HBase

ixAT Solution – ixatsolutions@gmail.com

Definition

- Apache HBase is the Hadoop database, a distributed, scalable, big data store.

- Modelled after Google's BigTable, white paper on Googles BigTable over here -- http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf
- HBase/BigTable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

Overview

- **HBase is a distributed column-oriented data store built on top of HDFS**
- **HBase is an Apache open source project whose goal is to provide storage for the Hadoop Distributed Computing**
- **Data is logically organized into tables, rows and columns**

Traditional RDBMS

Rows stored sequentially

Key	Fname	Lname	State	Zip	Phone	Age	Sex
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F

- Provides best performance when most queries are for multiple columns of a given row
- Access pattern is row based.
- Suited for OLTP models

Traditional RDBMS

Indexes

Indexes make accessing a row very fast

Key	RowID
1	0001B008D23A671A
2	0001B008D23A671B
3	0001B008D23A671C
4	0001B008D23A671D
5	0001B008D23A671E

WHERE key=4

Elmer Fudd calls customer service

Key	Fname	Lname	State	Zip	Phone	Age	Sex
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F

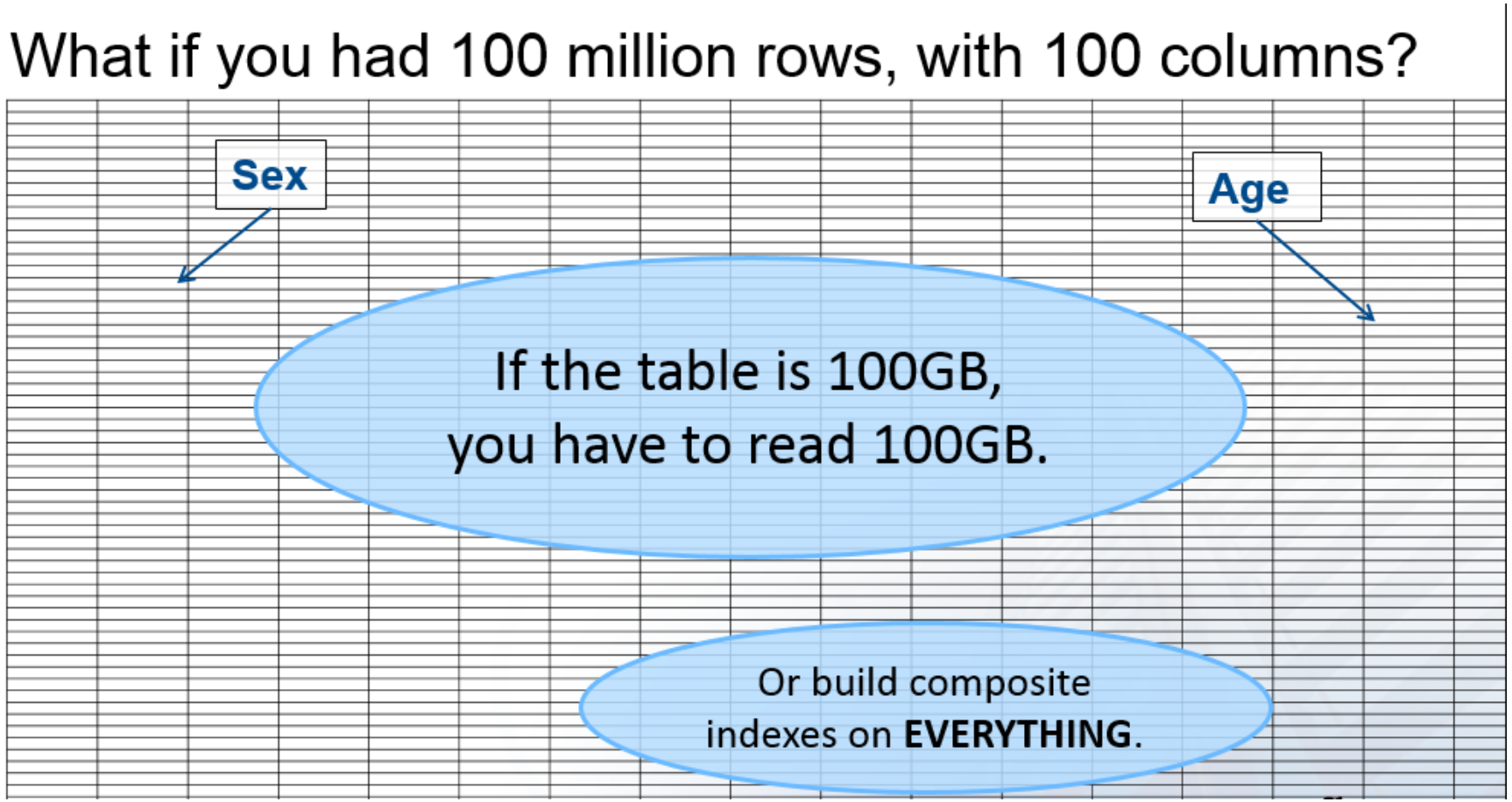
but don't help on analytical queries scanning many rows

e.g. What is the average age of all the males?

