No-SQL(Not Only SQL)

It is a database software which supports performing CRUD operations using SQL queries as well as using some other way.

There are some No SQL databases which also accept SQL syntax, which might not be very same what we write, but it might be database dependent.

Example of No-SQL Database

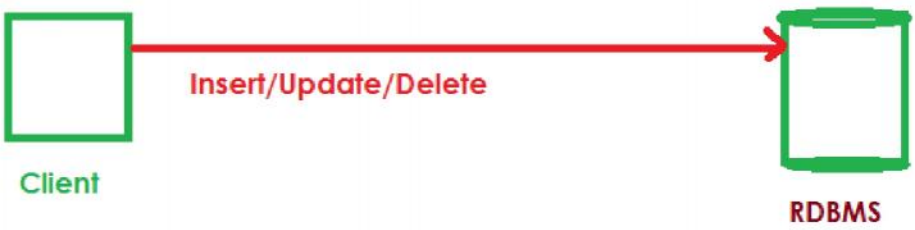
1. Cassandera
2. HBase
3. MongoDB
4. CouchBase
5. Neo4J
6. RIAK

Characteristics of No-SQL Database:

1. **Schema less:** With No-SQL, You can have a table where one row has 2 columns where as 2nd row has 20 columns,3rd row has 1 million columns.
2. **Distributed**: Most of the No-SQL database are distributed based on Horizontal scaling. Data are replicated and stored so that in case a node fails, still the data will not be lost. HBase works on top of Hadoop so no worries about the data failure.

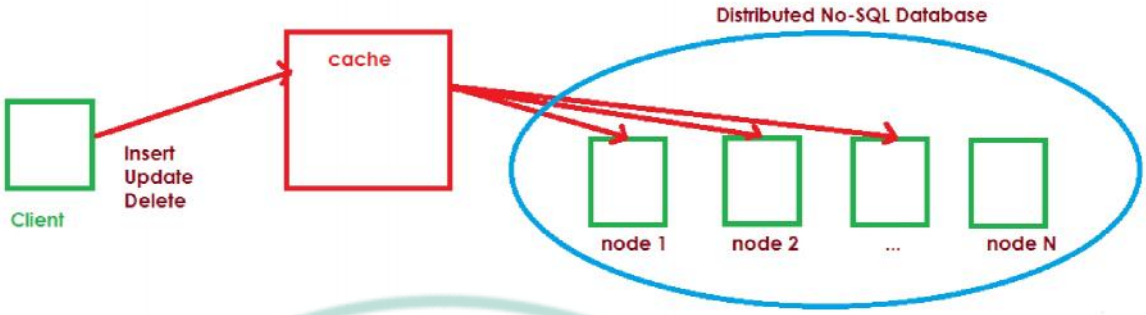
3. **Non-Relational:** Most of the NO-SQL database are Not Relational.
   a. There would not be any Primary key foreign key constraints among two tables.
   b. They don't support join.
   c. They cannot be normalized(data broken down to more tables)
   d. You will have to duplicate the data and store it on every tables.

4. **Eventual consistency:**
   In RDBMS if we insert the data in the data base it immediately gets persisted in the database



But In case of Distributed No-SQL you will see a cache where the data modification will be done initially, and when we freeze the changes, then it will update the final changes to all

the Nodes where the data is present. In case any node gets crashed then also the data will be still available in cache and other live nodes.



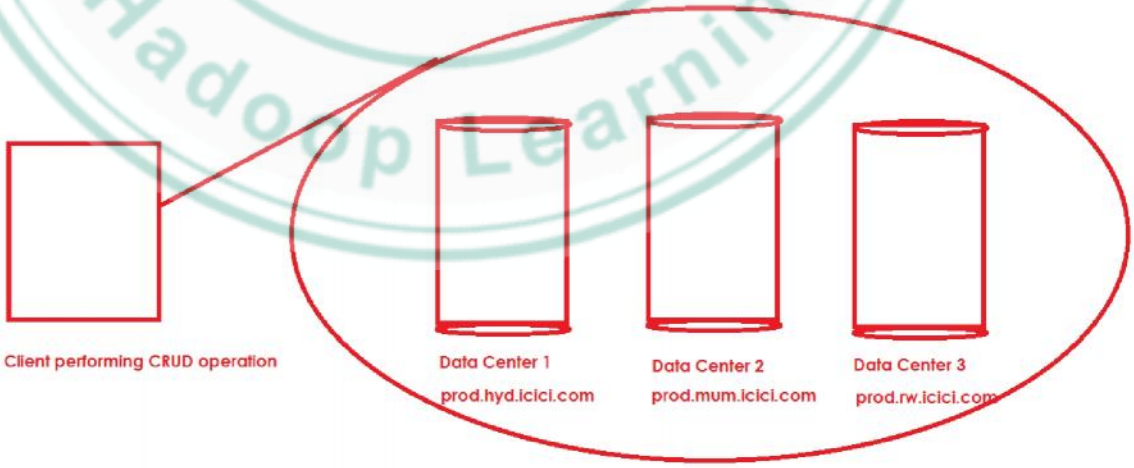
5. **ACID Property:** In all RDBMS,ACID Property is guaranteed. We perform every non-select operation in transaction. Few No-SQL databases(20%) supports ACID whereas most does not(80%).

   A-Atomicity
   C-consistency
   I-Isolation
   D-Durability

   More info here http://www.tutorialspoint.com/dbms/dbms_transaction.htm
   In short it is transaction.

   Assume that we have 3 operations,
   operation 1: Insert
   operation 2: Update
   operation 3: Delete
   We want this to be done using do All or Nothing principal. This is called as Transaction. If anyone operation is failed, then entire operation should be rolled back.
   This is not supported in most of No-SQL database.

6. Geographically Distributed: The Database are geographically distributed yet they work all together.



7. Opensource: Most of the no-sql database are opensource.

Categories of NO-SQL

They are categorized into 4

1. Key-Value: Redis , Riak , Mem-Cached
2. Document Oriented: Mongo DB, Couch Base
3. Column family: HBase , Cassandra , BigTable(google)
4. Graph Database: Neo4J

Aggregate Orientation:

In case of RDBMS, If you have 2 table Employee and dept like below

Employee

empId	empName	empDesg	deptId

dept

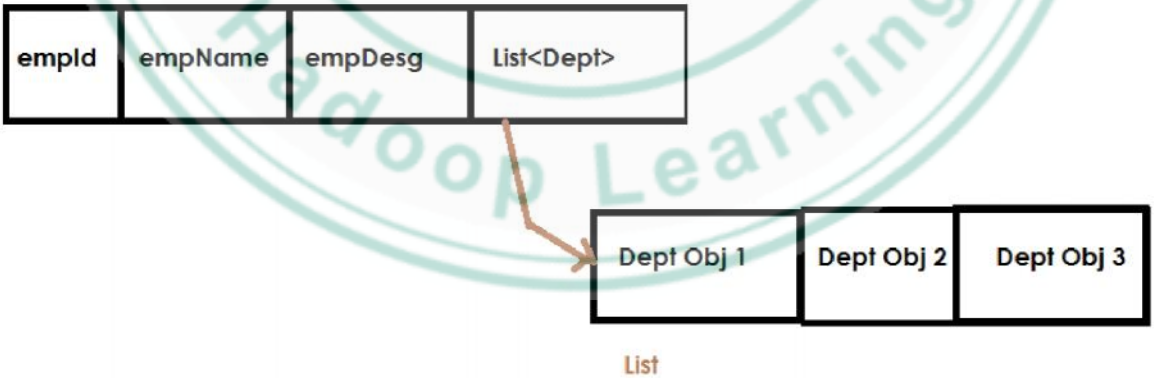
deptId	deptName	deptLoc

When we want to find the employee with their dept details, then You would have to join the result from two table.

SQL>SELECT e.empId, e.empName, d.deptName from employee e, dept d where e.deptId=d.deptId;

This is called Aggregation, as we have combined the data from 2 or more tables.

In case of No-SQL, We would be storing the data of these two tables together. There is no normalization, but the operations are faster as compare to joins. This is called Aggregation Oriented Database.



All the below No-SQL types supports Aggregation Orientation.

1. Key-Value
2. Document Oriented
3. Column family

So while designing the table and database, please design in such a way that similar related data are always put together rather them putting them separately.

Benefit of Aggregate Orientation

No-SQL is Distributed with replication factor, so the whole Aggregated data will be replicated as a single unit.
Easier for Programmer, as data are aggregated already and no need to write the join queries
Performance: Joins are costlier and resource hungry
Transactions: When You have whole related data in a single unit, we really don't need any transaction. If you want to update the employee address based on empId, you don't have to deal with 2 tables, its just about updating a row.

No-SQL Database Implementation

1.Key-Value Database: Looks like HashMap.
Key could be any unique data like row Id, empId.
Value would be Blob,to hold anything in Binary format.

Are some examples
hamsterdb
barkleydb
Redis
Riak

Data Storing Mechanism

Table is called as Bucket
Each rows are called document

Use Key-value oriented Database when you have to deal with data like user Preferences,credit card information ...

do-not use key-value oriented database when you have to deal with transactions,or data with relations with other tables,or you may have requirement to query the data based on the value
ie select * from employee where salary>5000;
Because the data are stored in key,value format,and salary would be present in the value but mixed with other data,so its difficult to perform a search effectively.For this reason people use indexing concept like solr.

Document database

key are unique id
value is whole document(JSON,XML,BSON document) which can be easily queried.The document can have map,collection etc
docuemnt can be nested

userid:123 --→ Key.The below are the complete value.
{
firstname:suraj,
likes:[guitar,cricket,hadoop],
last-city:[bangalore],
favourite-movie:[]
}

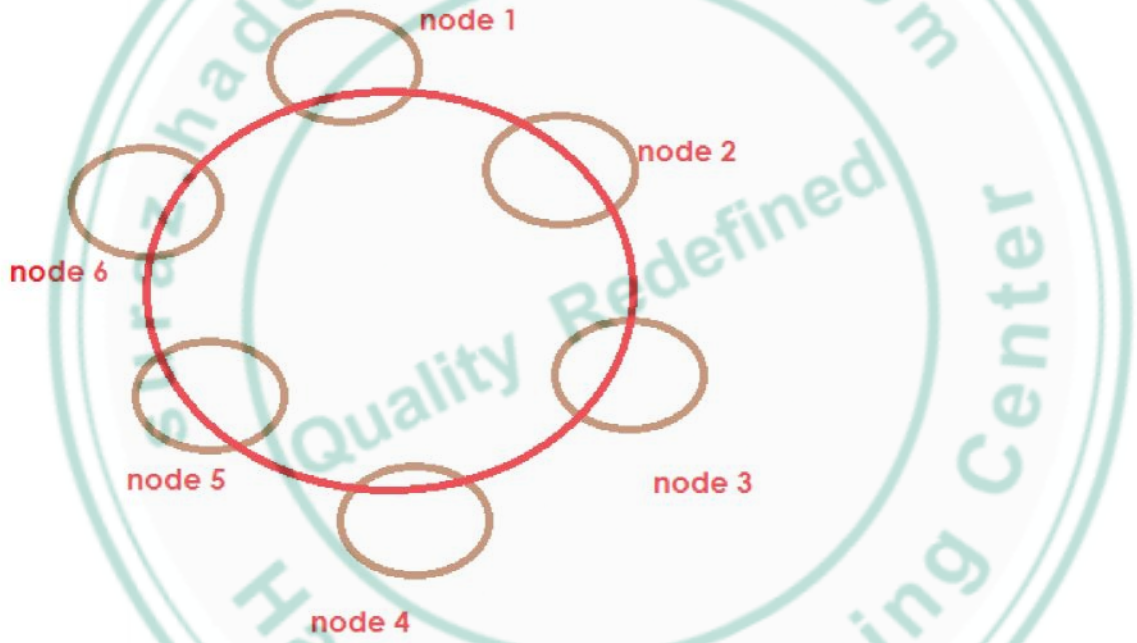
This kind of DB are suitable when you want to store the event based information.
Eg:
From where my webpage was visited.
What browser was used,
what is the client language.

They are also suitable for Content Management system.
Userid wise blogs,comments etc.

Donot use Document based dabaase when you are going to deal with complex transaction.
dont use document based database when your queries are dynamic ie keeps on changeing like search by state or capital in some case etc.else it will give you bad performance.

Column family Database

Cassandra: They dont have master node concept.If any node fails still the database work.They follow ring topology



HBase uses master node concept

Where you can use these database:-

1.Event logging
2.CMS
3.Expiring usage: In casandra you can configure to auto delete the record after certain interval(TTL-time to leave) say 60 days.it can be good when you want to expir the ads after 60 days,job posting after 60 days and so on..

