region

Start the region server as shown below.

$./bin/./local-regionservers.sh start 3

### Start HBase Shell

You can start HBase shell using the following command.

$cd bin

$./hbase shell

This will give you the HBase Shell Prompt as shown below.

2014-12-09 14:24:27,526 INFO [main] Configuration.deprecation:

hadoop.native.lib 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 0.98.8-hadoop2, r6cfc8d064754251365e070a10a82eb169956d5fe, Fri

Nov 14 18:26:29 PST 2014

hbase(main):001:0>

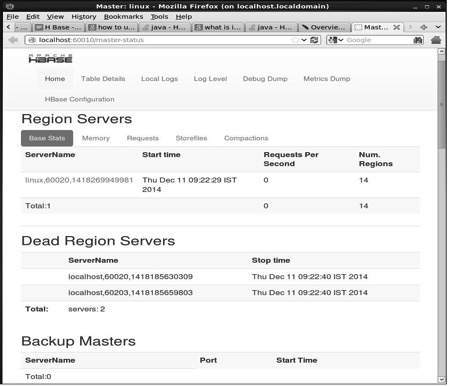
## HBase Web Interface

To access the web interface of HBase, type the following url in the browser.

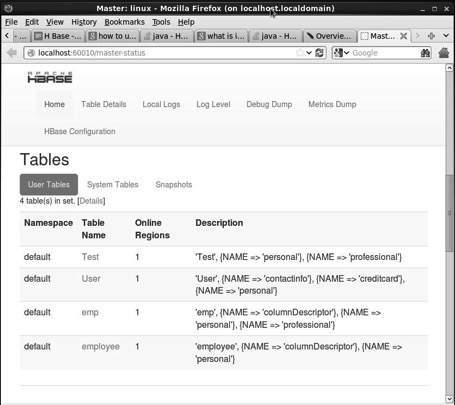
http://localhost:60010

This interface lists your currently running Region servers, backup masters and HBase tables.

### HBase Region servers and Backup Masters



### HBase Tables



## Setting Java Environment

We can also communicate with HBase using Java libraries, but before accessing HBase using Java API you need to set classpath for those libraries.

### Setting the Classpath

Before proceeding with programming, set the classpath to HBase libraries in **.bashrc** file. Open **.bashrc** in any of the editors as shown below.

$ gedit ~/.bashrc

Set classpath for HBase libraries (lib folder in HBase) in it as shown below.

export CLASSPATH = $CLASSPATH://home/hadoop/hbase/lib/\*

This is to prevent the “class not found” exception while accessing the HBase using java API

## HBase Shell

HBase contains a shell using which you can communicate with HBase. HBase uses the Hadoop File System to store its data. It will have a master server and region servers. The data storage will be in the form of regions (tables). These regions will be split up and stored in region servers.

The master server manages these region servers and all these tasks take place on HDFS. Given below are some of the commands supported by HBase Shell.

## General Commands

* **status** - Provides the status of HBase, for example, the number of servers.
* **version** - Provides the version of HBase being used.
* **table\_help** - Provides help for table-reference commands.
* **whoami** - Provides information about the user.

## Data Definition Language

These are the commands that operate on the tables in HBase.

* **create** - Creates a table.
* **list** - Lists all the tables in HBase.
* **disable** - Disables a table.
* **is\_disabled** - Verifies whether a table is disabled.
* **enable** - Enables a table.
* **is\_enabled** - Verifies whether a table is enabled.
* **describe** - Provides the description of a table.
* **alter** - Alters a table.
* **exists** - Verifies whether a table exists.
* **drop** - Drops a table from HBase.
* **drop\_all** - Drops the tables matching the ‘regex’ given in the command.
* **Java Admin API** - Prior to all the above commands, Java provides an Admin API to achieve DDL functionalities through programming. Under **org.apache.hadoop.hbase.client** package, HBaseAdmin and HTableDescriptor are the two important classes in this package that provide DDL functionalities.

## Data Manipulation Language

* **put** - Puts a cell value at a specified column in a specified row in a particular table.
* **get** - Fetches the contents of row or a cell.
* **delete** - Deletes a cell value in a table.
* **deleteall** - Deletes all the cells in a given row.
* **scan** - Scans and returns the table data.
* **count** - Counts and returns the number of rows in a table.
* **truncate** - Disables, drops, and recreates a specified table.
* **Java client API** - Prior to all the above commands, Java provides a client API to achieve DML functionalities, **CRUD** (Create Retrieve Update Delete) operations and more through programming, under org.apache.hadoop.hbase.client package. **HTable Put** and **Get** are the important classes in this package.

## 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

## 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

# **Admin API**

Class HBaseAdmin

**HBaseAdmin** is a class representing the Admin. This class belongs to the **org.apache.hadoop.hbase.client** package. Using this class, you can perform the tasks of an administrator. You can get the instance of Admin using **Connection.getAdmin()** method.

