NoSql: Mongo, HBASE

DS8003 – MGT OF BIG DATA AND TOOLS

Ryerson University

Instructor: Kanchana Padmanabhan

Today

- What we will look at today…
  - HBASE (Column (Family) store)
  - MongoDB (Document Store)

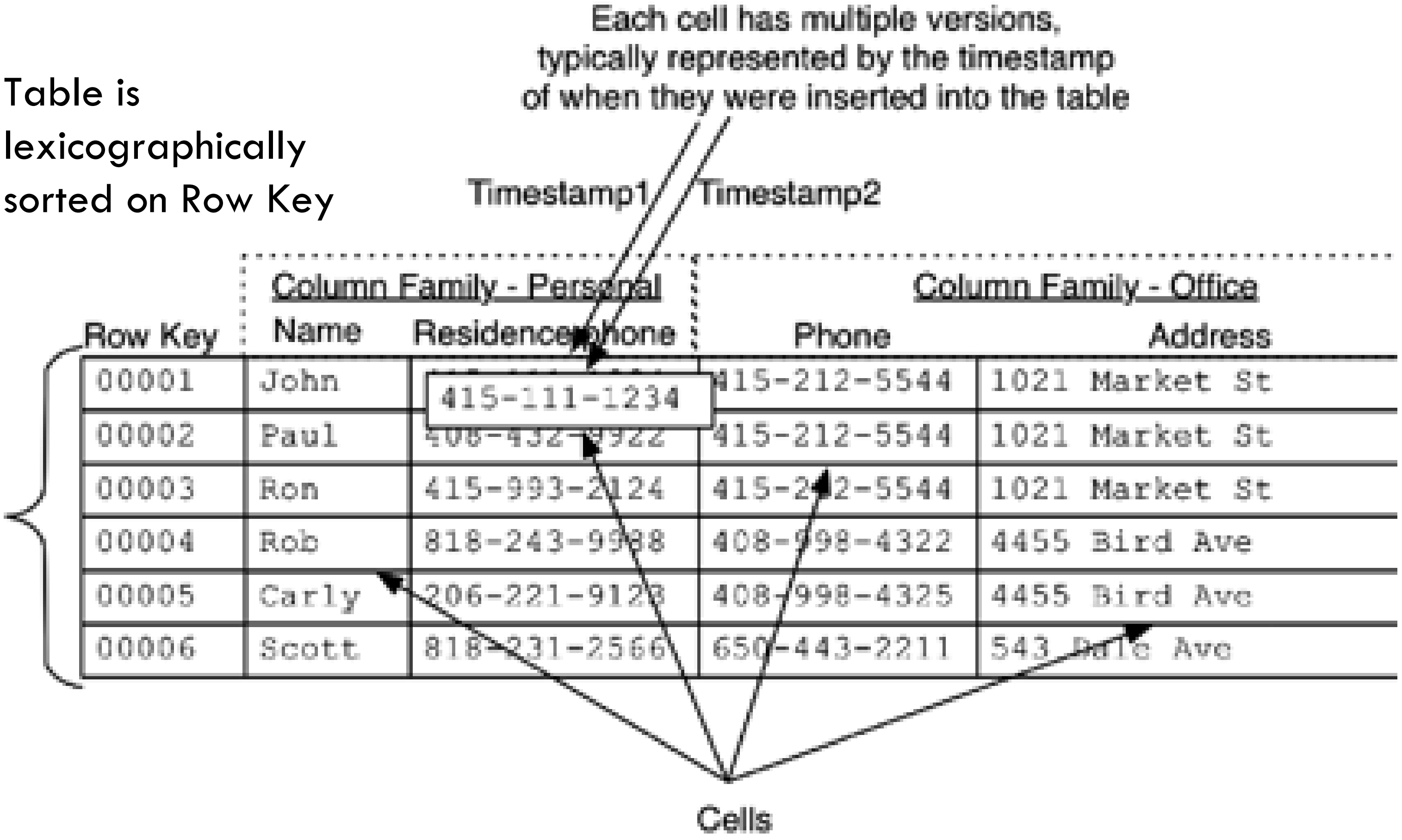
HBASE

- □ HBase is a distributed column family-oriented data store built on top of HDFS
- □ Based on Google’s BigTable
- □ NoSQL (non relational) key/value store
- □ Operations (queries) run in real-time (unlike HIVE)
- □ Random read write access to large data
- □ Facebook use it for messaging and real-time analytics [https://www.xplenty.com/blog/2014/05/hive-vs-hbase/]

http://amitjj.blogspot.ca/2012/11/how-hbase-works.html

Column family-oriented Data Model

Table is lexicographically sorted on Row Key



Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table

	Column Family - Personal		Column Family - Office	
Row Key	Name	Residence phone	Phone	Address
00001	John	415-111-1234	415-212-5544	1021 Market St
00002	Paul	408-432-9922	415-212-5544	1021 Market St
00003	Ron	415-993-2124	415-212-5544	1021 Market St
00004	Rob	818-243-9988	408-998-4322	4455 Bird Ave
00005	Carly	206-221-9123	408-998-4325	4455 Bird Ave
00006	Scott	818-231-2566	650-443-2211	543 Dale Ave

http://www.rittmanmead.com/2014/05/trickle-feeding-log-data-into-hbase-using-flume/

Column family-oriented Data Model

- □ Table – Similar to RDBMS (collection of rows)
- □ Row-Key – Uniquely identify each row in the table
- □ Column – Key-Value pair (can be added on the Fly)
- □ Column-Family – way to physically group Columns (specified at table creation time)
- □ Timestamp – versioning (history)
- □ Cell – Actual value stored in the table

Column Families are stored and accessed separately. This means that not all parts of a row are picked up in a single I/O operation

http://dbmsmusings.blogspot.ca/2010/03/distinguishing-two-major-types-of_29.html
http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/9353-login1210_khurana.pdf

Relational => HBASE

Name	Age	Gender	Company
K	22	F	Facebook
M	35	NULL	Verizon

Name (Row Key)	Personal Information		Professional Information
	Age	Gender	Company
K	22	F	Facebook
M	35	[redacted]	Verizon

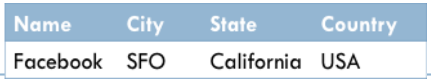
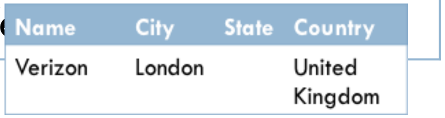
Relational => HBASE

Name	Age	Gender	Company
K	22	F	Facebook
M	35	M	Verizon

Name	City	State	Country
Facebook	SFO	Californi a	USA
Verizon	London		United Kingdom