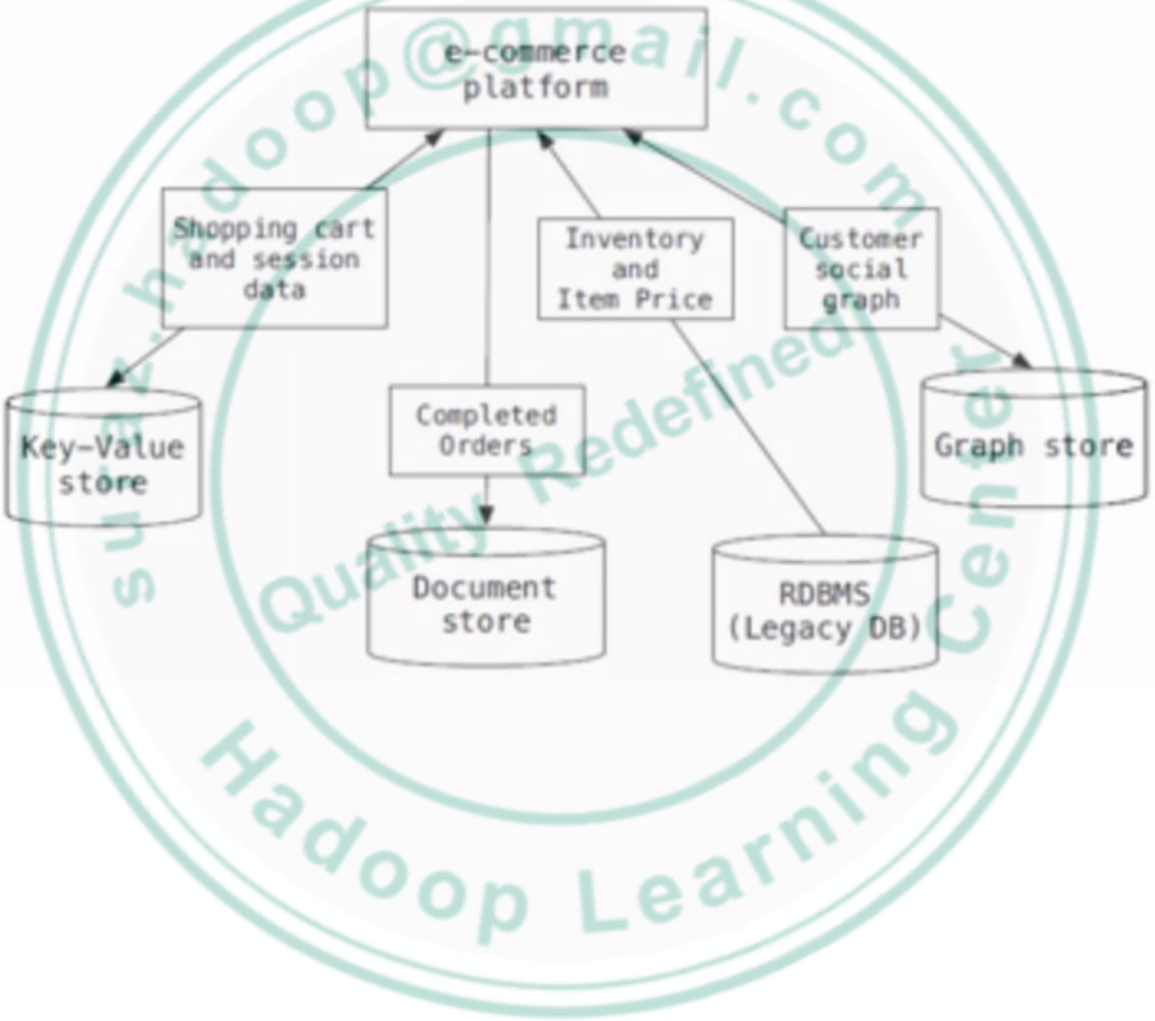
Where not to use:

1. ACID
2. Aggregate function(average,sum)
3. Dynamic query

4. You can use if the schema may change

Terminology:-
keyspace<==>schema(Like we have users or database name)
Keyspace contains cf1,cf2,....
cf are like tables
column has 3 values: column family:column Name: version

Polygot persistence:
Using more than one database to fulfill your complete requirement



Big Data Usage:-

Facebook adding more than 2 PB of data to Hadoop system Everyday
Tweeter 17TB of tweets every month.

Database Category:-

OLTP(Online Transaction Processing)-Bank fund transfer(Oracle,MySQL,DB2,Teradata)
OLAP(Online Analytical Processing)-Amazon, Ebay stores the data whenever customer purchases or clicks any links and later uses the same data to perform some analyses and come to some conclusion. They use some batch processing framework like Map-Reduce to generate some report on the processed dataset. It can be used by business to enhance the business.

Problem of RDBMS:

1. RDBMS shows the sign of weakness as the data starts growing. Say VISA card has daily 1 billion transactions, and when you store data of last 10 years, DB starts becoming weak as far as processing and storage is concerned. The query will become slow because of huge data.
2. When data is huge query becomes very slow.
3. Data Management will be too costly.
4. You can use Cache to hold the read only data, but it has got some limitation like RAM size and so on.

Note: Facebook, Yahoo, Microsoft uses mysql, Oracle, Teradata but they are used to hold different data and not the daily social media data. They can handle TBs of data but need to have experience server for the same.

Problem with HDFS:-

- Cannot handle real time Transactions
- Can't perform update on the existing HDFS data
- Random Read and Writes are not allowed. If you want to read the data, you need to start from the beginning. In the worst case, if the data is present at the very last node, then to read the data you need to process GB's or TBs of data. In best case, if data present in the initial nodes then it will just take very less time to read and process the data.

Real Time Configuration:-

300 GB of data (40 Nodes-i7-8 Processors-32GB of RAM) 1-2 hour execution time.

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.

Google gave Big Table to solve the (GFS) problem.
GFS is a paper& Big Table is a paper. HDFS, HBase are the implementations

HBase Introduction:-

1. HBase is an open-source(Product of Apache)Distributed, Version oriented, column family oriented type of Hadoop Database
2. It is the outcome of project called Big Table from Google. At the time of designing Map-reduce algorithm, Google also came up with an idea of Big Table (A distributed Database) where data will be stored across the cluster. A typical RDBMS(say Oracle) has to be installed on a single system and we start storing the data into it.
3. HBase says "Let's use HDFS to store data, and HBase will maintain the data in columnar format". There will be not any problem with size of the data, since it is distributed. The more node you add, you will get more space to store the data.
4. It is a platform to retrieve the data with Random Read write Access, which is not possible in HDFS.
5. HBase is good for Structured(csv, tsv) semi-structured(xml, tweets, blogs, email, html document).HBase can also store Unstructured data but the file should not be too large(greater than 2 GB).If the file is greater than 2 GB use HDFS to store the data instead of HBase.
6. HBase does not care about the data types in the column. The same column can store int for a row, whereas float for another row.

Key Feature of HBase:

HBase is a key/value store. Specifically it is a

1. **Sparse**
2. **Consistent**
3. **Distributed**
4. **Multidimensional**
5. **Sorted map**
6. **Horizontally Scalable**

Sparse:

You need not maintain empty column if you don't have suitable data. In HBase one row can have 2 columns, where as other row can have 100 columns. Unlike NULL in most relational databases, no storage is needed for absent information, there will be just no cell for a column that does not have any value.

Consistent

HBase makes two guarantees:
All changes with the same *rowkey* (see Multidimensional above) are atomic.
A reader will always read the last written (and committed) values.

Distributed

One key feature of HBase is that the data can be spread over 100s or 1000s of machines and reach billions of cells. HBase manages the load balancing automatically.

Multidimensional

The key itself has structure. Each key consists of the following parts:
row-key, column family, column, and time-stamp.
So the mapping is actually:

```
(rowkey, column family, column, timestamp) -> value
```

rowkey and *value* are just bytes (*column family* needs to be printable), so you can store anything that you can serialize into a byte[] into a cell.

Sorted-Map

HBase stores the data inform on key->value pair, where typically key will be rowId, and values will be other details. The row Ids are sorted lexically. Means Row 1 and then all rows whose id starts with Row 1 ... ,then Row 2 and all the id starting with row 2 ...

Row 1 ....
Row 11....
Row 2....
Row 21....
....

Where to Use HBase:-

1. **Random Read + Write or both** over huge amount of data. If you have thousands of operation to perform in one second over TB or PBs of data, then use HBase
2. **Simple Access Pattern:** When you don't have joins to perform to get the result, and all the similar data are stored in the same table. HBase may have TB of data in a table, and joining two or more Table with TB's of data will definitely impact the performance. So Joins are not supported here. HBase table data can be easily integrated with Map-reduce and hence each row from HBase can be an input to Mapper.map() & You can perform Map-Reduce join.
3. When data is sparse: It means
   first row may have data A,B,C,D,E
   second row may have A,,C,,E
   3rd row may have A,B,,,E

row1	A	B	C	D	E
row2	A	C	E		
row3	A	B	E		

Where not to use HBase

When the operation is too much of appending the data and less reading. Too much appending the data will increase the file size and later it becomes difficult to read the data.

If you are going to use adhoc-queries.

Assume you have a large data of 300 TB contains employee data, and you would like to find out the employees whose salary is >3000 USD. Internally it has to search all the records in the tables.

In HBase , Row-ID is well index where as the column data are not indexed, so it becomes trivial to search the whole data.

In HBase , there is only one indexing support, we don't have concept of secondary indexing support.

when you have less data say 2TB(single machine can hold the data) use RDBMS instead of HBASE

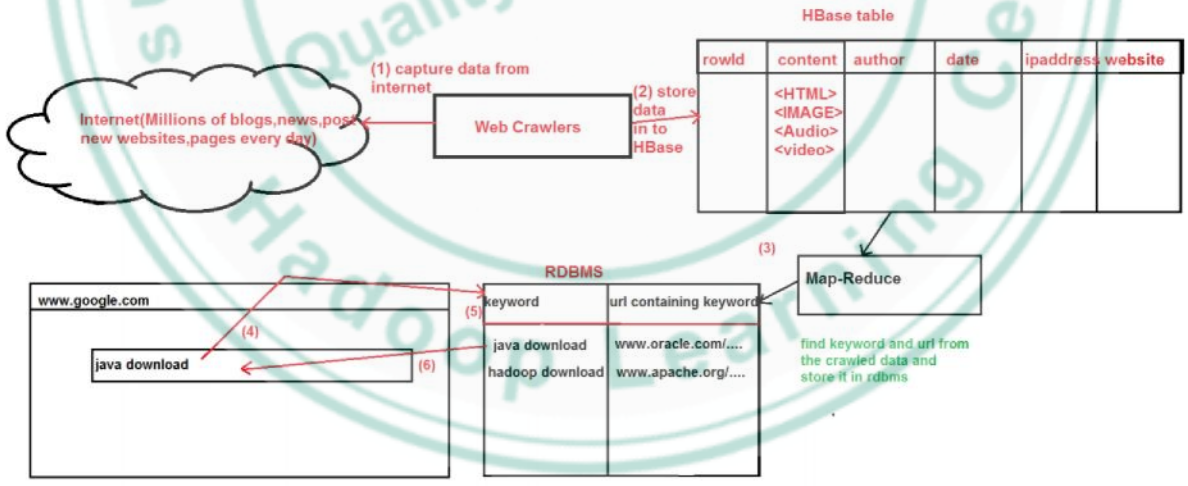
In short: HBase doesn't talk SQL, does not have an optimizer, does not support cross record transactions or joins

Search Engine using HBase

Search Engine are developed based on 2 things

1. Indexing
2. Searching

Everyday lots of blogs, post, websites are added to internet. There is a software called crawler keeping on searching the internet for any new documents and stores that into database(say HBase)



HBase is built on 2 things

HDFS
Zookeeper.

HBase can run on 3 modes

1. Local mode(single node)
2. pseudo distributed mode
3. Distributed mode

HBase can be accessed using

1. HBase shell
2. Java API

Row Oriented Vs Column Oriented

Row Oriented Database

RDBMS Purchase table: Here the data are stored sequentially based on row. If we talk from Memory diagram point of view, then 3 columns are stored adjacent to each other.
Oracle,MySql,DB2 all these databases are row-oriented databases.

customerName	product	Amount
Prasad	Tv	1000
Suman	Clock	200
Ravi	PC	1000

Row	Prasad	100
	TV	102
	1000	104
Row2	Suman	120
	Clock	122
	200	124
Row3	Ravi	130
	PC	132
	1000	134

Column Oriented Database:-

You will be able to put more similar data together, therefore you can compress similar data and it can save lot many spaces.

column-oriented database

CustomerName	Prasad	130
	Suman	132
	Ravi	134
Product	TV	130
	clock	132
	PC	134
Amount	1000	130
	200	132
	1000	134

HBase is not a column oriented database, but it follows nearly same pattern.

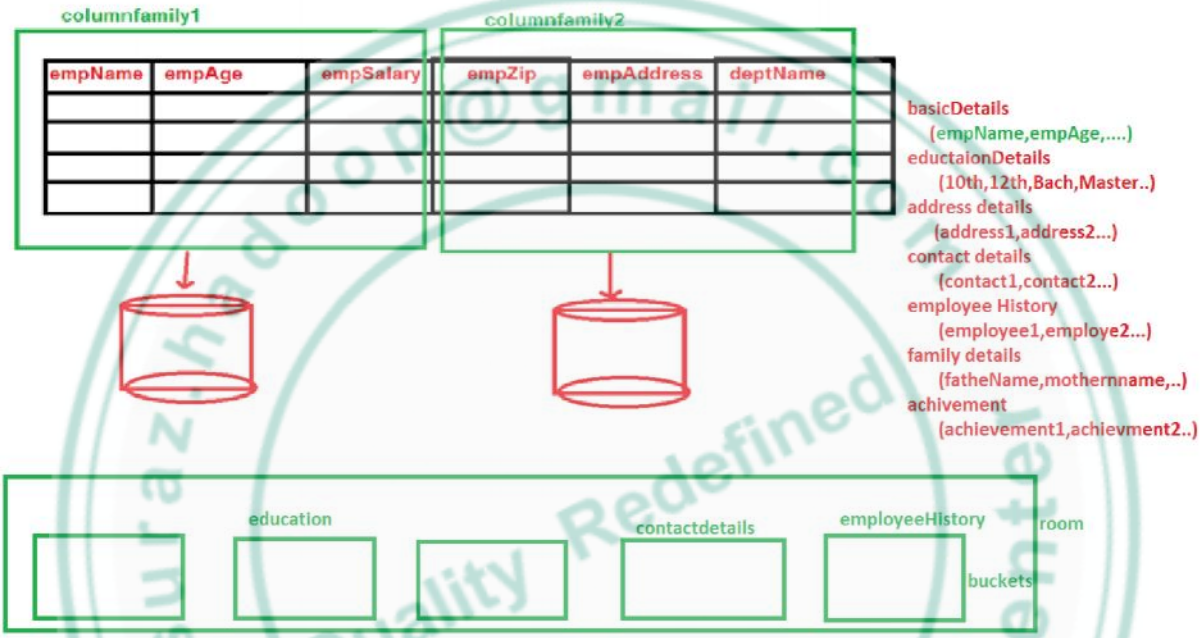
HBase Building Blocks

- The most basic unit is column.
- Combination of one or more column forms a column family.
- Combination of more than one column family forms a row
- Combination of one or more row forms a table

Table:

In Oracle database table name length should be max 30 character.
In HBase we don't have such limitation.

Row:-
Each row will be uniquely identified by rowId.
rowId will be generally a big value.
More than one column together can form a column family.
whenever you need the data together, please put them on same column family.



empName, empAge, empSalary will be stored together, whereas empZip, empAddress, deptName would be stored together.
While reading the data say empSalary, we don't need to read the file where empZip, empAddress, deptName resides.

Keep all the similar data together. You cannot modify column family after defining it. It takes a lot of effort.

Column qualifier:
Versioning:-

We will use cloudera 5. vm to demonstrate HBase.Open terminal and type the below to connect to hbase

```
$hbase shell
```

Note: In case you get any error while starting hbase or executing any command in HBase

Some Error list

ERROR: org.apache.hadoop.hbase.ipc.ServerNotRunningYetException: Server is not running yet
ERROR: Can't get master address from ZooKeeper; znode data == null
Go to cloudera manager and restart your hbase component.
Note: No need to use ; while executing hbase command. Instead it will give you error.

