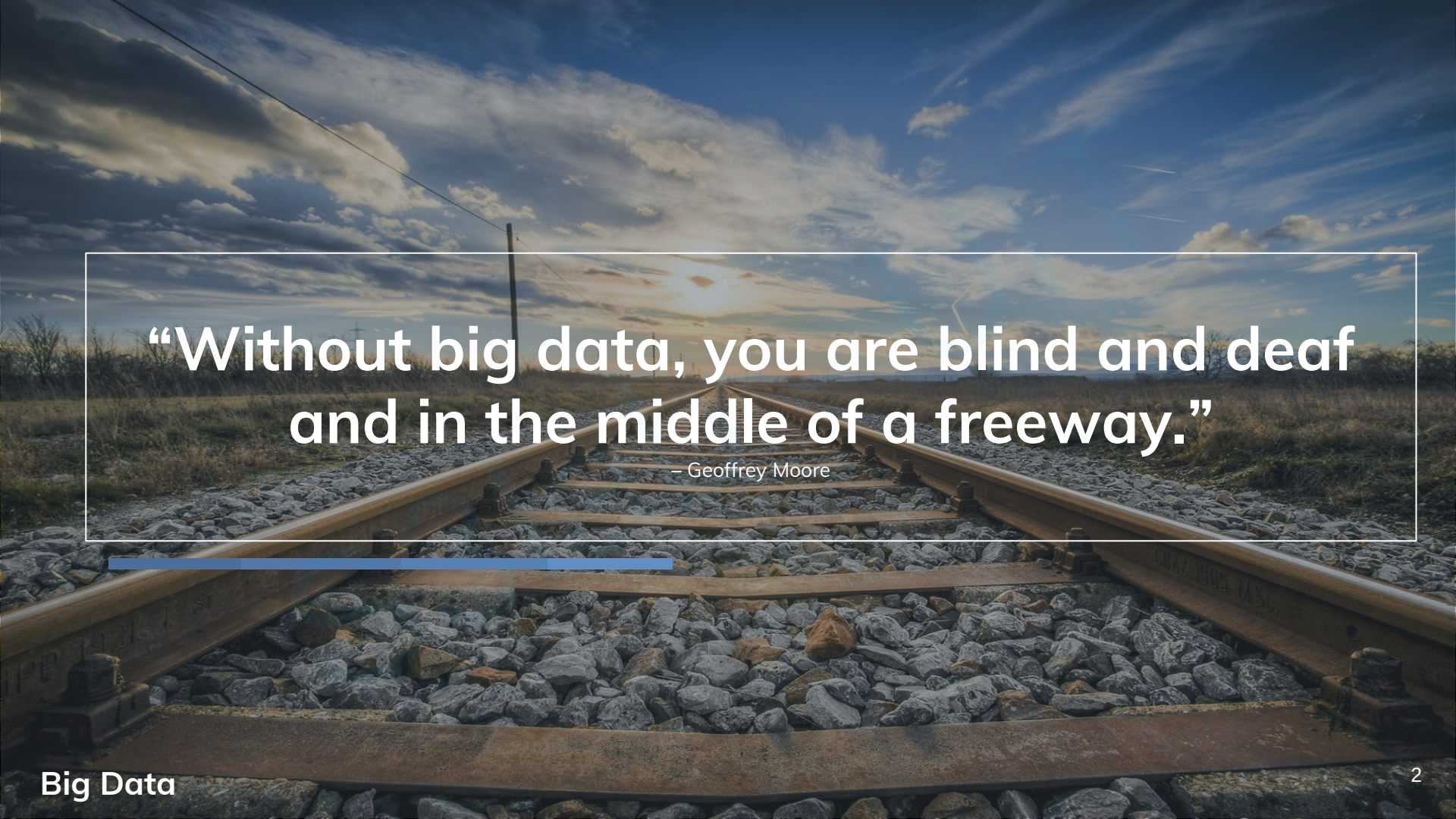
Big Data

NoSQL database and Hbase tutorial

Trong-Hop Do

S3Lab
Smart Software System Laboratory


“Without big data, you are blind and deaf and in the middle of a freeway.”
– Geoffrey Moore

NoSQL

Background

- Relational databases mainstay of business
- Web-based applications caused spikes
- Explosion of social media sites (Facebook, Twitter) with large data needs
- rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Hooking RDBMS to web-based application becomes trouble

Issues with Scaling up

- Best way to provide **ACID** and **Rich Query Model** is to have the dataset on a single machine
- Limits to scaling up (or vertical scaling: make a “single” machine more powerful) dataset is just too big!
- Scaling out (or horizontal scaling: adding more smaller / cheaper servers) is a better choice
- Different approaches for horizontal scaling (multi-node database):
  - Master / Slave
  - Sharding (partitioning)

Scaling out RDBMS

Master / Slave

- All writes are written to the master
- All reads performed against the replicated slave databases
- Critical reads may be incorrect as writes may not have been propagated down
- Large datasets can pose problems as master needs to duplicate data to slaves

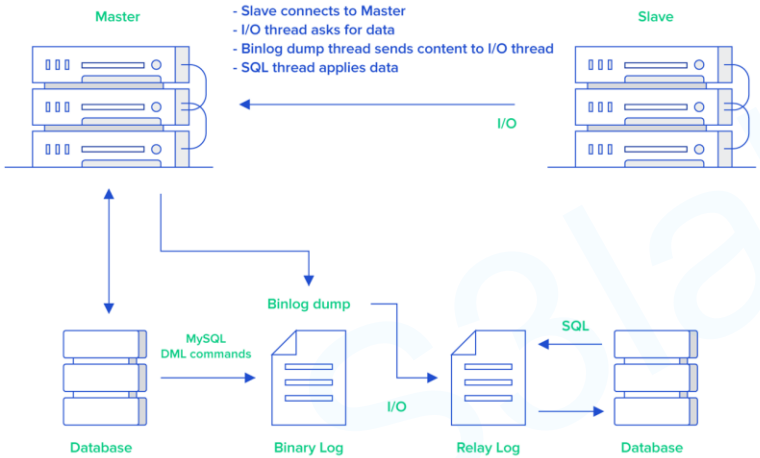
Scaling out RDBMS

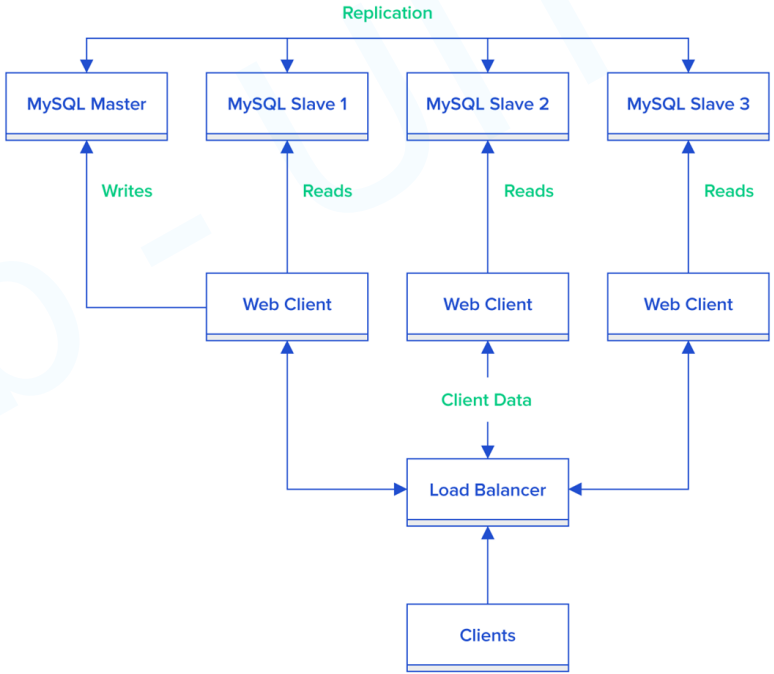
Sharding (Partitioning)

- Scales well for both reads and writes
- Not transparent, application needs to be partition-aware
- Can no longer have relationships/joins across partitions
- Loss of referential integrity across shards

Scaling out RDBMS

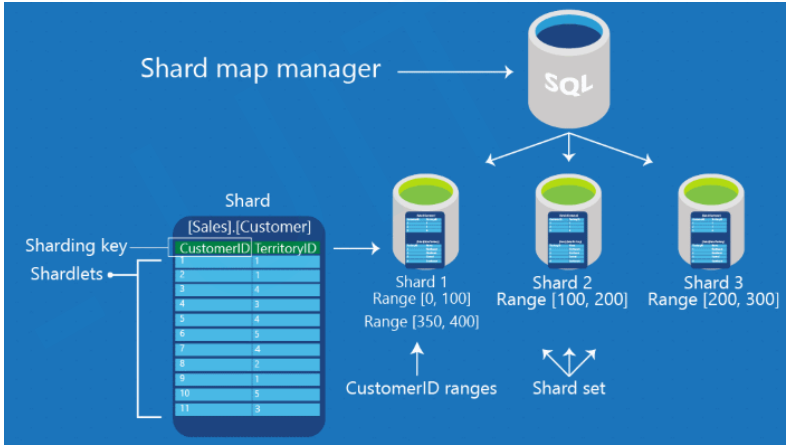
Master / Slave





Scaling out RDBMS

Sharding (Partitioning)



Original Table

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Vertical Partitions

VP1

CUSTOMER ID	FIRST NAME	LAST NAME
1	TAEKO	OHNUKI
2	O.V.	WRIGHT
3	SELDA	BAĞCAN
4	JIM	PEPPER

VP2

CUSTOMER ID	FAVORITE COLOR
1	BLUE
2	GREEN
3	PURPLE
4	AUBERGINE

Horizontal Partitions

HP1

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
1	TAEKO	OHNUKI	BLUE
2	O.V.	WRIGHT	GREEN

HP2

CUSTOMER ID	FIRST NAME	LAST NAME	FAVORITE COLOR
3	SELDA	BAĞCAN	PURPLE
4	JIM	PEPPER	AUBERGINE

Scaling out RDBMS

The other ways

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

What is NoSQL?

- This name stands for **Not Only SQL**
- The term NOSQL was introduced by Carl Strozzi in 1998 to name his file-based database
- It was again re-introduced by Eric Evans when an event was organized to discuss open source distributed databases
    - Eric states that "… but the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for. …"

What is NoSQL?

Key features (Advantages)

- non-relational
- don’t require schema
- data are replicated to multiple nodes (so, identical & fault-tolerant) and can be partitioned:
  - down nodes easily replaced
  - no single point of failure
- horizontal scalable

What is NoSQL?

Key features (Advantages)

- cheap, easy to implement (open-source)
- massive write performance
- fast key-value access

What is NoSQL?

Disadvantages

- Don’t fully support relational features
  - no join, group by, order by operations (except within partitions)
  - no referential integrity constraints across partitions
- No declarative query language (e.g., SQL) more programming
- Relaxed **ACID** (see **CAP** theorem) fewer guarantees
- No easy integration with other applications that support SQL

Who is using them?

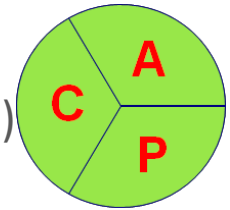
3 major papers for NoSQL

- Three major papers were the “seeds” of the NOSQL movement:
  - BigTable (Google)
  - DynamoDB (Amazon)
    - Ring partition and replication
    - Gossip protocol (discovery and error detection)
    - Distributed key-value data stores
    - Eventual consistency
  - CAP Theorem

The perfect storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a “perfect storm”
- Not a backlash against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NOSQL offerings

CAP Theorem



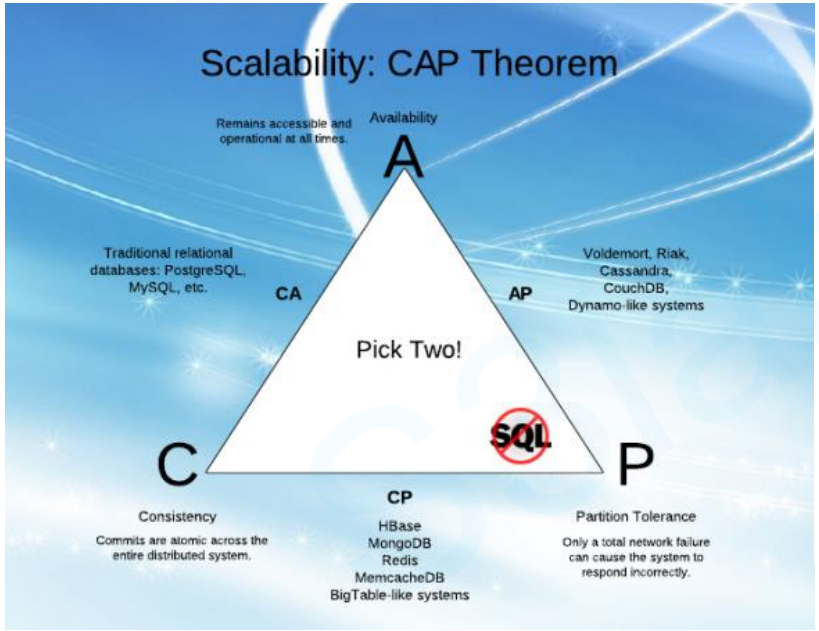
- Suppose three properties of a **distributed system** (sharing data)
  - **Consistency**:
    - Reads and writes are always executed atomically and are strictly consistent (linearizable). Put differently, all clients have the same view on the data at all times.
  - **Availability**:
    - Every non-failing node in the system can always accept read and write requests by clients and will eventually return with a meaningful response, i.e. not with an error message.
  - **Partition-tolerance**:
    - system properties (consistency and/or availability) hold even when network failures prevent some machines from communicating with others. A system can continue to operate in the presence of a network partitions

CAP Theorem

- Brewer’s CAP Theorem:
  - For any system sharing data, it is “impossible” to guarantee simultaneously all of these three properties
  - You can have at most two of these three properties for any shared-data system
- Very large systems will “partition” at some point:
  - That leaves either **C** or **A** to choose from (traditional DBMS prefers **C** over **A** and **P**)
  - In almost all cases, you would choose **A** over **C** (except in specific applications such as order processing)

CAP Theorem

Consistency



Data Model

- Key-Value
- Wide-Column
- Document
- Graph

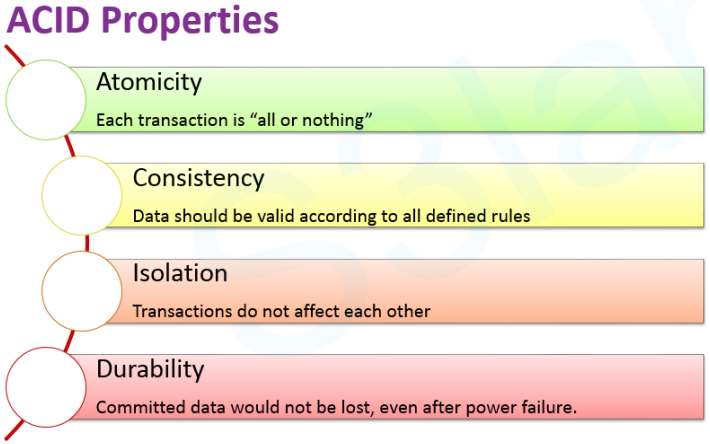
Consistency/Availability Trade-Off

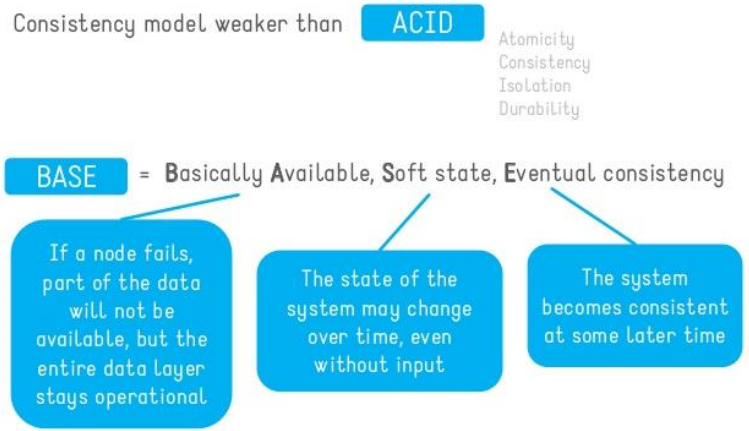
- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

CAP Theorem

Consistency

- Have 2 types of consistency:
  - Strong consistency – ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
  - Weak consistency – BASE (**B**asically **A**vailable **S**oft-state **E**ventual consistency)





CAP Theorem

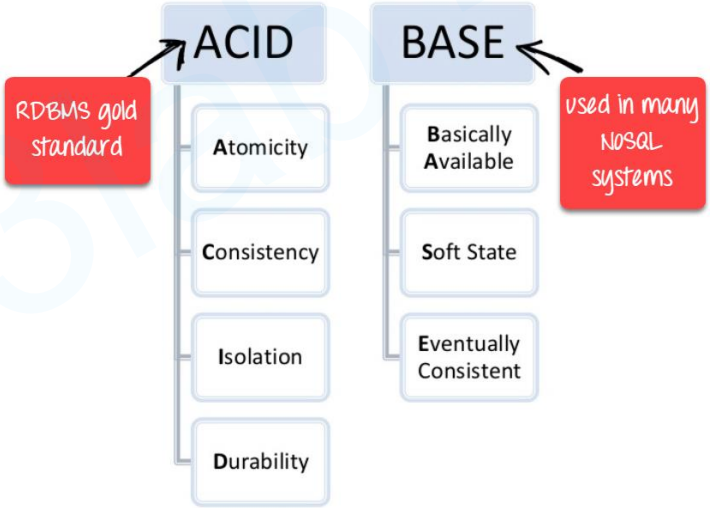
Consistency

- A consistency model determines rules for visibility and apparent order of updates
- Example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
  - **Does client B see the write from client A?**
  - Consistency is a continuum with tradeoffs
  - **For NOSQL, the answer would be: “maybe”**
  - CAP theorem states: “*strong consistency can't be achieved at the same time as availability and partition-tolerance*”

CAP Theorem

Consistency

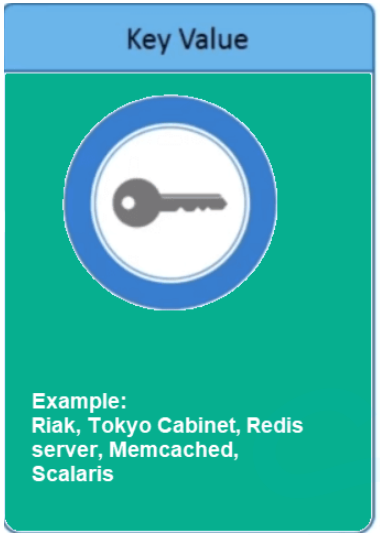
- Cloud computing
  - ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.