NO FOREIGN KEYS
NO JOINS
NO SECONDARY INDEXING

ALTERNATIVE: DENORMALIZE

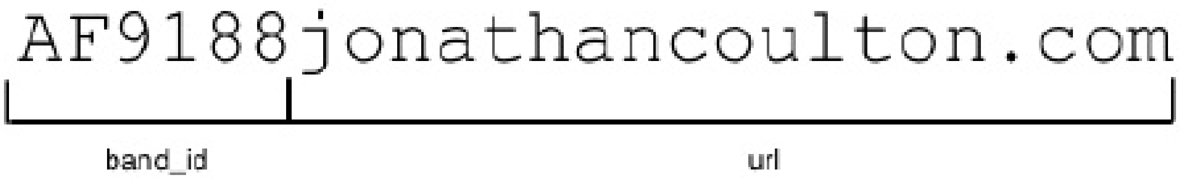
Name (Row Key)	Personal Information		Professional Information
	Age	Gender	Company
K	22	F	
M	35	[redacted]	Ve 

Name	City	State	Country
Facebook	SFO	California	USA

Name	City	State	Country
Verizon	London		United Kingdom

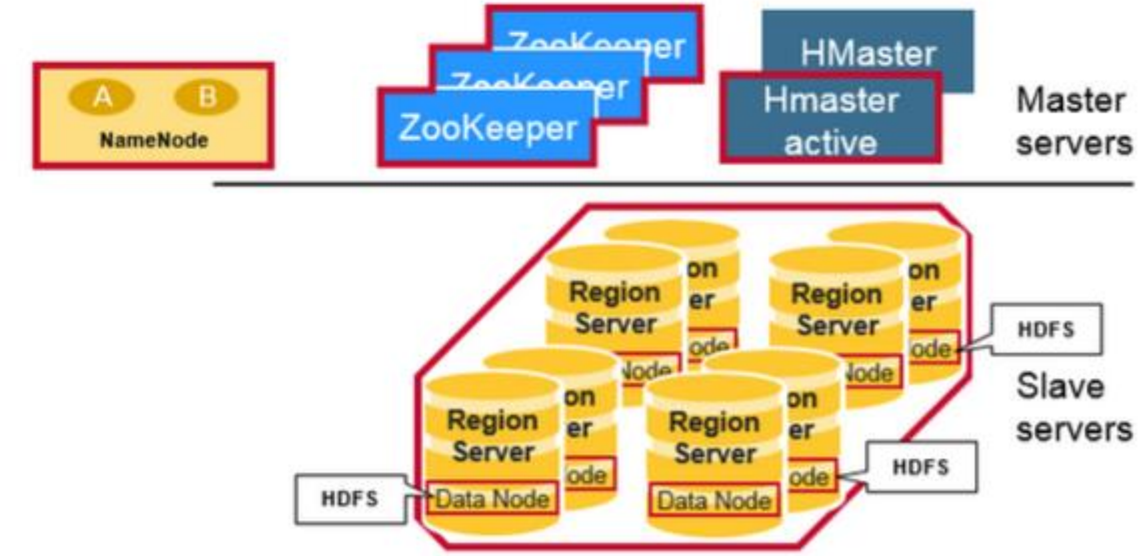
Choosing Row KEY

- Row keys are Strings
- Row keys are kept in strict lexicographic order.
- System is huge and distributed, this sorting feature is critical.
- For example, consider a table whose keys are domain names. It makes the most sense to list them in reverse notation (so "com.jimbojw.www" rather than "www.jimbojw.com") so that rows about a subdomain will be near the parent domain row
- “Com



http://jimbojw.com/wiki/index.php?title=Understanding_HBase_and_BigTable
http://www.slideshare.net/cloudera/5-h-base-schemahbasecon2012?qid=a9b9b2ce-6620-4085-9ee5-fd2c0bda8ca5&v=&b=&from_search=2

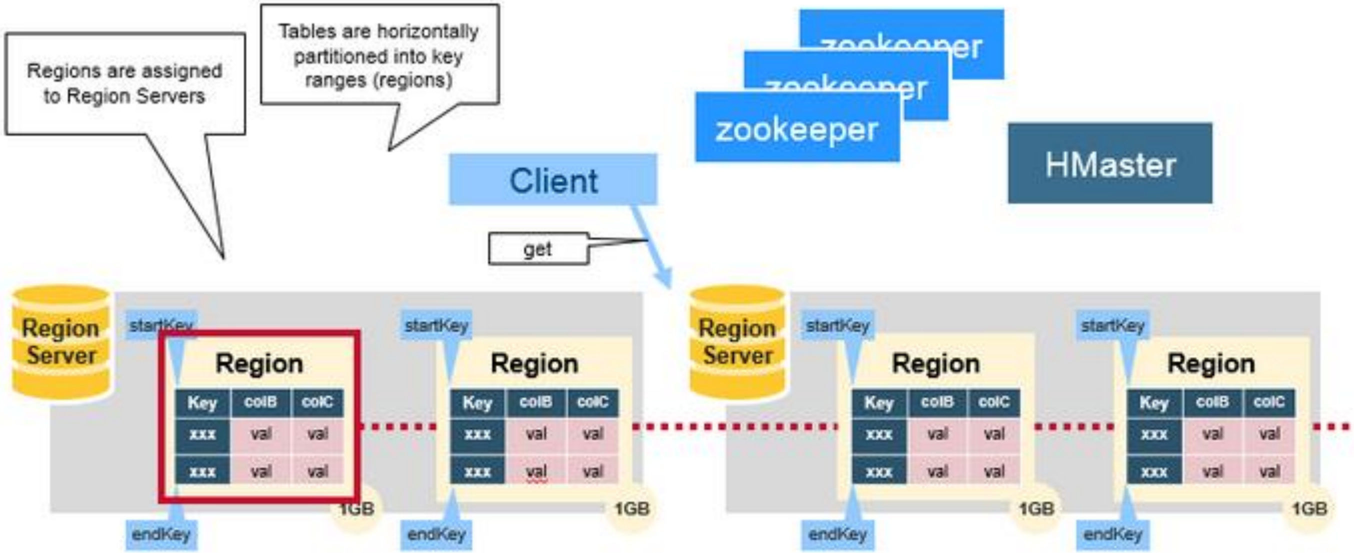
HBASE Architecture



- Region servers serve data for reads and writes
- HBase Master process: Region assignment, DDL (create, delete tables)
- Zookeeper maintains a live cluster state
- HBase data is stored in HDFS files