HBase Vs Hive

Hive stores data in HDFS and therefore read only. Latest hive version support read write as well (Hive-0.14 onwards)
HBase supports random read and write

Implementations of BigTable:-

1. Apache HBase
2. Apache Accumulo
3. Apache Cassandra
4. Level DB

Before you start

```
hbase>disable_all 't.*'
hbase> drop_all 't.*'
```

General HBase shell commands

Creating a table

The below command creates a table called testtable1 with a column family cf1.
A table must have at least one column family.

```
hbase(main):001:0> create 'testtable1','cf1'
0 row(s) in 1.5960 seconds
```

disable a table

Once you create a table, You may want to change the setting of the table or dropped the table.
In order to drop the table, you need to disable the table.
Use the below command to disable the table.

```
hbase(main):001:0> disable 'testtable1'
```

Note:

- You can change the setting of the table after creating it. But you need to disable the table first.
- You cannot read data from the table once you have disabled it. i.e. scan operation is not allowed. If you try to scan a disabled table, you will get "Error: table is disabled"
- You can still use **list** and **exists** command to check if the table is present or not(even if they are disabled).

Disable more than one table

In case you have many tables and you would like to disable more than one table based on some regular expression, Use the below command. The below command disables all the table starting with test.

```
hbase(main):004:0> disable_all 'test.*'
testtable1
Disable the above 1 tables (y/n)?
y
1 tables successfully disabled
```

Know if a table is disabled or not

In case you are not sure if the table is enabled or not, You can use the below command to check if the table is disabled. It returns true if disabled, false otherwise.

```
hbase(main):001:0> is_disabled 'testtable1'
false
```

Enabling a table

A disabled table can be enabled by using the below command. Once the table is enabled you can scan them.

```
hbase(main):001:0> enable 'testtable1'
```

Enabling all table

This will enable all the disabled table matching the given regular expression

```
hbase>enable_all 't.*'
```

Know if the table is enabled or not

This will show if the table is enabled or not. It returns true if enabled, false otherwise.

```
hbase>is_enabled  'testtable1'
```

Returns the below message if the table does not exists
ERROR: Unknown table testtable122!

Dropping a table

You cannot directly drop a table. You need to first disable the table. Then only you can drop it.
If you try to drop the table without disabling it, you will get the below error

```
Table must first be disabled:
```

General syntax:-

```
hbase> drop 't1'
hbase> drop 'ns1:t1'
```

Use the below command to disable and drop the table

```
hbase(main):001:0> disable 'testtable1'
hbase(main):001:0> drop 'testtable1'
0 row(s) in 1.5960 seconds
```

Dropping More than one table

Drop all of the tables matching the given regex. All the table should be disabled first.

```
hbase> drop_all 'tab.*'
```

Insert a record in the table

The below command inserts a record in testtable1.

1. row1 is the row-Id. It could be anything
2. cf1 is the column family where data needs to be inserted
3. val1 is the value that will get inserted

```
hbase(main):002:0> put 'testtable1','row1','cf1','val1'
0 row(s) in 0.0400 seconds
```

```
hbase(main):003:0> put 'testtable1','row2','cf1','val2'
0 row(s) in 0.0110 seconds
```

```
hbase(main):004:0> put 'testtable1','row2','cf1','val2'
0 row(s) in 0.0070 seconds
```

```
hbase(main):005:0> put 'testtable1','row2','cf1','val3'
0 row(s) in 0.0040 seconds
```

This contains 2 rows,row1 and row2(3 version maintained) but returns the last version

Note: Here we have not defined any column name, so data will be stored without any column name.

Display data from table.

```
hbase(main):007:0> scan 'testtable1'
ROW                COLUMN+CELL
 row1              column=cf1:, timestamp=1414442886980, value=val1
 row2              column=cf1:, timestamp=1414442913589, value=val3
2 row(s) in 0.0460 seconds
```

Note: row2 contains 3 values but scan will give the latest value only.

status

Show cluster status. Can be 'summary', 'simple', or 'detailed'. default status is summary

```
hbase>status
1 servers, 0 dead, 3.0000 average load
```

```
hbase> status 'summary'
1 servers, 0 dead, 3.0000 average load
```

```
hbase> status 'simple'
1 live servers
   quickstart.cloudera:60020 1443344364495
      requestsPerSecond=0.0, numberOfOnlineRegions=3, usedHeapMB=26, maxHeapMB=951,
numberOfStores=3, numberOfStorefiles=3, storefileUncompressedSizeMB=0, storefileSizeMB=0,
memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=59, writeRequestsCount=9,
rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=11,
currentCompactedKVs=11, compactionProgressPct=1.0, coprocessors=[]
0 dead servers
Aggregate load: 0, regions: 3
```

```
hbase(main):005:0> status 'detailed'
version 1.0.0-cdh5.4.2
0 regionsInTransition
master coprocessors: []
1 live servers
   quickstart.cloudera:60020 1443344364495
      requestsPerSecond=0.0, numberOfOnlineRegions=3, usedHeapMB=17, maxHeapMB=951,
numberOfStores=3, numberOfStorefiles=3, storefileUncompressedSizeMB=0, storefileSizeMB=0,
memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=65, writeRequestsCount=9,
rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0, totalCompactingKVs=11,
currentCompactedKVs=11, compactionProgressPct=1.0, coprocessors=[]
      "hbase:meta,,1"
         numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0,
storefileSizeMB=0, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=59,
writeRequestsCount=4, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0,
totalCompactingKVs=11, currentCompactedKVs=11, compactionProgressPct=1.0,
completeSequenceId=20, dataLocality=1.0
      "hbase:namespace,,1443056892077.4161f7115fc4d75cb25f5d1945569174."
         numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0,
storefileSizeMB=0, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=4,
writeRequestsCount=0, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0,
totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN,
completeSequenceId=7, dataLocality=1.0
```

```
    "testtable1,,1443348677379.eb8e49f6c815c8eeb2b026388a16ac5a."
      numberOfStores=1, numberOfStorefiles=1, storefileUncompressedSizeMB=0,
storefileSizeMB=0, memstoreSizeMB=0, storefileIndexSizeMB=0, readRequestsCount=2,
writeRequestsCount=5, rootIndexSizeKB=0, totalStaticIndexSizeKB=0, totalStaticBloomSizeKB=0,
totalCompactingKVs=0, currentCompactedKVs=0, compactionProgressPct=NaN,
completeSequenceId=12, dataLocality=1.0
0 dead servers
```

Version

This outputs the current HBase version

```
hbase(main):001:0> version
1.0.0-cdh5.4.2, rUnknown, Tue May 19 17:07:29 PDT 2015
```

whoami

This shows the current HBase user.

```
hbase> whoami
cloudera (auth:SIMPLE)
   groups: cloudera, default
```

list all tables

```
hbase> list
TABLE
testtable1
1 row(s) in 0.2230 seconds
```

Optional regular expression parameter could be used to filter the output

Display all tables starting with test

```
hbase> list 'test.*'
TABLE
testtable1
1 row(s) in 0.1590 seconds
```

Display all tables ending with table1

```
hbase> list   '.*table1'
TABLE
testtable1
1 row(s) in 0.1590 seconds
```

Display all tables that contains word tab

```
hbase> list   '.*tab.*'
TABLE
testtable1
1 row(s) in 0.1590 seconds
```

exists

check if the table exists. It will show you the table even if they are disabled.

```
hbase> exists 't1'
Table t1 does not exist
0 row(s) in 0.0840 seconds
```

```
hbase(main):005:0> exists 'testtable1'
Table testtable1 does exist
0 row(s) in 0.0140 seconds
```

Describe the table

check if the table exists

```
hbase> describe 'testtable1'
Table testtable1 is ENABLED
testtable1
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1',
COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE =>
'65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
1 row(s) in 0.0340 seconds
```

Creating a table in detail

Lets create table.

Create a table tab1 with only 1 column family f1,that can maintain up to 5 version.

```
hbase> create 'tab1', {NAME => 'f1', VERSIONS => 5}
```

Create table with Just 3 column family,and version =1 for f1 and f3, version=5 for f2

```
hbase> create 'tab2', {NAME => 'f1'}, {NAME => 'f2',VERSIONS=>5}, {NAME => 'f3'}
```

Use shorthand instead

```
hbase> create 'tab3', 'f1', 'f2', 'f3'
```

Create a table with a reference, using which you can call methods

```
hbase>t4 = create 'tab4', 'f1'
hbase(main):002:0> put 'tab4','row1','f1','val1'
0 row(s) in 0.0400 seconds

hbase(main):002:0>t4.put 'row2', 'f1', 'val2'

hbase(main):011:0> scan 'tab4'
ROW                    COLUMN+CELL
 row1                  column=f1:, timestamp=1443386162681, value=val1
 row2                  column=f1:, timestamp=1443386181947, value=val2
2 row(s) in 0.0310 seconds

hbase(main):011:0> scan 't4'    --This does not work

hbase(main):011:0> t4.scan    --This works
```

Note: If you log out and again login to hbase, the table persists but the variable t4 does not persist. Hence if you again try to perform t4.scan, You will get the below error.

```
NameError: undefined local variable or method `t4' for #<Object:0x60a23db3>
```

If you want to again hold the table into a reference variable, Please execute the following command

```
hbase(main):003:0> t4 = get_table 'tab4'
0 row(s) in 0.0170 seconds
=> Hbase::Table - tab4
```

Now you can again execute various methods on t4 reference.

```
hbase(main):004:0> t4.scan
```

CREATING COUMNS and writing data into columns of column families

```
hbase(main):002:0> put 'tab4','row3','f1:col1','val1'
hbase(main):002:0> put 'tab4','row3','f1:col2','val2'
hbase(main):002:0> put 'tab4','row4','f1:col3','val3'
hbase(main):002:0> put 'tab4','row4','f1:col4','val4'
```

Get

```
hbase(main):002:0> disable_all 't.*'
hbase(main):002:0> drop_all 't.*'
```

Lets Create a table to understand Get

```
hbase> create 'tab1', {NAME => 'f1',VERSIONS=>5}
hbase(main):002:0> put 'tab1','row1','f1:col1','val1'
hbase(main):002:0> put 'tab1','row2','f1:col1','val1'
hbase(main):002:0> put 'tab1','row2','f1:col1','val2'
hbase(main):002:0> put 'tab1','row2','f1:col1','val3'
```

```
hbase> scan 'tab1'
ROW                   COLUMN+CELL
 row1                 column=f1:col1, timestamp=1453840066866, value=val1
 row2                 column=f1:col1, timestamp=1453840099174, value=val3
2 row(s) in 0.0140 seconds
```

```
hbase> scan 'tab1',{VERSIONS => 10} --will not complain about 10,simply displays column till 10 possible versions
ROW                   COLUMN+CELL
 row1                 column=f1:col1, timestamp=1453840066866, value=val1
 row2                 column=f1:col1, timestamp=1453840099174, value=val3
 row2                 column=f1:col1, timestamp=1453840087676, value=val2
 row2                 column=f1:col1, timestamp=1453840077257, value=val1
2 row(s) in 0.0230 seconds
```

Note: In case you update a row with same column value, it will still maintain a new record with different timestamp.

```
row2                 column=f1:col1, timestamp=1454162863778, value=val3
row2                 column=f1:col1, timestamp=1454162694125, value=val3
```

```
hbase> get 'tab1','row2','f1' --gives latest version of all column within column family.
ROW                   COLUMN+CELL
 row2                 column=f1:col1, timestamp=1453840099174, value=val3
1 row(s) in 0.0230 seconds
```

```
hbase> get 'tab1','row2','f1:col1' --gives latest version of col1 column within column family.
ROW                   COLUMN+CELL
 row2                 column=f1:col1, timestamp=1453840099174, value=val3
1 row(s) in 0.0230 seconds
```

```
hbase> get 'tab1','row2',{ COLUMN => 'f1:col1', VERSIONS => 3} --gives all 3 versions
COLUMN                CELL
 f1:col1              timestamp=1453840099174, value=val3
 f1:col1              timestamp=1453840087676, value=val2
 f1:col1              timestamp=1453840077257, value=val1
3 row(s) in 0.0150 seconds
```

Help

Will show help related to all hbase command.

```
hbase(main):002:0> help
HBase Shell, version 1.0.0-cdh5.4.2, rUnknown, Tue May 19 17:07:29 PDT 2015
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for help on a specific command.
Commands are grouped. Type 'help "COMMAND_GROUP"', (e.g. 'help "general"') for help on a command group.
```

```
COMMAND GROUPS:
  Group name: general
  Commands: status, table_help, version, whoami

  Group name: ddl
  Commands: alter, alter_async, alter_status, create, describe, disable, disable_all, drop, drop_all,
enable, enable_all, exists, get_table, is_disabled, is_enabled, list, show_filters

  Group name: namespace
  Commands: alter_namespace, create_namespace, describe_namespace, drop_namespace,
list_namespace, list_namespace_tables

  Group name: dml
  Commands: append, count, delete, deleteall, get, get_counter, incr, put, scan, truncate,
truncate_preserve

  Group name: tools
  Commands: assign, balance_switch, balancer, catalogjanitor_enabled, catalogjanitor_run,
catalogjanitor_switch, close_region, compact, compact_mob, compact_rs, flush, major_compact,
major_compact_mob, merge_region, move, split, trace, unassign, wal_roll, zk_dump

  Group name: replication
  Commands: add_peer, append_peer_tableCFs, disable_peer, enable_peer, list_peers,
list_replicated_tables, remove_peer, remove_peer_tableCFs, set_peer_tableCFs,
show_peer_tableCFs

  Group name: snapshots
  Commands: clone_snapshot, delete_all_snapshot, delete_snapshot, list_snapshots,
restore_snapshot, snapshot

  Group name: configuration
  Commands: update_all_config, update_config

  Group name: quotas
  Commands: list_quotas, set_quota

  Group name: security
  Commands: grant, revoke, user_permission

  Group name: visibility labels
  Commands: add_labels, clear_auths, get_auths, list_labels, set_auths, set_visibility

SHELL USAGE:
Quote all names in HBase Shell such as table and column names.  Commas delimit
command parameters.  Type <RETURN> after entering a command to run it.
Dictionaries of configuration used in the creation and alteration of tables are
Ruby Hashes. They look like this:

  {'key1' => 'value1', 'key2' => 'value2', ...}
```

```
and are opened and closed with curley-braces.  Key/values are delimited by the
'=>' character combination.  Usually keys are predefined constants such as
NAME, VERSIONS, COMPRESSION, etc.  Constants do not need to be quoted.  Type
'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use
double-quote'd hexadecimal representation. For example:

  hbase> get 't1', "key\x03\x3f\xcd"
  hbase> get 't1', "key\003\023\011"
  hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added.
For more on the HBase Shell, see http://hbase.apache.org/book.html
```

Help on a particular group

Help are grouped into

- general
- ddl
- namespace
- dml
- tools
- replication
- snapshot
- configuration
- quotas
- security
- visibility labels

If you want to get help on 'general' group, you should type.

```
hbase(main):002:0> help 'general'
Command: status
Show cluster status. Can be 'summary', 'simple', 'detailed', or 'replication'. The
default is 'summary'. Examples:

hbase> status
  hbase> status 'simple'
  hbase> status 'summary'
  hbase> status 'detailed'
  hbase> status 'replication'
  hbase> status 'replication', 'source'
  hbase> status 'replication', 'sink'

Command: 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'
```

Or, if you have already created the table, you can get a reference to it:

```
hbase> t = get_table 't'
```

You can do things like call 'put' on the table:

```
hbase> t.put 'r', 'cf:q', 'v'
```

which puts a row 'r' with column family 'cf', qualifier 'q' and value 'v' into table t.

