-A APACHE HADOOP PROJECT

OUTLINE

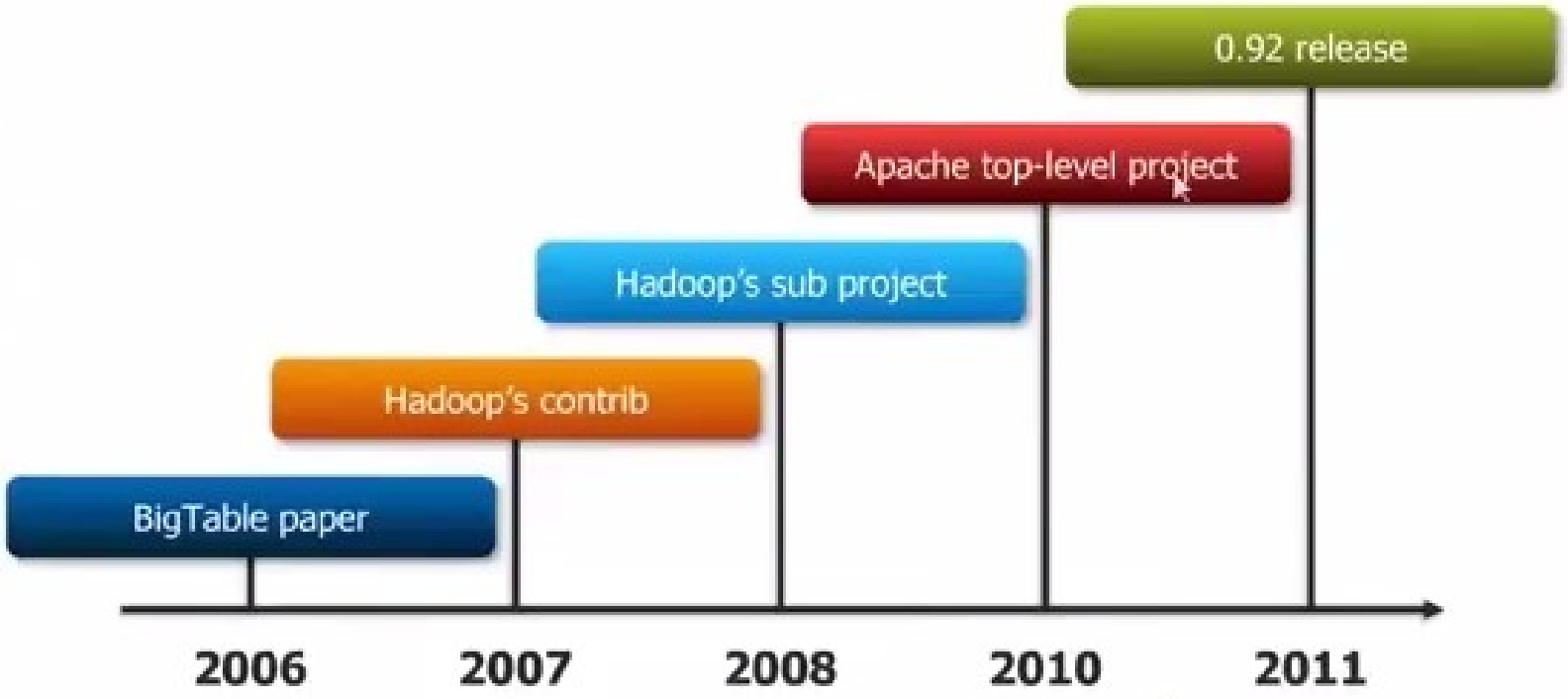
- History
- Why use Hbase?
- Hbase vs. HDFS
- What is Hbase?
- Hbase Data Model
- Hbase Architecture
- Acid properties in hbase
- Accessing hbase
- Hbase API
- Hbase vs. RDBMS
- Installation
- References

INTRODUCTION

- HBase is developed as part of **Apache Software Foundation's Apache Hadoop project** and runs on top of **HDFS** (Hadoop Distributed Filesystem) providing **BigTable**-like capabilities for Hadoop.

- Apache HBase began as a project by the company **Powerset** out of a need to process massive amounts of data for the purposes of natural language search.