https://www.mapr.com/blog/in-depth-look-hbase-architecture

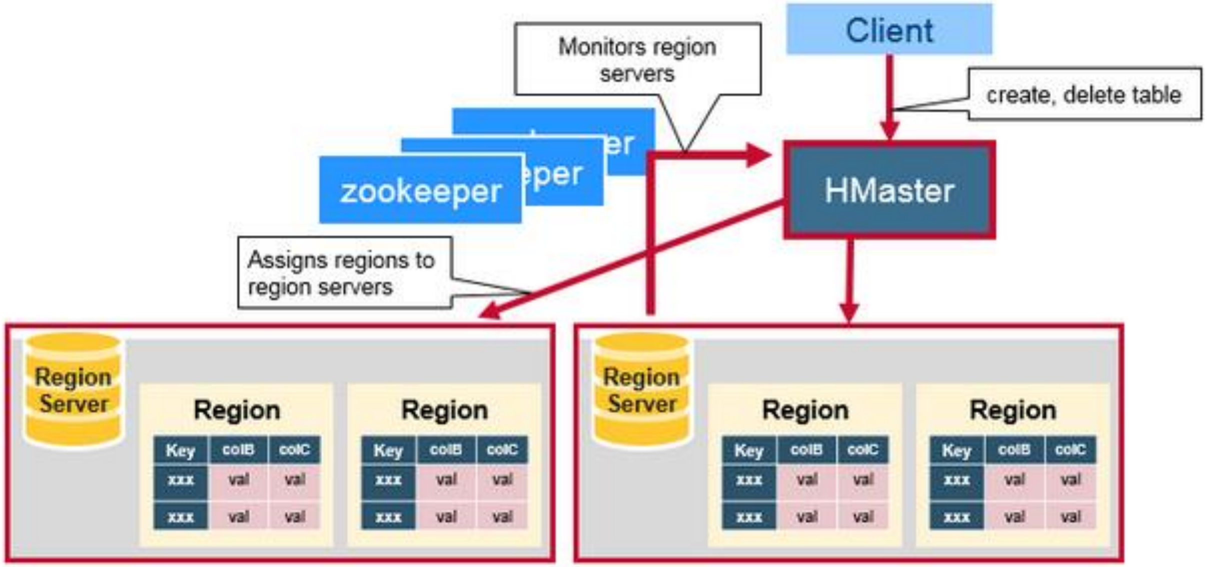
Region Server & Regions



- HBase Tables are divided horizontally by row key range into “Regions.” (1GB size)
- 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,”
- Region servers serve data for reads and writes.
- A region server can serve about 1,000 regions.
- Region = Contiguous Keys

https://www.mapr.com/blog/in-depth-look-hbase-architecture

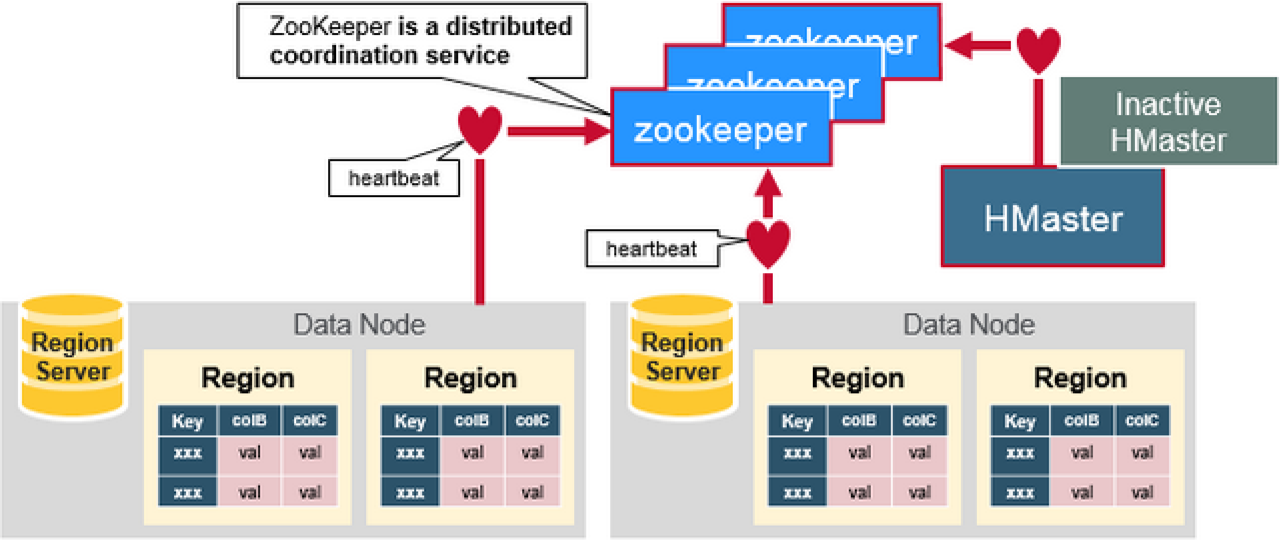
HBASE Master



- Region assignment, DDL (create, delete tables) operations
- Coordinating the region servers
- Monitoring all RegionServer instances
- Interface for creating, deleting, updating tables

https://www.mapr.com/blog/in-depth-look-hbase-architecture
https://blogs.apache.org/hbase/entry/hbase_who_needs_a_master

Zookeeper

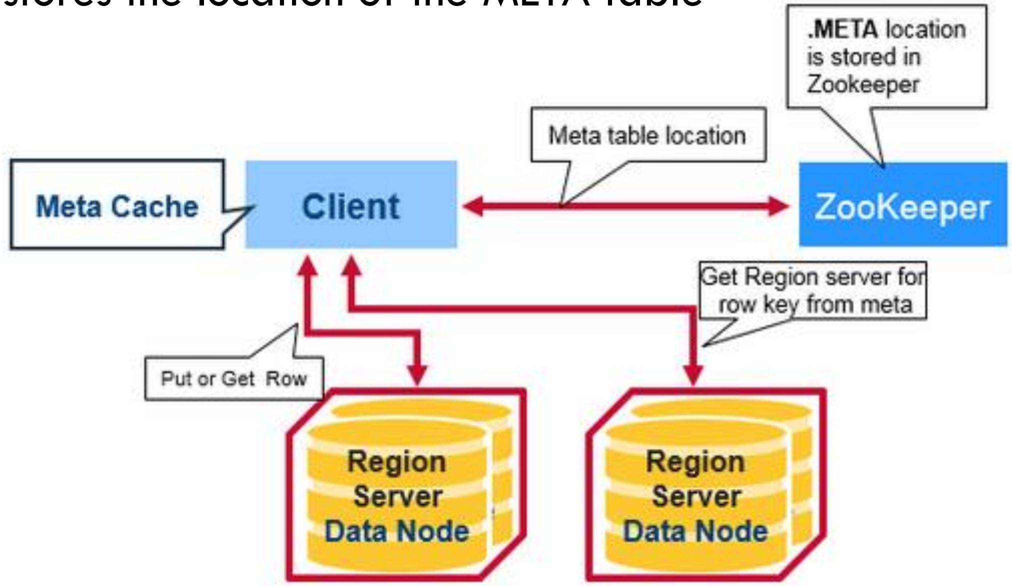


- HBase uses ZooKeeper as a distributed coordination service to maintain server state
- Zookeeper maintains which servers are alive and available
- Provides server failure notification