To read the data out, you can scan the table:

```
hbase> t.scan
```

which will read all the rows in table 't'.

Essentially, any command that takes a table name can also be done via table reference.
Other commands include things like: get, delete, deleteall,
get_all_columns, get_counter, count, incr. These functions, along with
the standard JRuby object methods are also available via tab completion.

For more information on how to use each of these commands, you can also just type:

```
hbase> t.help 'scan'
```

which will output more information on how to use that command.

You can also do general admin actions directly on a table; things like enable, disable,
flush and drop just by typing:

```
hbase> t.enable
hbase> t.flush
hbase> t.disable
hbase> t.drop
```

Note that after dropping a table, your reference to it becomes useless and further usage
is undefined (and not recommended).

Command: version
Output this HBase version

Command: whoami
Show the current hbase user.
Syntax : whoami
For example:

```
hbase> whoami
```

Edit a table to add more column family

We have a table called tab1,which contains just 1 column family that we added while creating the table.

Use desc or describe to get info about the table

```
hbase(main):002:0> desc 'tab1'
Table tab4 is ENABLED
tab4
COLUMN FAMILIES DESCRIPTION
{NAME => 'f1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_
SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL =
> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => '
false', BLOCKCACHE => 'true'}
1 row(s) in 0.0330 seconds
```

Lets add one more column family. You don't need to disable the table.

```
hbase(main):002:0> alter 'tab1','cf2'
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.3490 seconds
```

```
hbase(main):002:0> alter 'tab1',{NAME=>'cf3'}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.3490 seconds
```

```
hbase(main):002:0> alter 'tab1',{NAME=>'cf4',VERSIONS=>50}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.3490 seconds
hbase(main):002:0> describe 'tab1'
{NAME => 'cf2', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION
_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '1', TTL => 'FOREVER', MIN_VER
SIONS => '0', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY =>
'false', BLOCKCACHE => 'true'}
{NAME => 'cf3', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION
_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '1', TTL => 'FOREVER', MIN_VER
SIONS => '0', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY =>
'false', BLOCKCACHE => 'true'}
{NAME => 'cf4', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION
_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '50', TTL => 'FOREVER', MIN_VE
RSIONS => '0', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY =>
 'false', BLOCKCACHE => 'true'}
{NAME => 'f1', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_
```

```
SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL =
> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => '
false', BLOCKCACHE => 'true'}
4 row(s) in 0.0300 seconds
```

Performing multiple operations at the same time.

```
hbase> alter 'tab1', 'cf5', {NAME => 'cf6', IN_MEMORY => true}, {NAME => 'cf7', VERSIONS => 5}
```

The above command adds cf5,cf6,cf7 column family into the database.

```
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 7.1420 seconds
```

Deleting the column family from a table

Say currently the table has f1,f2,cf3, cf4,cf5,cf6 column family.

```
hbase> alter 'tab1', NAME => 'cf6', METHOD => 'delete'
hbase> alter 'tab1', 'delete' => 'cf5'
hbase> alter 'tab1', 'delete' => 'cf3', 'delete' => 'cf4'  --just deletes cf4,doesnot delete cf3
```

Change various scope of a table

There are lot many properties you can set in a hbase table
MAX_FILESIZE, READONLY, MEMSTORE_FLUSHSIZE, DEFERRED_LOG_FLUSH

--change MAX_SIZE of a region to 128 MB
hbase> alter 't1', MAX_FILESIZE => '134217728'

Namespace in HBase-Multitenancy

Multi-tenancy means one HBase instance can support multiple tenants(users or environments).
They were added in v0.96
It is like schema or User where you can have group of tables.
To create a namespace in hbase, use the below command

Create a namespace

```
hbase>create_namespace 'my_ns'
```

Create my_table in my_ns namespace

```
hbase>create 'my_ns:my_table', 'cf1'
```

Note: When you execute list command, it will display you all the tables of all the namespace.

Drop namespace

```
hbase>drop_namespace 'my_ns'
```

In Order to drop the namespace, all the table inside the namespace should be removed. then only you will be able to drop the namespace.

Alter namespace

```
hbase>alter_namespace 'my_ns', {METHOD => 'set', 'PROPERTY_NAME' =>
'PROPERTY_VALUE'}
```

Extra Read-Securing Namespace:- HBase-Admin

This task adds the security piece to the namespace feature. The work related to migration of the existing acl table to the new namespace is remaining and will be completed in the follow up patch. Permissions can be granted to a namespace by the hbase admin, by appending '@' to the namespace name. A user with write or admin permissions on a given namespace can create tables in that namespace. The other privileges (R, X, C) do not have any special meaning w.r.t namespaces. Any users of hbase can list tables in a namespace.

The following commands can only be executed by HBase admins.
1. Grant privileges for user on Namespace.
2. Revoke privileges for user on Namespace

Grant Command:
hbase> grant 'tenant-A' 'W' '@N1'
In the above example, the command will grant the user 'tenant-A' write privileges for a namespace named "N1".

Revoke Command:

hbase> revoke 'tenant-A"@N1'
In the above example, the command will revoke all privileges from user 'tenant-A' for namespace named "N1".

Lets see an example on how privileges work with namespaces.

User "Mike" request for a namespace named "hbase_perf" with the hbase admin.
whoami: hbase
hbase shell >> namespace_create 'hbase_perf'
hbase shell >> grant 'mike', 'W', '@hbase_perf'
Mike creates two tables "table20" and "table50" in the above workspace.
whoami: mike
hbase shell >> create 'hbase_perf.table20', 'family1'
hbase shell >> create 'hbase_perf.table50', 'family1'

Note: As Mike was able to create tables 'hbase_perf.table20', 'hbase_perf.table50', he becomes the owner of those tables.
This means he has "RWXCA" perms on those tables.
Another team member of Mike, Alice wants also to share the same workspace "hbase_perf".
HBase admin grants Alice also permission to create tables in "hbase_perf" namespace.
whoami: hbase
hbase shell >> grant 'alice', 'W', '@hbase_perf'
Now Alice can create new tables under "hbase_perf" namespace, but cannot read,write,alter,delete existing tables in the namespace.

whoami: alice
hbase shell >> namespace_list_tables 'hbase_perf'
hbase_perf.table20
hbase_perf.table50
hbase shell >> scan 'hbase_perf.table20'
AccessDeniedException

If Alice wants to read or write to existing tables in the "hbase_perf" namespace, hbase admins need to explicitly grant permission.

whoami: hbase
hbase shell >> grant 'alice', 'RW', 'hbase_perf.table20'
hbase shell >> grant 'alice', 'RW', 'hbase_perf.table50'

Where does HBase stores data in HDFS

Under `${base.rootdir}/data` we have subdirectories related to namespaces and table names.

For example Let's say we have this configuration in `hbase-site.xml`

```
<property>
      <name>hbase.rootdir</name>
      <value>hdfs://hadoop:8020/apps/hbase</value>
</property>
```

And we have namespace MyFirstNamespace with tables MyTable1, MyTable2 so the directory structure on hdfs will be:

```
/apps/hbase/data/MyFirstNamespace/MyTable1
/apps/hbase/data/MyFirstNamespace/MyTable2
```

NOTE : All the filter should be written in CAPS.otherwise it will throw error undefined local variable filter

those filter which accepts the argument then argument should be passed in ' ' single quotes

Filters

- It was availabe from hbase 0.92 version onwards.
- It allows you to perform server-side filtering when accessing HBase **Tables.**

```
hbase(main):010:0> show_filters

ColumnPrefixFilter
TimestampsFilter
PageFilter
MultipleColumnPrefixFilter
FamilyFilter
ColumnPaginationFilter
SingleColumnValueFilter
RowFilter
QualifierFilter
ColumnRangeFilter
ValueFilter
PrefixFilter
SingleColumnValueExcludeFilter
ColumnCountGetFilter
InclusiveStopFilter
DependentColumnFilter
FirstKeyOnlyFilter
KeyOnlyFilter
```

When you perform scan,it gives you all the columns.

SQL>select * from employee;

```
hbase(main):006:0> scan 'employee'
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row1               column=address:pincode, timestamp=1454166591130, value=500038
 row1               column=address:street, timestamp=1454166591083, value=20/A
 row1               column=basic:empId, timestamp=1454166590960, value=101
 row1               column=basic:empName, timestamp=1454166590999, value=Suraj
 row1               column=basic:empSalary, timestamp=1454166591037, value=5000
 row2               column=address:Lane, timestamp=1454166662089, value=Hitech city
 row2               column=address:pincode, timestamp=1454166662119, value=500032
 row2               column=address:street, timestamp=1454166662064, value=3B/H
 row2               column=basic:empId, timestamp=1454166661983, value=102
 row2               column=basic:empName, timestamp=1454166662017, value=Kiran
 row2               column=basic:empSalary, timestamp=1454166662044, value=7000
2 row(s) in 0.0410 seconds
```

keyOnlyFilter:

1. This doesnot take any argument
2. Gives you only the key Part of each row.
3. The key includes(Row Id, Column Family, Column Name, Time Stamp).
4. Does not give the value part.

```
hbase(main):004:0> scan 'employee',{ FILTER => "KeyOnlyFilter()"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=
 row1               column=address:pincode, timestamp=1454166591130, value=
 row1               column=address:street, timestamp=1454166591083, value=
 row1               column=basic:empId, timestamp=1454166590960, value=
 row1               column=basic:empName, timestamp=1454166590999, value=
 row1               column=basic:empSalary, timestamp=1454166591037, value=
 row2               column=address:Lane, timestamp=1454166662089, value=
 row2               column=address:pincode, timestamp=1454166662119, value=
 row2               column=address:street, timestamp=1454166662064, value=
 row2               column=basic:empId, timestamp=1454166661983, value=
 row2               column=basic:empName, timestamp=1454166662017, value=
 row2               column=basic:empSalary, timestamp=1454166662044, value=
2 row(s) in 0.0470 seconds
```

FirstKeyOnlyFilter: it will give you the filrst column from first columnfamily of each row

1. This doesnot take any argument
2. Gives you only the first key-value Part of each row.
3. The key includes(Row Id, Column Family, Column Name, Time Stamp).
4. The value includes the value.
5. In the below example it has given AddressColumn Family,Lane details for each row

```
hbase(main):004:0> scan 'employee',{ FILTER => "FirstKeyOnlyFilter()"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row2               column=address:Lane, timestamp=1454166662089, value=Hitech city
2 row(s) in 0.0330 seconds
```

prefixfilter: Row ID name Filters

1. This take one argument
2. You need to Provide a prefix of a row key to search for all the row maching the prefix.
3. It returns only those key-values present in a row that starts with the specified row prefix

```
hbase(main):004:0> scan 'employee',{ FILTER => "PrefixFilter('row1')"}
ROW                 COLUMN+CELL
```

```
row1          column=address:Lane, timestamp=1454166591108, value=Kondapur
row1          column=address:pincode, timestamp=1454166591130, value=500038
row1          column=address:street, timestamp=1454166591083, value=20/A
row1          column=basic:empId, timestamp=1454166590960, value=101
row1          column=basic:empName, timestamp=1454166590999, value=Suraj
row1          column=basic:empSalary, timestamp=1454166591037, value=5000
1 row(s) in 0.0320 seconds
```

The below command will result in both the row.row1 and row2

```
>scan 'employee',{ FILTER => "PrefixFilter('row')"} --Will give you all row whose id starts with row
```

ColumnPrefixFilter it will give you all column name under those column family whose name starts with parmeter passed in columnprefix filter

1. This take one argument
2. You need to Provide a column prefix
3. It returns only those key-values present in a column that starts with the specified column prefix.
4. The column prefix is the part of column name and not the column family

SQL>select * from employee where column_name starts with emp

```
hbase(main):004:0> scan 'employee',{ FILTER => "ColumnPrefixFilter('emp')"}
ROW           COLUMN+CELL
 row1         column=basic:empId, timestamp=1454166590960, value=101
 row1         column=basic:empName, timestamp=1454166590999, value=Suraj
 row1         column=basic:empSalary, timestamp=1454166591037, value=5000
 row2         column=basic:empId, timestamp=1454166661983, value=102
 row2         column=basic:empName, timestamp=1454166662017, value=Kiran
 row2         column=basic:empSalary, timestamp=1454166662044, value=7000
2 row(s) in 0.0300 seconds
```

The above example returns you all the columns that starts with emp(ie empId,empName,empSalary) for all the rows(ie row1,row2).

If you want to return the empId,empName,empSalary of employee with row1 id,then combine both prefixFilter and ColumnPrefixFilter as shown below

```
hbase(main):004:0> scan 'employee',{ FILTER => "PrefixFilter('row1') AND
ColumnPrefixFilter('emp')"}
ROW           COLUMN+CELL
 row1         column=basic:empId, timestamp=1454166590960, value=101
 row1         column=basic:empName, timestamp=1454166590999, value=Suraj
 row1         column=basic:empSalary, timestamp=1454166591037, value=5000
1 row(s) in 0.0170 seconds
```

MultipleColumnPrefixFilter

it will give you list of columns by filtering coulmns (under column family)

1. This take multiple argument ie a list of column prefixes.
2. You need to Provide a list of column prefix
3. It returns only those key-values present in a column that starts with the specified column prefixes.
4. The column prefix is the part of column name and not the column family

The below operation will return all rows(row1 and row2) those columns whose name starts with empId,pin(i.e,pinCode) and str(ie street)

```
hbase(main):004:0> scan 'employee',{ FILTER => "MultipleColumnPrefixFilter('empId','pin','str')"}
ROW                 COLUMN+CELL
 row1               column=address:pincode, timestamp=1454166591130, value=500038
 row1               column=address:street, timestamp=1454166591083, value=20/A
 row1               column=basic:empId, timestamp=1454166590960, value=101
 row2               column=address:pincode, timestamp=1454166662119, value=500032
 row2               column=address:street, timestamp=1454166662064, value=3B/H
 row2               column=basic:empId, timestamp=1454166661983, value=102
2 row(s) in 0.0460 seconds
```

ColumnCountGetFilter

it will fetch first N rows

1. This take single argument ie a number.
2. It returns the first limit number of columns in the table.
3. Use this filter to retrieve a maximum number of columns per row
4. This filter stops the scan once a row has been found that matches the number of column configured.