HISTORY


0.92 release
Apache top-level project
Hadoop's sub project
Hadoop's contrib
BigTable paper
2006
2007
2008
2010
2011

WHY USE HBASE?

- Storing large amounts of data.
- High throughput for a large number of requests.
- Storing unstructured or variable column data.
- Big data with random read writes.

HBASE VS. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes

- **HDFS** is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates

HBASE VS. HDFS

- **HBase** is designed to efficiently address the below points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates
- HBase updates are done by creating new versions of values

HBASE VS. HDFS

	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

WHAT IS HBASE?

- HBase is a Java implementation of Google's BigTable.
- Google defines BigTable as a "sparse, distributed, persistent multidimensional sorted map."

OPEN SOURCE

- Committers and contributors from diverse organizations like

Facebook, Cloudera, StumbleUpon,

TrendMicro, Intel, Horton works, Continuity etc.

SPARSE

- Sparse means that fields in rows can be empty or NULL but that doesn’t bring HBase to a screeching halt.
- HBase can handle the fact that we don’t (yet) know that information.
- Sparse data is supported with no waste of costly storage space.

SPARSE

- We can not only skip fields at no cost also dynamically add fields (or columns in terms of HBase) over time without having to redesign the schema or disrupt operations.

- HBase as a schema-less data store; that is, it’s fluid — we can add to, subtract from or modify the schema as you go along.

DISTRIBUTED AND PERSISTENT

- Persistent simply means that the data you store in HBase will persist or remain after our program or session ends.
- Just as HBase is an open source implementation of BigTable, HDFS is an open source implementation of GFS.
- HBase leverages HDFS to persist its data to disk storage.
- By storing data in HDFS, HBase offers reliability, availability, seamless scalability and high performance — all on cost effective distributed servers.

MULTIDIMENSIONAL SORTED MAP

- A map (also known as an associative array) is an abstract collection of key-value pairs, where the key is unique.
- The keys are stored in HBase and sorted in byte lexicographical order.
- Each value can have multiple versions, which makes the data model multidimensional. By default, data versions are implemented with a timestamp.

HBASE DATA MODEL

- HBase data stores consist of one or more tables, which are indexed by row keys.
- Data is stored in rows with columns, and rows can have multiple versions. By default, data versioning for rows is implemented with time stamps.
- Columns are grouped into column families, which must be defined up front during table creation.
- Column families are grouped together on disk, so grouping data with similar access patterns reduces overall disk access and increases performance.

HBASE DATA MODEL

Table 12-2 Logical View of Customer Contact Information in HBase

Row Key	*Column Family: {Column Qualifier:Version:Value}*
00001	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'} ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}
00002	CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe', ContactInfo: { 'SA': 1383859185577:'7 HBase Ave, CA 22222'}

