HBase

Challenges with traditional RDBMS

A relational database management system is software that stores, manages, queries, and retrieves data from a relational database (RDBMS). The RDBMS offers a user and application interface to the
database, as well as administrative operations for data storage, access, and performance management. Though there are some challenges with RDBMS, let us understand those challenges

● **Join** - The data in a relational database is assumed to be stored in a tabular fashion, and duplication is eliminated by joining the data together. While this method works well for small datasets, as the data becomes large and dispersed, combining data from many tables becomes a difficult task.

● **Transaction support** - When we start distributing data, the distributed data store's consistency becomes a problem.

● **Cost of RDBMS** - RDBMS can fix some of the issues, but they are expensive.

● New applications, such as web and mobile applications, require high availability and low latency.

● It structures differently for XML and JSON files.

What is NoSQL database, why NoSQL database?

Non-tabular databases, such as NoSQL databases, store data in a different way than relational tables. NoSQL databases are classified according to their data model. Document, key-value, wide-column, and graph are the most common types. They have adaptable schemas and can handle big amounts of data and high user loads with ease. When individuals say "NoSQL database," they're usually referring to any database that isn't relational. Some people claim "NoSQL" means "not only SQL," while others say it means "non SQL." In any case, most people agree that NoSQL databases are databases that don't use relational tables to store data.

History behind the NoSQL database?

As the cost of storage fell substantially in the late 2000s, NoSQL databases emerged. Gone were the days when avoiding data duplication necessitated creating a sophisticated, difficult-to-manage
data model. Because developers (rather than storage) were becoming the primary cost of software development , NoSQL databases were designed to maximize developer productivity.

Features of NoSQL database?

Each NoSQL database has its own set of features. Many NoSQL databases contain the following features at a high level

● **Flexible schemas** - Unlike SQL databases, where you must first identify and declare a table's schema before inserting data, in NoSQL databases you can just insert data. The fields in a single
collection do not have to be the same, and the data type for a field might vary between documents inside a collection.

● **Support for multiple data models** - While relational databases require data to be organized into tables and columns before being accessed and analyzed, NoSQL databases are incredibly flexible
when it comes to data management. They can easily consume structured, semi-structured, and unstructured data, whereas relational databases are quite rigid and only handle structured
data. Specific application requirements are handled by different data models. To more easily handle diverse agile application development requirements, developers and architects pick a NoSQL database.

● **Easily scalable via Peer-to-Peer architecture** - It's not that relational databases can't scale; it's that they can't scale easily or cheaply. This is because they're created with a classic master-slave architecture, which implies scaling up via larger and larger hardware servers rather than scaling out or worse by sharding.

● **Distribution capabilities** - NoSQL database is built to distribute data at a worldwide scale, which means it can write and read data from many locations involving multiple data canters and/or cloud regions. In contrast, relational databases rely on a centralized application that is location-dependent (e.g., a single location), particularly for write operations. Because data is spread with numerous copies where it needs to be, adopting a distributed database with a masterless design allows you to maintain continuous availability.

Different types of NoSQL databases?

- **Document databases**

A Document Data Model is a lot different than other data models because it stores data in JSON, BSON, or XML documents. In this data model, we can move documents under one document and apart from this, any particular elements can be indexed to run queries faster. Often documents are stored and retrieved in such a way that it becomes close to the data objects which are used in many applications which means very less translations are required to use data in applications. JSON is a native language that is often used to store and query data too.

So in the document data model, each document has a key-value pair below is an example for the same.

```
{
"Name" : "Vishal",
"Address" : "Delhi",
"Email" : "vishal@ineuron.ai",
"Contact" : "12345"
}
```

Working of Document Data Model:

This is a data model which works as a semi-structured data model in which the records and data associated with them are stored in a single document which means this data model is not completely unstructured. The main thing is that data here is stored in a document.