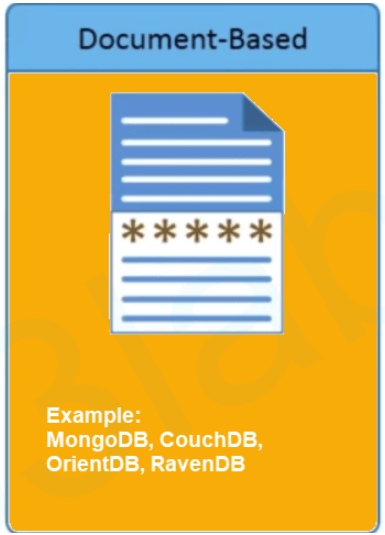


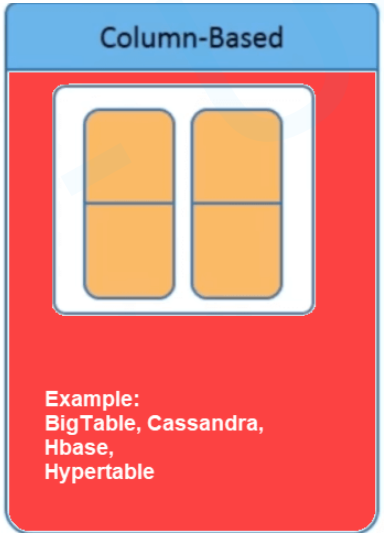
NoSQL

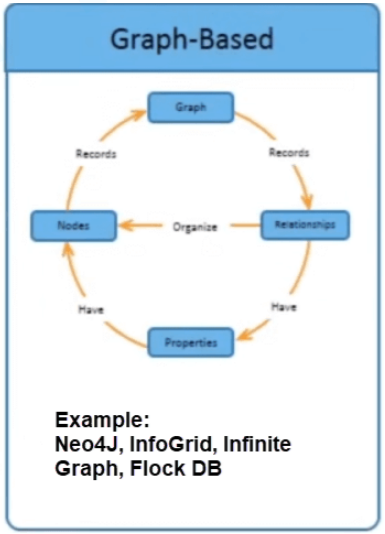
- “No-schema” is a common characteristics of most NOSQL storage systems
- Provide “flexible” data types
- Other or additional query languages than SQL
- Distributed – horizontal scaling
- Less structured data
- Supports big data

NoSQL Categories

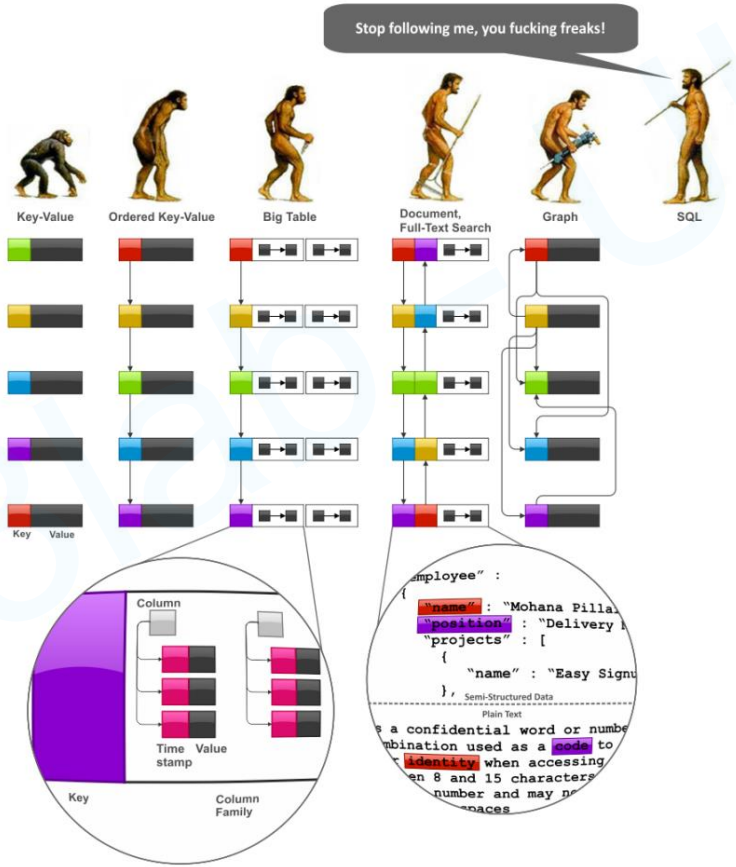








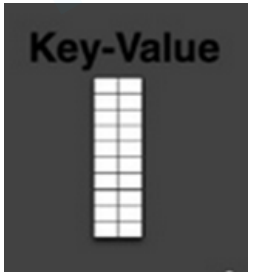
NoSQL Categories



NoSQL Categories

Key-value

- Focus on scaling to huge amounts of data
- Designed to handle massive load
- Based on Amazon's dynamo paper
- Data model: (global) collection of Key-value pairs
- Dynamo ring partitioning and replication
- Example: (DynamoDB)
  - items having one or more attributes (name, value)
  - An attribute can be single-valued or multivalued like set.
  - items are combined into a table



NoSQL Categories

Key-value

- Basic API access:
  - get(key): extract the value given a key
  - put(key, value): create or update the value given its key
  - delete(key): remove the key and its associated value
  - execute(key, operation, parameters): invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc)

NoSQL Categories

Key-value

- Pros:
  - very fast
  - very scalable (horizontally distributed to nodes based on key)
  - simple data model
  - eventual consistency
  - fault-tolerance
- Cons:
  - Can’t model more complex data structure such as objects

NoSQL Categories

Key-value

Name	Producer	Data model	Querying
SimpleDB	Amazon	set of couples (key, {attribute}), where attribute is a couple (name, value)	restricted SQL; select, delete, GetAttributes, and PutAttributes operations
Redis	Salvatore Sanfilippo	set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value	primitive operations for each value type
Dynamo	Amazon	like SimpleDB	simple get operation and put in a context
Voldemort	LinkedIn	like SimpleDB	similar to Dynamo

NoSQL Categories

Key-value

employee_id	first_name	last_name	address
1	John	Doe	New York
2	Benjamin	Button	Chicago
3	Mycroft	Holmes	London

```
employee:$employee_id:$attribute_name = $value

employee:1:first_name = "John"
employee:1:last_name = "Doe"
employee:1:address = "New York"

employee:2:first_name = "Benjamin"
employee:2:last_name = "Button"
employee:2:address = "Chicago"

employee:3:first_name = "Mycroft"
employee:3:last_name = "Holmes"
employee:3:address = "London"
```

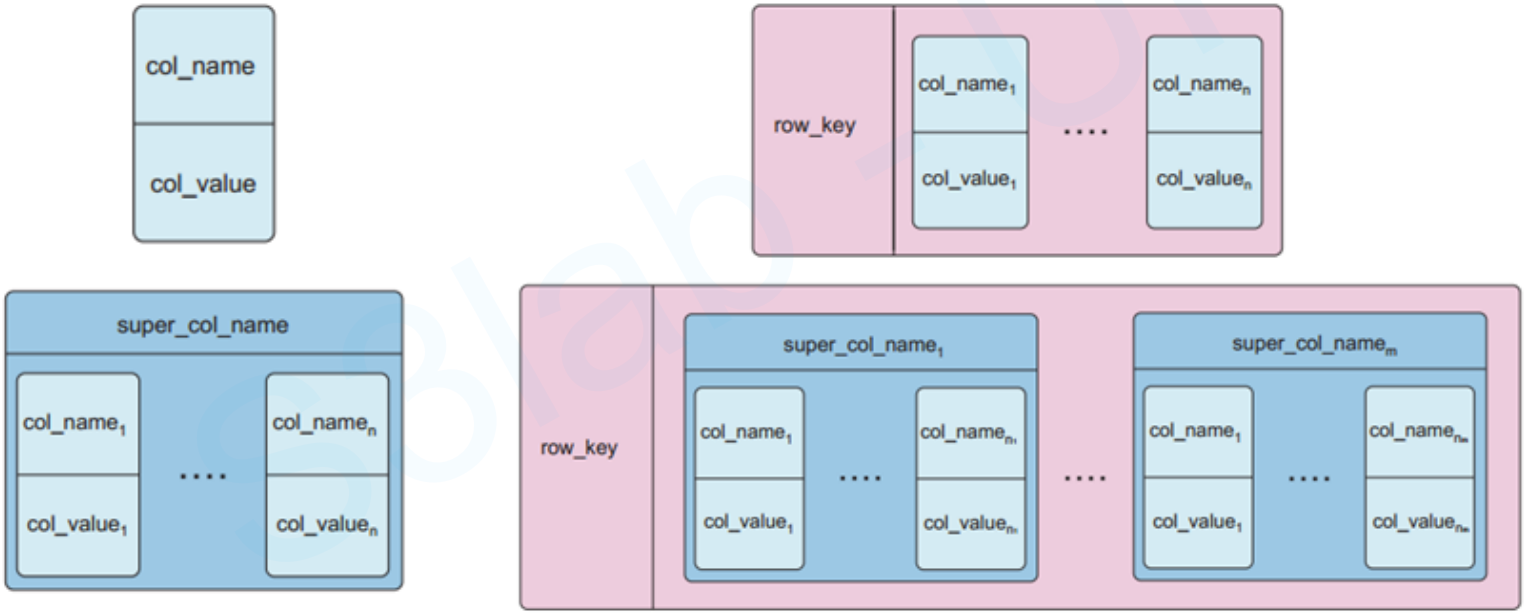
NoSQL Categories

Column-based

- Based on Google's BigTable paper
- Like column oriented relational databases (store data in column order) but with a twist
- Tables similarly to RDBMS, but handle semi-structured
- Data model:
  - Collection of Column Families
  - Column family = (key, value) where value = set of related columns (standard, super)
  - indexed by row key, column key and timestamp

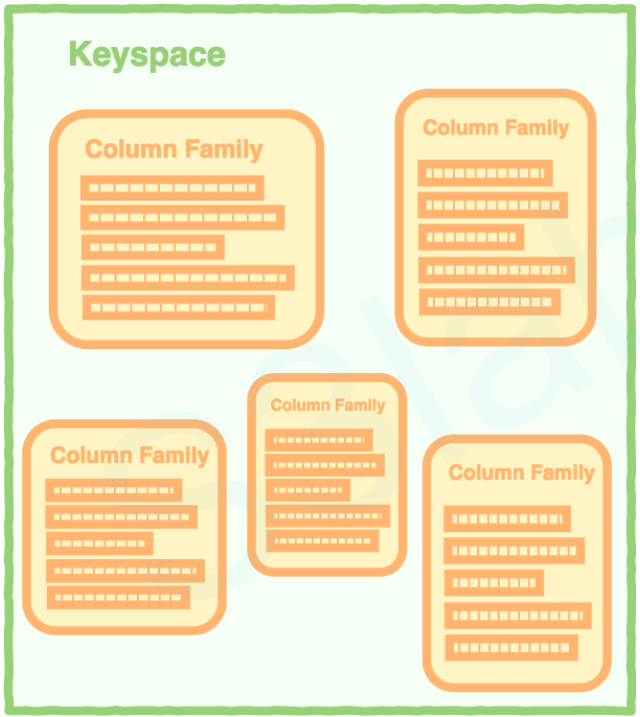
NoSQL Categories

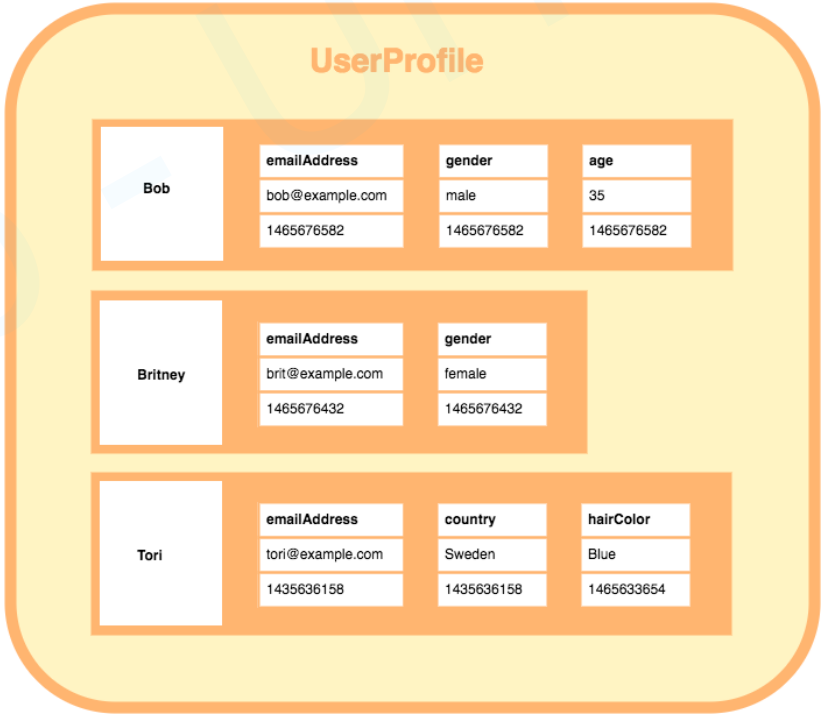
Column-based



NoSQL Categories

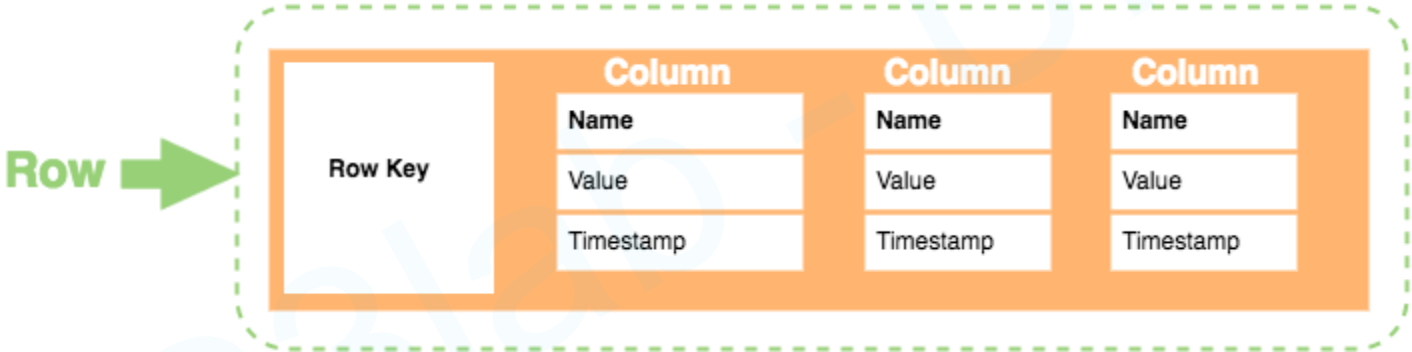
Column-based: Keyspace ~ Schema, Column Family ~ Table





NoSQL Categories

Column-based: Row structure



NoSQL Categories

Column-based

- One column family can have variable numbers of columns
- Cells within a column family are sorted “physically”
- Very sparse, most cells have null values
- Comparison: RDBMS vs column-based NOSQL
  - Query on multiple tables
    - RDBMS: must fetch data from several places on disk and glue together
    - Column-based NOSQL: only fetch column families of those columns that are required by a query (all columns in a column family are stored together on the disk, so multiple rows can be retrieved in one read operation data locality)

NoSQL Categories

Column-based

Operation	Column-Oriented Database	Row-Oriented Database
Aggregate Calulation of Single Column e.g. sum(price)	✔ fast	slow
Compression	✔ Higher. As stores similar data together	-
Retrieval of a few columns from a table with many columns	✔ Faster	has to skip over unnecessary data
Insertion/Updating of single new record	Slow	✔ Fast
Retrieval of a single record	Slow	✔ Fast

NoSQL Categories

Column-based

- Example: (Cassandra column family--timestamps removed for simplicity)

```
UserProfile = {
        Cassandra    =          {
                                                emailAddress:”casandra@apache.org” ,
age:”20”
                                   }
        TerryCho      =  {
                                                emailAddress:”terry.cho@apache.org” ,
gender:”male”
                                   }
        Cath                        =  {
                                                emailAddress:”cath@apache.org”,
                                                age:”20”,gender:”female”,address:”Seoul”
                                   }
                    }
```

NoSQL Categories

Column-based

Name	Producer	Data model	Querying
BigTable	Google	set of couples (key, {value})	selection (by combination of row, column, and time stamp ranges)
HBase	Apache	groups of columns (a BigTable clone)	JRUBY IRB-based shell (similar to SQL)
Hypertable	Hypertable	like BigTable	HQL (Hypertext Query Language)
CASSANDRA	Apache (originally Facebook)	columns, groups of columns corresponding to a key (supercolumns)	simple selections on key, range queries, column or columns ranges
PNUTS	Yahoo	(hashed or ordered) tables, typed arrays, flexible schema	selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)

NoSQL Categories

Document-based

- Can model more complex objects
- Inspired by Lotus Notes
- Data model: collection of documents
- Document: JSON (JavaScript Object Notation is a data model, key-value pairs, which supports objects, records, structs, lists, array, maps, dates, Boolean with nesting), XML, other semi-structured formats.