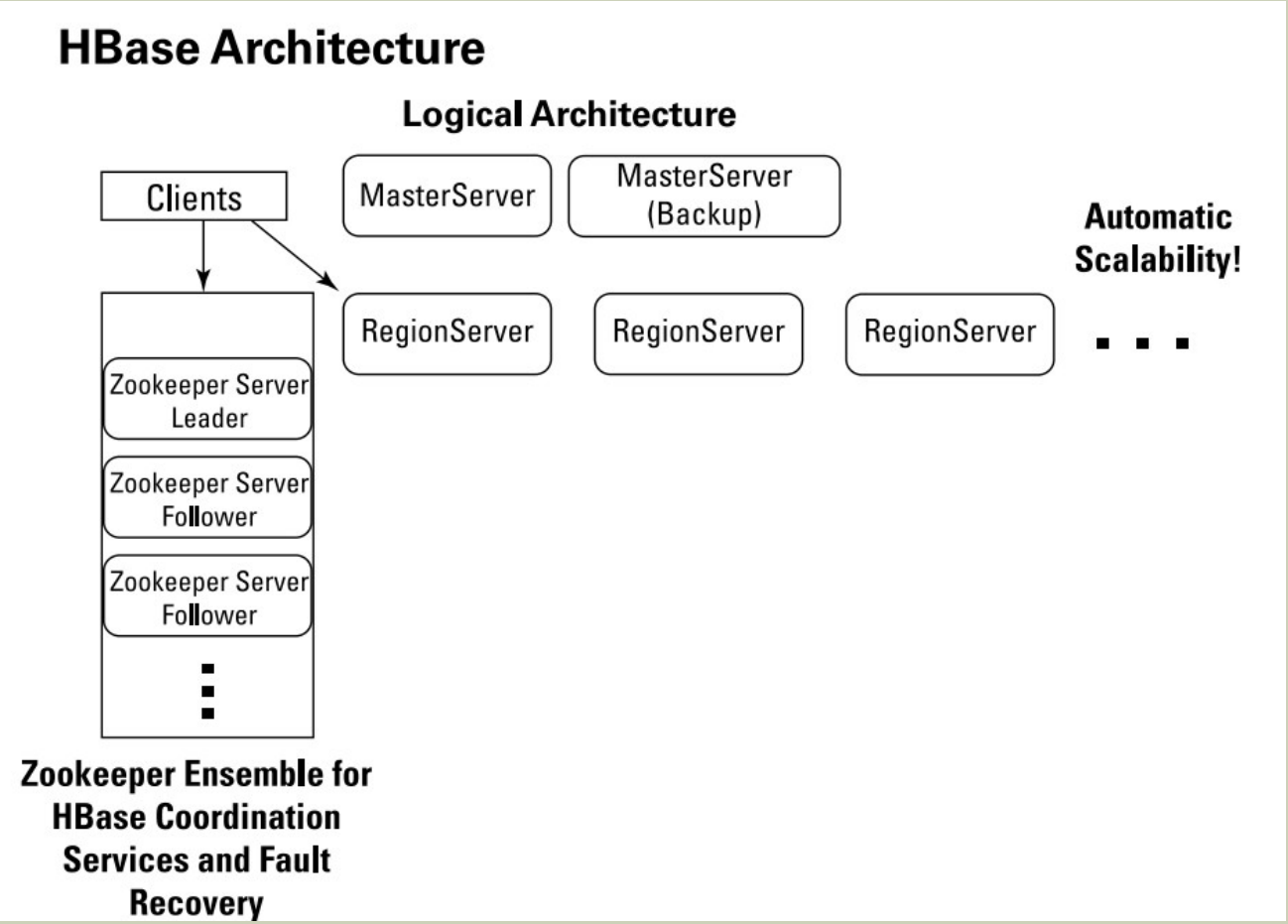
HBASE DATA MODEL

- Column qualifiers are specific names assigned to our data values.
- Unlike column families, column qualifiers can be virtually unlimited in content, length and number.
- Because the number of column qualifiers is variable new data can be added to column families on the fly, making HBase flexible and highly scalable.

HBASE DATA MODEL

- HBase stores the column qualifier with our value, and since HBase doesn’t limit the number of column qualifiers we can have, creating long column qualifiers can be quite costly in terms of storage.
- Values stored in HBase are time stamped by default, which means we have a way to identify different versions of our data right out of the box.
- The versioned data is stored in decreasing order, so that the most recent value is returned by default unless a query specifies a particular timestamp.