Phone	RowID
(207) 882-7323	0001B008D23A671D
(209) 375-6572	0001B008D23A671B
(212) 227-1810	0001B008D23A671C
(718) 938-3235	0001B008D23A671A
(978) 744-0991	0001B008D23A671E

WHERE phone='(207) 882-7323'

RDBMS



Columnar Store

Each column is stored as a separate section

Key	Fname	Lname	State	Zip	Phone	Age	Sex
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F

Each column for a given row is at the same offset (auto-indexing)

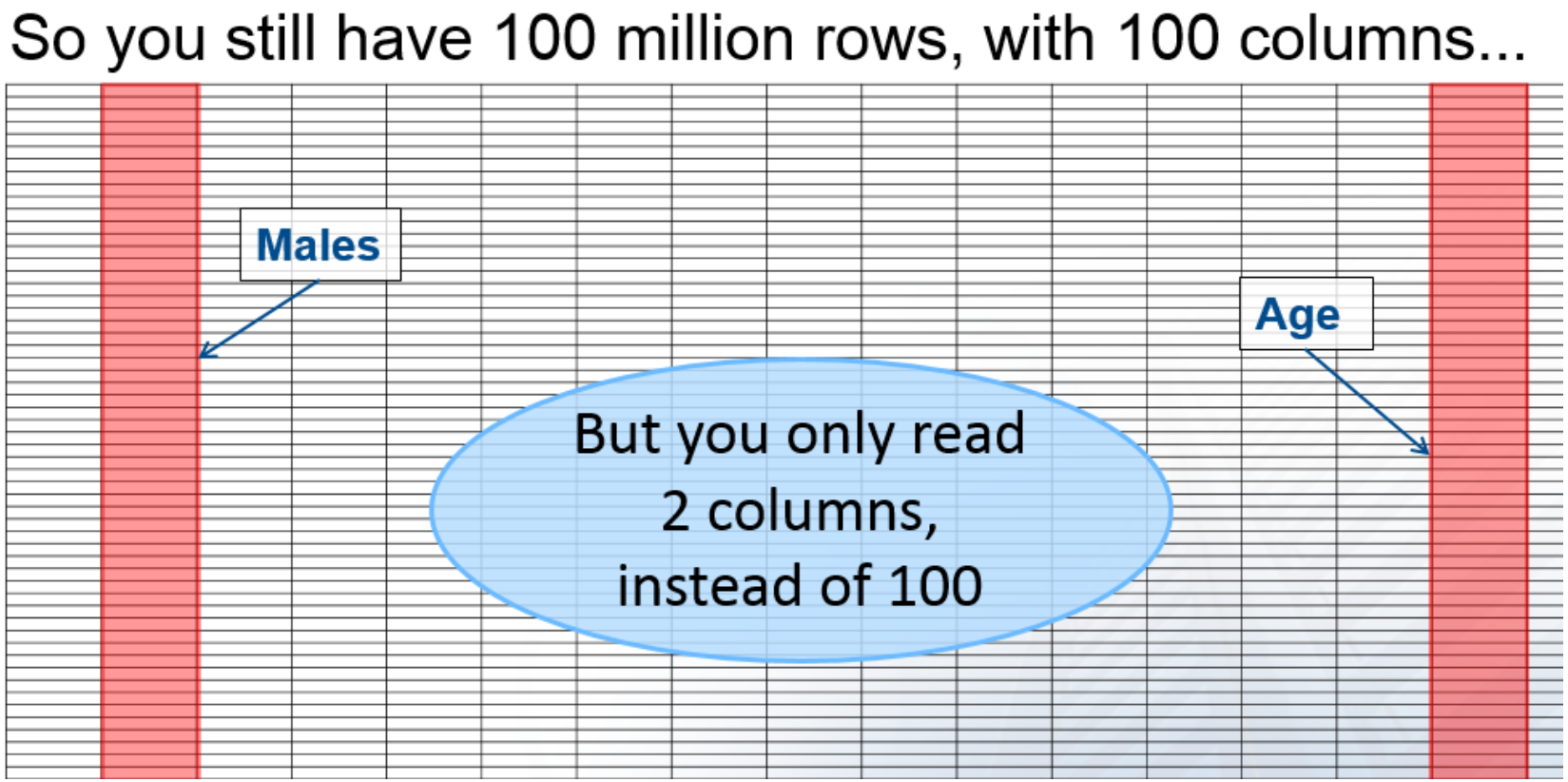
Columnar Store

You only read the sections that your anaytics is interested in

Key	Fname	Lname	State	Zip	Phone	Age	Sex
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F

Also get improved compression because all data in one file is the same data type.

Columnar Store – I/O Reduction



Columnar Store

Columnar databases produce automatic vertical partitioning

1	Bugs	Bunny	Brooklyn	NY	11217	(718) 938-3235
2	Yosemite	Sam	Wawona	CA	95389	(209) 375-6572
3	Daffy	Duck	New York	NY	10013	(212) 227-1810
4	Elmer	Fudd	Wiscasset	ME	04578	(207) 882-7323
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
8m	Snoopy	Brown	Springfield	MA	01105	(413) 781-6500

Columnar Store

Columnar databases produce automatic vertical partitioning

1	Bugs	Bunny	Brooklyn	NY	11217	(718) 938-3235
2	Yosemite	Sam	Wawona	CA	95389	(209) 375-6572
3	Daffy	Duck	New York	NY	10013	(212) 227-1810
4	Elmer	Fudd	Wiscasset	ME	04578	(207) 882-7323
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
:	:	:	:	:	:	:
8m	Snoopy	Brown	Springfield	MA	01105	(413) 781-6500

Partitioning

1	Bugs	Bunny	Brooklyn	NY	11217	(718) 938-3235
2	Yosemite	Sam	Wawona	CA	95389	(209) 375-6572
3	Daffy	Duck	New York	NY	10013	(212) 227-1810
4	Elmer	Fudd	Wiscasset	ME	04578	(207) 882-7323
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
8m	Snoopy	Brown	Springfield	MA	01105	(413) 781-6500
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Many Columnar stores provide Horizontal partitioning.