NoSQL Categories

Document-based

- **Example: (MongoDB) document**

{

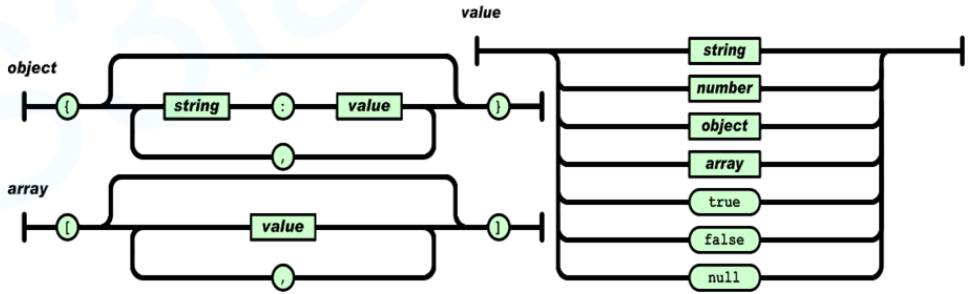
Name:"Jaroslav",

Address:"Malostranske nám. 25, 118 00 Praha 1”,

Grandchildren: {Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1“}

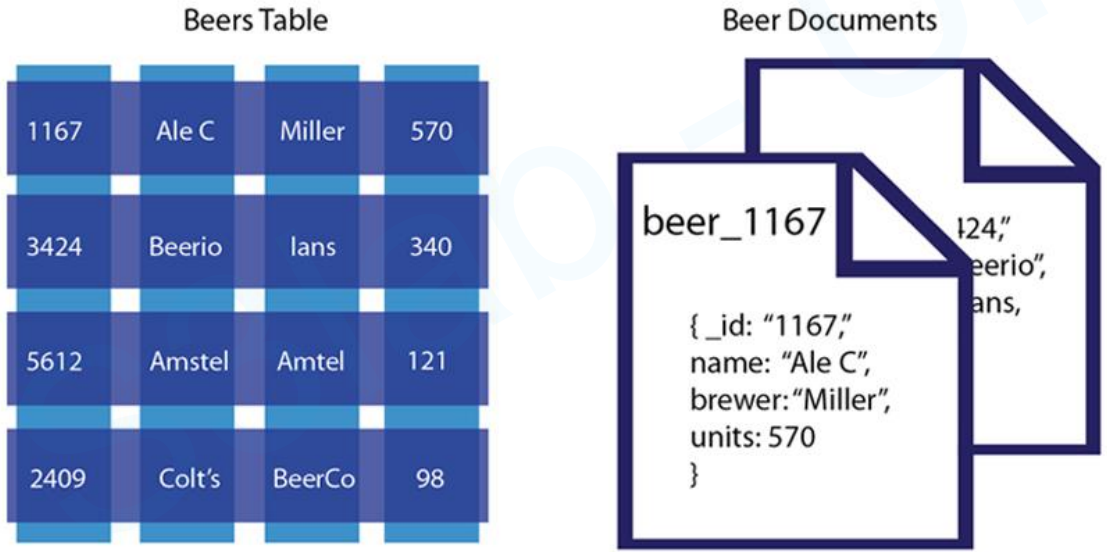
Phones: [“123-456-7890”, “234-567-8963”]

}



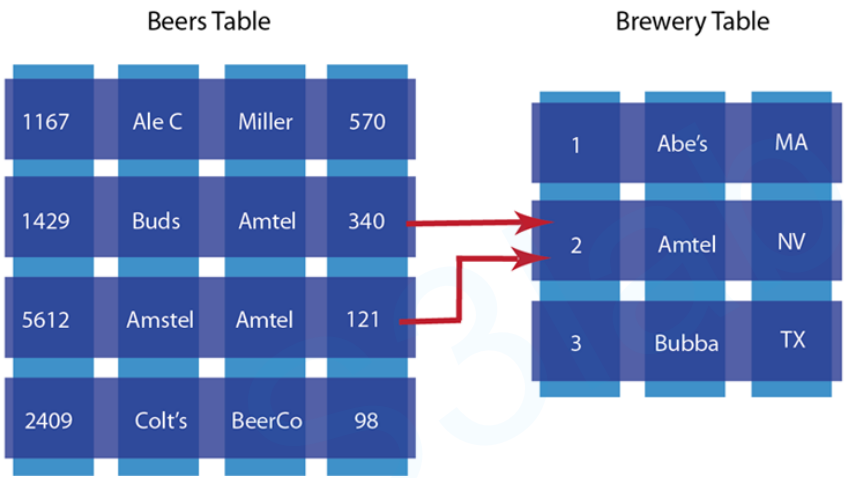
NoSQL Categories

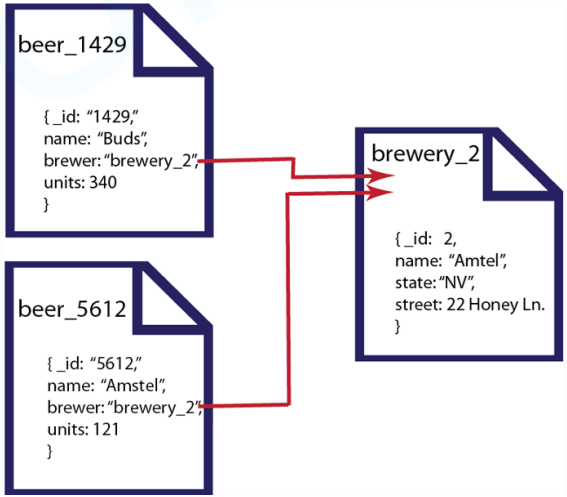
Document-based



NoSQL Categories

Document-based





NoSQL Categories

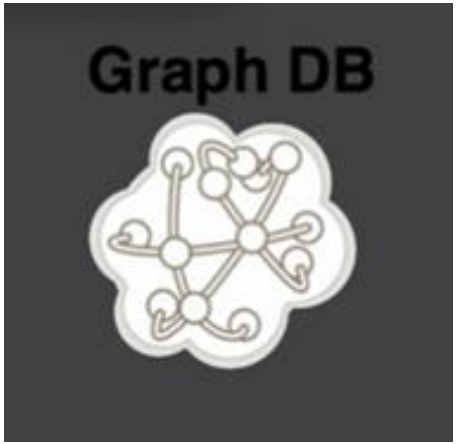
Document-based

Name	Producer	Data model	Querying
MongoDB	10gen	object-structured documents stored in collections; each object has a primary key called ObjectId	manipulations with objects in collections (find object or objects via simple selections and logical expressions, delete, update,)
Couchbase	Couchbase[1]	document as a list of named (structured) items (JSON document)	by key and key range, views via Javascript and MapReduce

NoSQL Categories

Graph-based

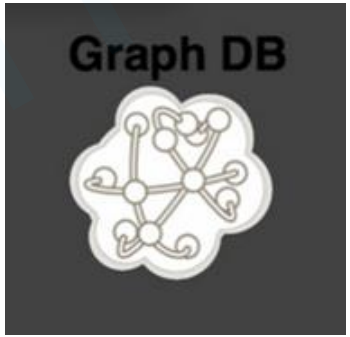
- Focus on modeling the structure of data (interconnectivity)
- A graph is composed of two elements: a node and a relationship.
- Scales to the complexity of data
- Inspired by mathematical Graph Theory (G=(E,V))
- Data model:
  - (Property Graph) nodes and edges
    - Nodes may have properties (including ID)
    - Edges may have labels or roles
  - Key-value pairs on both



NoSQL Categories

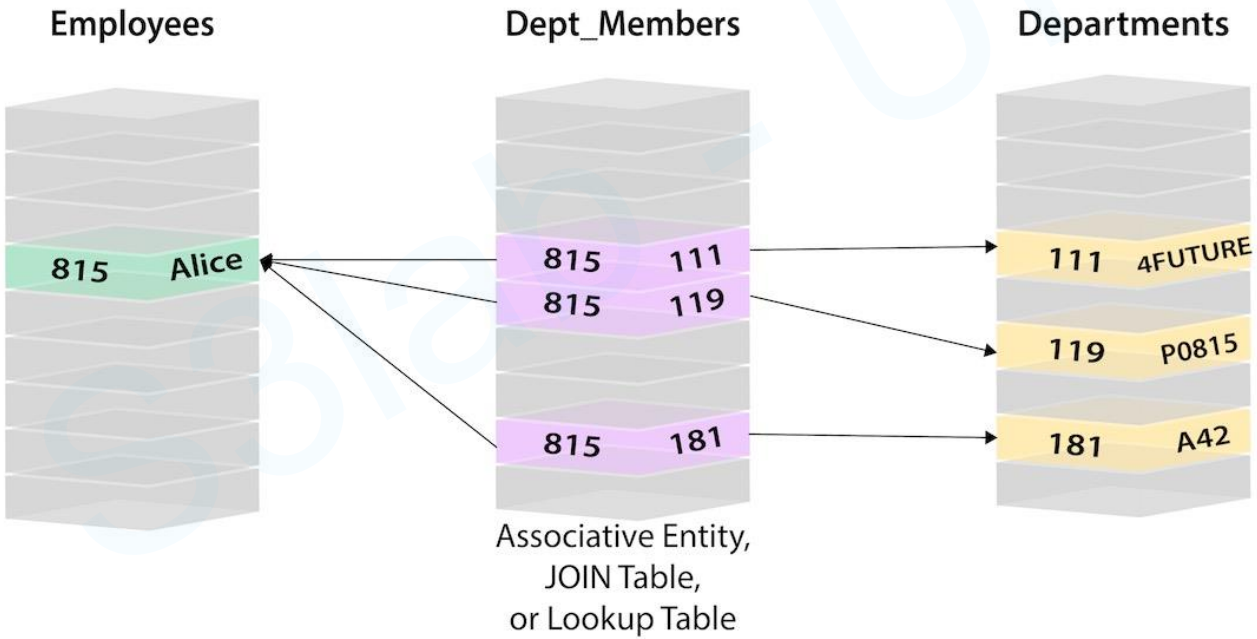
Graph-based

- Interfaces and query languages vary
- Single-step vs path expressions vs full recursion
- Example:
  - Neo4j, FlockDB, Pregel, InfoGrid …



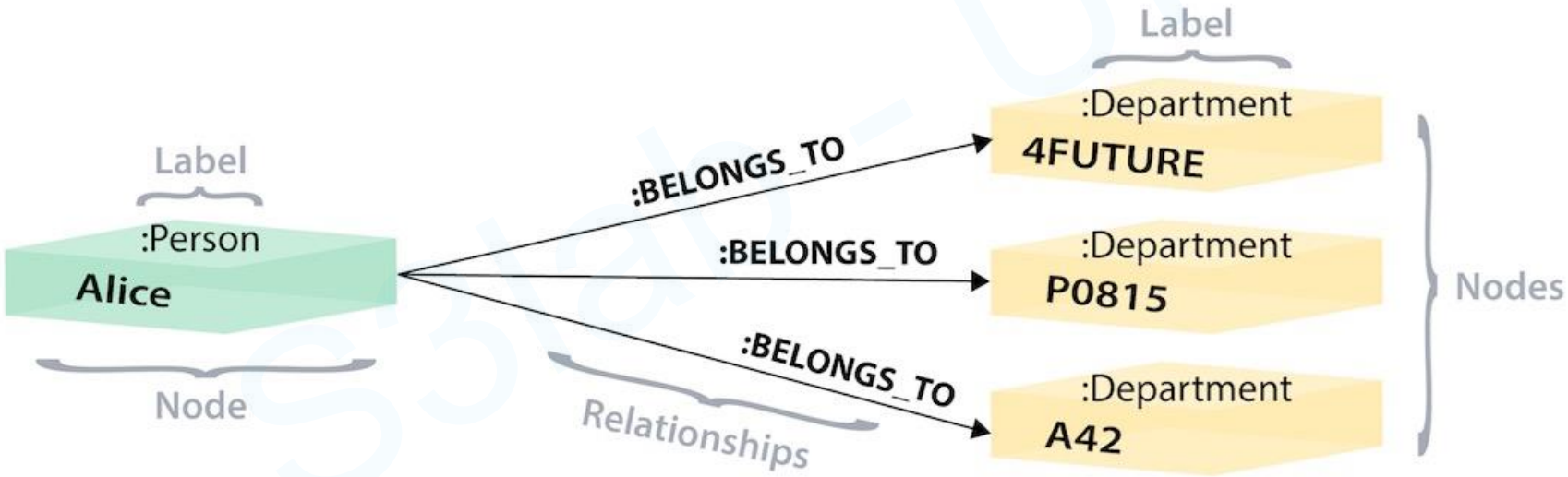
NoSQL Categories

Graph-based



NoSQL Categories

Graph-based



NoSQL Categories

Comparison

Attributes		NoSQL Databases								
Database model		Document-Stored		Wide-Column Stored				Key-Value Stored		Graph-oriented
	Features	MongoDB	CouchDB	DynamoBD	HBase	Cassandra	Accumulo	Redis	Riak	Neo4j
Design & Features	Data storage	Volatile memory File System	Volatile memory File System	SSD	HDFS		Hadoop	Volatile memory File System	Bitcask LevelDB Volatile memory	File System Volatile memory
	Query language	Volatile memory File System	JavaScript Memcached-protocol	API calls	API calls REST XML Thrift	API calls CQL Thrift		API calls	HTTP JavaScript REST Erlang	API calls REST SparQL Cypher Tinkerpop Gremlin
	Protocol	Custom, binary (BSON)	HTTP, REST	-	HTTP/REST Thrift	Thrift & custom binary CQL3	Thrift	Telnet-like	HTTP, REST	HTTP/REST Tembedding in Java
	Conditional entry updates	Yes	Yes	Yes	Yes	No	Yes	No	No	
	MapReduce	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	No
	Unicode	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	TTL for Entries	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	
	Compression	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	
Integrity	Integrity model	BASE	MVCC	ASID	Log Replication	BASE	MVCC	-	BASE	ASID
	Atomicity	Conditional	Yes	Yes	Yes	Yes	Conditional	Yes	No	Yes
	Consistency	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
	Isolation	No	Yes	Yes	No	No	-	Yes	Yes	Yes
	Durability (data storage)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes
	Transactions	No	No	No	Yes	No	Yes	Yes	No	Yes
	Referential integrity	No	No	No	No	No	No	Yes	No	Yes
	Revision control	No	Yes	Yes	Yes	No	Yes	No	Yes	No
Indexing	Secondary Indexes	Yes	Yes	No	Yes	Yes	Yes	-	Yes	-
	Composite keys	Yes	Yes	Yes	Yes	Yes	Yes	-	Yes	-
	Full text search	No	No	No	No	No	Yes	No	Yes	Yes
	Geospatial Indexes	Yes	No	No	No	No	Yes	-	-	Yes
	Graph support	No	No	No	No	No	Yes	No	Yes	Yes
Distribution	Horizontal scalable	Yes	Yes	Yes	Yes	Yes	Yes		Yes	No
	Replication	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
	Replication mode	Master-Slave-Replica Replication	Master-Slave Replication	-	Master-Slave Replication	Master-Slave Replication	-	Master-Slave Replication	Multi-master replication	-
	Sharding	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
	Shared nothing architecture	Yes	Yes	Yes	Yes	Yes	-	-	Yes	-
System	Value size max.	16MB	20MB	64KB	2TB	2GB	1EB	-	64MB	
	Operating system	Cross-platform	Ubuntu Red Hat Windows Mac OS X	Cross-platform	Cross-platform	Cross-platform	NIX 32 entries Operating system	Linux *NIX Mac OS X Windows	Cross-platform	Cross-platform
	Programming language	C++	Erlang C++ C Python	Java	Java	Java	Java	C C++	Erlang	Java