Scan employee till you get one column for any row

```
hbase(main):004:0> scan 'employee',{ FILTER => "ColumnCountGetFilter(1)"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
1 row(s) in 0.0180 seconds
```

Scan employee till you get 2 column for any row

```
hbase(main):004:0> scan 'employee',{ FILTER => "ColumnCountGetFilter(2)"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row1               column=address:pincode, timestamp=1454166591130, value=500038
1 row(s) in 0.0180 seconds
```

Scan employee till you get 3 column for any row.Need to check why it is returning both the record.

```
hbase(main):019:0> scan 'employee',{ FILTER => "ColumnCountGetFilter(3)"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row1               column=address:pincode, timestamp=1454166591130, value=500038
```

```
row1        column=address:street, timestamp=1454166591083, value=20/A
row2        column=address:Lane, timestamp=1454166662089, value=Hitech city
row2        column=address:pincode, timestamp=1454166662119, value=500032
row2        column=address:street, timestamp=1454166662064, value=3B/H
2 row(s) in 0.0180 seconds
```

PageFilter

1. This filter takes one argument a page size.
2. It returns page size number of rows from the table.
3. It means how many rows should be returned in a page.
4. If you specify 1,it will return just 1 row.If you specify 2,it returns 2 rows and so on.

```
hbase(main):019:0> scan 'employee',{ FILTER => "PageFilter(1)"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row1               column=address:pincode, timestamp=1454166591130, value=500038
 row1               column=address:street, timestamp=1454166591083, value=20/A
 row1               column=basic:empId, timestamp=1454166590960, value=101
 row1               column=basic:empName, timestamp=1454166590999, value=Suraj
 row1               column=basic:empSalary, timestamp=1454166591037, value=5000
1 row(s) in 0.0100 seconds
```

InclusiveStopFilter

1. This filter takes one argument a row key on which to stop scanning.
2. It returns all key-values present in rows up to and including the specified row.
3. The below will scan all the records till it gets row1 and then stops the search.

```
hbase(main):019:0> scan 'employee',{ FILTER => "InclusiveStopFilter('row1')"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row1               column=address:pincode, timestamp=1454166591130, value=500038
 row1               column=address:street, timestamp=1454166591083, value=20/A
 row1               column=basic:empId, timestamp=1454166590960, value=101
 row1               column=basic:empName, timestamp=1454166590999, value=Suraj
 row1               column=basic:empSalary, timestamp=1454166591037, value=5000
1 row(s) in 0.0100 seconds
```

Family Filter(Qualifier Filter)

it will apply filter to column name but with comparison operator

It works very similar to RowFilter,but applies the comparision to the column name available in the row.

This filter takes a compare operator and a comparator. It compares each qualifier name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that column.

The below operations will return you all the rows with column Lane.

SQL>select lane from employee;

```
hbase(main):024:0> scan 'employee',{ FILTER => "QualifierFilter(=,'binary:Lane')"}
ROW                 COLUMN+CELL
 row1               column=address:Lane, timestamp=1454166591108, value=Kondapur
 row2               column=address:Lane, timestamp=1454166662089, value=Hitech city
2 row(s) in 0.0080 seconds
```

The operator that you can use is

- LESS (<)
- LESS_OR_EQUAL (<=)
- EQUAL (=)
- NOT_EQUAL (!=)
- GREATER_OR_EQUAL (>=)
- GREATER (>)

Comparator you can use is:

- BinaryComparator - lexicographically compares against the specified byte array using the `Bytes.compareTo(byte[], byte[])` method.
- BinaryPrefixComparator - lexicographically compares against a specified byte array. It only compares up to the length of this byte array.
- RegexStringComparator - compares against the specified byte array using the given regular expression. Only `EQUAL` and `NOT_EQUAL` comparisons are valid with this comparator.
- SubStringComparator - tests whether or not the given substring appears in a specified byte array. The comparison is case insensitive. Only `EQUAL` and `NOT_EQUAL` comparisons are valid with this comparator.

ValueFilter

it will filter based on value of column name.

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

The below command will search for value called "Suraj" in every column of each column family of each row.

```
hbase(main):026:0> scan 'employee',{ FILTER => "ValueFilter(=,'binary:Suraj')"}
ROW                 COLUMN+CELL
 row1               column=basic:empName, timestamp=1454166590999, value=Suraj
1 row(s) in 0.0240 seconds
```

In case you want to search for "Suraj" in the empName column of basic column family,then you could try something like below.This increases the performance.

```
hbase(main):026:0> scan 'employee',{COLUMNS=>'basic:empName',LIMIT=>4, FILTER =>
"ValueFilter(=,'binary:Suraj')"}
ROW                  COLUMN+CELL
 row1                column=basic:empName, timestamp=1454166590999, value=Suraj
1 row(s) in 0.0180 seconds
```

SingleColumnValueFilter

This filter takes a column family, a qualifier, a compare operator and a comparator. If the specified column is not found – all the columns of that row will be emitted. If the column is found and the comparison with the comparator returns true, all the columns of the row will be emitted. If the condition fails, the row will not be emitted.

This filter also takes two additional optional boolean arguments – filterIfColumnMissing and setLatestVersionOnly

If the filterIfColumnMissing flag is set to true the columns of the row will not be emitted if the specified column to check is not found in the row. The default value is false.

If the setLatestVersionOnly flag is set to false, it will test previous versions (timestamps) too. The default value is true.

These flags are optional and if you must set neither or both.

SQL>Select * from employee where empName='Suraj';

```
hbase(main):026:0> scan 'employee',{ FILTER => "
SingleColumnValueFilter('basic','empName',=,'binary:Suraj')"}
ROW                  COLUMN+CELL
 row1                column=address:Lane, timestamp=1454166591108, value=Kondapur
 row1                column=address:pincode, timestamp=1454166591130, value=500038
 row1                column=address:street, timestamp=1454166591083, value=20/A
 row1                column=basic:empId, timestamp=1454166590960, value=101
 row1                column=basic:empName, timestamp=1454166590999, value=Suraj
 row1                column=basic:empSalary, timestamp=1454166591037, value=5000
1 row(s) in 0.0330 seconds
```

Note: If you have some rows (say row5) which doesnot have basic:empName column,then its result will also be imported.

Some Assignments:-

Select empName,empSalary,pinCode of all the employees.

```
hbase(main):026:0> scan 'employee',{COLUMNS=>['basic:empName','basic:empSalary',
'address:pincode']}
ROW                  COLUMN+CELL
 row1                column=address:pincode, timestamp=1454166591130, value=500038
 row1                column=basic:empName, timestamp=1454166590999, value=Suraj
 row1                column=basic:empSalary, timestamp=1454166591037, value=5000
 row2                column=address:pincode, timestamp=1454166662119, value=500032
 row2                column=basic:empName, timestamp=1454166662017, value=Kiran
 row2                column=basic:empSalary, timestamp=1454166662044, value=7000
```

Select empName,empSalary,pinCode of all the employees where the salary is greater than 6000

```
hbase(main):026:0> scan 'employee',{COLUMNS=>['basic:empName','basic:empSalary',
'address:pincode'], FILTER => "SingleColumnValueFilter('basic','empSalary',>,'binary:6000')"}
ROW                  COLUMN+CELL
row2                column=address:pincode, timestamp=1454166662119, value=500032
 row2                column=basic:empName, timestamp=1454166662017, value=Kiran
 row2                column=basic:empSalary, timestamp=1454166662044, value=7000
1 row(s) in 0.0150 seconds
```

SQL>Select empName,empSalary,pincode of all the employees where the salary is greater than 6000 or pinCode='500038'

```
hbase(main):026:0> scan 'employee',{COLUMNS=>['basic:empName','basic:empSalary',
'address:pincode'], FILTER => "SingleColumnValueFilter('basic','empSalary',>,'binary:6000') OR
SingleColumnValueFilter('address','pincode',=,'binary:500038')  "}
row1                column=address:pincode, timestamp=1454166591130, value=500038
 row1                column=basic:empName, timestamp=1454166590999, value=Suraj
 row1                column=basic:empSalary, timestamp=1454166591037, value=5000
 row2                column=address:pincode, timestamp=1454166662119, value=500032
 row2                column=basic:empName, timestamp=1454166662017, value=Kiran
 row2                column=basic:empSalary, timestamp=1454166662044, value=7000
2 row(s) in 0.0250 seconds
```

We will use the Employee table to work with Hbase and Java.

Basic			address		
empId	empName	empSalary	Street	Lane	pincode
101	Suraj	5000	20/A	Kondapur	500038
102	Kiran	7000	3B/H	Hitech city	500032

Steps:-

1. **Open HBase shell and execute the below commands to create Employee table and insert records into the table.**

```
hbase>disable "employee"
drop "employee"
create "employee", "basic", "address"
put "employee","row1","basic:empId","101"
put "employee","row1","basic:empName","Suraj"
put "employee","row1","basic:empSalary","5000"
put "employee","row1","address:street","20/A"
put "employee","row1","address:Lane","Kondapur"
put "employee","row1","address:pincode","500038"

put "employee","row2","basic:empId","102"
put "employee","row2","basic:empName","Kiran"
put "employee","row2","basic:empSalary","7000"
put "employee","row2","address:street","3B/H"
put "employee","row2","address:Lane","Hitech city"
put "employee","row2","address:pincode","500032"
```

2. **Create a Java Project in eclipse "HBaseProject", and add the below lib to class path.**

Note: In case of Cloudera5.5.0,below jars are main jars needed to compile the program and all remaining jars are dependent jars needed to execute the program. All the jars should be placed in the build path.

a. hadoop-common-X.jar,
b. hbase-common-X.jar,
c. hbase-client-X.jar

3. Create a Java Class with a main method and write the below logic to create a HBase Connection.

```
// Create configuration object having all configuration properties
Configuration conf = HBaseConfiguration.create();

// Point to Employee table
HTable table = new HTable(conf, "employee");
```

4. Selecting a row:- Write the below Logic to select just one row

```
Get g = new Get(Bytes.toBytes("row1"));
Result result1 = table.get(g);

byte[] empId = result1.getValue(Bytes.toBytes("basic"),Bytes.toBytes("empId"));
byte[] empName = result1.getValue(Bytes.toBytes("basic"),Bytes.toBytes("empName"));
byte[] empSalary = result1.getValue(Bytes.toBytes("basic"),Bytes.toBytes("empSalary"));
byte[] lane = result1.getValue(Bytes.toBytes("address"),Bytes.toBytes("Lane"));
byte[] pin = result1.getValue(Bytes.toBytes("address"),Bytes.toBytes("pincode"));
byte[] street = result1.getValue(Bytes.toBytes("address"),Bytes.toBytes("street"));

System.out.println("empID:" + Bytes.toString(empId));
System.out.println("empName:" + Bytes.toString(empName));
System.out.println("empSalary:" + Bytes.toString(empSalary));
System.out.println("lane:" + Bytes.toString(lane));
System.out.println("pin:" + Bytes.toString(pin));
System.out.println("street:" + Bytes.toString(street));
```

5. Select More than one row: To Select more than one row, Say row1,row2,Write the below logic

```
// Create Get Objects
Get get1 = new Get(Bytes.toBytes("row1"));
Get get2 = new Get(Bytes.toBytes("row2"));

// Prepare an arraylist and store all get Objects there
ArrayList<Get> getList = new ArrayList<Get>();
getList.add(get1);
getList.add(get2);

// Call get() on table by passing arrayList,this will return Result
// array holding all the results
Result[] resultArr = table.get(getList);

for (Result result : resultArr) {
        byte[] empId = result.getValue(Bytes.toBytes("basic"),Bytes.toBytes("empId"));
    byte[] empName = result.getValue(Bytes.toBytes("basic"),Bytes.toBytes("empName"));
    byte[] empSalary = result.getValue(Bytes.toBytes("basic"),Bytes.toBytes("empSalary"));
    byte[] lane = result.getValue(Bytes.toBytes("address"),Bytes.toBytes("Lane"));
    byte[] pin = result.getValue(Bytes.toBytes("address"),Bytes.toBytes("pincode"));
    byte[] street = result.getValue(Bytes.toBytes("address"),Bytes.toBytes("street"));

    System.out.println("\n\n");
    System.out.println("empID:" + Bytes.toString(empId));
    System.out.println("empName:" + Bytes.toString(empName));
    System.out.println("empSalary:" + Bytes.toString(empSalary));
    System.out.println("lane:" + Bytes.toString(lane));
    System.out.println("pin:" + Bytes.toString(pin));
    System.out.println("street:" + Bytes.toString(street));
    }
```

Output:-

```
empID:101
empName:Suraj
empSalary:5000
lane:Kondapur
pin:500038
street:20/A

empID:102
empName:kiran
empSalary:7000
lane:Hitech City
pin:500032
street:30B
```

6. Single Put Operation: Write the below code

```
// Create Put Objects
Put put=new Put(Bytes.toBytes("row3"));
put.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empId"), Bytes.toBytes("103"));
put.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empName"), Bytes.toBytes("Karan"));
put.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empSalary"), Bytes.toBytes("9000"));

put.addColumn(Bytes.toBytes("address"),Bytes.toBytes("Lane"), Bytes.toBytes("25/A"));
put.addColumn(Bytes.toBytes("address"),Bytes.toBytes("pincode"), Bytes.toBytes("500089"));
put.addColumn(Bytes.toBytes("address"),Bytes.toBytes("street"), Bytes.toBytes("SR Nagar"));

table.put(put);
//After perform put,manually check in the table using scan command
```

7. Multiple Put Operations:Write the below code

```
// Create Put Objects
Put put1=new Put(Bytes.toBytes("row4"));
put1.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empId"), Bytes.toBytes("104"));
put1.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empName"), Bytes.toBytes("Rabina"));
put1.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empSalary"), Bytes.toBytes("9000"));

put1.addColumn(Bytes.toBytes("address"),Bytes.toBytes("Lane"), Bytes.toBytes("45/A"));
put1.addColumn(Bytes.toBytes("address"),Bytes.toBytes("pincode"), Bytes.toBytes("50002"));
put1.addColumn(Bytes.toBytes("address"),Bytes.toBytes("street"), Bytes.toBytes("Lingampally"));

// Create Put Objects
Put put2=new Put(Bytes.toBytes("row5"));
put2.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empId"), Bytes.toBytes("105"));
put2.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empName"), Bytes.toBytes("Nabu"));
put2.addColumn(Bytes.toBytes("basic"),Bytes.toBytes("empSalary"), Bytes.toBytes("8700"));

put2.addColumn(Bytes.toBytes("address"),Bytes.toBytes("Lane"), Bytes.toBytes("55/A"));
put2.addColumn(Bytes.toBytes("address"),Bytes.toBytes("pincode"), Bytes.toBytes("500089"));
put2.addColumn(Bytes.toBytes("address"),Bytes.toBytes("street"), Bytes.toBytes("Manikonda"));

//Add Put objects into ArrayList
ArrayList<Put> putList=new ArrayList<Put>();
putList.add(put1);
putList.add(put2);

//Perform put operation
table.put(putList);

table.close();
//After perform put,manually check in the table using scan command
```