https://www.mapr.com/blog/in-depth-look-hbase-architecture

HBASE READ/WRITE

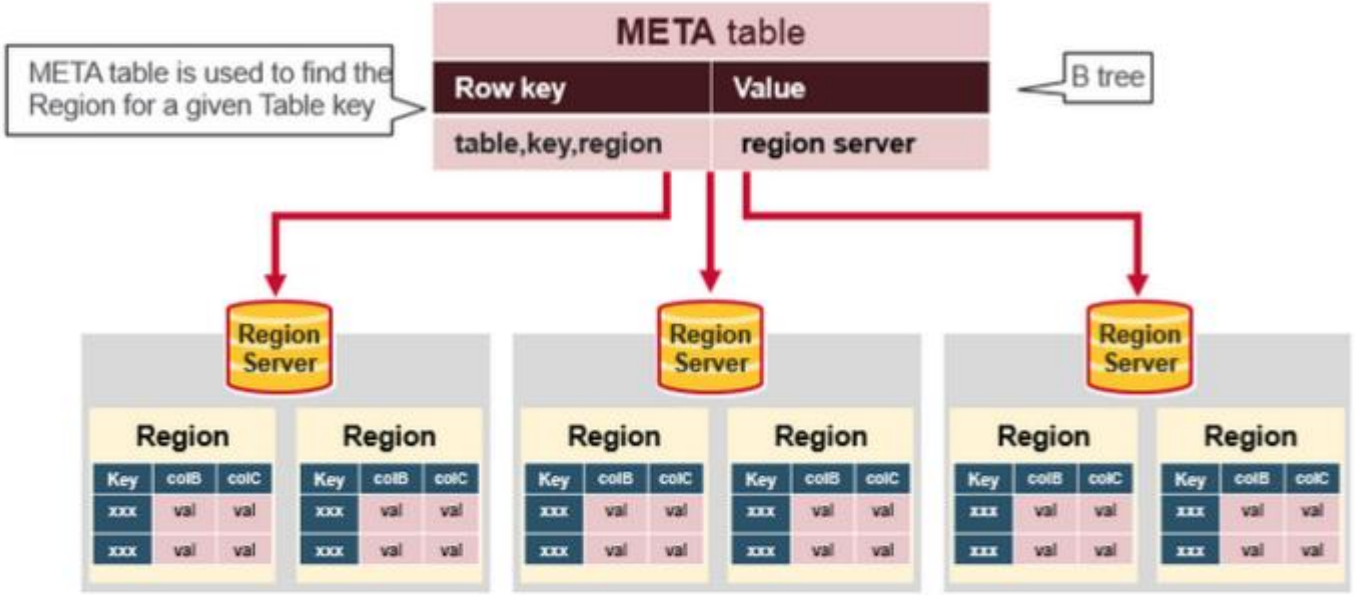
- META table holds the location of the regions in the cluster.
- ZooKeeper stores the location of the META table



- □ The client gets the Region server that hosts the META table from ZooKeeper.
- □ The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
- □ It will get the Row from the corresponding Region Server.

http://hbase.apache.org/0.94/book/arch.catalog.html
https://www.mapr.com/blog/in-depth-look-hbase-architecture

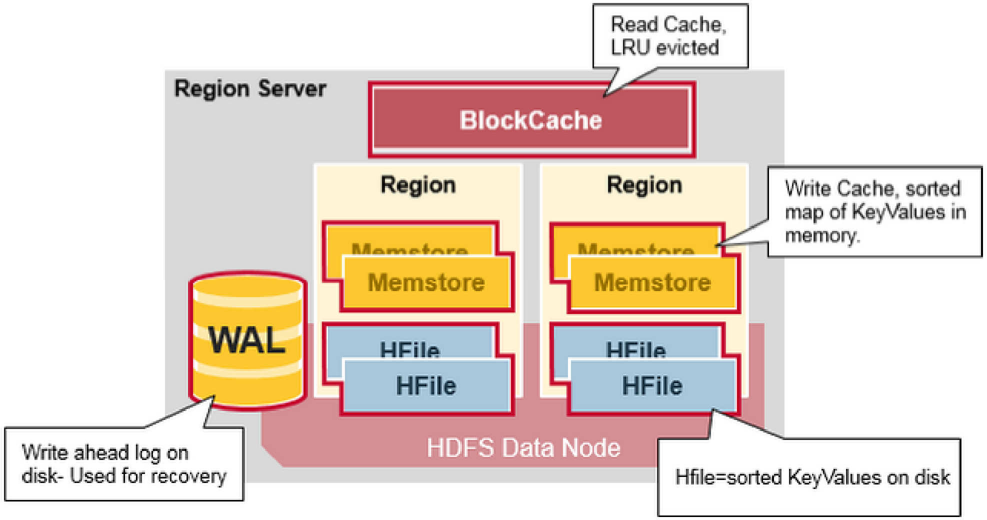
HBase Meta Table



- This META table is an HBase table that keeps a list of all regions in the system.
- The .META. table is like a b tree.
- The .META. table structure is as follows:
- - Key: region start key,region id
- - Values: RegionServer