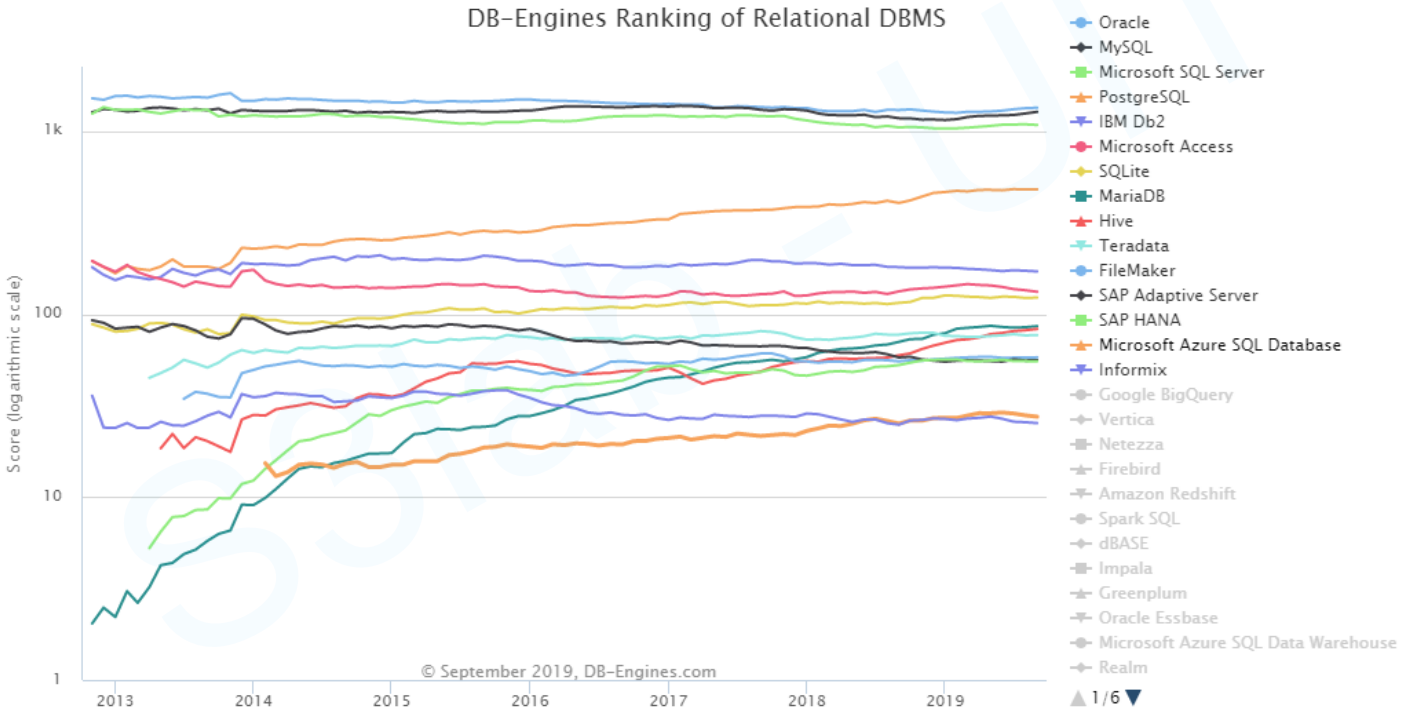
Conclusion

- NOSQL database cover only a part of data-intensive cloud applications (mainly Web applications)
- Problems with cloud computing:
  - SaaS (Software as a Service or on-demand software) applications require enterprise-level functionality, including ACID transactions, security, and other features associated with commercial RDBMS technology, i.e. NOSQL should not be the only option in the cloud
  - Hybrid solutions:
    - Voldemort with MySQL as one of storage backend
    - deal with NOSQL data as semi-structured data
      -> integrating RDBMS and NOSQL via SQL/XML

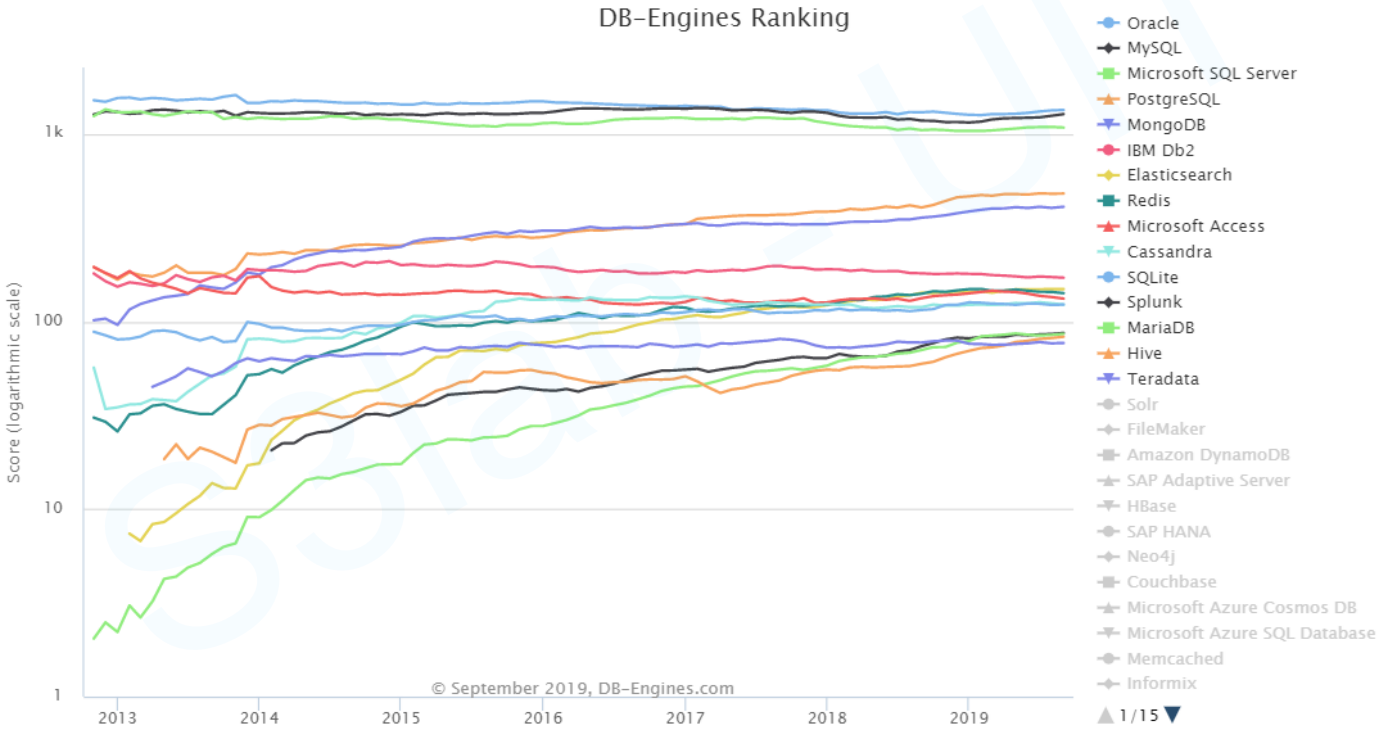
Conclusion

- next generation of highly scalable and elastic RDBMS: NewSQL databases (from April 2011)
  - they are designed to scale out horizontally on shared nothing machines,
  - still provide ACID guarantees,
  - applications interact with the database primarily using SQL,
  - the system employs a lock-free concurrency control scheme to avoid user shut down,
  - the system provides higher performance than available from the traditional systems.
- Examples: MySQL Cluster (most mature solution), VoltDB, Clustrix, ScalArc, etc.

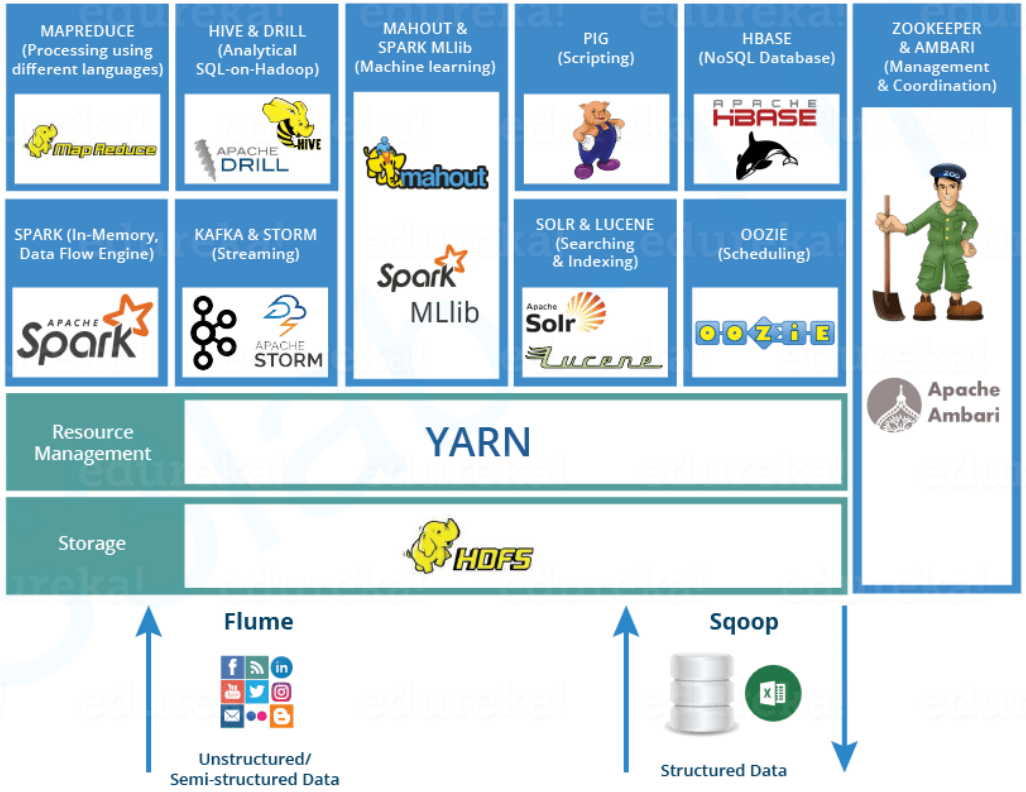
Conclusion


DB-Engines Ranking of Relational DBMS
Score (logarithmic scale)
1k
100
10
1
2013
2014
2015
2016
2017
2018
2019
© September 2019, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
IBM Db2
Microsoft Access
SQLite
MariaDB
Hive
Teradata
FileMaker
SAP Adaptive Server
SAP HANA
Microsoft Azure SQL Database
Informix
Google BigQuery
Vertica
Netezza
Firebird
Amazon Redshift
Spark SQL
dBASE
Impala
Greenplum
Oracle Essbase
Microsoft Azure SQL Data Warehouse
Realm
1/6

Conclusion


DB-Engines Ranking
Score (logarithmic scale)
1k
100
10
1
2013
2014
2015
2016
2017
2018
2019
© September 2019, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
IBM Db2
Elasticsearch
Redis
Microsoft Access
Cassandra
SQLite
Splunk
MariaDB
Hive
Teradata
Solr
FileMaker
Amazon DynamoDB
SAP Adaptive Server
HBase
SAP HANA
Neo4j
Couchbase
Microsoft Azure Cosmos DB
Microsoft Azure SQL Database
Memcached
Informix
1/15

Hadoop Ecosystem



HBase tutorial

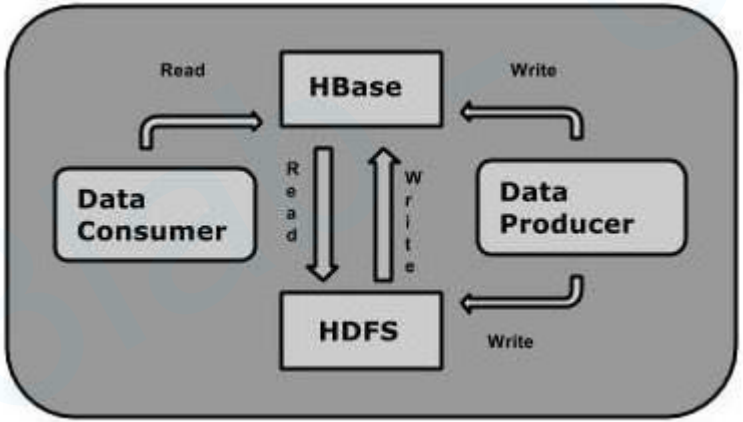
Hbase tutorial

What is HBase?

- HBase is a distributed column-oriented database built on top of the Hadoop file system.
- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).
- It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.
- One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access

Hbase tutorial

What is HBase?



Hbase tutorial

HDFS vs HBase

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

Hbase tutorial

What is HBase?

- HBase is a column-oriented database and the tables in it are sorted by row. The table schema defines only column families, which are the key value pairs.