Thus one could parallelize the query and achieve scalability

Columnar Store – Easy to add a new Column

Row-oriented: Usually requires rebuilding table

Key	Fname	Lname	State	Zip	Phone	Age	Sex	Golf
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M	Y
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M	N
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M	Y
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M	Y
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F	N

Addition of column shifts every row

Column-oriented: Just create another file

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
ME
MA

Zip
11217
95389
10013
04578
01970

Phone
(718) 938-3235
(209) 375-6572
(212) 227-1810
(207) 882-7323
(978) 744-0991

Age
34
52
35
43
57

Sex
M
M
M
M
F

Golf
Y
N
Y
Y
N

Columnar Store – Easy to add a new Column

Row-oriented: Usually requires rebuilding table

Key	Fname	Lname	State	Zip	Phone	Age	Sex	Golf
1	Bugs	Bunny	NY	11217	(718) 938-3235	34	M	Y
2	Yosemite	Sam	CA	95389	(209) 375-6572	52	M	N
3	Daffy	Duck	NY	10013	(212) 227-1810	35	M	Y
4	Elmer	Fudd	ME	04578	(207) 882-7323	43	M	Y
5	Witch	Hazel	MA	01970	(978) 744-0991	57	F	N

Addition of column shifts every row

Column-oriented: Just create another file

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
ME
MA

Zip
11217
95389
10013
04578
01970

Phone
(718) 938-3235
(209) 375-6572
(212) 227-1810
(207) 882-7323
(978) 744-0991

Age
34
52
35
43
57

Sex
M
M
M
M
F

Golf
Y
N
Y
Y
N

Columnar Store

- Insert's
- Deletes
- Updates

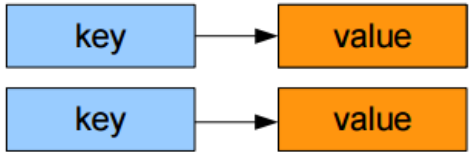
Other types of NoSQL's….

- Columnar
- KeyValue
- Document
- GraphDB

KeyValue Stores

Key / value stores (opaque)

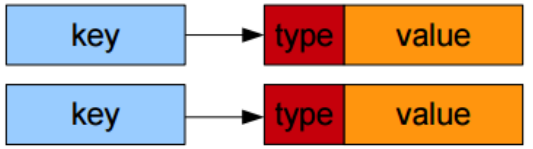
- Keys are mapped to values
- Values are treated as BLOBs (opaque data)
- No type information is stored
- Values can be heterogenous



Example values:
- { name: „foo“, age: 25, city: „bar“ } => JSON, but store will not care about it
- \xde\xad\xb0\x0b => binary, but store will not care about it

Key / value stores (typed)

- Keys are mapped to values
- Values have simple type information attached
- Type information is stored per value
- Values can still be heterogenous



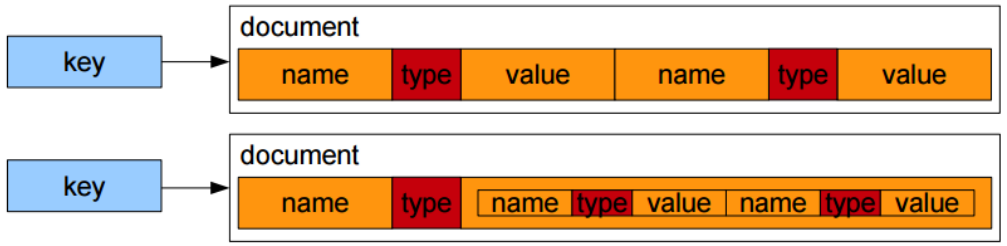
Example values:
- number: 25 => numeric, store can do something with it
- list: [1, 2, 3] => list, store can do something with it

Examples: Redis, Aerospike, Voldemort, Dynamite, RocksDB….

Document Stores

Document stores (non-shaped)

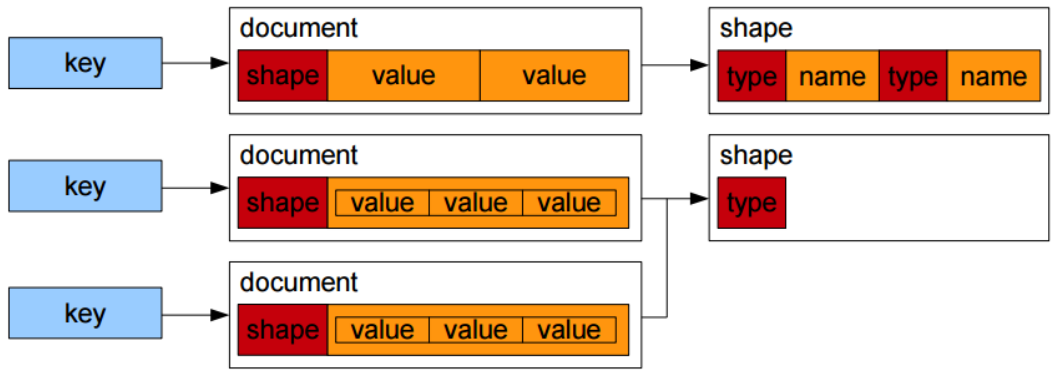
- Keys are mapped to documents
- Documents consist of attributes
- Attributes are name/typed value pairs, which may be nested
- Type information is stored per attribute
- Documents can be heterogenous
- Documents may be organised in collections or databases