Features:

- **Document Type Model:** As we all know data is stored in documents rather than tables or graphs, so it becomes easy to map things in many programming languages.
- **Flexible Schema:** Overall schema is very much flexible to support this statement one must know that not all documents in a collection need to have the same fields.
- **Distributed and Resilient:** Document data models are very much dispersed which is the reason behind horizontal scaling and distribution of data.

- **Manageable Query Language:** These data models are the ones in which query language allows the developers to perform CRUD (Create Read Update Destroy) operations on the data model.

Examples of Document Data Models :

- Amazon DocumentDB
- MongoDB
- Cosmos DB
- ArangoDB
- CouchDB

Advantages:

- **Schema-less:** These are very good in retaining existing data at massive volumes because there are absolutely no restrictions in the format and the structure of data storage.
- **Faster creation of document and maintenance:** It is very simple to create a document and apart from this maintenance requires is almost nothing.
- **Open formats:** It has a very simple build process that uses XML, JSON, and its other forms.
- **Built-in versioning:** It has built-in versioning which means as the documents grow in size there might be a chance they can grow in complexity. Versioning decreases conflicts.

Disadvantages:

- **Weak Atomicity:** It lacks in supporting multi-document ACID transactions. A change in the document data model involving two collections will require us to run two separate queries i.e. one for each collection. This is where it breaks atomicity requirements.
- **Consistency Check Limitations:** One can search the collections and documents that are not connected to an author collection but doing this might create a problem in the performance of database performance.
- **Security:** Nowadays many web applications lack security which in turn results in the leakage of sensitive data. So it becomes a point of concern, one must pay

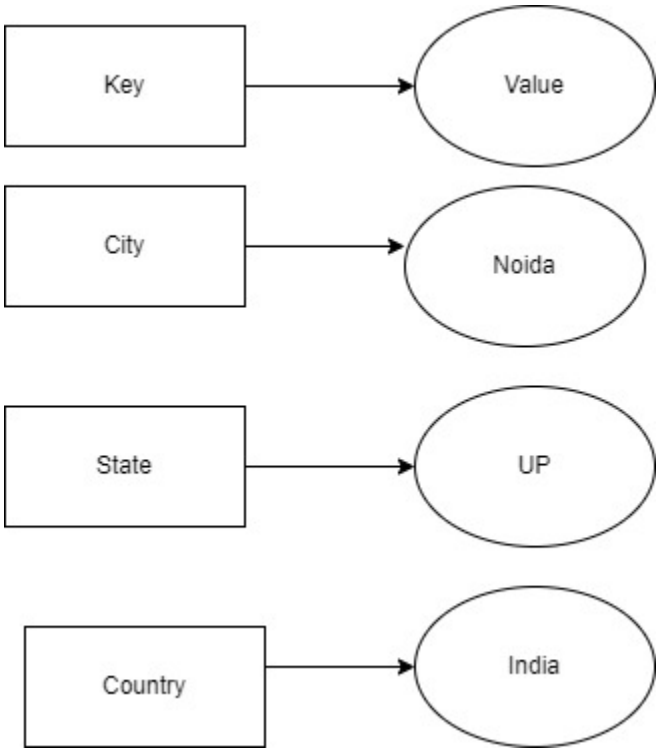
attention to web app vulnerabilities.

Applications of Document Data Model :

- **Content Management:** These data models are very much used in creating various video streaming platforms, blogs, and similar services Because each is stored as a single document and the database here is much easier to maintain as the service evolves over time.
- **Book Database:** These are very much useful in making book databases because as we know this data model lets us nest.
- **Catalog:** When it comes to storing and reading catalog files these data models are very much used because it has a fast reading ability if incase Catalogs have thousands of attributes stored.
- **Analytics Platform:** These data models are very much used in the Analytics Platform.

● Key-value databases

A key-value data model or database is also referred to as a key-value store. It is a non-relational type of database. In this, an associative array is used as a basic database in which an individual key is linked with just one value in a collection. For the values, keys are special identifiers. Any kind of entity can be valued. The collection of key-value pairs stored on separate records is called key-value databases and they do not have an already defined structure.