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

Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Hbase tutorial

Column Oriented and Row Oriented

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

Hbase tutorial

HBase and RDBMS

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

Hbase tutorial

Features of HBase

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

Hbase tutorial

Where to Use HBase

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

Hbase architecture

Components of HBase Architecture

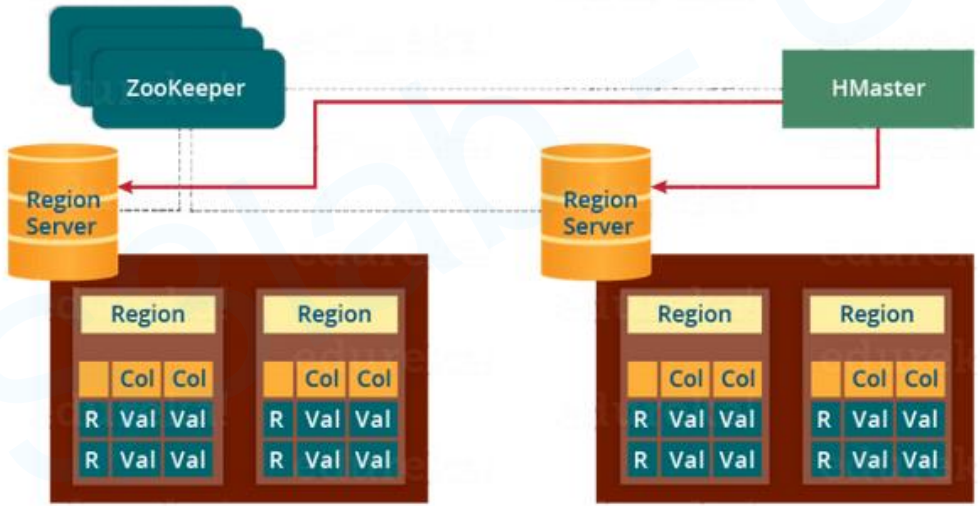


Figure: Components of HBase

Hbase architecture

HMaster

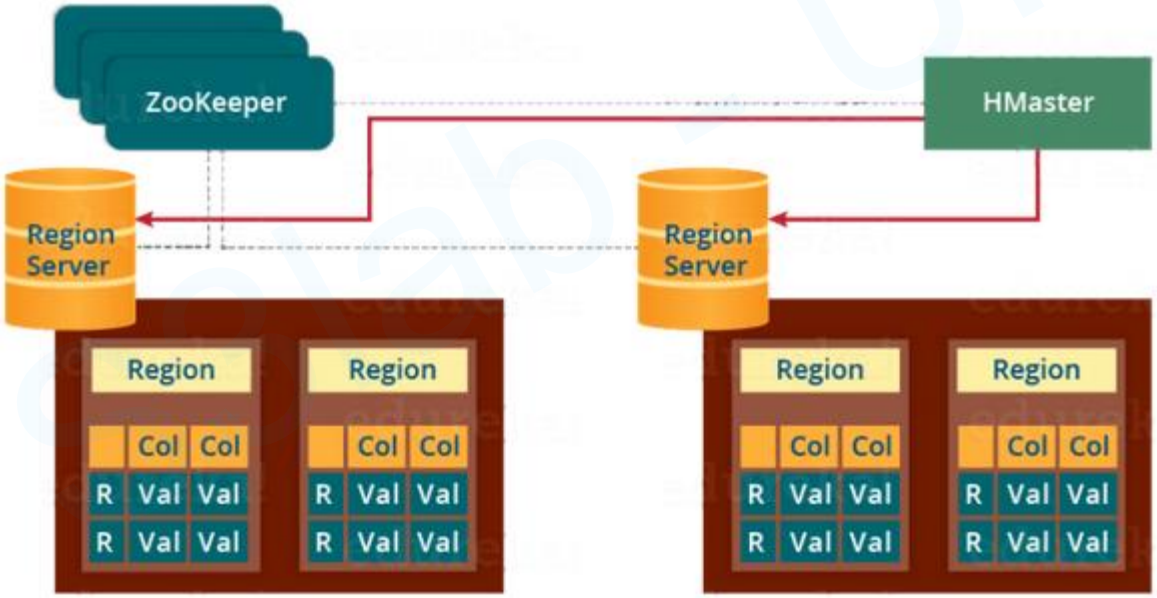


Figure: Components of HBase

Hbase architecture

ZooKeeper – The Coordinator

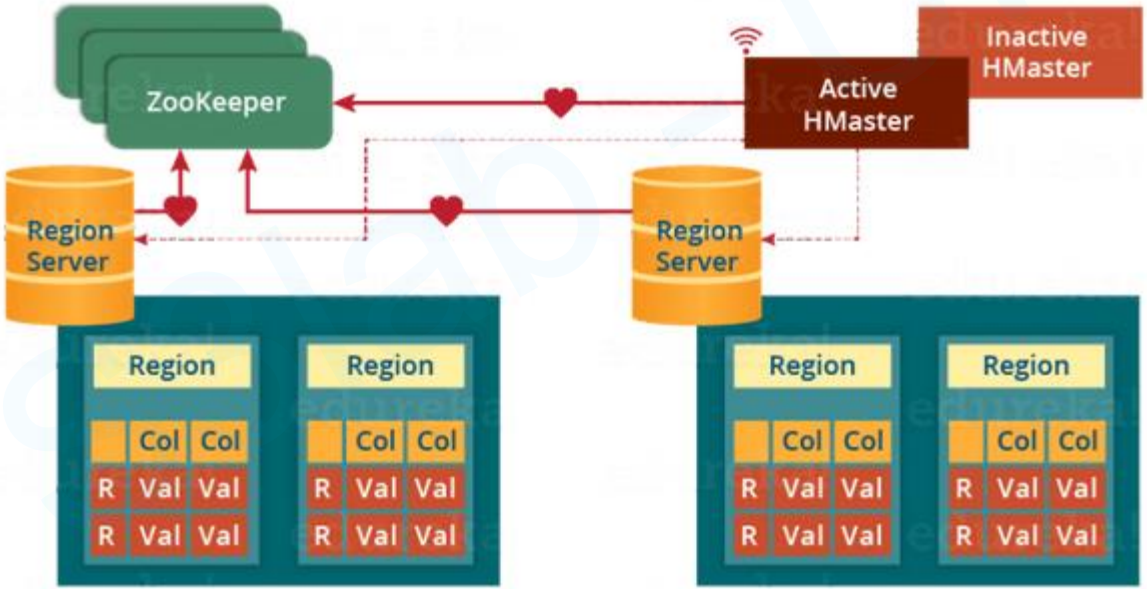


Figure: ZooKeeper as Coordination Service

Hbase architecture

META table

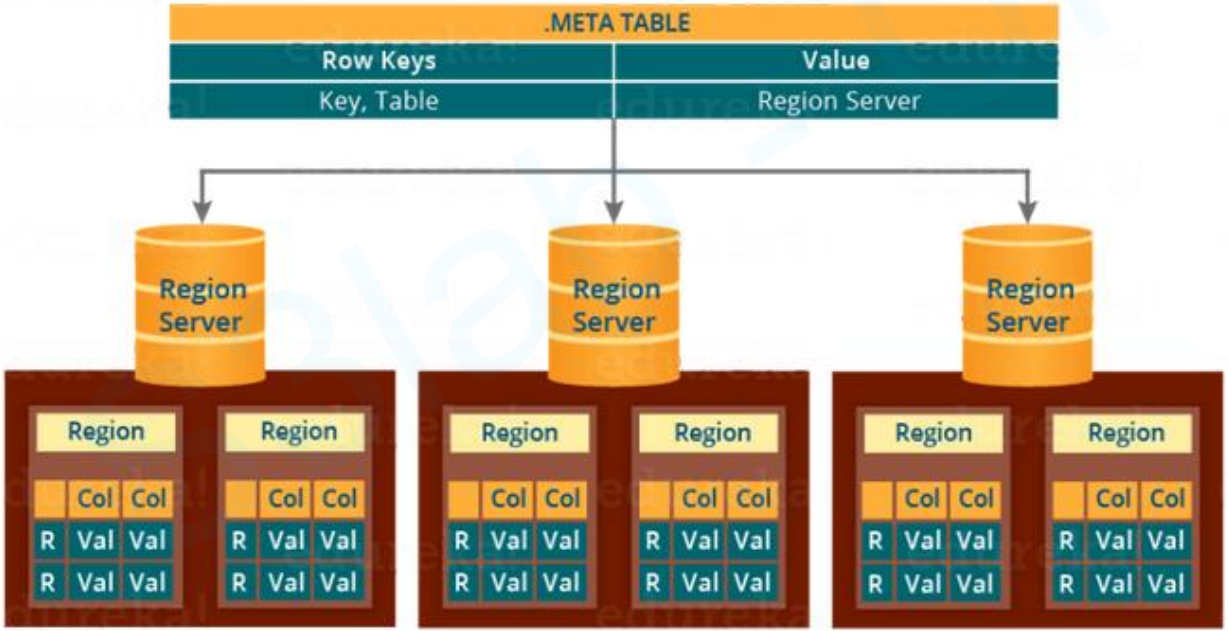


Figure: META Table

Hbase architecture

Components of Region Server

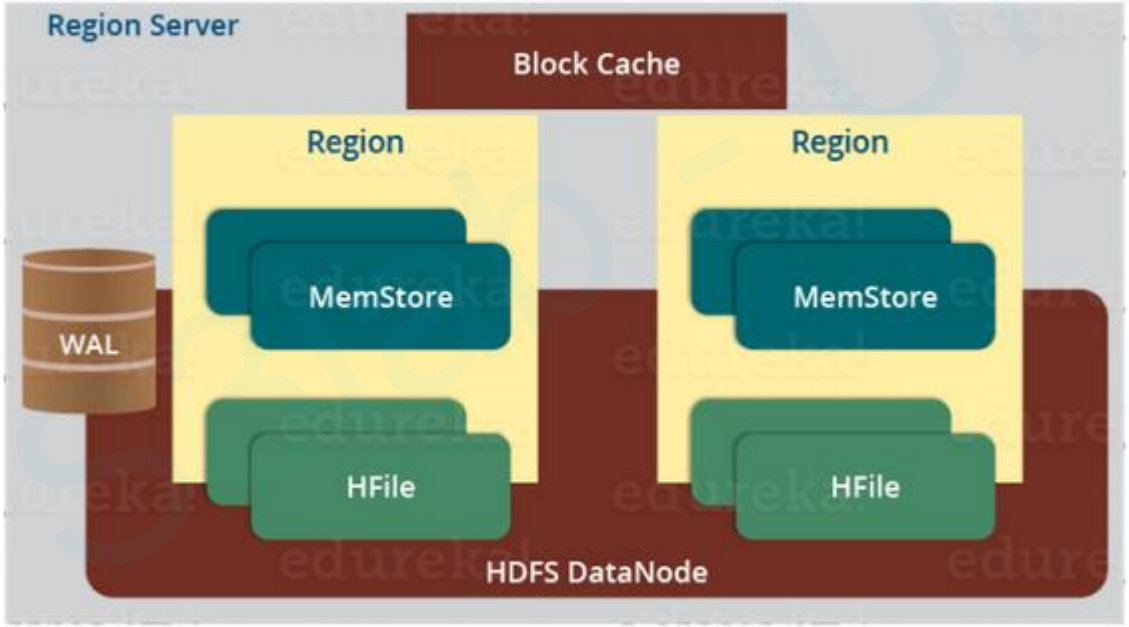


Figure: Region Server Components

Hbase architecture

Hbase write mechanism

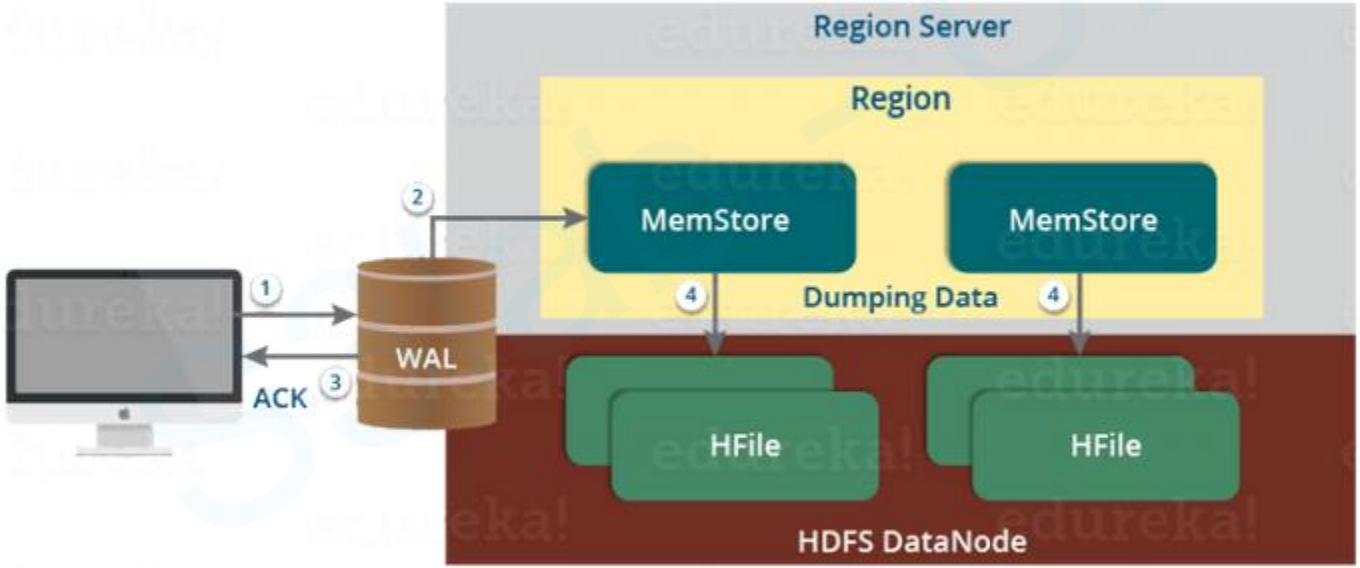


Figure: Write Mechanism in HBase

Hbase tutorial

Hbase tutorial

- Accessing HBase by using the HBase Shell
- Command: **hbase shell**

```
[cloudera@quickstart ~]$ hbase shell
2020-10-13 06:35:21,354 INFO  [main] Configuration.deprecation: hadoop.native.li
b is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.13.0, rUnknown, Wed Oct  4 11:16:18 PDT 2017

hbase(main):001:0>
```

Hbase tutorial

- Check the shell functioning before proceeding further. Use the **list** command for this purpose. List is a command used to get the list of all the tables in HBase.

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.1460 seconds

=> []
hbase(main):002:0>
```

Hbase tutorial

- Command: **status**

This command returns the status of the system including the details of the servers running on the system. Its syntax is as follows:

```
hbase(main):002:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load
```

- Command: table_help

This command guides you what and how to use table-referenced commands. Given below is the syntax to use this command.

Hbase tutorial

Creating a Table using HBase Shell

Command: create '<table name>','<column family>'

Given below is a sample schema of a table named emp. It has two column families: "personal data" and "professional data".

Row key	personal data	professional data

```
hbase(main):006:0> create 'emp', 'personal data', 'professional data'
0 row(s) in 1.4800 seconds

=> Hbase::Table - emp
hbase(main):007:0> list
TABLE
emp
1 row(s) in 0.0110 seconds

=> ["emp"]
```

verify whether the table is created using the list command

Hbase tutorial

Creating a Table using HBase Shell

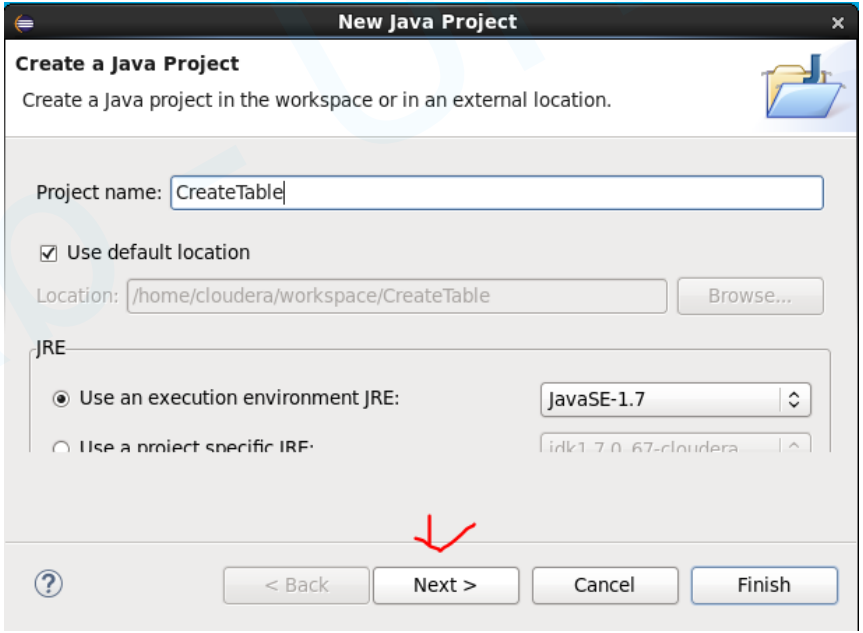
Check the table

```
hbase(main):009:0> describe 'emp'
Table emp is ENABLED
emp
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VE
RSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
 BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'professional data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0'
, VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FAL
SE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
2 row(s) in 0.0490 seconds
```

Hbase tutorial

Creating a Table Using java API

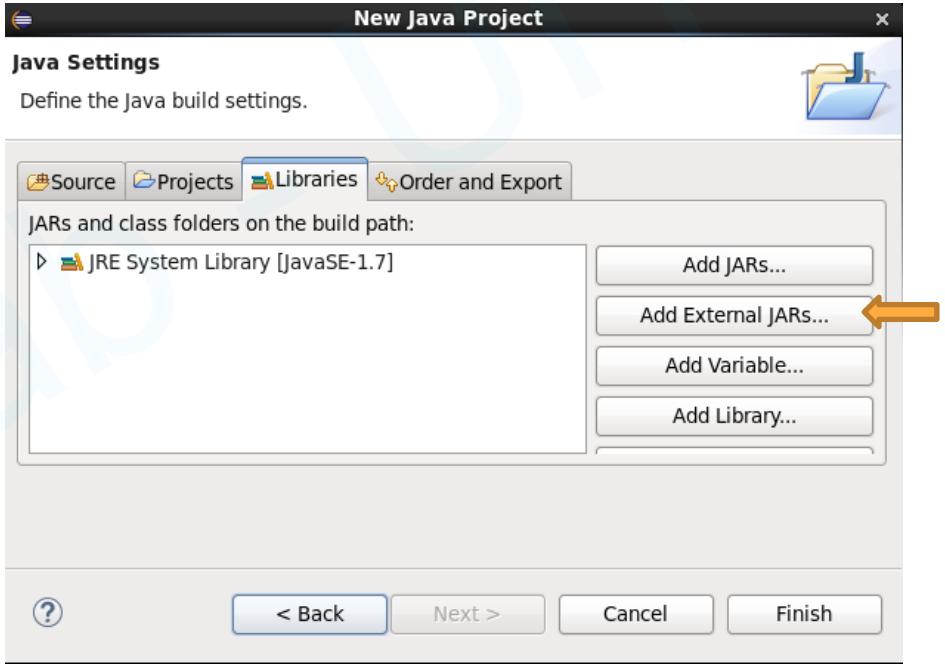
- Create Java Project



Hbase tutorial

Creating a Table Using java API

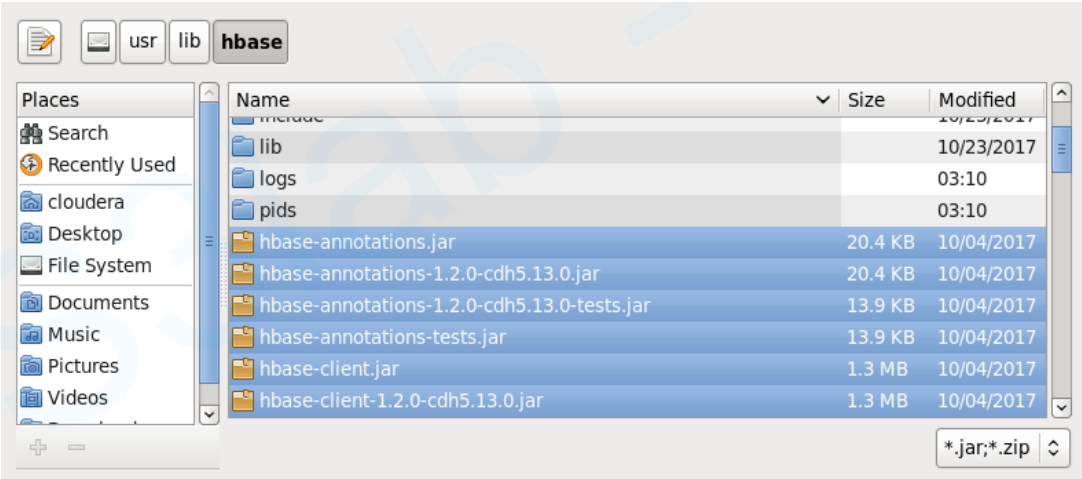
- Add External JARs



Hbase tutorial

Creating a Table Using java API

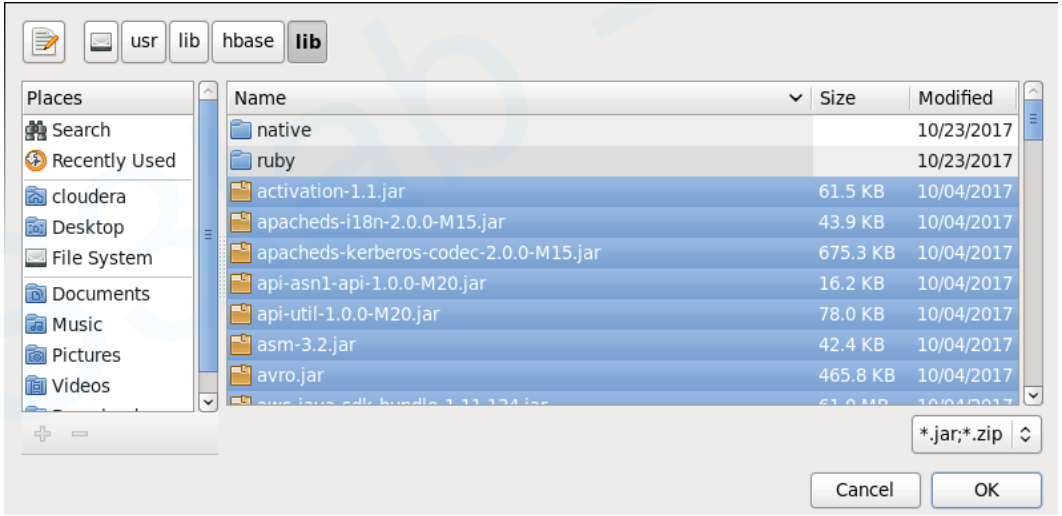
- Add all .jar files in **/usr/lib/hbase**



Hbase tutorial

Creating a Table Using java API

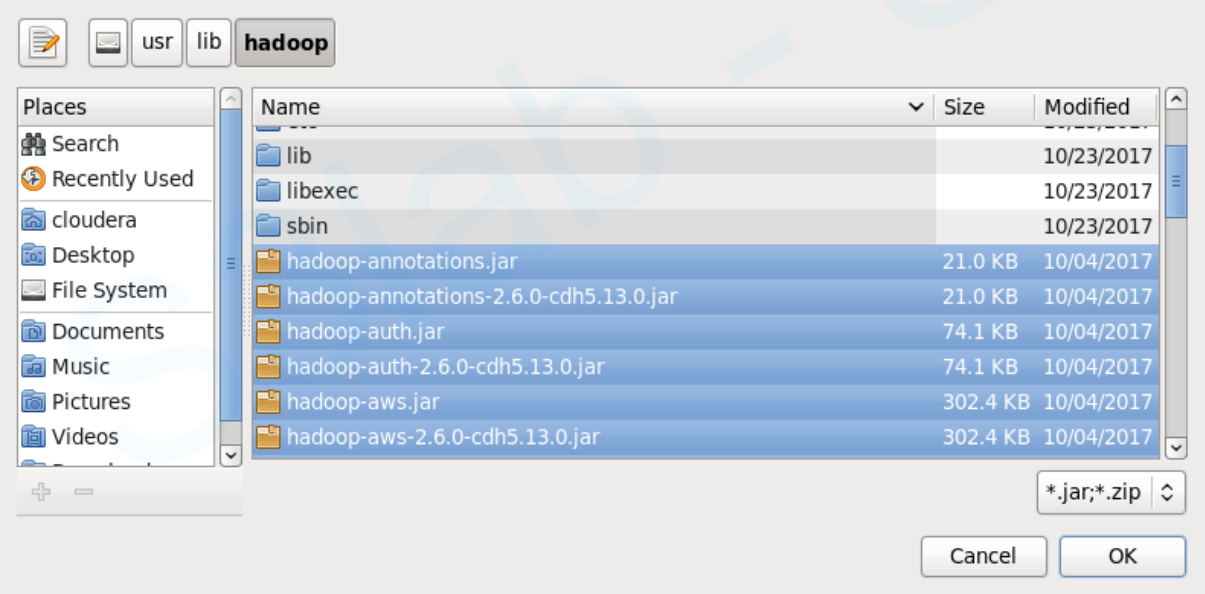
- Add all .jar files in **/usr/lib/hbase/lib**



Hbase tutorial

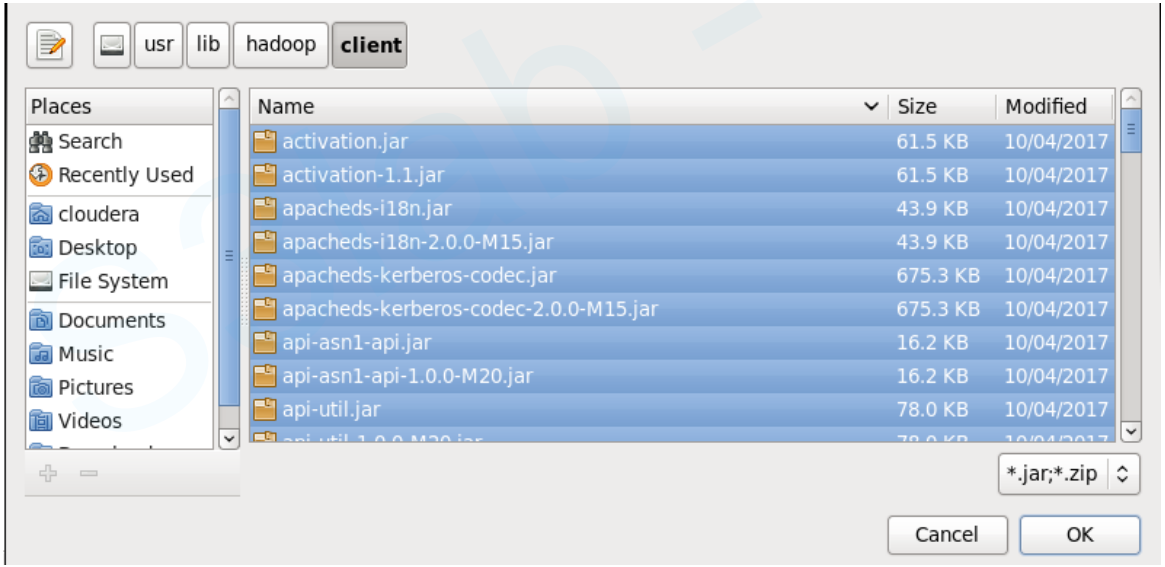
Creating a Table Using java API

- Add all .jar files in **/usr/lib/hadoop**



Hbase tutorial

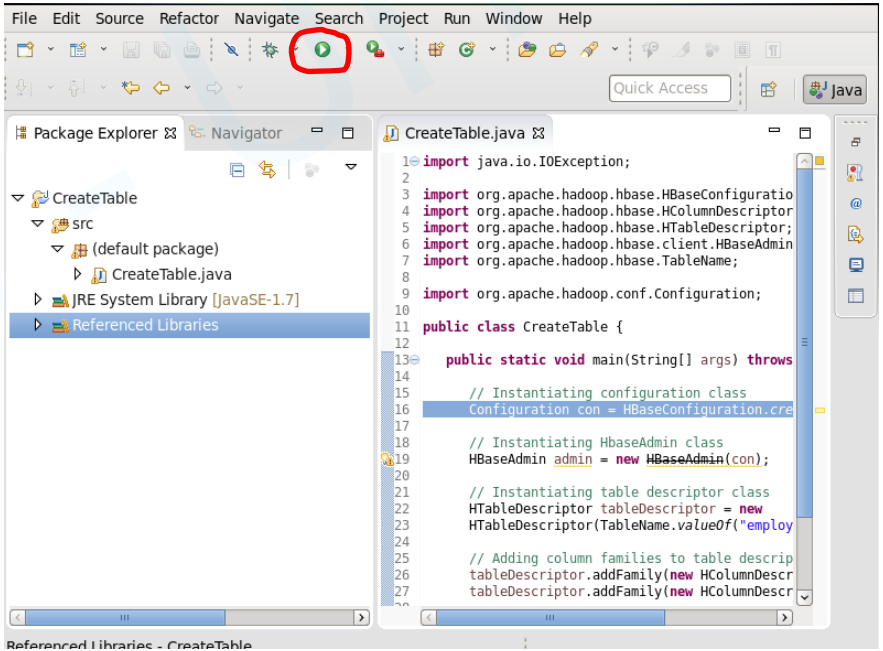
- Add all .jar files in **/usr/lib/hadoop/clident**



Hbase tutorial

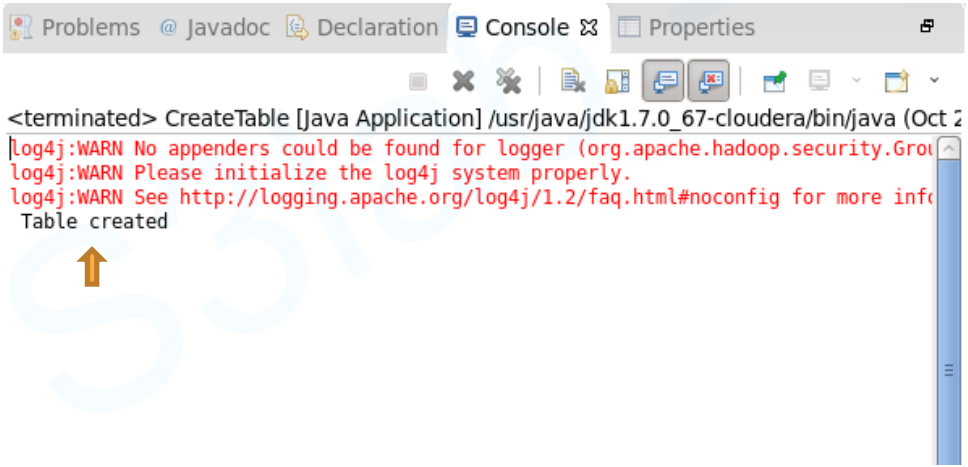
Creating a Table Using java API

- Create new .java file and run it



Hbase tutorial

- The console output should be like this



Hbase tutorial

Creating a Table Using java API

- Check if the table **employee** has been created