Example documents:
- { name: „foo“, age: 25, city: „bar“ } => attributes and sub-attributes
- { name: { first: „foo“, last: „bar“ }, age: 25 } are typed and can be indexed

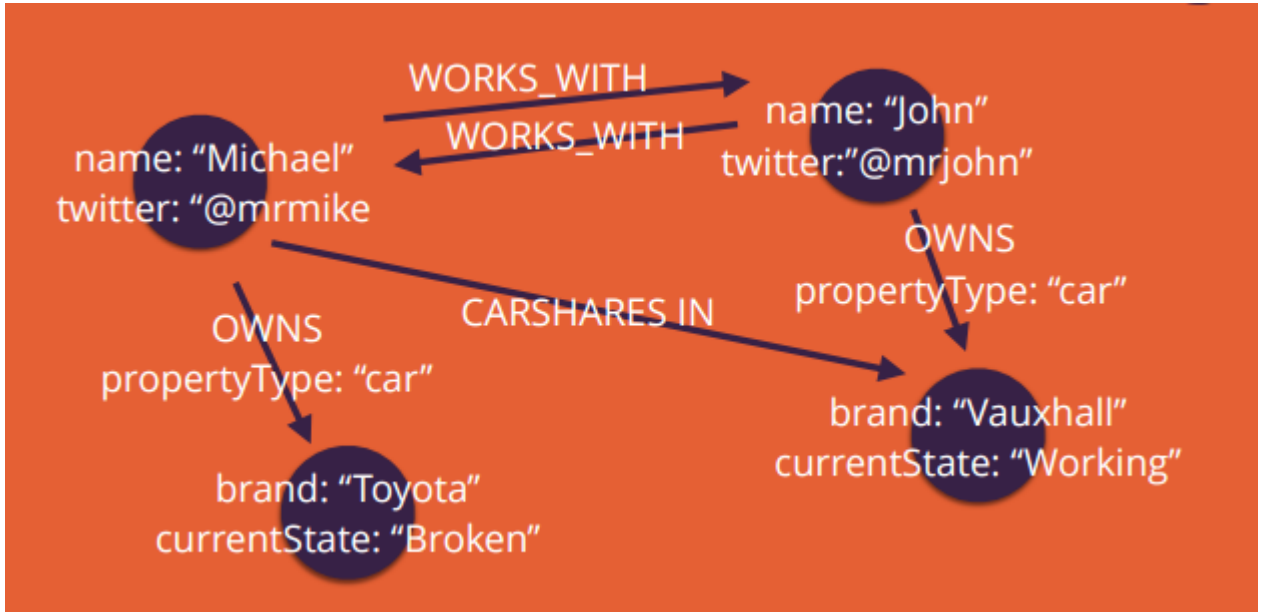
Document stores (shaped)

- Same as document stores, but...
- ...document type information is stored in shapes
- ...documents with similar structure (attribute names and types) point to the same shape