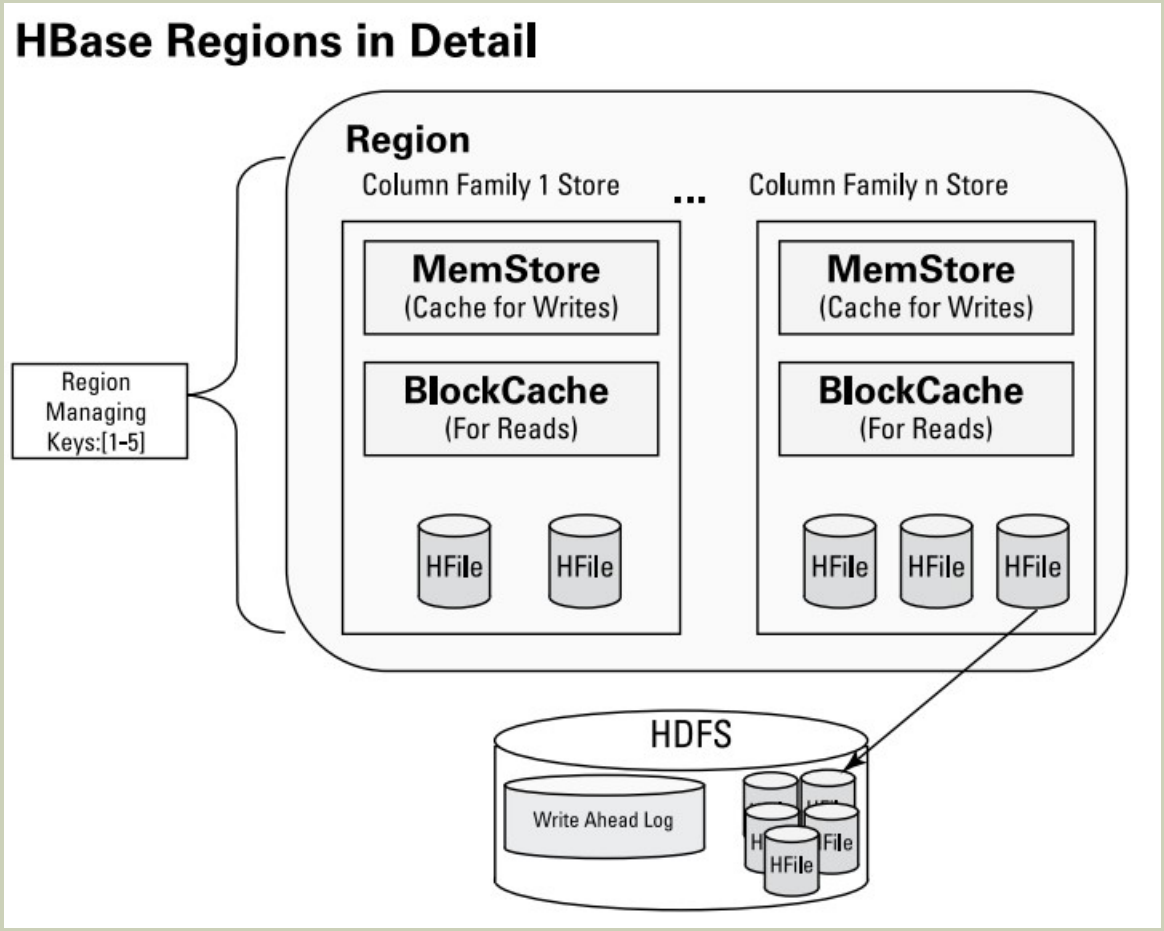
HBASE ARCHITECTURE



HBASE ARCHITECTURE: REGION SERVERS

- RegionServers are the software processes (often called daemons) we activate to store and retrieve data in HBase. In production environments, each RegionServer is deployed on its own dedicated compute node.
- When a table grows beyond a configurable limit HBase system automatically splits the table and distributes the load to another RegionServer. This is called **auto-sharding**.
- As tables are split, the splits become regions. Regions store a range of key-value pairs, and each RegionServer manages a configurable number of regions.