8. Single Put Operations:Write the below code

```
// Create Delete Objects
Delete delete=new Delete(Bytes.toBytes("row5"));//Deletes entire row if nothing is specified

//deletes empId column from basic column family of row4
delete.addColumn(Bytes.toBytes("basic"), Bytes.toBytes("empId"));

//deletes 1st version of empId(means the version which was first created
//delete.addColumn(Bytes.toBytes("basic"), Bytes.toBytes("empId"),1);

//deletes 5th versions and all the old versions ie 1,2,3,4,5
//delete.addColumn(Bytes.toBytes("basic"), Bytes.toBytes("empId"),5);

//deletes Just basic  column family of row4
delete.addFamily(Bytes.toBytes("basic"));

//deletes  All 1st,2nd,3rd version  of row4 from basic column family
//delete.addFamily(Bytes.toBytes("basic"),3);

//Perform delete operation
table.delete(delete);

table.close();
//After perform put,manually check in the table using scan command
```

9. Multiple Row Delete:Perform below steps:

```
// Create Delete Objects
Delete delete1=new Delete(Bytes.toBytes("row3"));//Deletes entire row if nothing is specified
```

```
delete1.addColumn(Bytes.toBytes("basic"), Bytes.toBytes("empId")); //deletes empId column from basic column family o
f row3
delete1.addFamily(Bytes.toBytes("address"));

Delete delete2=new Delete(Bytes.toBytes("row2"));//Deletes entire row if nothing is specified
delete2.addColumn(Bytes.toBytes("basic"), Bytes.toBytes("empId")); //deletes empId column from basic column family o
f row2
delete2.addColumn(Bytes.toBytes("address"),Bytes.toBytes("Lane"));
delete2.addColumn(Bytes.toBytes("address"),Bytes.toBytes("pincode"));

ArrayList<Delete> deleteList=new ArrayList<Delete>();
deleteList.add(delete1);
deleteList.add(delete2);

//Perform delete operation
table.delete(deleteList);

table.close();
//After perform put,manually check in the table using scan command
```

Assignments:-

WAP to read the data from **NASDAQ_daily_prices_A.csv**

exchange,stock_symbol,date,stock_price_open,stock_price_high,stock_price_low,stock_price_close,stock_volume,stock_price_adj_close

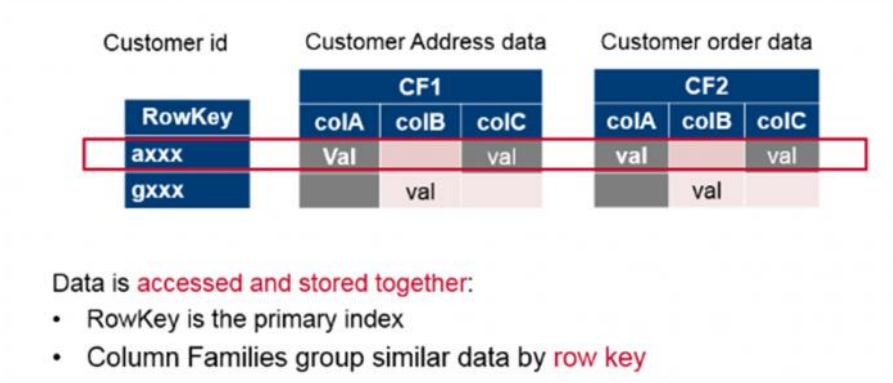
Divide the dataset into 2 column family & store the data into it

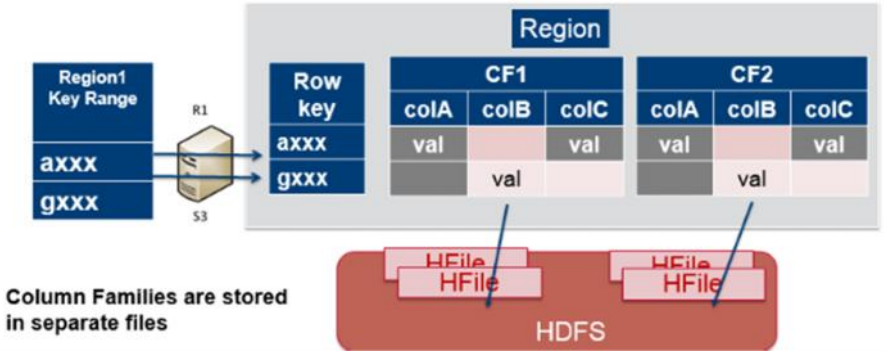
exchange,stock_symbol,date, stock_volume ->Basic

stock_price_open,stock_price_high,stock_price_low,stock_price_close,

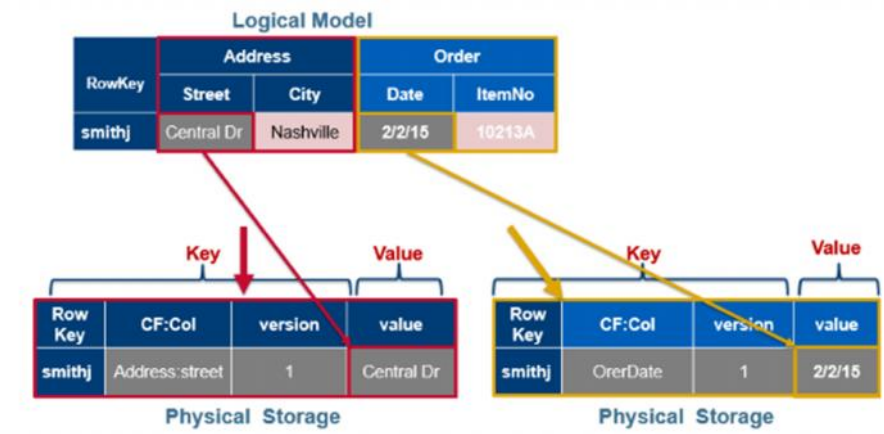
stock_price_adj_close->Price

Database normalization eliminates redundant data, which makes storage efficient. However, a normalized schema causes joins for queries, in order to bring the data back together again. While HBase does not support relationships and joins, data that is accessed together is stored together so it avoids the limitations associated with a relational model.
Distributed Joins are not efficient.





The data is stored in HBase table cells. The entire cell, with the added structural information, is called Key Value. The entire cell, the row key, column family name, column name, timestamp, and value are stored for every cell for which you have set a value. The key consists of the row key, column family name, column name, and timestamp. When the max number of versions is exceeded, extra records will be eventually removed.



Architecture of HBase:-

Hbase Architecture contains 3 Servers to work with hbase

HMaster: Handles Region assignment, DDL (create, delete tables) operations are handled.this component is installed at NameNode.

RegionServer: When accessing data, clients communicate with HBase RegionServers directly

Zookeeper: maintains a live cluster state.

QuorumPeer

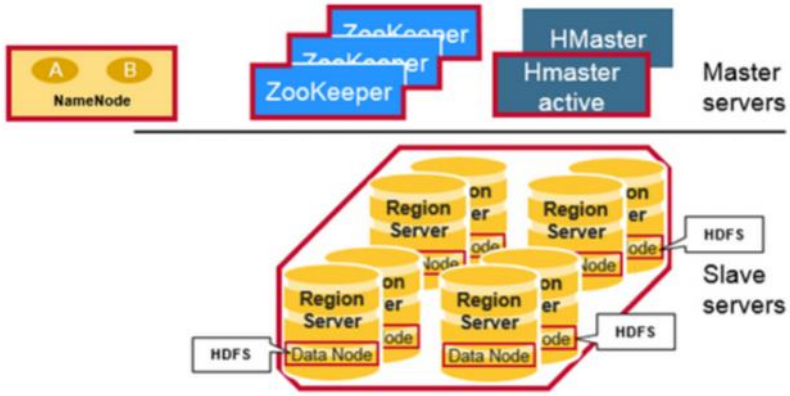
HBase's version of ZooKeeper's QuorumPeer. When HBase is set to manage ZooKeeper, this class is used to start up QuorumPeer instances. By doing things in here rather than directly calling to ZooKeeper, we have more control over the process. This class uses `ZKConfig` to parse the zoo.cfg and inject variables from HBase's site.xml configuration in

A distributed Apache HBase (TM) installation depends on a running ZooKeeper cluster. All participating nodes and clients need to be able to access the running ZooKeeper ensemble. Apache HBase by default manages a ZooKeeper "cluster" for you. It will start and stop the ZooKeeper ensemble as part of the HBase start/stop process. You can also manage the ZooKeeper ensemble independent of HBase and just point HBase at the cluster it should use. To toggle HBase management of ZooKeeper, use the `HBASE_MANAGES_ZK` variable in `conf/hbase-env.sh`. This variable, which defaults to `true`, tells HBase whether to start/stop the ZooKeeper ensemble servers as part of HBase start/stop.

The Hadoop DataNode stores the data that the Region Server is managing. All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the

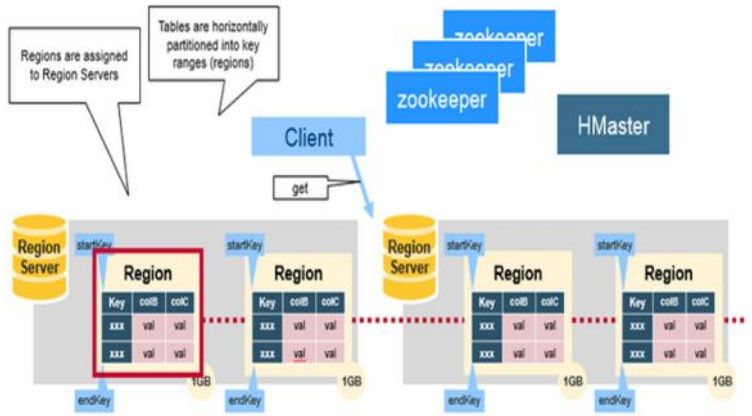
RegionServers. HBase data is local when it is written, but when a region is moved, it is not local until compaction.

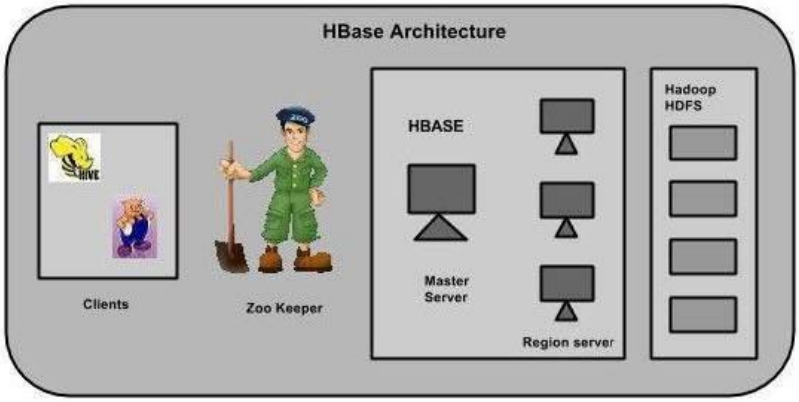
The NameNode maintains metadata information for all the physical data blocks that comprise the files.

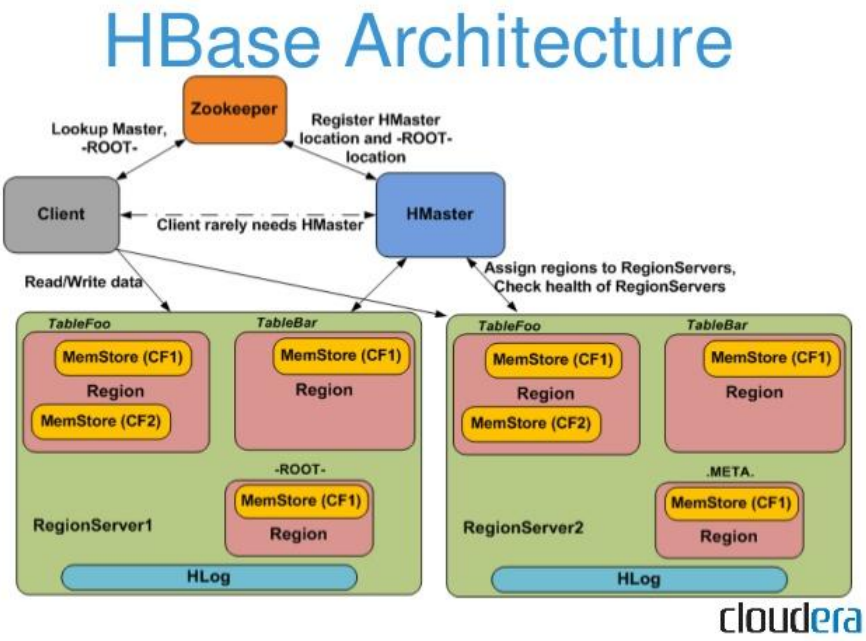


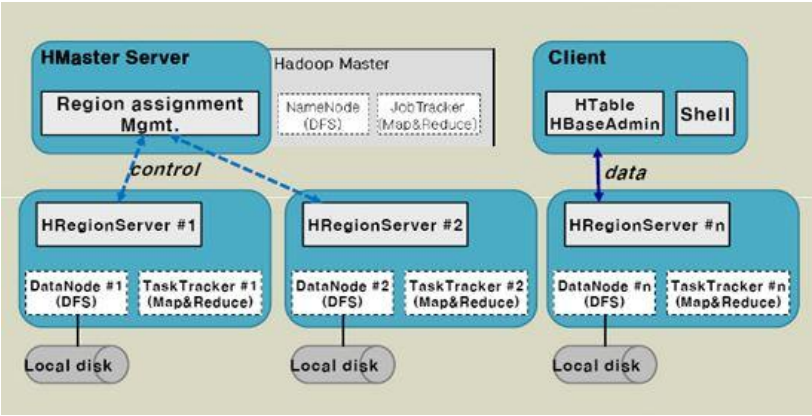
Regions

HBase Tables are divided horizontally by row key range into “Regions.” A region contains all rows in the table between the region’s start key and end key. Regions are assigned to the nodes in the cluster, called “Region Servers,” and these serve data for reads and writes. A region server can serve about 1,000 regions.