How do key-value databases work?

A number of easy strings or even a complicated entity are referred to as a value that is associated with a key by a key-value database, which is utilized to monitor the entity. Like in many programming paradigms, a key-value database resembles a map object or array, or dictionary, however, which is put away in a tenacious manner and controlled by a DBMS.

An efficient and compact structure of the index is used by the key-value store to have the option to rapidly and dependably find value using its key. For example, Redis is a key-value store used to track lists, maps, heaps, and primitive types (which are simple data structures) in a constant database. Redis can uncover a very basic point of interaction to query and manipulate value types, just by supporting a predetermined number of value types, and when arranged, is prepared to do high throughput.

When to use a key-value database:

- User session attributes in an online app like finance or gaming, which is referred to as real-time random data access.
- Caching mechanism for repeatedly accessing data or key-based design.
- The application is developed on queries that are based on keys.

Features:

- One of the most un-complex kinds of NoSQL data models.
- For storing, getting, and removing data, key-value databases utilize simple functions.
- Querying language is not present in key-value databases.
- Built-in redundancy makes this database more reliable.

Advantages:

- It is very easy to use. Due to the simplicity of the database, data can accept any kind, or even different kinds when required.
- Its response time is fast due to its simplicity, given that the remaining environment near it is very much constructed and improved.
- Key-value store databases are scalable vertically as well as horizontally.
- Built-in redundancy makes this database more reliable.

Disadvantages:

- As querying language is not present in key-value databases, transportation of queries from one database to a different database cannot be done.
- The key-value store database is not refined. You cannot query the database without a key.

Some examples of key-value databases:

Here are some popular key-value databases which are widely used:

- **Couchbase:** It permits SQL-style querying and searching for text.
- **Amazon DynamoDB:** The key-value database which is mostly used is Amazon DynamoDB as it is a trusted database used by a large number of users. It can easily handle a large number of requests every day and it also provides various security options.
- **Riak:** It is the database used to develop applications.
- **Aerospike:** It is an open-source and real-time database working with billions of exchanges.

- **Berkeley DB:** It is a high-performance and open-source database providing scalability.

● Column-oriented databases / Columnar database

A columnar database is used in a database management system (**DBMS**) which helps to store data in columns rather than rows. It is responsible for speeding up the time required to return a particular query. It also is responsible for greatly improving the disk I/O performance. It is helpful in data analytics and data warehousing. The major motive of Columnar Database is to effectively read and write data. Here are some examples for Columnar Database like Monet DB, Apache Cassandra, SAP Hana, Amazon Redshift.

Columnar Database VS Row Database:

Both Columnar and Row databases are a few methods used for processing big data analytics and data warehousing. But their approach is different from each other.

For example:

- Row Database: “Customer 1: Name, Address, Location.”(The fields for each new record are stored in a long row).
- Columnar Database: “Customer 1: Name, Address, Location.”(Each field has its own set of columns).

Example:

Here is an example of a simple database table with four columns and three rows.

ID Number	Last Name	First Name	Bonus
534782	Miller	Ginny	6000
585523	Parker	Peter	8000
479148	Stacy	Gwen	2000

In a Columnar DBMS, the data stored is in this format:

```
534782, 585523, 479148; Miller, Parker, Stacy; Ginny, Peter, Gwen; 6000, 8000, 2000.
```

In a Row-oriented DBMS, the data stored is in this format:

```
534782, Miller, Ginny, 6000; 585523, Parker, Peter, 8000; 479148, Stacy, Gwen, 2000.
```

When to use the Columnar Database:

1. Queries that involve only a few columns.
2. Compression but column-wise only.
3. Clustering queries against a huge amount of data.