Examples: MongoDB, ArangoDB, RethinkDB, most of the JSON based Search Engines

GraphDB Stores

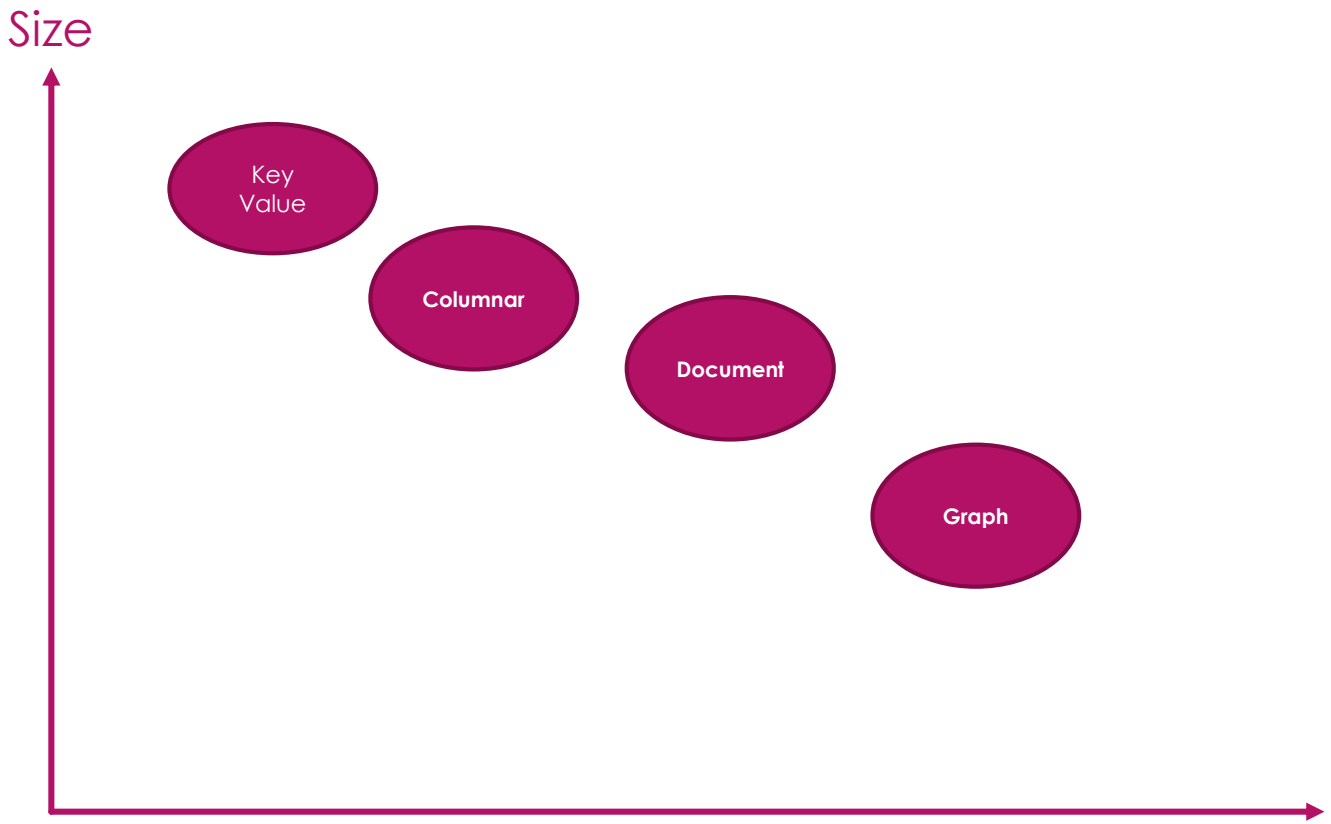


Examples: Neo4J, TitanDB

Reference

- http://nosql-database.org/

Comparision



OLTP vs OLAP

OLTP

- "transactional“ processing
- retrieve or modify individual records (mostly few records)
- use indexes to quickly find relevant records
- queries often triggered by end user actions and should complete instantly
- ACID properties may be important
- mixed read/write workload
- working set should fit in RAM

OLAP

- analytical processing / reporting
- derive new information from existing data (aggregates, transformations, calculations)
- queries often run on many records or complete data set
- data set may exceed size of RAM easily
- mainly read or even read-only workload
- ACID properties often not important, data can often be regenerated
- queries often run interactively
- common: not known in advance which aspects are interesting
- so pre-indexing "relevant" columns is difficult

SQL vs N.SQL

SQL

The Good

- High performance for transactions.
- Adheres to ACID
- Highly structured, very portable
- Small amounts of data
- SMALL IS LESS THAN 500GB
- Supports many tables with different types of data
- Can fetch ordered data
- Compatible with lots of tools

The Bad

- Complex queries take a long time
- The relational model takes a long time to learn
- Not really scalable
- Not suited for rapid development

N.SQL

The Good

- Fits well for volatile data
- Implement BASE properties
- High read and write throughput
- Scales really well
- Rapid development is possible
- In general it's faster than SQL

The Bad

Key/Value pairs need to be packed/unpacked all the time

Still working on getting security for these working as well as SQL

Lack of relations from one key to another

Coming back to the Overview

- **HBase is a distributed column-oriented data store built on top of HDFS**
  - This should be clear now

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*** is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)

- HBase updates are done by creating new versions of values

HBase vs. HDFS (Cont'd)