```
hbase(main):001:0> list
TABLE
emp
employee
2 row(s) in 0.2330 seconds

=> ["emp", "employee"]
hbase(main):002:0>
```

Hbase tutorial

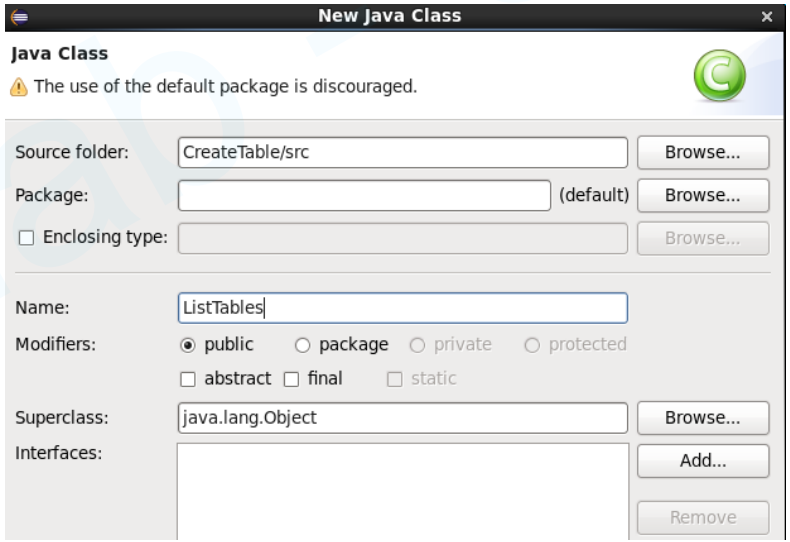
Creating a Table Using java API

```
hbase(main):010:0> describe 'employee'
Table employee is ENABLED
employee
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSION
S => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOC
KSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'professional', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VER
SIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE',
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
2 row(s) in 0.0200 seconds
```

Hbase tutorial

Listing Tables Using Java API

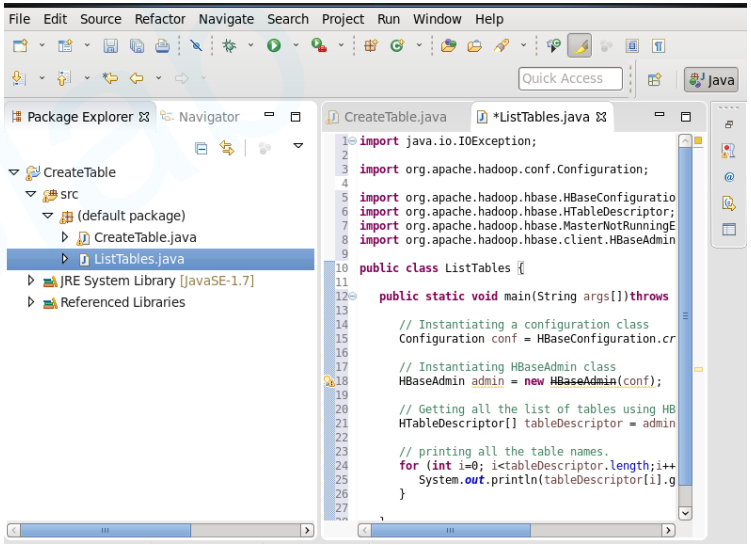
- Create new Java file in the same project



Hbase tutorial

Listing Tables Using Java API

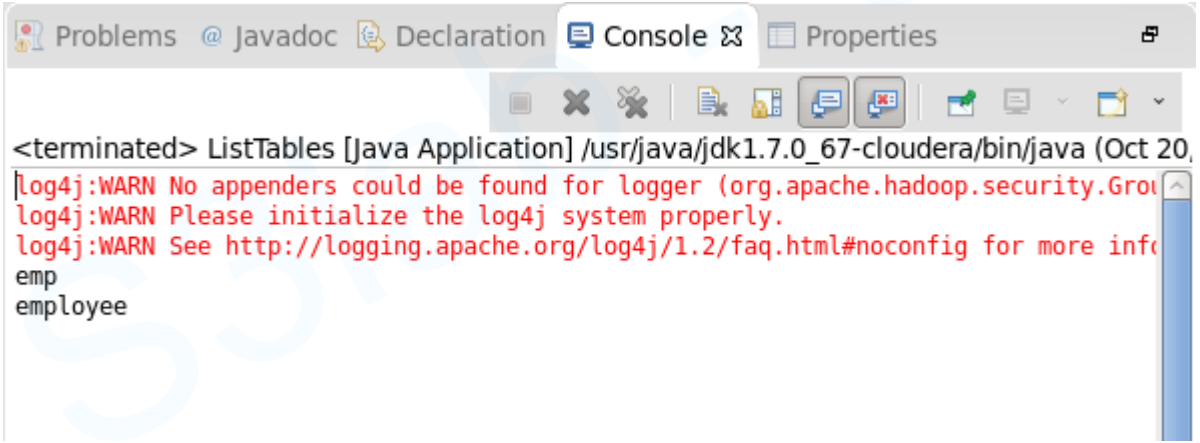
- Paste the code and execute it



Hbase tutorial

Listing Tables Using Java API

- The console output should be like this

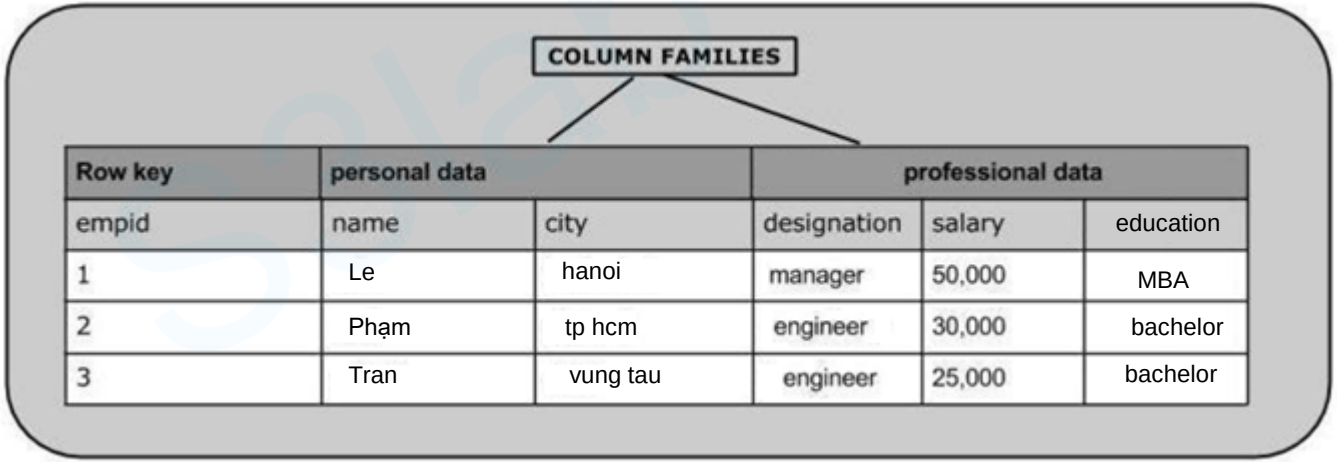


Hbase tutorial

Writing Data to HBase

Using **put** command, you can insert rows into a table. Its syntax is as follows:

```
put '<table name>','<row name> ','<colfamily:colname>','<value>'
```



Row key	personal data		professional data		
empid	name	city	designation	salary	education
1	Le	hanoi	manager	50,000	MBA
2	Phạm	tp hcm	engineer	30,000	bachelor
3	Tran	vung tau	engineer	25,000	bachelor

Hbase tutorial

Writing Data to HBase

- Let us insert the first row values into the emp table as shown below

```
hbase(main):011:0> put 'emp','1','personal data:name','Le'
0 row(s) in 0.1250 seconds

hbase(main):012:0> put 'emp','1','personal data:city','hanoi'
0 row(s) in 0.0050 seconds

hbase(main):013:0> put 'emp','1','professional data:task','manager'
0 row(s) in 0.0130 seconds

hbase(main):015:0> put 'emp','1','professional data:salary','5000'
0 row(s) in 0.0060 seconds

hbase(main):016:0> put 'emp','1','professional data:education','MBA'
0 row(s) in 0.0080 seconds
```

Hbase tutorial

Writing Data to HBase

- Check the content of the table

```
hbase(main):017:0> scan 'emp'
ROW                           COLUMN+CELL
 1                            column=personal data:city, timestamp=1603200548353, value=hanoi
 1                            column=personal data:name, timestamp=1603200496978, value=Le
 1                            column=professional data:education, timestamp=1603200731642, value=MBA
 1                            column=professional data:salary, timestamp=1603200695604, value=5000
 1                            column=professional data:task, timestamp=1603200595474, value=manager
1 row(s) in 0.0150 seconds
```

Hbase tutorial

Writing Data to HBase

- Copy these lines and paste in the Hbase shell (you can type them if you want)

hbasePut

```
put 'emp','1','personal data:name','Le'
put 'emp','1','personal data:city','hanoi'
put 'emp','1','professional data:task','manager'
put 'emp','1','professional data:salary','5000'
put 'emp','1','professional data:education','MBA'
put 'emp','2','personal data:name','Pham'
put 'emp','2','personal data:city','tp hcm'
put 'emp','2','professional data:task','engineer'
put 'emp','2','professional data:salary','3000'
put 'emp','2','professional data:education','bachelor'
put 'emp','3','personal data:name','Tran'
put 'emp','3','personal data:city','tp hcm'
put 'emp','3','professional data:task','vung tau'
put 'emp','3','professional data:salary','2500'
put 'emp','3','professional data:education','bachelor'
```

Hbase tutorial

Writing Data to HBase

- Check the content of the table

```
hbase(main):028:0> scan 'emp'
ROW                         COLUMN+CELL
 1                          column=personal data:city, timestamp=1603201878949, value=hanoi
 1                          column=personal data:name, timestamp=1603201878923, value=Le
 1                          column=professional data:education, timestamp=1603201880630, value=MBA
 1                          column=professional data:salary, timestamp=1603201878997, value=5000
 1                          column=professional data:task, timestamp=1603201878979, value=manager
 2                          column=personal data:city, timestamp=1603201992646, value=tp hcm
 2                          column=personal data:name, timestamp=1603201992611, value=Pham
 2                          column=professional data:education, timestamp=1603201992708, value=bachelor
 2                          column=professional data:salary, timestamp=1603201992686, value=3000
 2                          column=professional data:task, timestamp=1603201992671, value=engineer
 3                          column=personal data:city, timestamp=1603201992738, value=tp hcm
 3                          column=personal data:name, timestamp=1603201992721, value=Tran
 3                          column=professional data:education, timestamp=1603201992787, value=bachelor
 3                          column=professional data:salary, timestamp=1603201992774, value=2500
 3                          column=professional data:task, timestamp=1603201992756, value=vung tau
3 row(s) in 0.0490 seconds
```

Hbase tutorial

Writing Data to HbaseUsing Java API

Step 1: Instantiate the Configuration Class