Advantages of Columnar Database:

1. Columnar databases can be used for different tasks such as when the applications that are related to big data comes into play then the column-oriented databases have greater attention in such case.
2. The data in the columnar database has a highly compressible nature and has different operations like (AVG), (MIN, MAX), which are permitted by the compression.
3. Efficiency and Speed: The speed of Analytical queries that are performed is faster in columnar databases.
4. Self-indexing: Another benefit of a column-based DBMS is self-indexing, which uses less disk space than a relational database management system containing the same data.

Limitation of Columnar Database:

1. For loading incremental data, traditional databases are more relevant as compared to column-oriented databases.
2. For Online transaction processing (OLTP) applications, Row oriented databases are more appropriate than columnar databases.

● **Graph databases**

A graph database is a type of NoSQL database that is designed to handle data with complex relationships and interconnections. In a graph database, data is stored as nodes and edges, where nodes represent entities and edges represent the relationships between those entities.

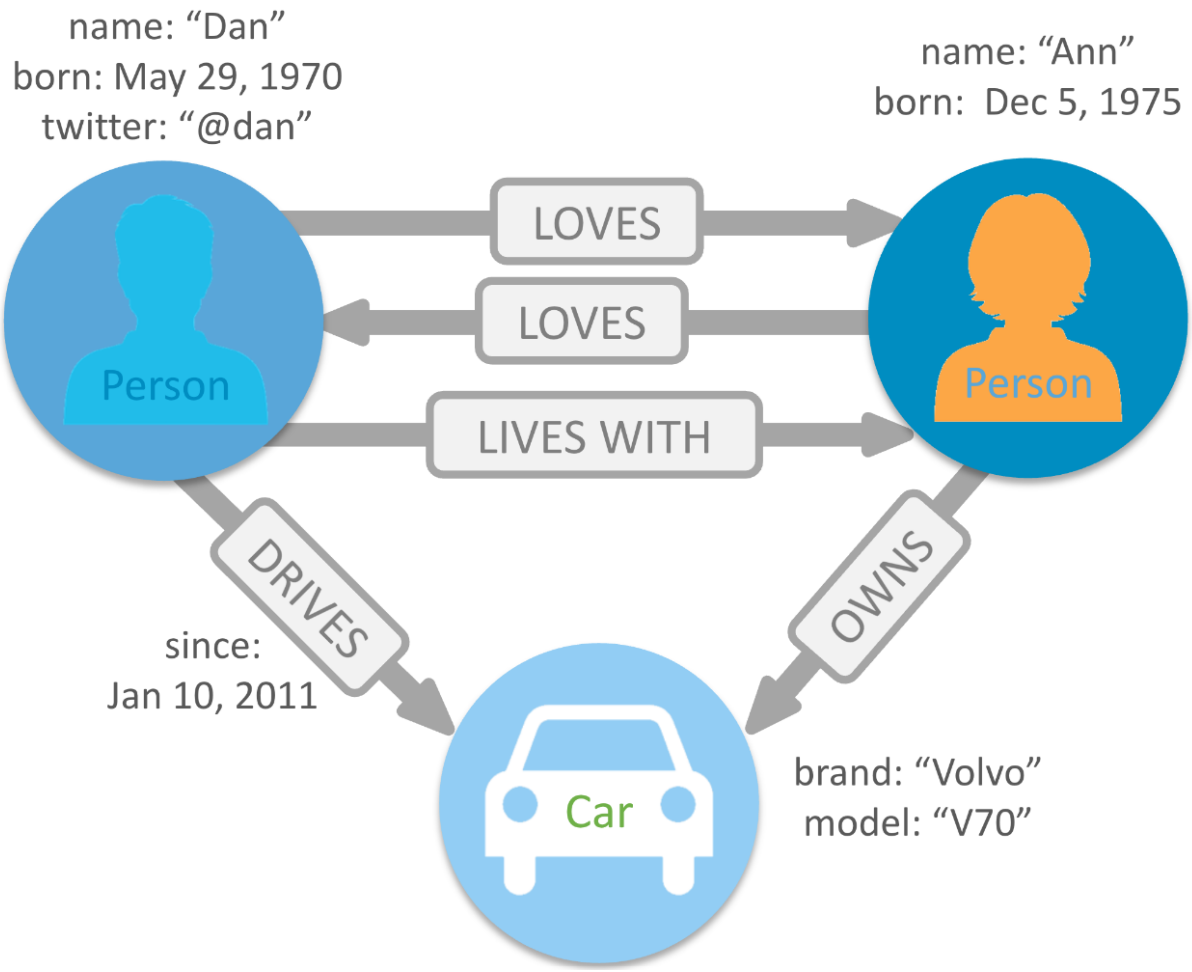
1. Graph databases are particularly well-suited for applications that require deep and complex queries, such as social networks, recommendation engines, and fraud detection systems. They can also be used for other types of applications, such as supply chain management, network and infrastructure management, and bioinformatics.
2. One of the main advantages of graph databases is their ability to handle and represent relationships between entities. This is because the relationships between entities are as important as the entities themselves, and often cannot be easily represented in a traditional relational database.
3. Another advantage of graph databases is their flexibility. Graph databases can handle data with changing structures and can be adapted to new use cases without requiring significant changes to the database schema. This makes them particularly useful for applications with rapidly changing data structures or complex data requirements.
4. However, graph databases may not be suitable for all applications. For example, they may not be the best choice for applications that require simple queries or that deal primarily with data that can be easily represented in a traditional relational database. Additionally, graph databases may require more specialized knowledge and expertise to use effectively.