	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

If application has neither random reads or writes → Stick to HDFS

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

Kerberos

HBase: Keys and Column Families

Each record is divided into ***Column Families***

Each row has a ***Key***

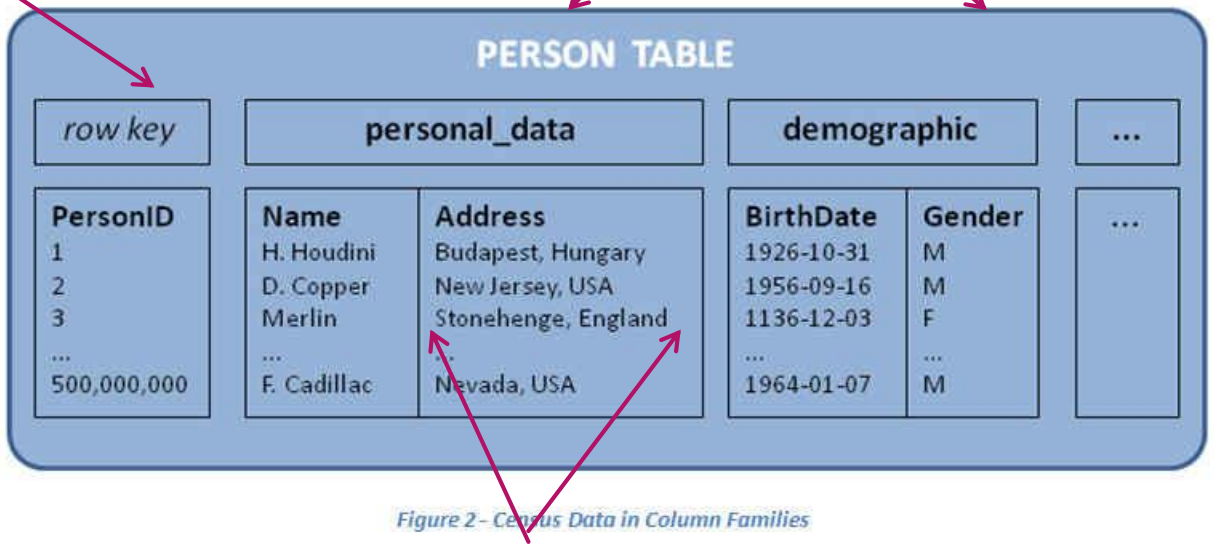


Figure 2 - Census Data in Column Families

Each column family consists of one or more ***Columns***

HBase: Keys and Column Families

- **Key**
  - Byte array
  - Serves as the primary key for the table
  - Indexed far fast lookup
- **Column Family**
  - Has a name (string)
  - Contains one or more related columns
- **Column**
  - Belongs to one column family
  - Included inside the row
    - ***familyName:columnName***

Column family named "Contents"

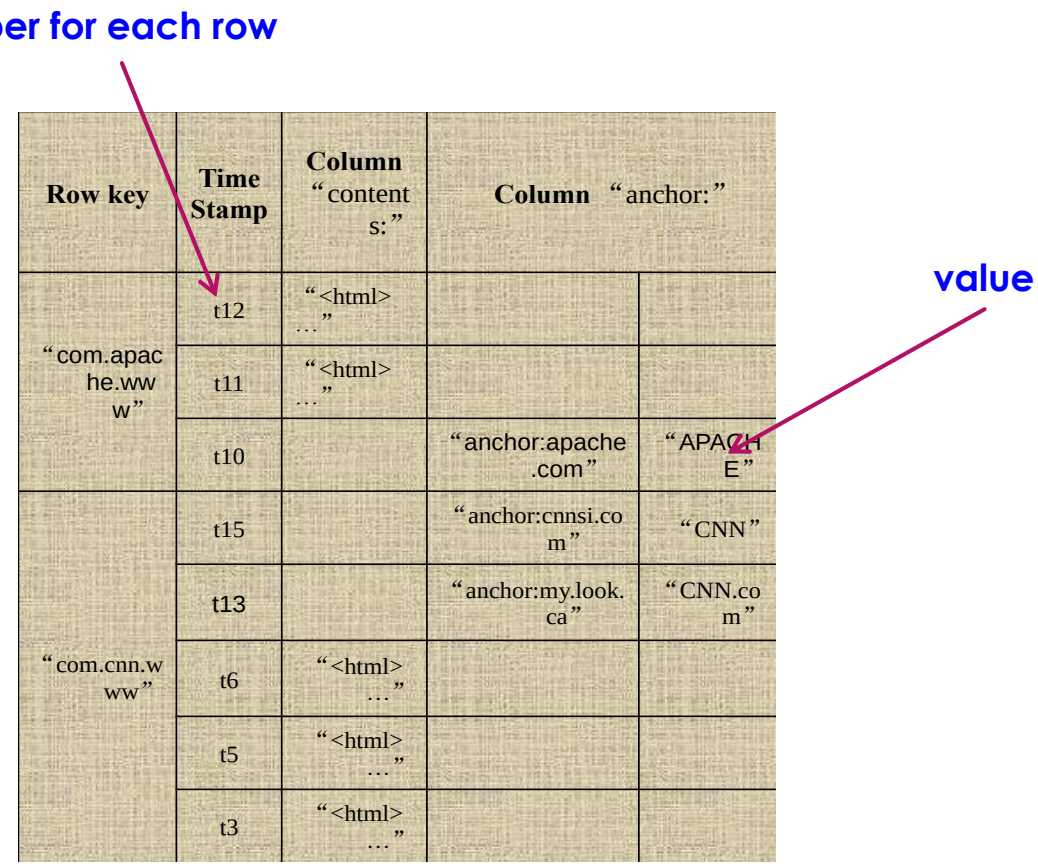
Column family named "anchor"

Row key	Time Stamp	Column "contents:"	Column "anchor:"	
"com.apache.www"	t12	"<html>…"		
	t11	"<html>…"		
	t10		"anchor:apache.com"	"APACHE"
"com.cnn.www"	t15		"anchor:cnnsi.com"	"CNN"
	t13		"anchor:my.look.ca"	"CNN.com"
	t6	"<html>…"		
	t5	"<html>…"		
	t3	"<html>…"		

Column named "apache.com"

HBase: Versions and Values

- **Version Number**
  - Unique within each key
  - By default→ System's timestamp
  - Data type is Long
- **Value (Cell)**
  - Byte array



Notes on Data Model

- HBase schema consists of several ***Tables***
- Each table consists of a set of ***Column Families***
  - Columns are not part of the schema
- HBase has ***Dynamic Columns --- Sparse***
  - Because column names are encoded inside the cells
  - Different cells can have different columns

"Roles" column family has different columns in different cells

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Notes on Data Model (Cont'd)

- The ***version number*** can be user-supplied
  - Even does not have to be inserted in increasing order
  - Version number are unique within each key
- Table can be very sparse
  - Many cells are empty
- ***Keys*** are indexed as the primary key