```
Configuration conf = HbaseConfiguration.create();
```

Step 2:Instantiate the HTable Class

```
HTable hTable = new HTable(conf, tableName);
```

Step 3: Instantiate the PutClass

```
Put p = new Put(Bytes.toBytes("row id"));
```

This class requires the row name you want to insert the data into, in string format.

Step 4: Insert Data

```
p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes("column name"),Bytes.toBytes("value"));
```

Step 5: Save the Data in Table

```
hTable.put(p);
```

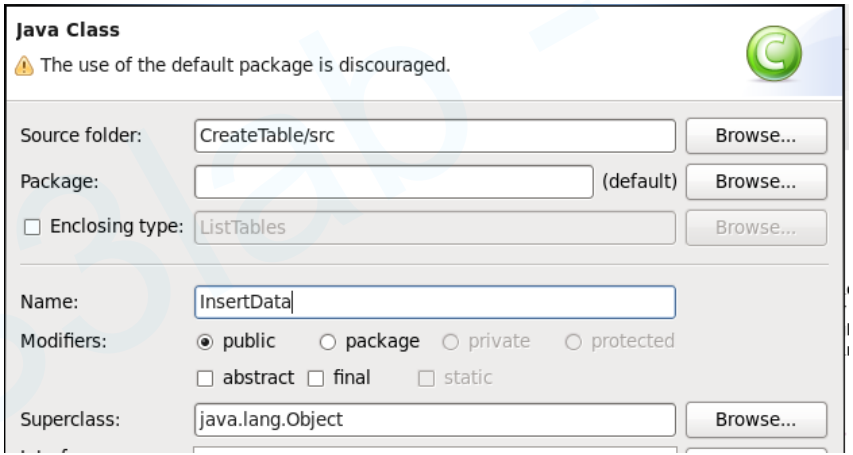
Step 6: Close the HTable Instance

```
hTable.close();
```

Hbase tutorial

Writing Data to HbaseUsing Java API

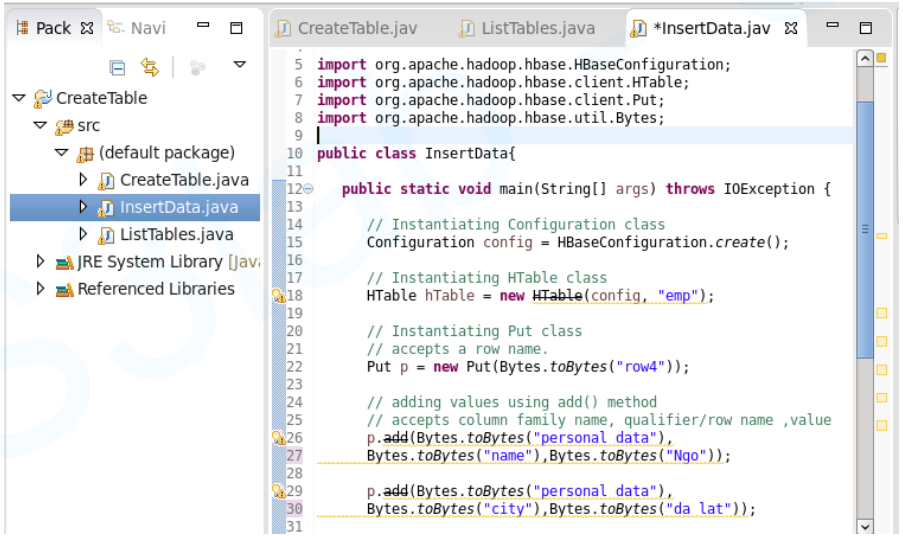
- Create new .java file



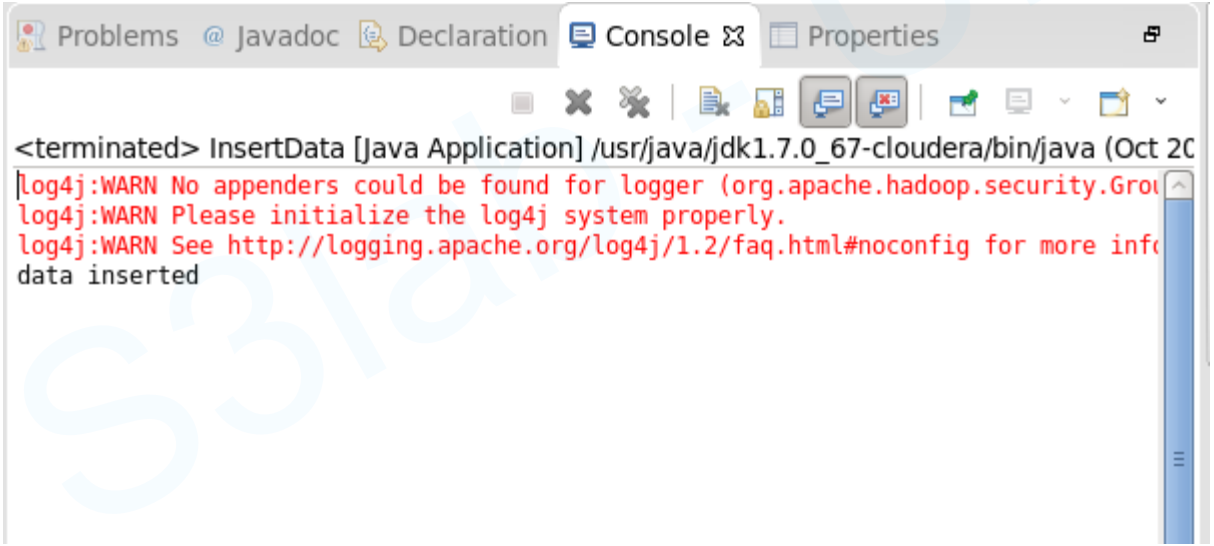
Hbase tutorial

Writing Data to HbaseUsing Java API

- Paste the code, save, and run it



Hbase tutorial



Hbase tutorial

- Check the result

```
hbase(main):029:0> scan 'emp'
ROW                         COLUMN+CELL
 1                          column=personal data:city, timestamp=1603201878949, value=hanoi
 1                          column=personal data:name, timestamp=1603201878923, value=Le
 1                          column=professional data:education, timestamp=1603201880630, value=MBA
 1                          column=professional data:salary, timestamp=1603201878997, value=5000
 1                          column=professional data:task, timestamp=1603201878979, value=manager
 2                          column=personal data:city, timestamp=1603201992646, value=tp hcm
 2                          column=personal data:name, timestamp=1603201992611, value=Pham
 2                          column=professional data:education, timestamp=1603201992708, value=bachelor
 2                          column=professional data:salary, timestamp=1603201992686, value=3000
 2                          column=professional data:task, timestamp=1603201992671, value=engineer
 3                          column=personal data:city, timestamp=1603201992738, value=tp hcm
 3                          column=personal data:name, timestamp=1603201992721, value=Tran
 3                          column=professional data:education, timestamp=1603201992787, value=bachelor
 3                          column=professional data:salary, timestamp=1603201992774, value=2500
 3                          column=professional data:task, timestamp=1603201992756, value=vung tau
 row4                       column=personal data:city, timestamp=1603204408960, value=da lat
 row4                       column=personal data:name, timestamp=1603204408960, value=Ngo
 row4                       column=professional data:education, timestamp=1603204408960, value=bachelor
 row4                       column=professional data:salary, timestamp=1603204408960, value=2000
 row4                       column=professional data:task, timestamp=1603204408960, value=developer
4 row(s) in 0.0400 seconds
```

Hbase tutorial

- Edit the code

```
Put p = new Put(Bytes.toBytes("row4"));  ⇨  Put p = new Put(Bytes.toBytes("4"));
```

- Then run the code

Hbase tutorial

Writing Data to HbaseUsing Java API

- Scan the table to see the result.
- In the table, “4” and “row4” are different rows

```
hbase(main):030:0> scan 'emp'
ROW                     COLUMN+CELL
 1                      column=personal data:city, timestamp=1603201878949, value=hanoi
 1                      column=personal data:name, timestamp=1603201878923, value=Le
 1                      column=professional data:education, timestamp=1603201880630, value=MBA
 1                      column=professional data:salary, timestamp=1603201878997, value=5000
 1                      column=professional data:task, timestamp=1603201878979, value=manager
 2                      column=personal data:city, timestamp=1603201992646, value=tp hcm
 2                      column=personal data:name, timestamp=1603201992611, value=Pham
 2                      column=professional data:education, timestamp=1603201992708, value=bachelor
 2                      column=professional data:salary, timestamp=1603201992686, value=3000
 2                      column=professional data:task, timestamp=1603201992671, value=engineer
 3                      column=personal data:city, timestamp=1603201992738, value=tp hcm
 3                      column=personal data:name, timestamp=1603201992721, value=Tran
 3                      column=professional data:education, timestamp=1603201992787, value=bachelor
 3                      column=professional data:salary, timestamp=1603201992774, value=2500
 3                      column=professional data:task, timestamp=1603201992756, value=vung tau
 4                      column=personal data:city, timestamp=1603205278909, value=da lat
 4                      column=personal data:name, timestamp=1603205278909, value=Ngo
 4                      column=professional data:education, timestamp=1603205278909, value=bachelor
 4                      column=professional data:salary, timestamp=1603205278909, value=2000
 4                      column=professional data:task, timestamp=1603205278909, value=developer
 row4                   column=personal data:city, timestamp=1603204408960, value=da lat
 row4                   column=personal data:name, timestamp=1603204408960, value=Ngo
 row4                   column=professional data:education, timestamp=1603204408960, value=bachelor
 row4                   column=professional data:salary, timestamp=1603204408960, value=2000
 row4                   column=professional data:task, timestamp=1603204408960, value=developer
5 row(s) in 0.1000 seconds
```

Hbase tutorial

Reading Data using HBase Shell

- Command: get ‘<table name>’ , ‘<row id>’

```
hbase(main):032:0> get 'emp','row4'
COLUMN                          CELL
 personal data:city             timestamp=1603204408960, value=da lat
 personal data:name             timestamp=1603204408960, value=Ngo
 professional data:educatio     timestamp=1603204408960, value=bachelor
 n
 professional data:salary       timestamp=1603204408960, value=2000
 professional data:task         timestamp=1603204408960, value=developer
5 row(s) in 0.0150 seconds
```

Hbase tutorial

Reading a Specific Column using HBase Shell

- Command: get 'table name', ‘row id’, {COLUMN ⇒ ‘column family:column name ’}

```
hbase(main):034:0> get 'emp', 'row4', {COLUMN => 'personal data:city'}
COLUMN                  CELL
 personal data:city     timestamp=1603204408960, value=da lat
1 row(s) in 0.0050 seconds
```

Hbase tutorial

Updating Data using HBase Shell

- Command: put ‘table name’,’row id’,'Column family:column name',’new value’

```
hbase(main):040:0> put 'emp','1','personal data:city','haiphong'
0 row(s) in 0.0190 seconds

hbase(main):041:0> scan 'emp'
ROW                         COLUMN+CELL
 1                          column=personal data:city, timestamp=1603208058069, value=haiphong
 1                          column=personal data:name, timestamp=1603201878923, value=Le
 1                          column=professional data:education, timestamp=1603201880630, value=MBA
 1                          column=professional data:salary, timestamp=1603201878997, value=5000
 1                          column=professional data:task, timestamp=1603201878979, value=manager
 2                          column=personal data:city, timestamp=1603201992646, value=tp hcm
```

Hbase tutorial

Deleting a Specific Cell in a Table

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

```
hbase(main):035:0> delete 'emp', 'row4', 'personal data:city'
0 row(s) in 0.0320 seconds

hbase(main):036:0> scan 'emp'
ROW                         COLUMN+CELL
 1                          column=personal data:city, timestamp=1603201878949, value=hanoi
 1                          column=personal data:name, timestamp=1603201878923, value=Le
 1                          column=professional data:education, timestamp=1603201880630, value=MBA
 1                          column=professional data:salary, timestamp=1603201878997, value=5000
 1                          column=professional data:task, timestamp=1603201878979, value=manager


 row4                       column=personal data:name, timestamp=1603204408960, value=Ngo
 row4                       column=professional data:education, timestamp=1603204408960, value=bachelor
 row4                       column=professional data:salary, timestamp=1603204408960, value=2000
 row4                       column=professional data:task, timestamp=1603204408960, value=developer
5 row(s) in 0.0230 seconds
```

Hbase tutorial

Deleting all the cells in a row

- Command: deleteall ‘<table name>’, ‘<row id>’

```
hbase(main):037:0> deleteall 'emp','row4'
0 row(s) in 0.0080 seconds

hbase(main):038:0> scan 'emp'
ROW                         COLUMN+CELL
 1                          column=personal data:city, timestamp=1603201878949, value=hanoi
 1                          column=personal data:name, timestamp=1603201878923, value=Le
 1                          column=professional data:education, timestamp=1603201880630, value=MBA
 1                          column=professional data:salary, timestamp=1603201878997, value=5000
 1                          column=professional data:task, timestamp=1603201878979, value=manager
 2                          column=personal data:city, timestamp=1603201992646, value=tp hcm
 2                          column=personal data:name, timestamp=1603201992611, value=Pham
 2                          column=professional data:education, timestamp=1603201992708, value=bachelor
 2                          column=professional data:salary, timestamp=1603201992686, value=3000
 2                          column=professional data:task, timestamp=1603201992671, value=engineer
 3                          column=personal data:city, timestamp=1603201992738, value=tp hcm
 3                          column=personal data:name, timestamp=1603201992721, value=Tran
 3                          column=professional data:education, timestamp=1603201992787, value=bachelor
 3                          column=professional data:salary, timestamp=1603201992774, value=2500
 3                          column=professional data:task, timestamp=1603201992756, value=vung tau
 4                          column=personal data:city, timestamp=1603205278909, value=da lat
 4                          column=personal data:name, timestamp=1603205278909, value=Ngo
 4                          column=professional data:education, timestamp=1603205278909, value=bachelor
 4                          column=professional data:salary, timestamp=1603205278909, value=2000
 4                          column=professional data:task, timestamp=1603205278909, value=developer
4 row(s) in 0.0230 seconds
```

Hbase tutorial

Deleting a Column Family

- Command: alter ‘ <table name> ’, ‘delete’ ⇒ ‘ <column family> ’

```
hbase(main):044:0> alter 'emp' , 'delete' => 'personal data'
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.1930 seconds

hbase(main):045:0> scan 'emp'
ROW                         COLUMN+CELL
 1                          column=professional data:education, timestamp=1603201880630, value=MBA
 1                          column=professional data:salary, timestamp=1603201878997, value=5000
 1                          column=professional data:task, timestamp=1603201878979, value=manager
 2                          column=professional data:education, timestamp=1603201992708, value=bachelor
 2                          column=professional data:salary, timestamp=1603201992686, value=3000
 2                          column=professional data:task, timestamp=1603201992671, value=engineer
 3                          column=professional data:education, timestamp=1603201992787, value=bachelor
```

Hbase tutorial

VERSION

- When you put data into HBase, a timestamp is required.

- The timestamp can be generated automatically by the RegionServer or can be supplied by you.
- The timestamp must be unique per version of a given cell, because the timestamp identifies the version.
- To modify a previous version of a cell, for instance, you would issue a Put with a different value for the data itself, but the **same timestamp**.

Command: **put ‘table name’,’row id’,'Column family:column name',’new value’, timestamp**

Hbase tutorial

VERSION

- Doing a put always creates a new version of a cell, at a certain timestamp.