Some popular graph databases include Neo4j, OrientDB, and ArangoDB. These databases provide a range of features, including support for different data models, scalability, and high availability, and can be used for a wide variety of applications.

As we all know the graph is a pictorial representation of data in the form of nodes and relationships which are represented by edges. A graph database is a type of database used to represent the data in the form of a graph. It has three components: nodes, relationships, and properties. These components are used to model the data. The concept of a Graph Database is based on the theory of graphs. It was introduced in the year 2000. They are commonly referred to **NoSql** databases as data is stored using nodes, relationships and properties instead of traditional databases. A graph database is very useful for heavily interconnected data. Here relationships between data are given priority and therefore the relationships can be easily visualized. They are flexible as new data can be added without hampering the old ones. They are useful in the fields of social networking, fraud detection, **AI Knowledge** graphs etc.

The description of components are as follows:

- **Nodes:** represent the objects or instances. They are equivalent to a row in database. The node basically acts as a vertex in a graph. The nodes are grouped by applying a label to each member.
- **Relationships:** They are basically the edges in the graph. They have a specific direction, type and form patterns of the data. They basically establish relationship between nodes.
- **Properties:** They are the information associated with the nodes.



Some examples of Graph Databases software are Neo4j, Oracle NoSQL DB, Graph base etc. Out of which Neo4j is the most popular one.

In traditional databases, the relationships between data is not established. But in the case of Graph Database, the relationships between data are prioritized. Nowadays mostly interconnected data is used where one data is connected directly or indirectly. Since the concept of this database is based on graph theory, it is flexible and works very fast for associative data. Often data are interconnected to one another which also helps to establish further relationships. It works fast in the querying part as well because with the help of relationships we can quickly find the desired nodes. Join operations are not required in this database which reduces the cost. The relationships and properties are stored as first-class entities in Graph Database.

Graph databases allow organizations to connect the data with external sources as well. Since organizations require a huge amount of data, often it becomes cumbersome to store data in the form of tables. For instance, if the organization wants to find a particular data that is connected with another data in another table, so first join operation is performed between the tables, and then search for the data is done row by row. But Graph database solves this big problem. They store the relationships and properties along with the data. So if the organization needs to search for a particular data, then with the help of relationships and properties the nodes can be found without joining or without traversing row by row. Thus the searching of nodes is not dependent on the amount of data.