https://www.mapr.com/blog/in-depth-look-hbase-architecture

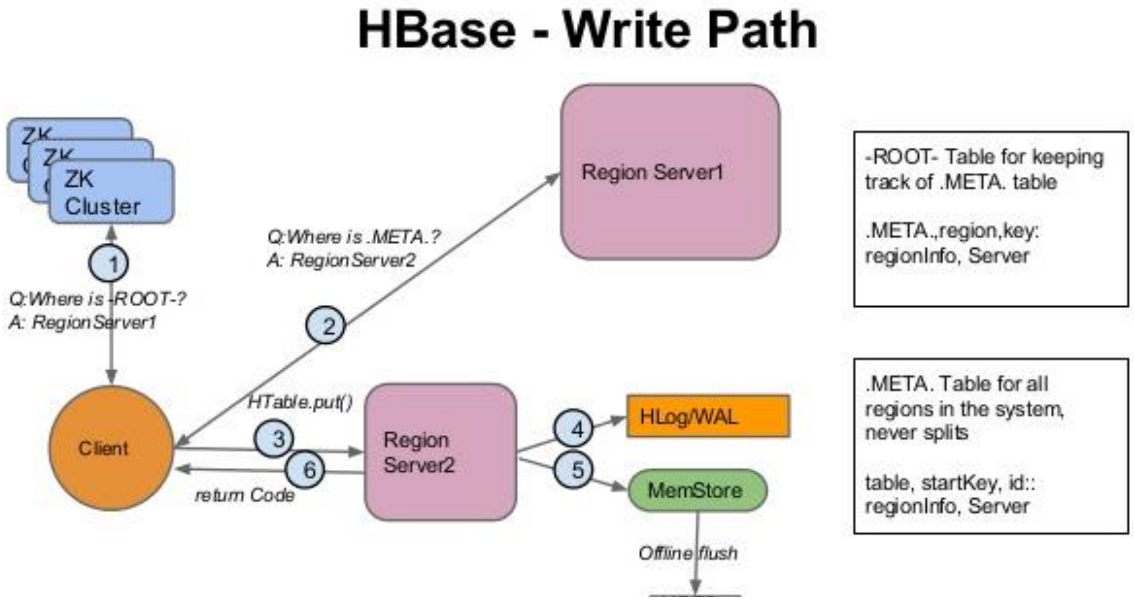
Region Server Components



- **WAL:** Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.
- **BlockCache:** is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.
- **MemStore:** is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
- Hfiles store the rows as sorted KeyValues on disk.

https://www.mapr.com/blog/in-depth-look-hbase-architecture

HBASE Write



-ROOT- Table for keeping track of .META. table

.META.,region,key: regionInfo, Server

.META. Table for all regions in the system, never splits

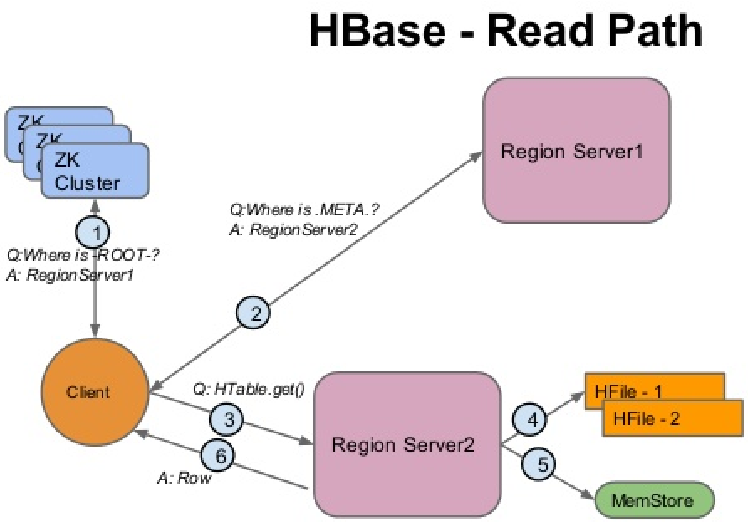
table, startKey, id:: regionInfo, Server

When the client issues a Put request,
The first step is to write the data to the write-ahead log, the WAL
Edits are appended to the end of the WAL file that is stored on disk.

- The WAL is used to recover not-yet-persisted data in case a server crashes
- Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client..
- Eventually, the data in Memstore gets flushed onto disk files called HFiles

https://www.mapr.com/blog/in-depth-look-hbase-architecture
http://www.slideshare.net/sameertiwari33/storage-for-big-data-30957881

HBASE READ

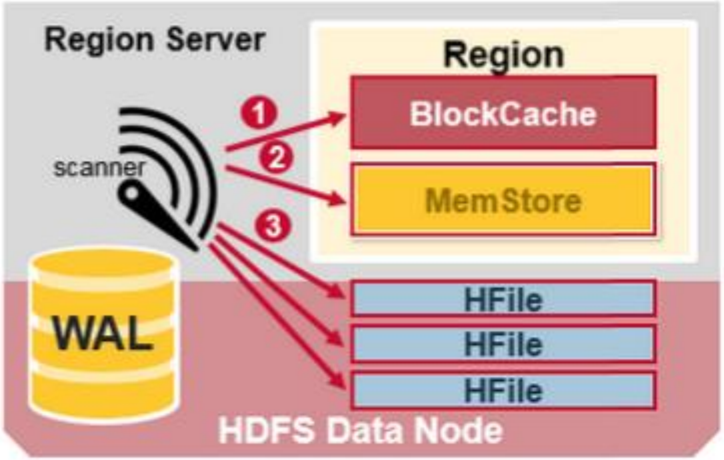


-ROOT- Table for keeping track of .META. table

.META.,region,key: regionInfo, Server

.META. Table for all regions in the system, never splits

table, startKey, id:: regionInfo, Server



- First, the scanner looks for the Row cells in the Block cache - the read cache. Recently Read Key Values are cached here, and Least Recently Used are evicted when memory is needed.
- Next, the scanner looks in the MemStore, the write cache in memory containing the most recent writes.
- If the scanner does not find all of the row cells in the MemStore and Block Cache, then HBase will use the Block Cache indexes and bloom filters to load HFiles into memory, which may contain the target row cells.

http://www.slideshare.net/sameertiwari33/storage-for-big-data-30957881

Other info

- Data Replication: Relies on HDFS
- Crash Recovery/Data Recovery: Performed using WAL
- Minor Compaction: Smaller HFiles are combined into one large Hfile.
- Major Compaction: Merges and rewrites all the HFiles in a region to one HFile per column family, and in the process, drops deleted or expired cells. Heavy process. Needs to be scheduled