Methods and Description

|  |  |
| --- | --- |
| **S.No.** | **Methods and Description** |
| 1 | **void createTable(HTableDescriptor desc)**  Creates a new table. |
| 2 | **void createTable(HTableDescriptor desc, byte[][] splitKeys)**  Creates a new table with an initial set of empty regions defined by the specified split keys. |
| 3 | **void deleteColumn(byte[] tableName, String columnName)**  Deletes a column from a table. |
| 4 | **void deleteColumn(String tableName, String columnName)**  Delete a column from a table. |
| 5 | **void deleteTable(String tableName)**  Deletes a table. |

Class Descriptor

This class contains the details about an HBase table such as:

* the descriptors of all the column families,
* if the table is a catalog table,
* if the table is read only,
* the maximum size of the mem store,
* when the region split should occur,
* co-processors associated with it, etc.

Constructors

|  |  |
| --- | --- |
| **S.No.** | **Constructor and summary** |
| 1 | **HTableDescriptor(TableName name)**  Constructs a table descriptor specifying a TableName object. |

Methods and Description

|  |  |
| --- | --- |
| **S.No.** | **Methods and Description** |
| 1 | **HTableDescriptor addFamily(HColumnDescriptor family)**  Adds a column family to the given descriptor |

Create table

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

### 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

## 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'

ROW COLUMN + CELL

1 column = personal data:city, timestamp = 1417516501, value = hyderabad

1 column = personal data:name, timestamp = 1417525058, value = ramu

1 column = professional data:designation, timestamp = 1417532601, value = manager

1 column = professional data:salary, timestamp = 1417524244109, value = 50000

2 column = personal data:city, timestamp = 1417524574905, value = chennai

2 column = personal data:name, timestamp = 1417524556125, value = ravi

2 column = professional data:designation, timestamp = 14175292204, value = sr:engg

2 column = professional data:salary, timestamp = 1417524604221, value = 30000

3 column = personal data:city, timestamp = 1417524681780, value = delhi

3 column = personal data:name, timestamp = 1417524672067, value = rajesh

3 column = professional data:designation, timestamp = 14175246987, value = jr:engg

3 column = professional data:salary, timestamp = 1417524702514, value = 25000

3 row(s) in 0.0400 seconds

## 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

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

hbase> describe 'table name'

Given below is the output of the describe command on the **emp** table.

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

DESCRIPTION

ENABLED

'emp', {NAME ⇒ 'READONLY', DATA\_BLOCK\_ENCODING ⇒ 'NONE', BLOOMFILTER

⇒ 'ROW', REPLICATION\_SCOPE ⇒ '0', COMPRESSION ⇒ 'NONE', VERSIONS ⇒

'1', TTL true

⇒ 'FOREVER', MIN\_VERSIONS ⇒ '0', KEEP\_DELETED\_CELLS ⇒ 'false',

BLOCKSIZE ⇒ '65536', IN\_MEMORY ⇒ 'false', BLOCKCACHE ⇒ 'true'}, {NAME

⇒ 'personal

data', DATA\_BLOCK\_ENCODING ⇒ 'NONE', BLOOMFILTER ⇒ 'ROW',

REPLICATION\_SCOPE ⇒ '0', VERSIONS ⇒ '5', COMPRESSION ⇒ 'NONE',

MIN\_VERSIONS ⇒ '0', TTL

⇒ 'FOREVER', KEEP\_DELETED\_CELLS ⇒ 'false', BLOCKSIZE ⇒ '65536',

IN\_MEMORY ⇒ 'false', BLOCKCACHE ⇒ 'true'}, {NAME ⇒ 'professional

data', DATA\_BLO

CK\_ENCODING ⇒ 'NONE', BLOOMFILTER ⇒ 'ROW', REPLICATION\_SCOPE ⇒ '0',

VERSIONS ⇒ '1', COMPRESSION ⇒ 'NONE', MIN\_VERSIONS ⇒ '0', TTL ⇒

'FOREVER', K

EEP\_DELETED\_CELLS ⇒ 'false', BLOCKSIZE ⇒ '65536', IN\_MEMORY ⇒

'false', BLOCKCACHE ⇒ 'true'}, {NAME ⇒ 'table\_att\_unset',

DATA\_BLOCK\_ENCODING ⇒ 'NO

NE', BLOOMFILTER ⇒ 'ROW', REPLICATION\_SCOPE ⇒ '0', COMPRESSION ⇒

'NONE', VERSIONS ⇒ '1', TTL ⇒ 'FOREVER', MIN\_VERSIONS ⇒ '0',

KEEP\_DELETED\_CELLS

⇒ 'false', BLOCKSIZE ⇒ '6

## 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

Updating all regions with the new schema...

0/1 regions updated.

1/1 regions updated.

Done.

0 row(s) in 2.3050 seconds

### Table 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

Updating all regions with the new schema...

0/1 regions updated.

1/1 regions updated.

Done.