Types of Graph Databases:

- **Property Graphs:** These graphs are used for querying and analyzing data by modelling the relationships among the data. It comprises of vertices that has information about the particular subject and edges that denote the relationship. The vertices and edges have additional attributes called properties.
- **RDF Graphs:** It stands for Resource Description Framework. It focuses more on data integration. They are used to represent complex data with well defined semantics. It is represented by three elements: two vertices, an edge that reflect the subject, predicate and object of a sentence. Every vertex and edge is represented by URI(Uniform Resource Identifier).

When to Use Graph Database?

- Graph databases should be used for heavily interconnected data.
- It should be used when amount of data is larger and relationships are present.

- It can be used to represent the cohesive picture of the data.

How Graph and Graph Databases Work?

Graph databases provide graph models They allow users to perform traversal queries since data is connected. Graph algorithms are also applied to find patterns, paths and other relationships this enabling more analysis of the data. The algorithms help to explore the neighboring nodes, clustering of vertices analyze relationships and patterns. Countless joins are not required in this kind of database.

Example of Graph Database:

- Recommendation engines in E commerce use graph databases to provide customers with accurate recommendations, updates about new products thus increasing sales and satisfying the customer’s desires.
- Social media companies use graph databases to find the “friends of friends” or products that the user’s friends like and send suggestions accordingly to user.
- To detect fraud Graph databases play a major role. Users can create graph from the transactions between entities and store other important information. Once created, running a simple query will help to identify the fraud.

Advantages of Graph Database:

- Potential advantage of Graph Database is establishing the relationships with external sources as well
- No joins are required since relationships is already specified.
- Query is dependent on concrete relationships and not on the amount of data.
- It is flexible and agile.
- it is easy to manage the data in terms of graph.
- Efficient data modeling: Graph databases allow for efficient data modeling by representing data as nodes and edges. This allows for more flexible and scalable data modeling than traditional relational databases.
- Flexible relationships: Graph databases are designed to handle complex relationships and interconnections between data elements. This makes them well-suited for applications that require deep and complex queries, such as social networks, recommendation engines, and fraud detection systems.

- High performance: Graph databases are optimized for handling large and complex datasets, making them well-suited for applications that require high levels of performance and scalability.
- Scalability: Graph databases can be easily scaled horizontally, allowing additional servers to be added to the cluster to handle increased data volume or traffic.
- Easy to use: Graph databases are typically easier to use than traditional relational databases. They often have a simpler data model and query language, and can be easier to maintain and scale.

Disadvantages of Graph Database:

- Often for complex relationships speed becomes slower in searching.
- The query language is platform dependent.
- They are inappropriate for transactional data
- It has smaller user base.
- Limited use cases: Graph databases are not suitable for all applications. They may not be the best choice for applications that require simple queries or that deal primarily with data that can be easily represented in a traditional relational database.
- Specialized knowledge: Graph databases may require specialized knowledge and expertise to use effectively, including knowledge of graph theory and algorithms.
- Immature technology: The technology for graph databases is relatively new and still evolving, which means that it may not be as stable or well-supported as traditional relational databases.
- Integration with other tools: Graph databases may not be as well-integrated with other tools and systems as traditional relational databases, which can make it more difficult to use them in conjunction with other technologies.
- Overall, graph databases on NoSQL offer many advantages for applications that require complex and deep relationships between data elements. They are highly flexible, scalable, and performant, and can handle large and complex datasets. However, they may not be suitable for all applications, and may require specialized knowledge and expertise to use effectively.

Future of Graph Database:

Graph Database is an excellent tool for storing data but it cannot be used to completely replace the traditional database. This database deals with a typical set of interconnected data. Although Graph Database is in the developmental phase it is becoming an important part as business and organizations are using big data and Graph databases help in complex analysis. Thus these databases have become a must for today's needs and tomorrow success.

When NoSQL databases are used?

The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. Big data and real-time web apps both use NoSQL. Every day, firms like Twitter, Facebook, and Google, for example, collect gigabytes of user data.