HBASE ARCHITECTURE

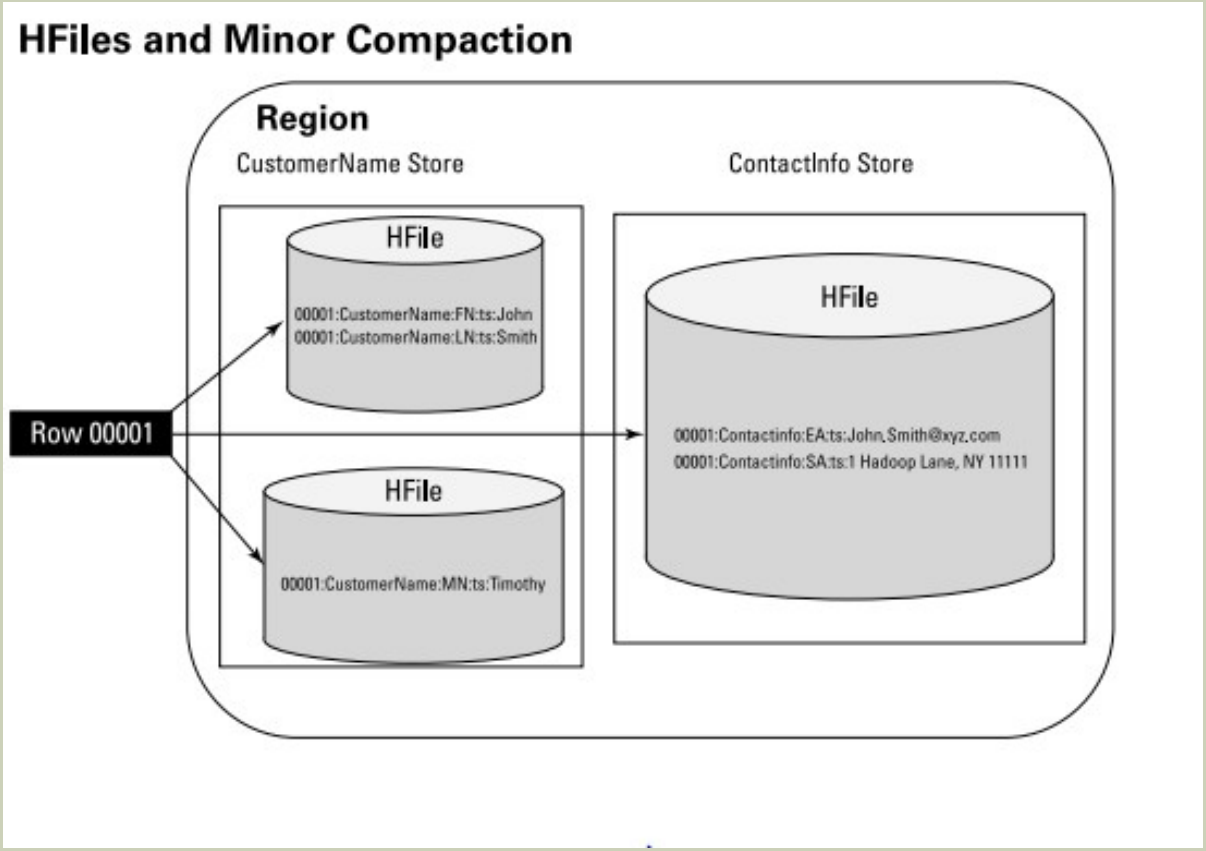


HBASE ARCHITECTURE: REGION SERVERS

- Each column family store object has a read cache called the BlockCache and a write cache called the MemStore.
- The BlockCache helps with random read performance.
- The Write Ahead Log (WAL, for short) ensures that our Hbase writes are reliable.
- The design of HBase is to flush column family data stored in the MemStore to one HFile per flush. Then at configurable intervals HFiles are combined into larger HFiles.

HBASE ARCHITECTURE: COMPACTIONS

Compaction, the process by which HBase cleans up after itself, comes in two flavors: major and minor.



HBASE ARCHITECTURE: COMPACTIONS

- Minor compactions combine a configurable number of smaller HFiles into one larger HFile.
- Minor compactions are important because without them, reading a particular row can require many disk reads and cause slow overall performance.
- A major compaction seeks to combine *all* HFiles into one large HFile. In addition, a major compaction does the cleanup work after a user deletes a record.

HBASE ARCHITECTURE: MASTER SERVER

Responsibilities of a Master Server:

- Monitor the region servers in the Hbase clusters.
- Handle metadata operations.
- Assign regions.
- Manage region server failover.

HBASE ARCHITECTURE: MASTER SERVER

- Oversee load balancing of regions across all available region servers.
- Manage and clean catalog tables.
- Clear the WAL.
- Provide a coprocessor framework for observing master operations.

There should always be a backup MasterServer in any HBase cluster incase of failover of the actual MasterServer.

HBASE ARCHITECTURE: ZOOKEEPER

- HBase clusters can be huge and coordinating the operations of the MasterServers, RegionServers, and clients can be a daunting task, but that's where **Zookeeper** enters the picture.

- Zookeeper is a distributed cluster of servers that collectively provides reliable coordination and synchronization services for clustered applications.

HBASE ARCHITECTURE: CAP THEOREM

- HBase provides a high degree of reliability. HBase can tolerate any failure and still function properly.

- HBase provides “Consistency” and “Partition Tolerance” but is not always “Available.”