HBASE COMMANDS

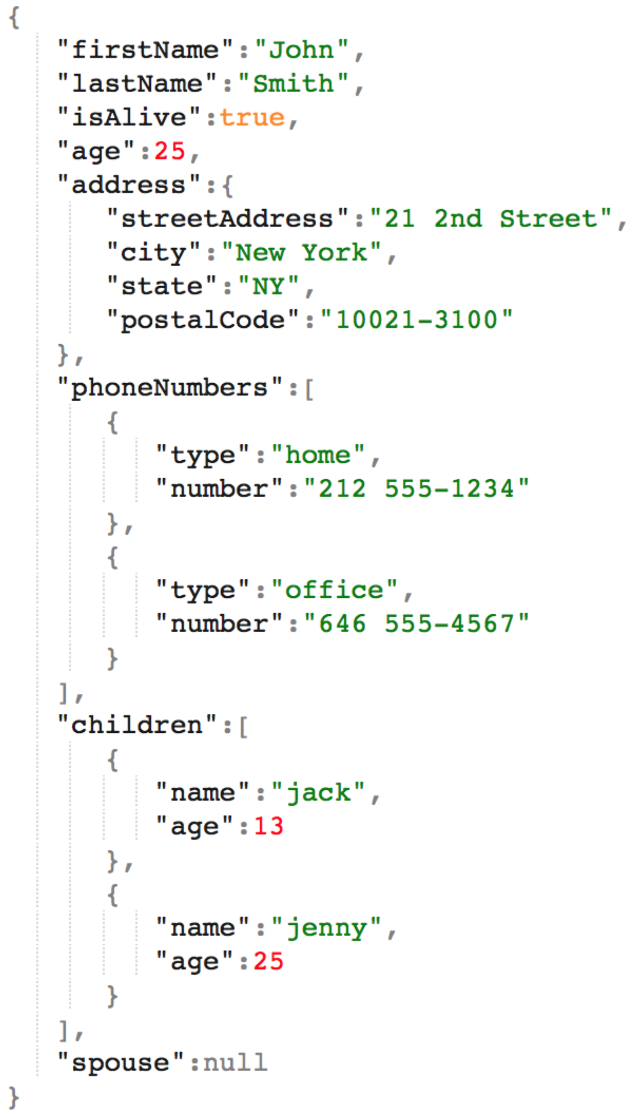
- hbase shell
- hbase> create ‘Movies’,{NAME=>’info’},{Name=>’director’}
- hbase> describe ‘Movies’
- hbase> put ‘Movies’,’1’,’info:title’,’Godfather’
- hbase> put ‘Movies’,’1’,’info:star’,’Marlon Brando’
- hbase> put ‘Movies’,’1’,’info:star’,’Al Pacino’
- hbase> put ‘Movies’,’1’,’info:type’,’Crime’
- hbase> put ‘Movies’,’1’,’info:type’,’Drama
- hbase> get ‘Movies’, ‘1’
- hbase> put ‘Movies’,’2’,’info:star’,’Samuel L. Jackson’
- hbase> scan ‘Movies’
- hbase> delete ‘Movies’,’1’,’info:star’
- hbase> disable ‘Movies’
- hbase> drop ‘Movies’

References

- http://www.cyanny.com/2014/03/13/hbase-architecture-analysis-part2-process-architecture/
- http://www.slideshare.net/sameertiwari33/storage-for-big-data-30957881
- https://www.mapr.com/blog/in-depth-look-hbase-architecture
- http://www.netwoven.com/2013/10/hbase-overview-of-architecture-and-data-model/
- http://hbase.apache.org/0.94/book/arch.catalog.html
- https://blogs.apache.org/hbase/entry/hbase_who_needs_a_master
- http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/9353-login1210_khurana.pdf

JSON

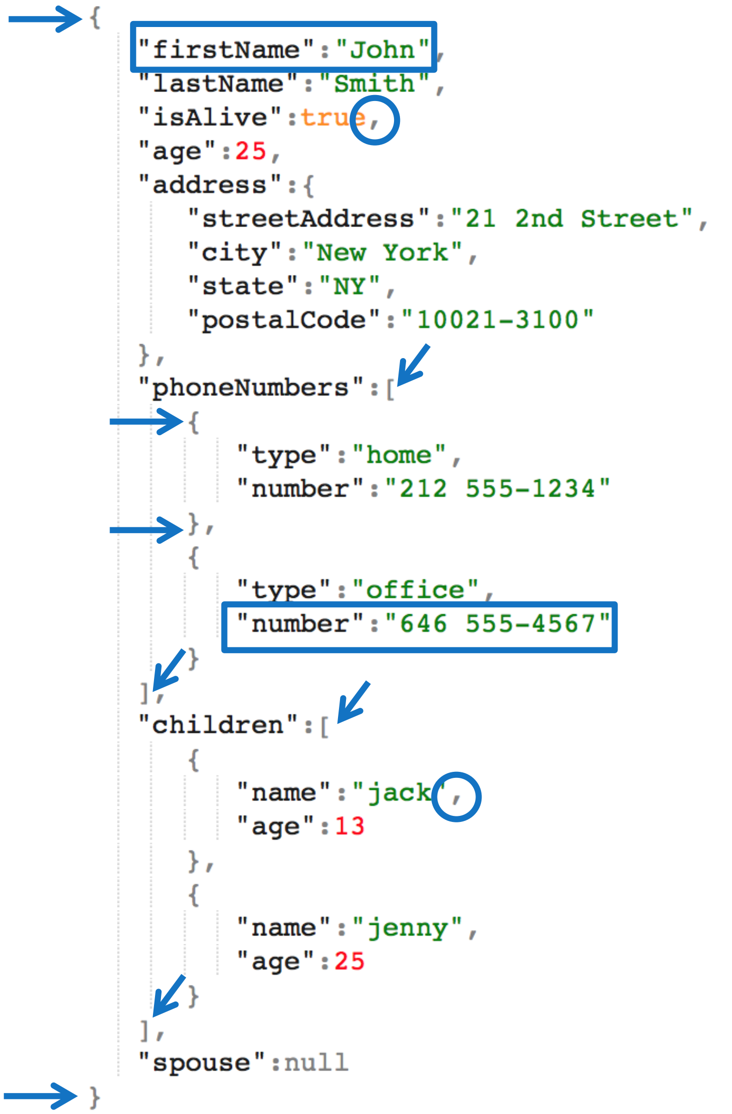
- What is JSON?
  - JSON stands for **J**ava**S**cript **O**bject **N**otation
  - JSON is a light-weight data-interchange format
  - JSON is language independent
  - JSON is self-describing and easy to understand



```
{
    "firstName":"John",
    "lastName":"Smith",
    "isAlive":true,
    "age":25,
    "address":{
        "streetAddress":"21 2nd Street",
        "city":"New York",
        "state":"NY",
        "postalCode":"10021-3100"
    },
    "phoneNumbers":[
        {
            "type":"home",
            "number":"212 555-1234"
        },
        {
            "type":"office",
            "number":"646 555-4567"
        }
    ],
    "children":[
        {
            "name":"jack",
            "age":13
        },
        {
            "name":"jenny",
            "age":25
        }
    ],
    "spouse":null
}
```