Advantages of NoSQL databases

The limitations of traditional relational database technology prompted the development of NoSQL databases. NoSQL databases are frequently more scalable and give better performance than relational databases.

● **Handle large columns of data at high speed with scale-out architecture** - The most common way to construct SQL databases is to use a scale-up architecture, which is based on leveraging
larger machines with more CPUs and memory to boost speed. In the Internet and cloud computing eras, NoSQL databases were developed, making it easier to construct a scale-out architecture.
Scalability is achieved in a scale-out architecture by distributing data storage and processing operations over a large cluster of computers. More computers are added to the cluster to boost capacity.

● **Store unstructured, semi-structured or structured data** - NoSQL databases have become popular because they allow data to be stored in more understandable or similar ways to how it is used by applications. When data is saved or retrieved for use, fewer changes are necessary. Many various types of data can be saved and accessed more simply, whether structured, unstructured, or
semi-structured. Furthermore, many NoSQL databases' schemas are flexible and under the developers' control, making it easier to adapt the database to new types of data. This eliminates

inefficiencies in the development process caused by requesting a SQL database be redesigned by a database administrator.

● **Enable easy updates to schema and fields** - NoSQL databases have become popular because they store data in simple straightforward forms that can be easier to understand than the type of data models used in SQL databases. Furthermore, NoSQL databases frequently allow developers to update the data structure directly. Document databases don’t have a set data structure to start with, so a new document type can be stored just as easily as what is currently being stored. New values and columns can be added to key-value and column-oriented stores without affecting the current structure. Developers of graph databases update nodes with new characteristics and arcs with new meanings in response to new types of data.

● **Developer-friendly** - Developers have been the primary drivers of NoSQL database adoption, as they find it easier to design many types of applications than with relational databases. JSON is used
by document databases like MongoDB to transform data into something that resembles code. This gives the developer complete control over the data's structure. Furthermore, NoSQL databases store data in forms that are similar to the types of data objects used in applications, requiring fewer transformations when moving data in and out.

Disadvantages of NoSQL databases

● Not all NoSQL databases take atomicity and data integrity into consideration. They can resist what is referred to as "ultimate consistency."