Regions are assigned to Region Servers
Tables are horizontally partitioned into key ranges (regions)
zookeeper
zookeeper
zookeeper
HMaster
Client
get
Region Server
Region
Region Server
Region
Region


HBase Architecture
HBASE
Hadoop HDFS
Clients
Zoo Keeper
Master Server
Region server





HBase system consists of

1. Master Servers
2. Region Servers
3. Regions
4. Stores

Master Server:

HMasterServer is an implementation.

responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes

In a distributed cluster, the Master typically runs on the NameNode.
it is possible to run HBase in a multiple master setup, in which there is a single active master

Region Server:

Are installed in each Slave node.
It Hosts and manages the region
Automatically splits a big region into 2 regions equally from the middle.
Client interact with Region Server directly.
Handles Read/Write Requests.

Regions:

HBase tables are partitioned into multiple regions with each region storing a range of the table's rows, and multiple regions are assigned by the master to a region server

Tables are split into regions. Regions are vertically divided by column family into Stores.
Regions contain an in-memory data store (MemStore) and a persistent data store (HFile), and all regions on a region server share a reference to the write-ahead log (WAL) (HLog)which is used to store new data that hasn't yet been persisted to permanent storage and to recover from region server crashes. Each region holds a specific range of row keys, and when a region exceeds a configurable size, HBase automatically splits the region into two child regions, which is the key to scaling HBase.

Stores:-

They are the file where actually data are stored into.

Zookeeper:-

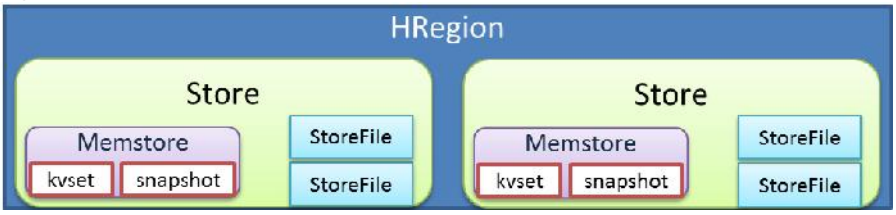
HBase utilize Zookeeper(a distributed coordination service) to manage region assignments to region servers, and to recover from region server crashes by loading the crashed region server's regions onto other functioning region servers.
Client(Map-reduce,Pig,Hive) interacts with HBase after authentication done by zookeeper.
Once authentication is done, Client directly interacts with RegionServer.However zookeeper is going to talk to HMaster for authentication.

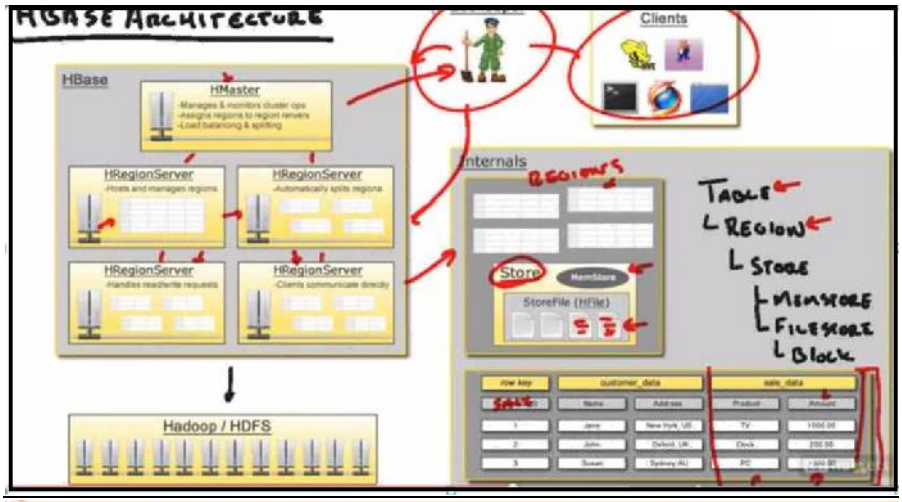
How HBase works

As a table grows, more and more regions are created and spread across the entire cluster. When clients request a specific row key or scan a range of row keys, HBaseServer tells them the regions on which those keys exist, and the clients then communicate directly with the region servers where those regions exist. This design minimizes the number of disk seeks required to find any given row, and optimizes HBase toward disk transfer when returning data. This is in contrast to relational databases, which might need to do a large number of disk seeks before transferring data from disk, even with indexes.

Clients interact with HBase via one of several available APIs, including a native Java API as well as a REST-based interface and several RPC interfaces (Apache Thrift, Apache Avro). You can also use DSLs to HBase from Groovy, Jython, and Scala.

RegionServer contains Store





Row Key	Column Family "Metadata"	Column Family "Photo"
emp_id1	firstname:Suman,lastname:jain	<suman.gif>
emp_id2	firstname:Rithik,lastname:Verma	<rithik.gif>
emp_id3	firstname:Lokesh,lastname:Kumar	

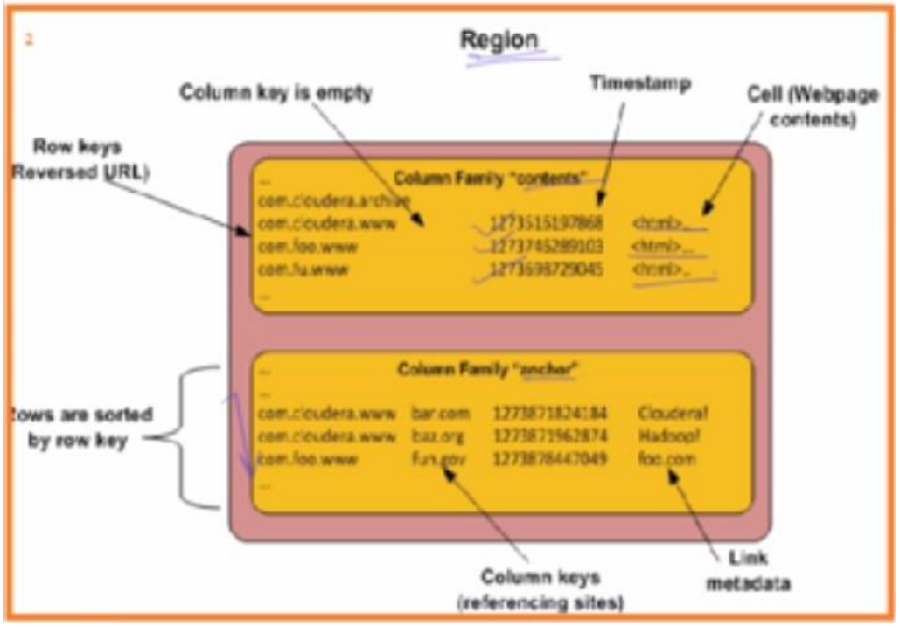


Table and column families

Table can be used to group all the related columns together, but within the column itself, there could be 2 more column that can be closely associated together. We create a column family for that.

A small talk about region and regionserver:-

A region is a continuous block on rows stored sequentially.
When a region grows too large, it gets splitted into multiple regions by hbase framework, and the two regions will be managed by regionservers.

How Region server gets splitted?

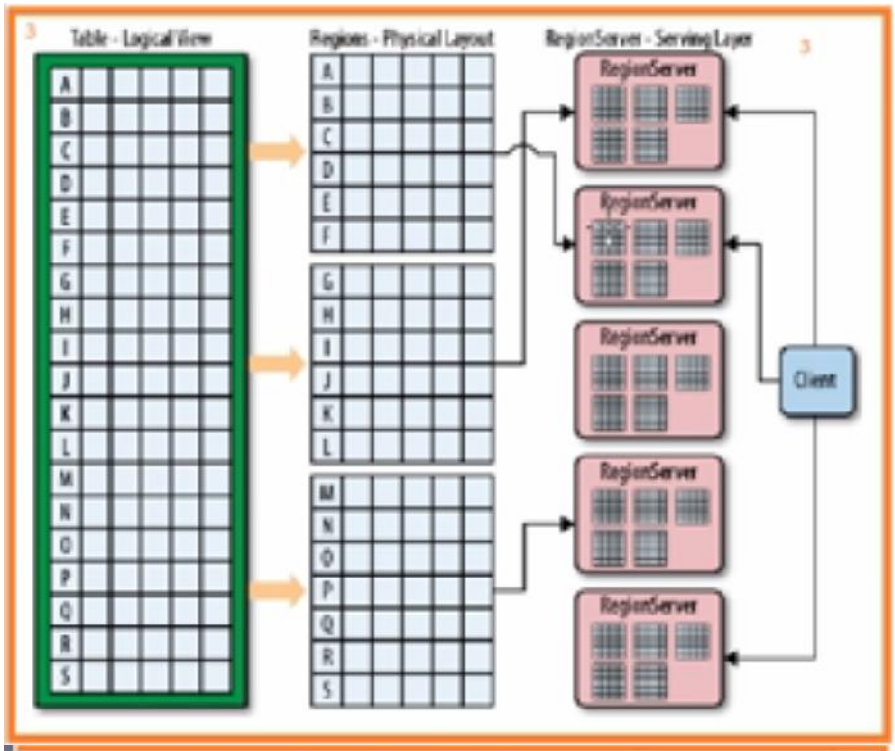
Let's say the split size of the HBase region server is 64MB(configurable) and u kept on adding new rows, and eventually the size grew 64.1 MB, in that case it get splitted into 2 parts and it gets divided right from the middle,ie 32.05 MB each.This is called dynamic partitioning or dynamic sharding/Partitioning/splitting.

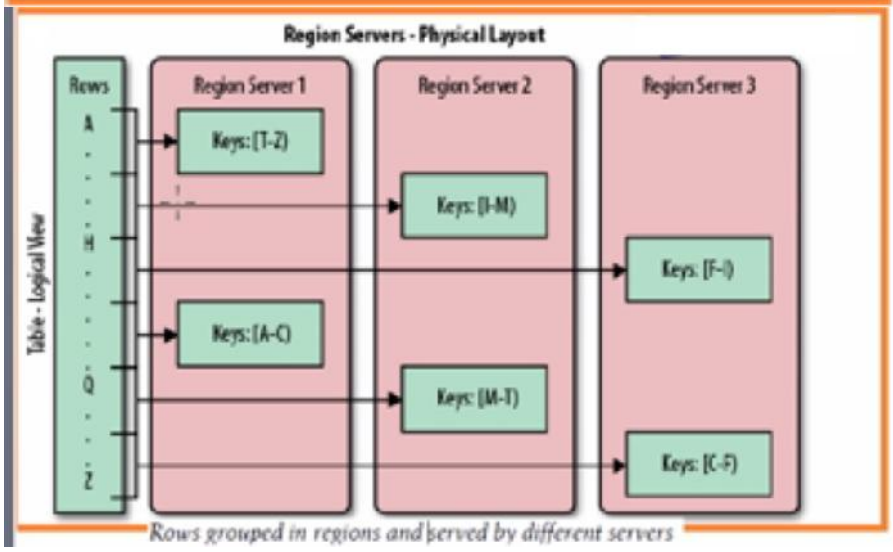
Sometime the 2 regions can also shrink together due to row getting deleted from 2 bigger regions.

Column Family:-

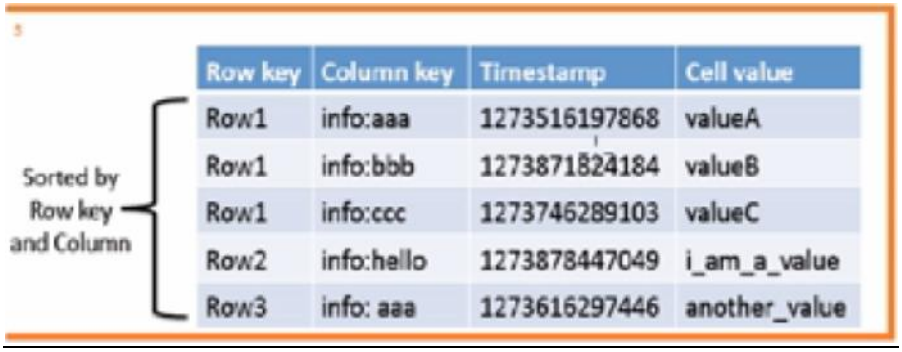
You can also enable compression type(gz,lzo) in column family
You can specify how many versions can be maintained in column family.
You can specify TTL(Time to leave)
You can also keep the data inmemory

Commented [SG1]:





HBase is a sorted Distributed Map



Row key	Column key	Timestamp	Cell value
Row1	info:aaa	1273516197868	valueA
Row1	info:bbb	1273871824184	valueB
Row1	info:ccc	1273746289103	valueC
Row2	info:hello	1273878447049	i_am_a_value
Row3	info: aaa	1273616297446	another_value

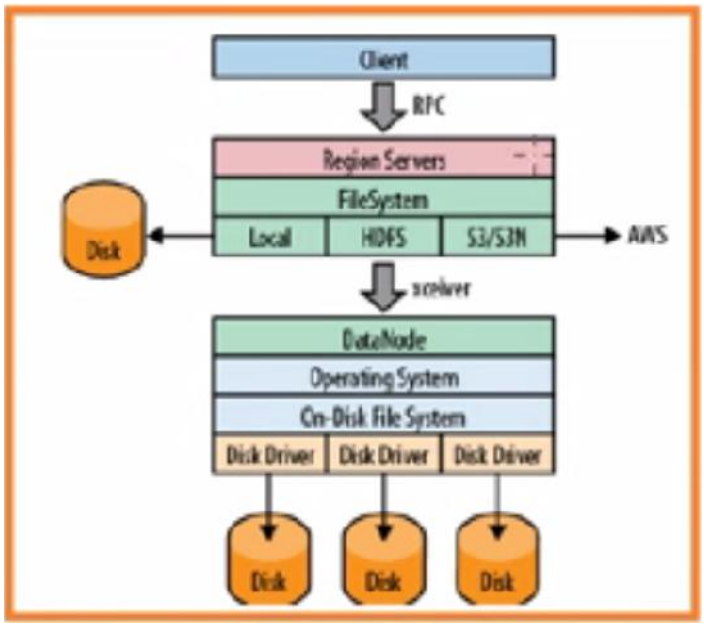
Row1 has 3 columns, column names are aaa, bbb, ccc they all belong to info column family. These columns has valueA, valueB, ValueC.
In case a column has more than 2 values, you will see more than 2 records but the timestamp will tell you which one is the recent value.