Has two columns [cnnsi.com & my.look.ca]

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

HBase Physical Model

- Each column family is stored in a separate file (called ***HTables***)
- Key & Version numbers are replicated with each column family
- Empty cells are not stored

HBase maintains a multi-level index on values: *<key, column family, column name, timestamp>*

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Row Key	Time Stamp	Column Family anchor
"com.cnn.www"	t9	anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

Example

`info` Column Family

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

`roles` Column Family

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

HBase Regions

- Each HTable (column family) is partitioned horizontally into ***regions***
  - Regions are counterpart to HDFS blocks

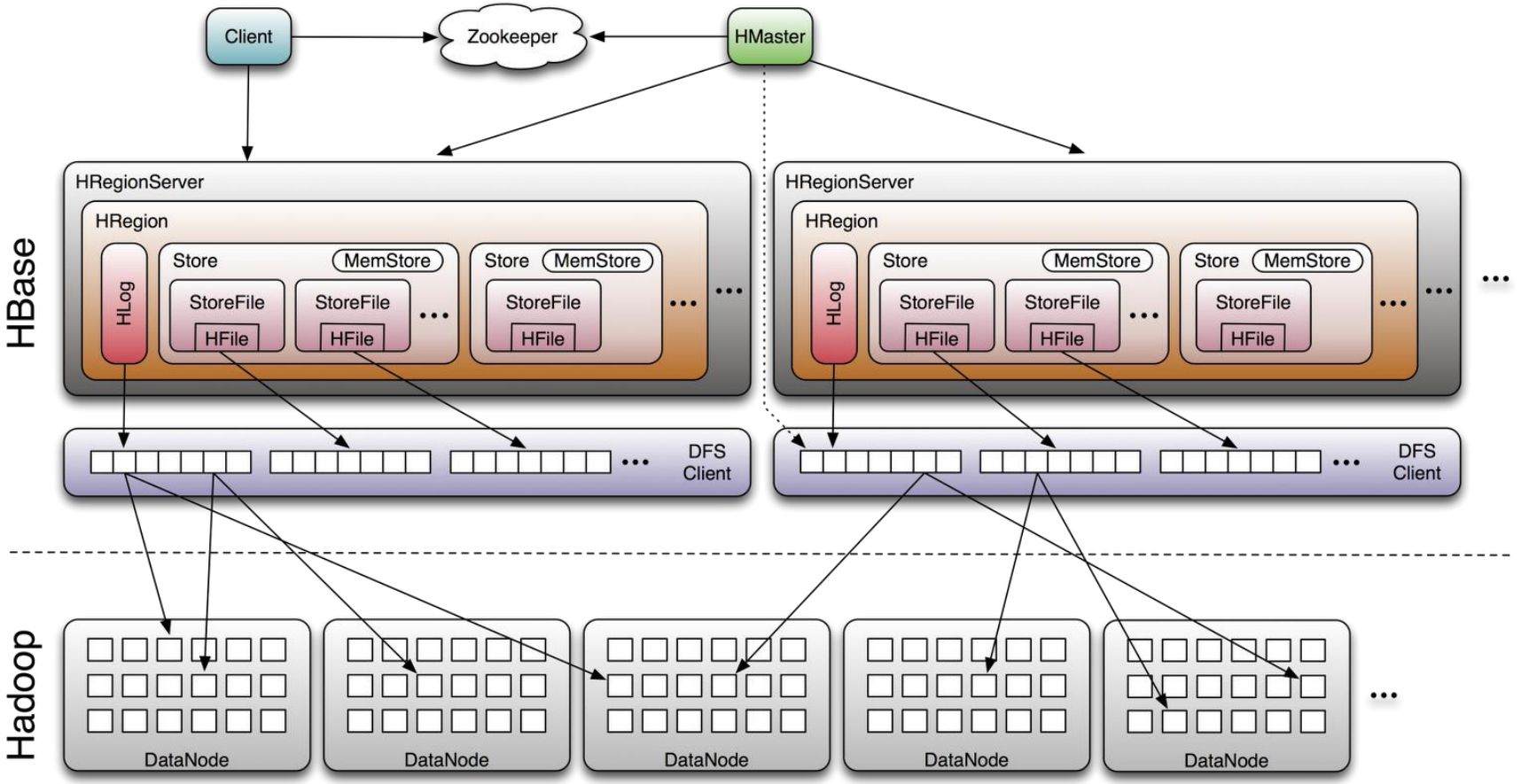
Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Each will be one region