ACID PROPERTIES IN HBASE

- When compared to an RDBMS, HBase isn’t considered an ACID-compliant database.
- However it guarantees the following aspects-
- Atomic
- Consistency
- Durability

ACCESSING HBASE

- Java API
- REST/HTTP
- Apache Thrift
- Hive/Pig for analytics

HBASE API

Types of access:

- Gets: Gets a row’s data based on the row key.
- Puts: Inserts a row with data based on the row key.
- Scans: Finding all matching rows based on the row key.

  Scan logic can be increased by using filters.

GETS

```
1 Get g = new Get(ROW_KEY_BYTES);
2 Result r= table.get(g);
3 byte[] byteArray =
  r.getValue(COLFAM_BYTS,COLDESC_BYTS);
4 String columnValue =
  Bytes.toString(byteArray);
```

PUTS

```
1 Put p = new
  Put(Bytes.toBytes(ROW_KEY_BYTES);
2 p.add(COLFAM_BYTES, COLDESC_BYTES,
  Bytes.toBytes("value"));
3
4 table.put(p);
```

HBASE VS. RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

INSTALLATION

- HBase requires that a JDK be installed.
  http://java.com/en/download/index.jsp
- Choose a download site from the list of Apache Download Mirrors given in the Apache website.
  http://www.apache.org/dyn/closer.cgi/hbase/
- Extract the downloaded file, and change to a newly-created directory.
- For HBase 0.98.5 and later, we are required to set the JAVA_HOME environment variable before starting Hbase using conf/hbase-env.sh.

INSTALLATION

- The JAVA_HOME variable should be set to a directory which contains the executable file bin/java.

- Edit conf/hbase-site.xml, which is the main HBase configuration file.

- The bin/start-hbase.sh script is provided as a convenient way to start HBase.

```
$ ./bin/hbase shell
hbase(main):001:0>
```

INSTALLATION

- Connect to your running instance of HBase using the **hbase shell** command.

- Use the create command to create a new table. You must specify the table name and the ColumnFamily name.

  hbase> **create 'test', 'cf'**

  0 row(s) in 1.2200 seconds

- Use the list command to see the List Information About your Table.

  hbase> **list 'test'**

  TABLE

  test

  1 row(s) in 0.0350 seconds

  => ["test"]

INSTALLATION

- To put data into your table, use the put command.

  hbase> **put 'test', 'row1', 'cf:a', 'value1'**
  0 row(s) in 0.1770 seconds

- Use the **scan** command to scan the table for data.

  hbase> **scan 'test'**
  ROW COLUMN+CELL
  row1 column=cf:a, timestamp=1403759475114, value=value1
  1 row(s) in 0.0440 seconds

INSTALLATION

- To get a single row of data at a time, use the **get** command.

```
hbase> get 'test','row1'
COLUMN          CELL
cf:a             timestamp=1403759475114, value=value1
1 row(s) in 0.0230 seconds
```

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

```
hbase> disable 'test‘
0 row(s) in 1.6270 seconds

hbase> enable 'test'
0 row(s) in 0.4500 seconds
```

INSTALLATION

- To drop (delete) a table, use the drop command.
  hbase> drop 'test'
  0 row(s) in 0.2900 seconds

- To exit the HBase Shell use bin/stop-hbase.sh script.
  $ ./bin/stop-hbase.sh
  stopping hbase..... $

- For the detailed installation procedure look at,
  http://hbase.apache.org/cygwin.html

POWERED BY HBASE

REFERENCES

- https://www.usenix.org/system/files/conference/fast14/fast14-paper_harter.pdf
- http://www.manning.com/dimidukkhurana/HBiAsample_ch1.pdf
- https://research.facebook.com/publications/1420502254864214/analysis-of-hdfs-under-hbase-a-facebook-messages-case-study/
- http://blog.cloudera.com/blog/2012/09/the-action-on-hbase-in-action/
- http://www.informationweek.com/big-data/software-platforms/big-data-debate-will-hbase-dominate-nosql/d/d-id/1111048
- http://hbasecon.com/archive.html
- http://jimbojw.com/wiki/index.php?title=Understanding_Hbase_and_BigTable
- ftp://61.135.158.199/pub/books/HBase%20The%20Definitive%20Guide.pdf

QUESTIONS?

THANK YOU