The above table will be sorted on ascending order of row Key,(row1 then row2,then row3) followed by sorted order of column Key(aaa then bbb,ccc) followed by timestamp,followed by cell Value.
In case 1 column has 2 different values, there timestamp will be in descending order followed by there values,so that the recent updated value will always be at the top.

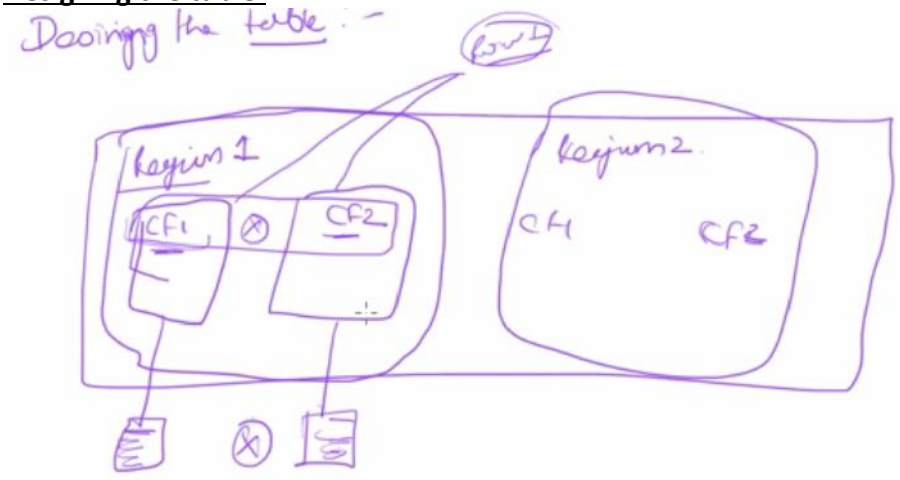
Database vs RDBMS

	RDBMS	HBase
Data layout	Row or column-oriented	Column Family-oriented
Transactions	Yes	Single row only
Query language	SQL	get/put/scan
Security	Authentication/Authorization	TBD
Indexes	Yes	Row-key only
Max data size	TBs	PB+
Read/write throughput limits	1000s queries/second	Millions of queries/second

How data is read using HBase



Designing the table:



Regions are nothing but collections of rows
(row1 ... row 10) →Region 1
(row11 ... row 20) →Region 2

We would have collections of column family at each row label.
Row with same row-key will always be stored under the same node.

When even we want specify the row-Key and want to get all the column data associated with that row-key then internally a join happens among the data stored for that particular row.

Data-coordinates

Data accessed in the hbase is in form of key,value pair.
Key has 4 parts

1. table name
2. column family
3. column name
4. version number: You will define this while creating a column family.

If you dont specify which version to get, it will always return you the latest version.

Hbase Storage Technique

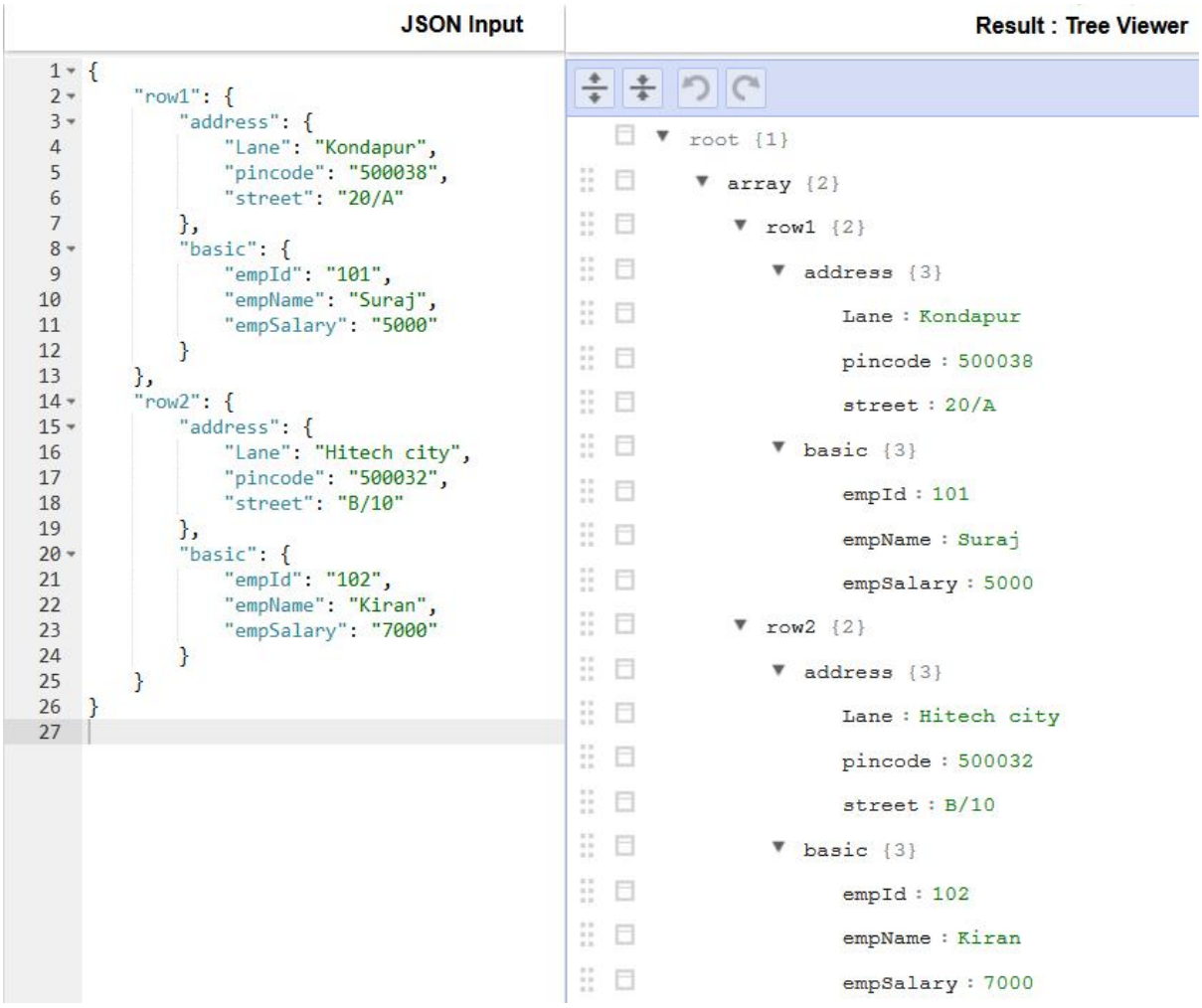
- each row will have a unique id called rowid
- we have concept called column family
- we have a concept called qualifier(columns)
- We can group one or more column in a column family
- HBase uses key value pair to store data, unstructured(both)

Employee

Basic			address		
empId	**empName**	**empSalary**	**Street**	**Lane**	**pincode**
101	Suraj	5000	20/A	Kondapur	500038
102	Kiran	7000	3B/H	Hitech city	500032

```
>create "employee", "basic",  "address"
>put "employee","row1","basic:empId","101"
>put "employee","row1","basic:empName","Suraj"
>put "employee","row1","basic:empSalary","5000"
>put "employee","row1","address:street","20/A"
>put "employee","row1","address:Lane","Kondapur"
>put "employee","row1","address:pincode","500038"
```


```
{
  "row1": {
    "address": {
      "Lane": "Kondapur",
      "pincode": "500038",
      "street": "20/A"
    },
    "basic": {
      "empId": "101",
      "empName": "Suraj",
      "empSalary": "5000"
    }
  },
  "row2": {
    "address": {
      "Lane": "Hitech city",
      "pincode": "500032",
      "street": "B/10"
    },
    "basic": {
      "empId": "102",
      "empName": "Kiran",
      "empSalary": "7000"
    }
  }
}
```



Insert:-

```
Update:- put 'table name','row ','Column family:column name','new value'


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


select:- get 'emp', '1'
```

Executed in cloudera 4.4

Objective:-

The objective of this session is to develop a map-reduce program that can read the student data and write it to hbase table. The file student1.txt is present in dataset folder where this document is present.Copy dataset folder into your VM desktop.Launch terminal and navigate to desktop and issue the below command to put the file in hdfs

```
$ hadoop fs -mkdir /user/cloudera/marks
$hadoop fs -put student1.txt /user/cloudera/marks/
```

Verify your input file

```
$hadoop fs -cat /user/cloudera/marks/student1.txt
```

Sample data:-

mahesh,980,SMCS,Orissa,mahesh@gmail.com
Kumar,950,AOG,Orissa,kumar@rediff.in
Deepak,900,Saint Pauls,Jharkhand,deepak@gmail.com
prasad,890,Saint Marys,Andhra Pradesh,prasad@smcs.in
Preeti,900,KV,Andhra Pradesh,preeti@gmail.com
Suman,975,KV,Orissa,suman@rediff.in
Ratan,980,SMCS,Andhra Pradesh,ratan@smcs.in
Deepak,995,st Anns,Jharkhand,deepak@stanns.in

create hbase table to store the data:-

```
$hbase shell
hbase>disable 'student' //For 2nd time,If you want to re-start the process
hbase>drop 'student'    //for 2nd time
hbase>create 'student','info','address'
hbase>list                  //shows all tables
hbase>scan 'student'      //select * from student,Table name should be enclosed
```

Here we are using 2 column family
info : Name,Marks,SchoolName
address : state,email

Insert 1 record into the table manually

Lets insert **mahesh,980,SMCS,Orissa,mahesh@gmail.com into the table**

```
put 'student','1','info:name','mahesh'
put 'student','1','info:marks','980'
put 'student','1','info:school','smcs'
put 'student','1','address:state','orissa'
put 'student','1','address:email','mahesh@gmail.com'
1 is the row id here
```

Verify Your data

```
hbase(main):006:0> scan 'student'
ROW                    COLUMN+CELL
 1                     column=info:email, timestamp=1445718391512, value=mahesh@gmail.com
 1                     column=info:marks, timestamp=1445718381147, value=980
 1                     column=info:name, timestamp=1445718281275, value=mahesh
 1                     column=address:school, timestamp=1445718391460, value=smcs
 1                     column=address:state, timestamp=1445718391487, value=orissa
1 row(s) in 0.0150 seconds
```

Lets write the Map-reduce Program and store all of them into hbase table

1. Create a java project(MapReduce2HbaseWrite) & add the below jar files

marked are common jars for both the projects

2. Execute the project present in this document

Executed in cloudera 4.4

Objective:-

This is the continuation of previous session. In previous session we had store the student details in the table called student.
Here we are using 2 column family
info : Name,Marks,SchoolName
address : state,email

```
$hbase shell
hbase>scan 'student'
ROW                COLUMN+CELL
1                  column=address:email, timestamp=1445735144019, value=mahesh@gmail.com
1                  column=address:state, timestamp=1445735143998, value=orissa
1                  column=info:marks, timestamp=1445735143937, value=980
1                  column=info:name, timestamp=1445735143896, value=mahesh
1                  column=info:school, timestamp=1445735143976, value=smcs
2                  column=address:email, timestamp=1445735152701, value=mahesh@gmail.com
2                  column=address:state, timestamp=1445735152701, value=Orissa
2                  column=info:marks, timestamp=1445735152701, value=980
2                  column=info:name, timestamp=1445735152701, value=mahesh
2                  column=info:school, timestamp=1445735152701, value=SMCS
3                  column=address:email, timestamp=1445735152701, value=kumar@rediff.in
3                  column=address:state, timestamp=1445735152701, value=Orissa
3                  column=info:marks, timestamp=1445735152701, value=950
3                  column=info:name, timestamp=1445735152701, value=Kumar
3                  column=info:school, timestamp=1445735152701, value=AOG
4                  column=address:email, timestamp=1445735152701, value=deepak@gmail.com
4                  column=address:state, timestamp=1445735152701, value=Jharkhand
4                  column=info:marks, timestamp=1445735152701, value=900
4                  column=info:name, timestamp=1445735152701, value=Deepak
4                  column=info:school, timestamp=1445735152701, value=Saint Pauls
5                  column=address:email, timestamp=1445735152701, value=prasad@smcs.in
5                  column=address:state, timestamp=1445735152701, value=Andhra Pradesh
5                  column=info:marks, timestamp=1445735152701, value=890
5                  column=info:name, timestamp=1445735152701, value=prasad
5                  column=info:school, timestamp=1445735152701, value=Saint Marys
6                  column=address:email, timestamp=1445735152701, value=preeti@gmail.com
6                  column=address:state, timestamp=1445735152701, value=Andhra Pradesh
6                  column=info:marks, timestamp=1445735152701, value=900
6                  column=info:name, timestamp=1445735152701, value=Preeti
6                  column=info:school, timestamp=1445735152701, value=KV
7                  column=address:email, timestamp=1445735152701, value=suman@rediff.in
7                  column=address:state, timestamp=1445735152701, value=Orissa
7                  column=info:marks, timestamp=1445735152701, value=975
7                  column=info:name, timestamp=1445735152701, value=Suman
7                  column=info:school, timestamp=1445735152701, value=KV
8                  column=address:email, timestamp=1445735152701, value=ratan@smcs.in
8                  column=address:state, timestamp=1445735152701, value=Andhra Pradesh
8                  column=info:marks, timestamp=1445735152701, value=980
8                  column=info:name, timestamp=1445735152701, value=Ratan
8                  column=info:school, timestamp=1445735152701, value=SMCS
9                  column=address:email, timestamp=1445735152701, value=deepak@stanns.in
9                  column=address:state, timestamp=1445735152701, value=Jharkhand
9                  column=info:marks, timestamp=1445735152701, value=995
9                  column=info:name, timestamp=1445735152701, value=Deepak
9                  column=info:school, timestamp=1445735152701, value=st Anns
9 row(s) in 0.0590 seconds
```

`1-9 is the row Id here`

Description:-

As we know Map-Reduce
(k1,V1)-> Map ->(k2,V2) ->Reduce(K3,V3)

In All Previous Examples, our input source was HDFS files. But here we want to read the data from HBase table directly into our Mapper.map(-,-,-).We would find the state wise topper, and store in into HBase table

TableMapper & TableReduce
Hbase provide the below 2 abstract classes to read and write from/to HBase

1. **org.apache.hadoop.hbase.mapreduce.TableMapper**
2. **org.apache.hadoop.hbase.mapreduce.TableReducer**

<key,value> Input to TableMapper = <ImmutableBytesWritable,Result>
ImmutableBytesWritable holds the row_key of the current row being processed.
Result holds the value (result) of the row

These input key,value is fixed for TableMapper. But the output key value from TableMapper is user defined.

Output of TableReducer =<Writable,Writable>

1. Create a java project(MapReduce2HbaseWrite) & add the below jar files

2. Execute the program given along with this document