```
hbase(main):005:0> get 'emp', '1' , {COLUMN => 'professional data:task'}
COLUMN                          CELL
 professional data:task         timestamp=1603201878979, value=manager
1 row(s) in 0.0160 seconds
```

- Default update

```
hbase(main):008:0> put 'emp', '1' , 'professional data:task', 'CEO'
0 row(s) in 0.0770 seconds

hbase(main):009:0> get 'emp', '1' , {COLUMN => 'professional data:task',VERSION => 2}
COLUMN                          CELL
 professional data:task         timestamp=1603248439909, value=CEO
1 row(s) in 0.0060 seconds
```

Hbase tutorial

Change the maximum number of versions

- Get 2 versions of that cell

```
hbase(main):013:0> get 'emp', '1' , {COLUMN => 'professional data:task',VERSIONS => 2}
COLUMN                          CELL
 professional data:task         timestamp=1603248439909, value=CEO
1 row(s) in 0.0050 seconds
```

- We receive only the latest version of that cell (which is 'CEO')
- The reason is because the maximum number of versions defaults to 1
- Use **alter** command to change the maximum number of versions of that family column

```
hbase(main):014:0> alter 'emp', {NAME => 'professional data', VERSIONS => 3}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.3850 seconds
```

Hbase tutorial

Get all versions of a cell

- Let’s try again to get 2 versions of that cell

```
hbase(main):015:0> get 'emp', '1' , {COLUMN => 'professional data:task',VERSIONS => 2}
COLUMN                          CELL
 professional data:task         timestamp=1603248439909, value=CEO
 professional data:task         timestamp=1603201878979, value=manager
2 row(s) in 0.0080 seconds
```

Hbase tutorial

Update a specific version

Let’s get all versions of the cell

```
hbase(main):015:0> get 'emp', '1' , {COLUMN => 'professional data:task',VERSIONS => 2}
COLUMN                        CELL
 professional data:task       timestamp=1603248439909, value=CEO
 professional data:task       timestamp=1603201878979, value=manager
2 row(s) in 0.0080 seconds
```

Command: **put ‘table name’,’row id’,'Column family:column name','new value’, timestamp**

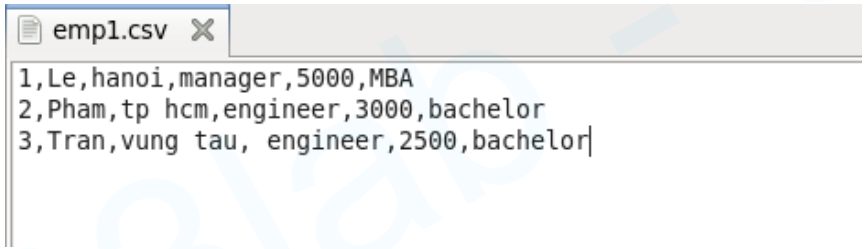
```
hbase(main):018:0> put 'emp', '1' , 'professional data:task', 'helpdesk',1603248439909
0 row(s) in 0.0110 seconds
```

```
hbase(main):020:0> get 'emp', '1' , {COLUMN => 'professional data:task',VERSIONS => 3}
COLUMN                        CELL
 professional data:task       timestamp=1603248439909, value=helpdesk
 professional data:task       timestamp=1603201878979, value=manager
2 row(s) in 0.0060 seconds
```

Hbase tutorial

Load CSV file from HDFS to HBase

- Create a csv file (e.g. using **gedit**)



- Put the file to HDFS

```
[cloudera@quickstart ~]$ hdfs dfs -put emp1.csv
```

Hbase tutorial

Load CSV file from HDFS to HBase

- Navigate to HBase directory

```
[cloudera@quickstart ~]$ cd /usr/lib/hbase
[cloudera@quickstart hbase]$
```

- Command: **hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=',' -Dimporttsv.columns= ...**

```
[cloudera@quickstart hbase]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=',' -Dimporttsv.columns='HBASE_ROW_KEY,perso
nal:name,personal:city,professional:task,professional:salary,professional:education' employee /user/cloudera/emp1.csv
2020-10-20 22:30:53,376 INFO  [main] zookeeper.RecoverableZooKeeper: Process identifier=hconnection-0x451b2519 connecting to ZooKeeper ensemble=l
ocalhost:2181
2020-10-20 22:30:53,387 INFO  [main] zookeeper.ZooKeeper: Client environment:zookeeper.version=3.4.5-cdh5.13.0--1, built on 10/04/2017 18:04 GMT
```

```
        ImportTsv
                Bad Lines=0
        File Input Format Counters
                Bytes Read=105
        File Output Format Counters
                Bytes Written=0
```

Important: no space before and after ',' when listing columns

```
-Dimporttsv.columns='HBASE_ROW_KEY, personal:name, personal:city, professional:task,
```

won't run

Hbase tutorial

Load CSV file from HDFS to HBase

- Scan the table in hbase shell

```
hbase(main):031:0> scan 'employee'
ROW                                   COLUMN+CELL
 1                                    column=personal:city, timestamp=1603256907282, value=hanoi
 1                                    column=personal:name, timestamp=1603256907282, value=Le
 1                                    column=professional:education, timestamp=1603256907282, value=MBA
 1                                    column=professional:salary, timestamp=1603256907282, value=5000
 1                                    column=professional:task, timestamp=1603256907282, value=manager
 2                                    column=personal:city, timestamp=1603256907282, value=tp hcm
 2                                    column=personal:name, timestamp=1603256907282, value=Pham
 2                                    column=professional:education, timestamp=1603256907282, value=bachelor
 2                                    column=professional:salary, timestamp=1603256907282, value=3000
 2                                    column=professional:task, timestamp=1603256907282, value=engineer
 3                                    column=personal:city, timestamp=1603256907282, value=vung tau
 3                                    column=personal:name, timestamp=1603256907282, value=Tran
 3                                    column=professional:education, timestamp=1603256907282, value=bachelor
 3                                    column=professional:salary, timestamp=1603256907282, value=2500
 3                                    column=professional:task, timestamp=1603256907282, value= engineer
3 row(s) in 0.0270 seconds
```

Hbase tutorial

Load data from Hive to HBase

- Check the Hive table **student**

```
hive> select * from student;
OK
123451  Quynh   Hadoop  22
123452  Tai     Java    22
123453  Truong  Python  23
123454  Nghia   Hadoop  24
123455  Thuy    Java    23
123456  Hao     Python  24
123457  Hien    Hadoop  22
123458  Phuong  Java    23
123459  Hai     Python  23
123460  Phuong  Hadoop  24
Time taken: 0.882 seconds, Fetched: 10 row(s)
hive> describe student;
OK
id                      int
name                    string
course                  string
age                     int
Time taken: 0.079 seconds, Fetched: 4 row(s)
```

Hbase tutorial

Create HBase-Hive Mapping table

- Create another hive table which actually points to an HBase table
- **hbase_student** is the name of Hive table
- **studen_hbase** is the name of Hbase table linked to the Hive table above

- Command: create table hbase_student (id int,name string,course string,age int) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,personal:name,personal:course,additional:age") TBLPROPERTIES ("hbase.table.name" = "studen_hbase");

```
hive> create table hbase_student (id int,name string,course string,age int) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' WITH SERDEPR
OPERTIES ("hbase.columns.mapping" = ":key,personal:name,personal:course,additional:age") TBLPROPERTIES ("hbase.table.name" = "studen_hbase");
OK
Time taken: 3.204 seconds
```

Hbase tutorial

Load data from Hive to HBase

- Check if the new table has been created in Hbase, then check its schema

```
hbase(main):034:0> list
TABLE
emp
employee
studen_hbase
3 row(s) in 0.0040 seconds

=> ["emp", "employee", "studen_hbase"]
hbase(main):035:0> describe 'studen_hbase'
Table studen_hbase is ENABLED
studen_hbase
COLUMN FAMILIES DESCRIPTION
{NAME => 'additional', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1', COMPRESSIO
N => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false', BL
OCKCACHE => 'true'}
{NAME => 'personal', 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', BLOC
KCACHE => 'true'}
2 row(s) in 0.0130 seconds
```

Hbase tutorial

Load data from Hive to HBase

- Migrate hive table data to HBase

```
hive> insert into table hbase_student select * from student;
Query ID = cloudera_20201020231515_7295e24d-11a1-4350-81b3-7d1f193d0ef3
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1603246922983_0003, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1603246922983_0003/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1603246922983_0003
Hadoop job information for Stage-0: number of mappers: 1; number of reducers: 0
2020-10-20 23:15:19,749 Stage-0 map = 0%,  reduce = 0%
2020-10-20 23:15:28,451 Stage-0 map = 100%,  reduce = 0%, Cumulative CPU 1.69 sec
MapReduce Total cumulative CPU time: 1 seconds 690 msec
Ended Job = job_1603246922983_0003
MapReduce Jobs Launched:
Stage-Stage-0: Map: 1   Cumulative CPU: 1.69 sec   HDFS Read: 12230 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 690 msec
OK
```

Hbase tutorial

Load data from Hive to HBase

- Check the table in Hive

```
hive> select * from hbase_student;
OK
123451  Quynh   Hadoop  22
123452  Tai     Java    22
123453  Truong  Python  23
123454  Nghia   Hadoop  24
123455  Thuy    Java    23
123456  Hao     Python  24
123457  Hien    Hadoop  22
123458  Phuong  Java    23
123459  Hai     Python  23
123460  Phuong  Hadoop  24
Time taken: 0.115 seconds, Fetched: 10 row(s)
```

Hbase tutorial

Load CSV file from HDFS to HBase

- Check the Hbase table

```
hbase(main):037:0> scan 'studen_hbase'
ROW                             COLUMN+CELL
 123451                         column=additional:age, timestamp=1603260927982, value=22
 123451                         column=personal:course, timestamp=1603260927982, value=Hadoop
 123451                         column=personal:name, timestamp=1603260927982, value=Quynh
 123452                         column=additional:age, timestamp=1603260927982, value=22
 123452                         column=personal:course, timestamp=1603260927982, value=Java
 123452                         column=personal:name, timestamp=1603260927982, value=Tai
 123453                         column=additional:age, timestamp=1603260927982, value=23
 123453                         column=personal:course, timestamp=1603260927982, value=Python
 123453                         column=personal:name, timestamp=1603260927982, value=Truong
 123454                         column=additional:age, timestamp=1603260927982, value=24
 123454                         column=personal:course, timestamp=1603260927982, value=Hadoop
 123454                         column=personal:name, timestamp=1603260927982, value=Nghia
 123455                         column=additional:age, timestamp=1603260927982, value=23
 123455                         column=personal:course, timestamp=1603260927982, value=Java
 123455                         column=personal:name, timestamp=1603260927982, value=Thuy
 123456                         column=additional:age, timestamp=1603260927982, value=24
 123456                         column=personal:course, timestamp=1603260927982, value=Python
 123456                         column=personal:name, timestamp=1603260927982, value=Hao
 123457                         column=additional:age, timestamp=1603260927982, value=22
```

Hbase tutorial

Dropping a Table using HBase Shell

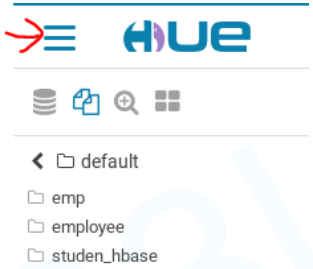
- Using the drop command, you can delete a table. Before dropping a table, you have to disable it.

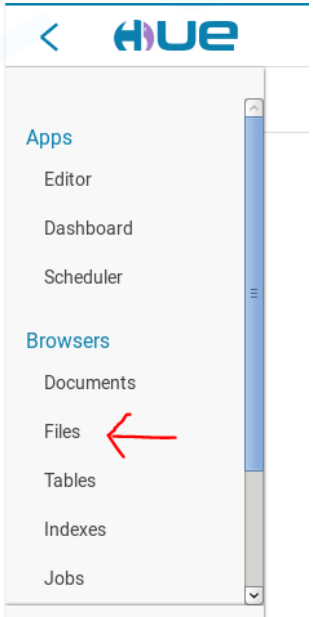
```
hbase(main):039:0> disable 'emp'
0 row(s) in 2.4110 seconds

hbase(main):040:0> drop 'emp'
0 row(s) in 1.3980 seconds

hbase(main):041:0> list
TABLE
employee
studen_hbase
2 row(s) in 0.0240 seconds

=> ["employee", "studen_hbase"]
```

Hbase tutorial

Hfile stored in HDFS

- Open HUE and use File Browsers

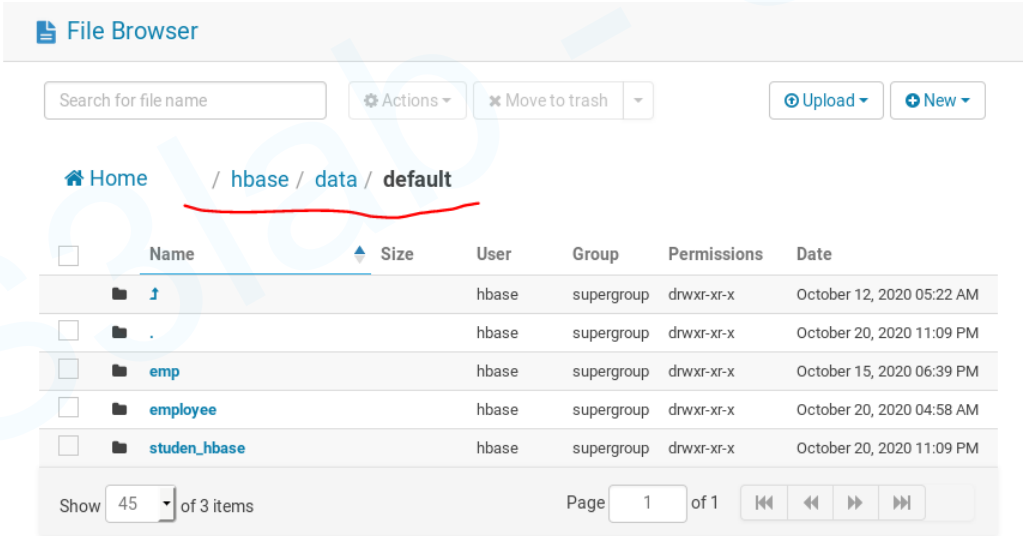




Hbase tutorial

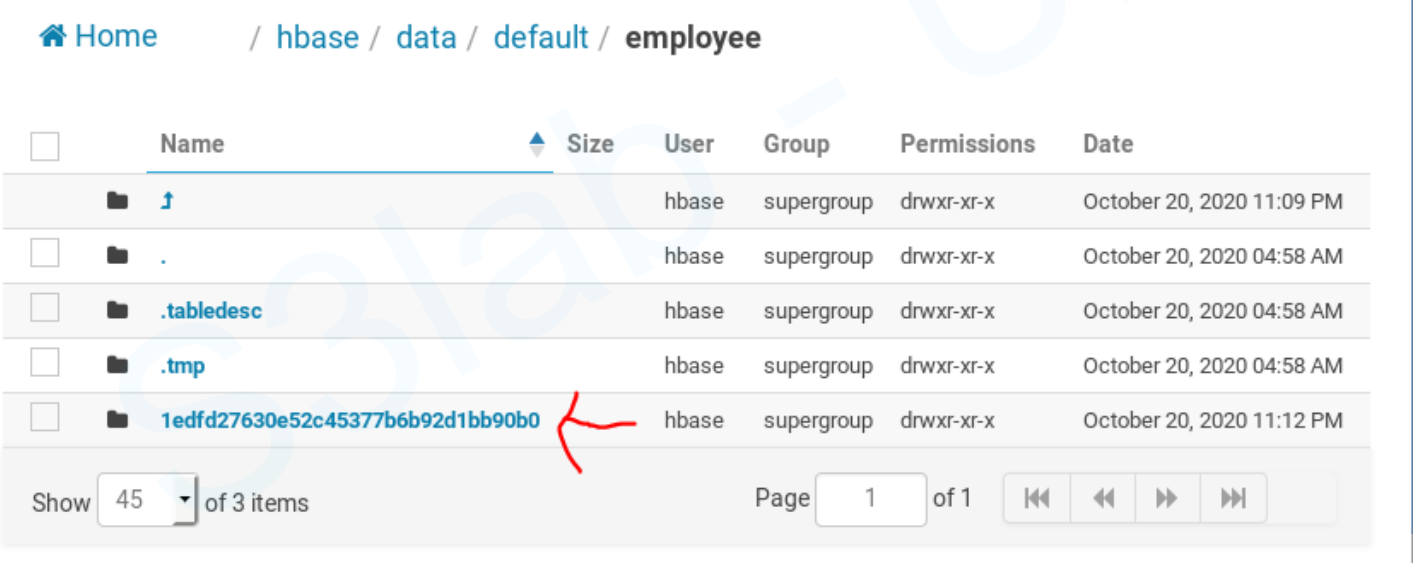
Hfile stored in HDFS

- Navigate to **/hbase/data/default**



Hbase tutorial

Hfile stored in HDFS



Hbase tutorial

Hfile stored in HDFS

/ hbase / data / default / employee / **1edfd27630e52c45377b6b92d1bb90b0**

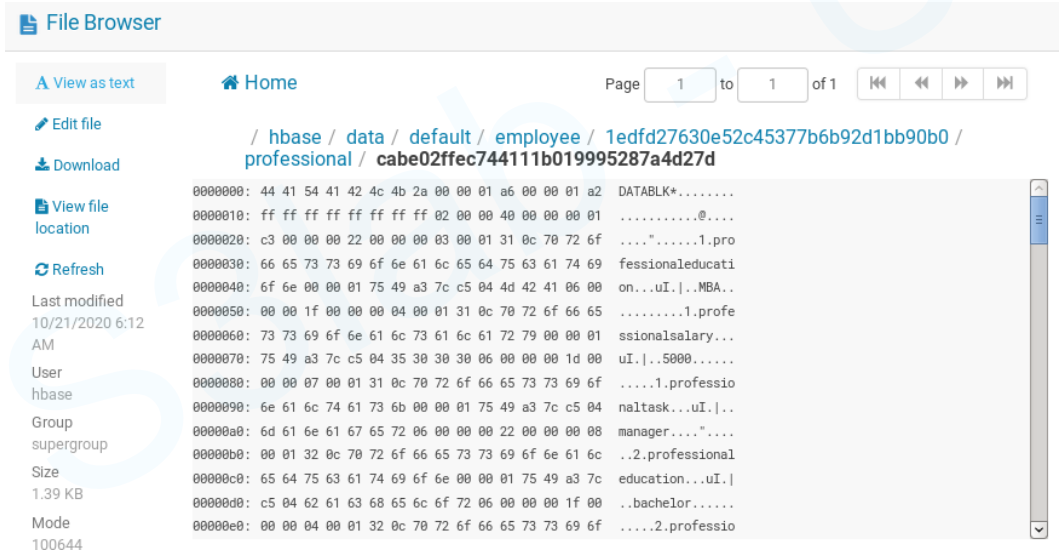
Name	Size	User	Group	Permissions	Date
↑		hbase	supergroup	drwxr-xr-x	October 20, 2020 04:58 AM
.		hbase	supergroup	drwxr-xr-x	October 20, 2020 11:12 PM
.regioninfo	43 bytes	hbase	supergroup	-rw-r--r--	October 20, 2020 04:58 AM
.tmp		hbase	supergroup	drwxr-xr-x	October 20, 2020 11:12 PM
personal		hbase	supergroup	drwxr-xr-x	October 20, 2020 11:12 PM
professional		hbase	supergroup	drwxr-xr-x	October 20, 2020 11:12 PM
recovered.edits		hbase	supergroup	drwxr-xr-x	October 20, 2020 07:23 PM

Important note: different column families are stored separatedly. When you query a row, the region server will have to grap data in multiple places (which will slow down your system).

Hbase tutorial

Hfile stored in HDFS

- Check the content of the Hfile (shown in binary and text format)



ther