JSON Syntax Rules

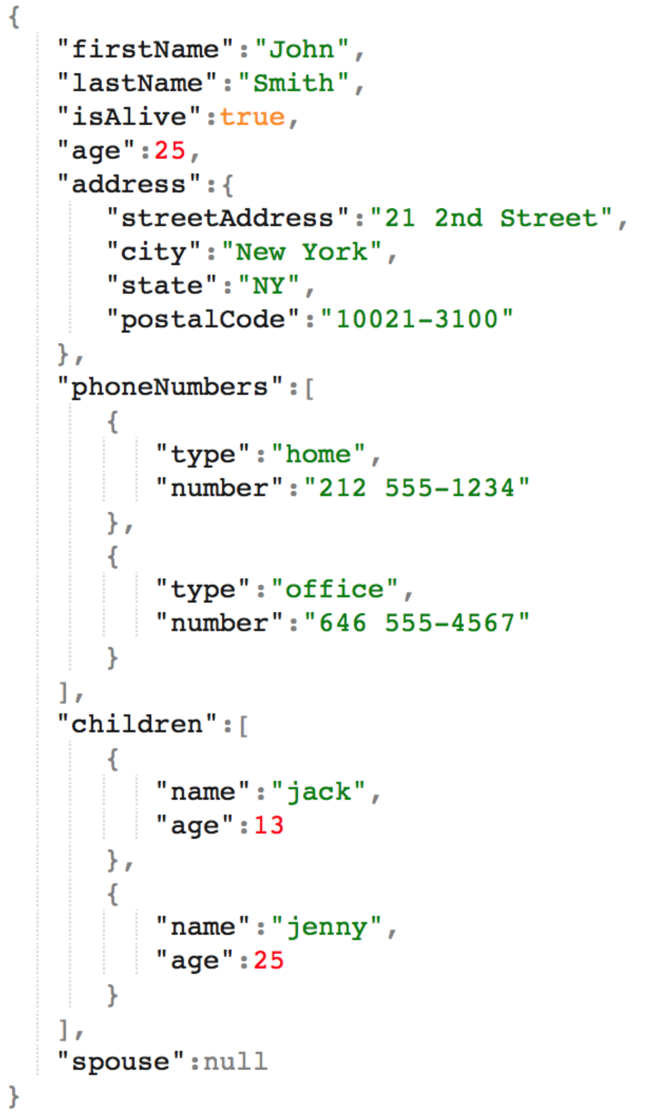
- Syntax
  - Curly braces ‘{ }’hold objects
  - Data is in name/value pairs
    - A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value
  - Data is separated by commas ‘,’
  - Square brackets ‘[ ]’hold arrays



```
{
   "firstName":"John",
   "lastName":"Smith",
   "isAlive":true,
   "age":25,
   "address":{
      "streetAddress":"21 2nd Street",
      "city":"New York",
      "state":"NY",
      "postalCode":"10021-3100"
   },
   "phoneNumbers":[
      {
         "type":"home",
         "number":"212 555-1234"
      },
      {
         "type":"office",
         "number":"646 555-4567"
      }
   ],
   "children":[
      {
         "name":"jack",
         "age":13
      },
      {
         "name":"jenny",
         "age":25
      }
   ],
   "spouse":null
}
```

More About JSON Syntax

- JSON **Values**
  - A number (integer or floating point)
  - A string (in double quotes)
  - A Boolean (true or false)
  - An **array** (in square brackets)
  - An **object** (in curly braces)
  - null
- JSON **Objects**
  - are enclosed inside curly braces { }
  - can contain multiple **name/value pairs**
- JSON **Arrays**
  - are enclosed inside square brackets [ ]
  - can contain multiple **objects**



```json
{
    "firstName":"John",
    "lastName":"Smith",
    "isAlive":true,
    "age":25,
    "address":{
        "streetAddress":"21 2nd Street",
        "city":"New York",
        "state":"NY",
        "postalCode":"10021-3100"
    },
    "phoneNumbers":[
        {
            "type":"home",
            "number":"212 555-1234"
        },
        {
            "type":"office",
            "number":"646 555-4567"
        }
    ],
    "children":[
        {
            "name":"jack",
            "age":13
        },
        {
            "name":"jenny",
            "age":25
        }
    ],
    "spouse":null
}
```

JSON Object vs Pig Tuple

Pig tuple ( )

Pig schema

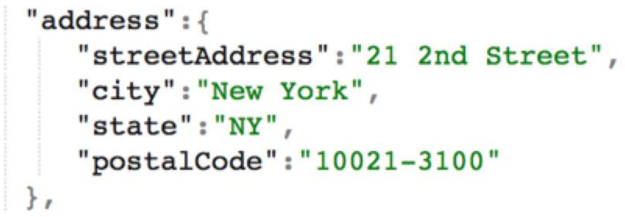
(streetAddress: chararray,
city: chararray,
state: chararray
postalCode: chararray)

Pig tuple

(21 2nd Street, New York, NY, 10021-3100)

JSON object { }

JSON object



```
"address":{
   "streetAddress":"21 2nd Street",
   "city":"New York",
   "state":"NY",
   "postalCode":"10021-3100"
},
```

JSON Arrays vs Pig Bag

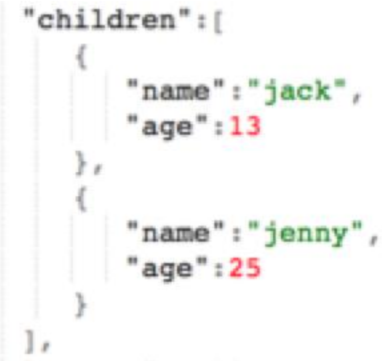
Pig bag { ( ), ( ), ( ) }

Pig schema bag: {tuple: (name: chararray, age:int)}

Pig tuple { (jack, 13), (jenny, 25) }

JSON array of objects

JSON
array of objects



```
"children":[
    {
        "name":"jack",
        "age":13
    },
    {
        "name":"jenny",
        "age":25
    }
],
```

MONGO DB

- Document oriented database
- Expressive query language and secondary indexes
- NOT ON HDFS But Sharding
- But Master-Slave Model – Like MYSQL
- No Foreign Keys Constraints
- No Join
- Map-Reduce –like pipeline for aggregation