0 row(s) in 2.2140 seconds

### 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'

ROW COLUMN+CELL

row1 column = personal:city, timestamp = 1418193767, value = hyderabad

row1 column = personal:name, timestamp = 1418193806767, value = raju

row1 column = professional:designation, timestamp = 1418193767, value = manager

row1 column = professional:salary, timestamp = 1418193806767, value = 50000

1 row(s) in 0.0160 seconds

Now let us delete the column family named **professional** using the alter command.

hbase(main):007:0> alter 'employee','delete'⇒'professional'

Updating all regions with the new schema...

0/1 regions updated.

1/1 regions updated.

Done.

0 row(s) in 2.2380 seconds

Now verify the data in the table after alteration. Observe the column family ‘professional’ is no more, since we have deleted it.

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

ROW COLUMN + CELL

row1 column = personal:city, timestamp = 14181936767, value = hyderabad

row1 column = personal:name, timestamp = 1418193806767, value = raju

1 row(s) in 0.0830 seconds

## 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'

Table emp does exist

0 row(s) in 0.0750 seconds

==================================================================

hbase(main):015:0> exists 'student'

Table student does not exist

0 row(s) in 0.0480 seconds

## 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.\*'

raja

rajani

rajendra

rajesh

raju

Drop the above 5 tables (y/n)?

y

5 tables successfully dropped

## 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.

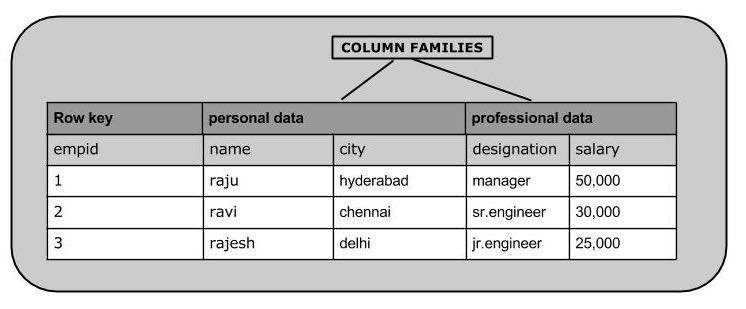
./bin/stop-hbase.sh

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'

ROW COLUMN+CELL

1 column=personal data:city, timestamp=1417524216501, value=hyderabad

1 column=personal data:name, timestamp=1417524185058, value=ramu

1 column=professional data:designation, timestamp=1417524232601,

value=manager

1 column=professional data:salary, timestamp=1417524244109, value=50000

2 column=personal data:city, timestamp=1417524574905, value=chennai

2 column=personal data:name, timestamp=1417524556125, value=ravi

2 column=professional data:designation, timestamp=1417524592204,

value=sr:engg

2 column=professional data:salary, timestamp=1417524604221, value=30000

3 column=personal data:city, timestamp=1417524681780, value=delhi

3 column=personal data:name, timestamp=1417524672067, value=rajesh

3 column=professional data:designation, timestamp=1417524693187,

value=jr:engg

3 column=professional data:salary, timestamp=1417524702514,

value=25000

## 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'

ROW COLUMN + CELL

row1 column = personal:name, timestamp = 1418035791555, value = raju

row1 column = personal:city, timestamp = 1418274645907, value = Delhi

row1 column = professional:designation, timestamp = 141857555,value = manager

row1 column = professional:salary, timestamp = 1418039555, value = 50000

1 row(s) in 0.0100 seconds

## 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'

COLUMN CELL

personal : city timestamp = 1417521848375, value = hyderabad

personal : name timestamp = 1417521785385, value = ramu

professional: designation timestamp = 1417521885277, value = manager

professional: salary timestamp = 1417521903862, value = 50000

4 row(s) in 0.0270 seconds

## 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'}

COLUMN CELL

personal:name timestamp = 1418035791555, value = raju

1 row(s) in 0.0080 seconds

## 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'

ROW COLUMN + CELL

2 column = personal data:city, timestamp = 1417524574905, value = chennai

2 column = personal data:name, timestamp = 1417524556125, value = ravi

2 column = professional data:designation, timestamp = 1417524204, value = sr:engg

2 column = professional data:salary, timestamp = 1417524604221, value = 30000

3 column = personal data:city, timestamp = 1417524681780, value = delhi

3 column = personal data:name, timestamp = 1417524672067, value = rajesh

3 column = professional data:designation, timestamp = 1417523187, value = jr:engg

3 column = professional data:salary, timestamp = 1417524702514, value = 25000

## Scaning using HBase Shell

The **scan** command is used to view the data in HTable. Using the scan command, you can get the table data. Its syntax is as follows:

scan ‘<table name>’

### Example

The following example shows how to read data from a table using the scan command. Here we are reading the **emp** table.

hbase(main):010:0> scan 'emp'

ROW COLUMN + CELL

1 column = personal data:city, timestamp = 1417521848375, value = hyderabad

1 column = personal data:name, timestamp = 1417521785385, value = ramu

1 column = professional data:designation, timestamp = 1417585277,value = manager

1 column = professional data:salary, timestamp = 1417521903862, value = 50000

1 row(s) in 0.0370 seconds

## 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