HBase Components

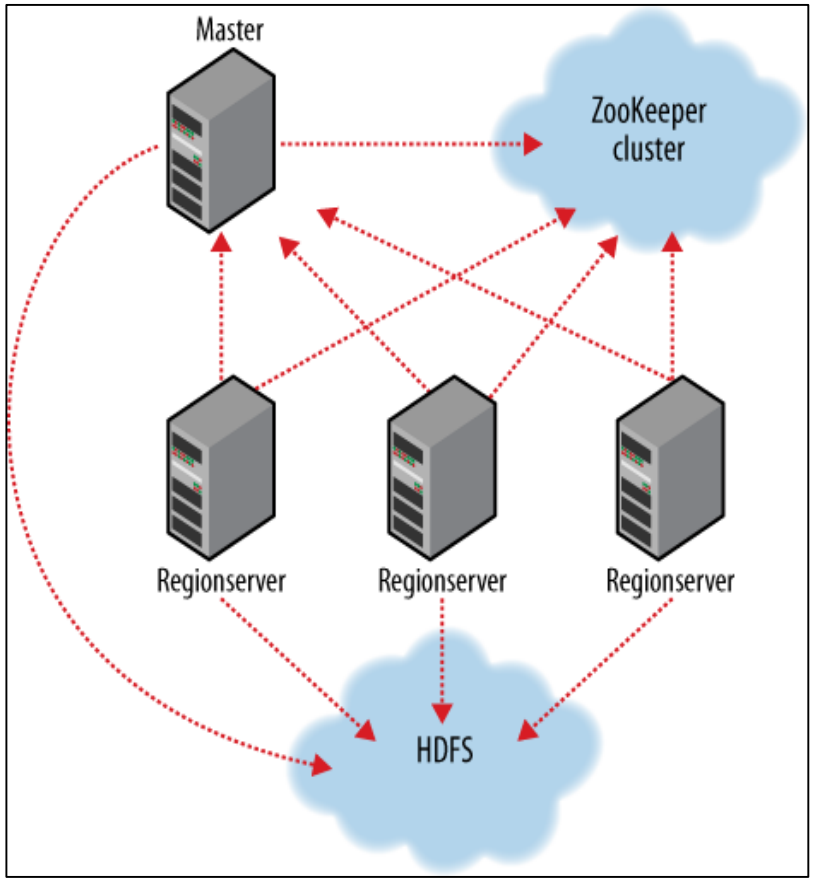
- **Region**
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- **RegionServer (many slaves)**
  - Manages data regions
  - Serves data for reads and writes (***using a log***)
- **Master**
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Architecture



ZooKeeper

- HBase depends on ZooKeeper
- By default HBase manages the ZooKeeper instance
  - E.g., starts and stops ZooKeeper
- HMaster and HRegionServers register themselves with ZooKeeper



Installation

- Download the binary distribution from Apache Website
- Untar, Set Path etc…
- If you want to run in localmode set hbase.cluster.distributed to false in hbase-site.xml
- Else set the configuration shown in the right to run hbase.
- Start using start-hbase.sh
- Run the client using hbase shell

If you have CDH VM, Hbase would start automatically

```
<configuration>
	<property>
	<name>hbase.rootdir</name> <!- point to your hdfs-->
	<value>hdfs://localhost:8020/hbase</value>
	</property>

	<property>
	<name>hbase.cluster.distributed</name>
	<value>true</value>
	</property>

	<property>
	<name>hbase.zookeeper.quorum</name>
	<value>localhost</value>
	</property>

	<property>
	<name>dfs.replication</name>
	<value>1</value>
	</property>

	<property>
	<name>hbase.zookeeper.property.clientPort</name>
	<value>2181</value>
	</property>

	<property>
	<name>hbase.zookeeper.property.dataDir</name>
	<value>/home/hdtester/hbase/zookeeper</value>
	</property>
</configuration>
```

An Example – emp table

```
table help ----> To get commonly used commands on a table
```

- Create a Table

```
create 'emp','identifier','compensation','address'
```

- Now insert one row (with Key 1111)

```
put 'emp','1111','identifier:empno','e1'
put 'emp','1111','identifier:ename','JAMES'
put 'emp','1111','compensation:salary','3000'
put 'emp','1111','compensation:pf','30'
put 'emp','1111','address:city','HYD'
```

- Select Data

```
scan 'emp'
```

- Update a row

```
put 'emp','1111','identifier:ename',‘JONES'
```

- Drop a table

```
disable 'emp'
drop 'emp'
```

- Insert some more rows

```
put 'emp','1112','identifier:empno','e2'
put 'emp','1112','identifier:ename','SMITH'
put 'emp','1112','compensation:salary','5000'
put 'emp','1112','compensation:pf','30'
put 'emp','1112','address:city','BLR'
put 'emp','1113','identifier:empno','e2'
put 'emp','1113','identifier:ename','SMITH'
put 'emp','1113','compensation:salary','5000'
put 'emp','1113','compensation:pf','30'
put 'emp','1113','address:city','HYD'
put 'emp','1113','address:street','KOTI'
put 'emp','1113','address:doorno','2333-333'
put 'emp','11110','identifier:empno','e3'
put 'emp','11110','identifier:ename','CLARK'
put 'emp','11110','compensation:salary','5000'
put 'emp','11110','compensation:pf','30'
```

- Delete a cell

```
delete 'emp','1111','identifier:ename',‘JONES'
```

- Select a row

```
get 'emp','1111'
```

- Delete a row

```
deleteall 'emp','1111'
```

JAVA API

- Create Table - https://github.com/iXat-Training/Hadoop101/blob/master/26_Hbase/src/main/java/com/ixat/hbase/HBaseTests/CreateTable.java
- Java Table Select- https://github.com/iXat-Training/Hadoop101/blob/master/26_Hbase/src/main/java/com/ixat/hbase/HBaseTests/SelectTest.java
- MR Job - https://github.com/iXat-Training/Hadoop101/blob/master/26_Hbase/src/main/java/com/ixat/hbase/HBaseTests/AnalyzeDataMR.java