http://thejackalofjavascript.com/mapreduce-in-mongodb/

https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu

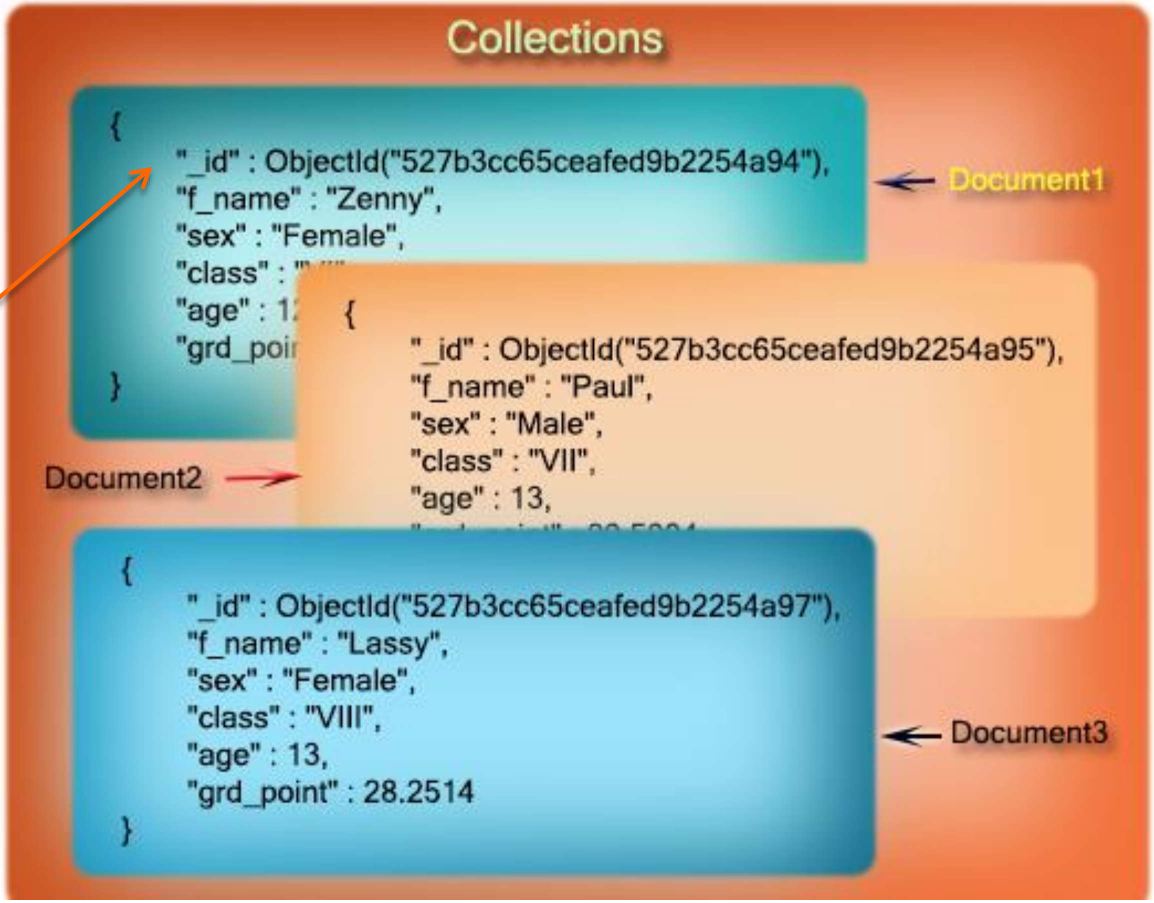
Document Oriented Database

- All data is treated in JSON
- Records do not need to have a uniform structure, i.e. different records may have different columns.
- The types of the values of individual columns can be different for each record.
- Columns can have more than one value (arrays).
- Records can have a nested structure

http://db-engines.com/en/article/Document+Stores

MONGO DB Data Model

Unique “primary” key



```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a94"),
    "f_name" : "Zenny",
    "sex" : "Female",
    "class" : "
    "age" : 1
    "grd_poi
}
```

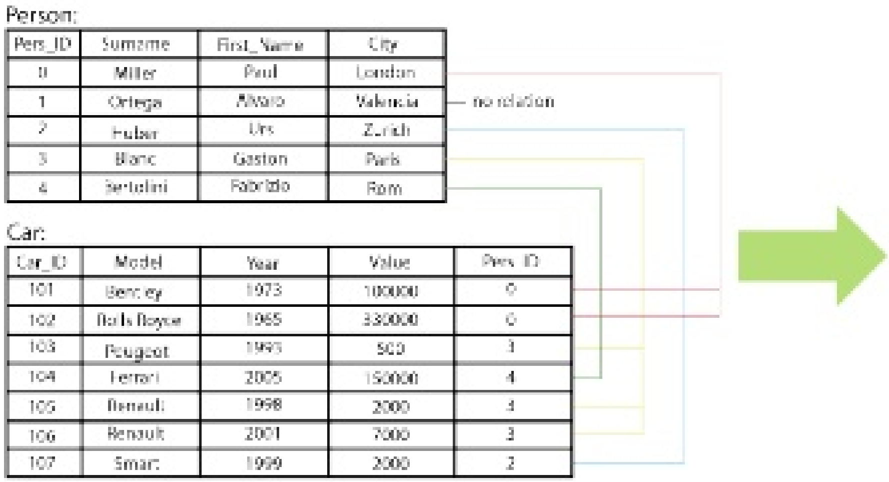
```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a95"),
    "f_name" : "Paul",
    "sex" : "Male",
    "class" : "VII",
    "age" : 13,
```

```
{
    "_id" : ObjectId("527b3cc65ceafed9b2254a97"),
    "f_name" : "Lassy",
    "sex" : "Female",
    "class" : "VIII",
    "age" : 13,
    "grd_point" : 28.2514
}
```

http://www.w3resource.com/mongodb/databases-documents-collections.php

MONGO DB Data Model

Relational



Person:

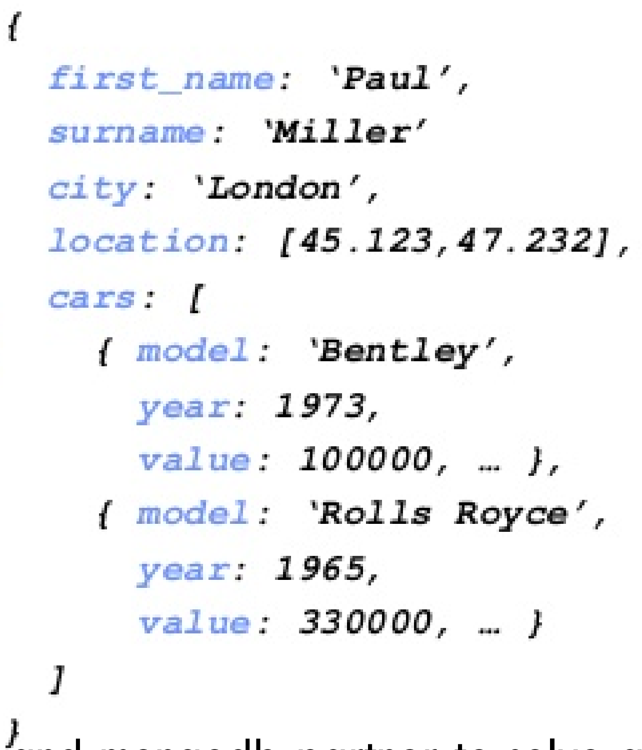
Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

no relation

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB



```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

http://www.slideshare.net/Pentaho/pentaho-and-mongodb-partner-to-solve-government-big-data-challenges

RDBMS vs. HBASE vs. MONGO

RDBMS	MONGODB	HBASE
Table	Collection	Table
Row	Document	Column Family
No Equivalent	Shard	Region
GROUP_BY	Aggregation Pipeline	MapReduce

https://www.mongodb.com/compare/mongodb-hbase

Install & Run MONGO DB

- https://www.mongodb.org/downloads#production
- tar –xvzf mongo******.tar.gz
- cd mongo******
- mkdir –p /data/db
- export LC_ALL=C
- Bin/mongo
- Show dbs
- use test;
- db.teams.save({country:"England",GroupName:"D"})

Database

Collection

Document

Tutorial: http://www.tutorialspoint.com/mongodb/index.htm