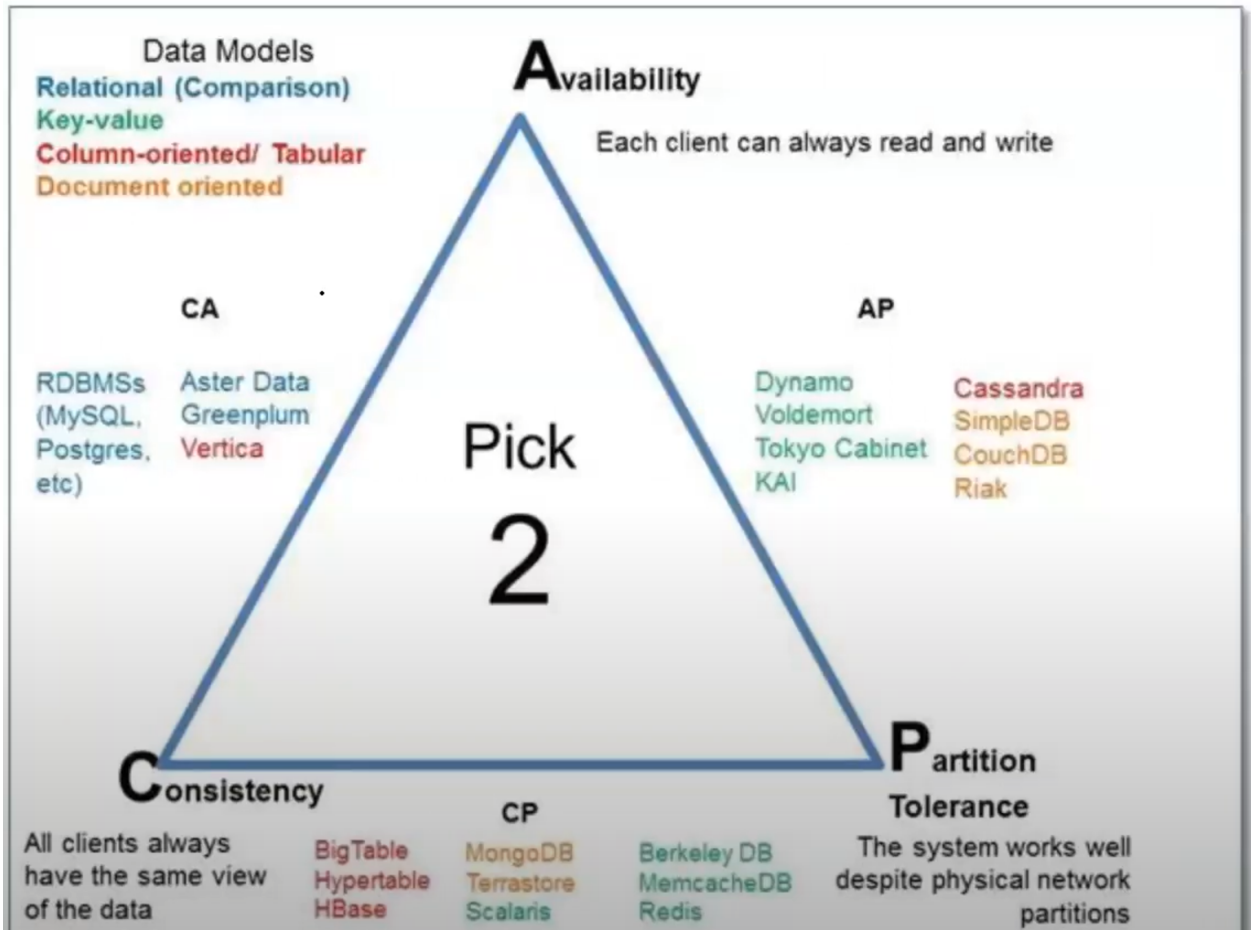
● SQL instructions have compatibility concerns. The query language of new databases has its own peculiarities, and it is not yet 100 percent compatible with SQL used in relational databases. Support for work query issues in a NoSQL database are more complicated.

● There is a lack of standardization. There are various NoSQL databases, but none of them follow the same standards as relational databases. These databases are expected to have an
unclear future.

● Support for multiple platforms, some systems still require significant changes in order to work on non-Linux operating systems.

● Usability is a problem. They frequently have access to consoles or management tools that aren't really useful.

CAP Theorem



HBase

HBase is a type of "NoSQL" database. "NoSQL" is a general term meaning that the database isn't an RDBMS which supports SQL as its primary access language, but there are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database. Technically speaking, HBase is really more a "Data Store" than "Data Base" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc.

However, HBase has many features which supports both linear and modular scaling. HBase clusters expand by adding RegionServers that are hosted on commodity class servers. If a cluster expands from 10 to 20 RegionServers, for example, it doubles both in terms of storage and as well as processing capacity. An RDBMS can scale well, but only up to a point - specifically, the size of a single database server - and for the best

performance requires specialized hardware and storage devices. HBase features of note are:

- Strongly consistent reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic RegionServer failover
- Hadoop/HDFS Integration: HBase supports HDFS as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

Disadvantages of HBase

1. No support SQL structure
2. No transaction support
3. Sorted only on key
4. Memory issues on the cluster

When should we use HBase?

HBase isn’t suitable for every problem.

First, make sure you have enough data. If you have hundreds of millions or billions of rows, then HBase is a good candidate. If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle.

Second, make sure you can live without all the extra features that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.) An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver, for example. Consider moving from an RDBMS to HBase as a complete redesign as opposed to a port.

Third, make sure you have enough hardware. Even HDFS doesn’t do well with anything less than 5 DataNodes (due to things such as HDFS block replication which has a default of 3), plus a NameNode.

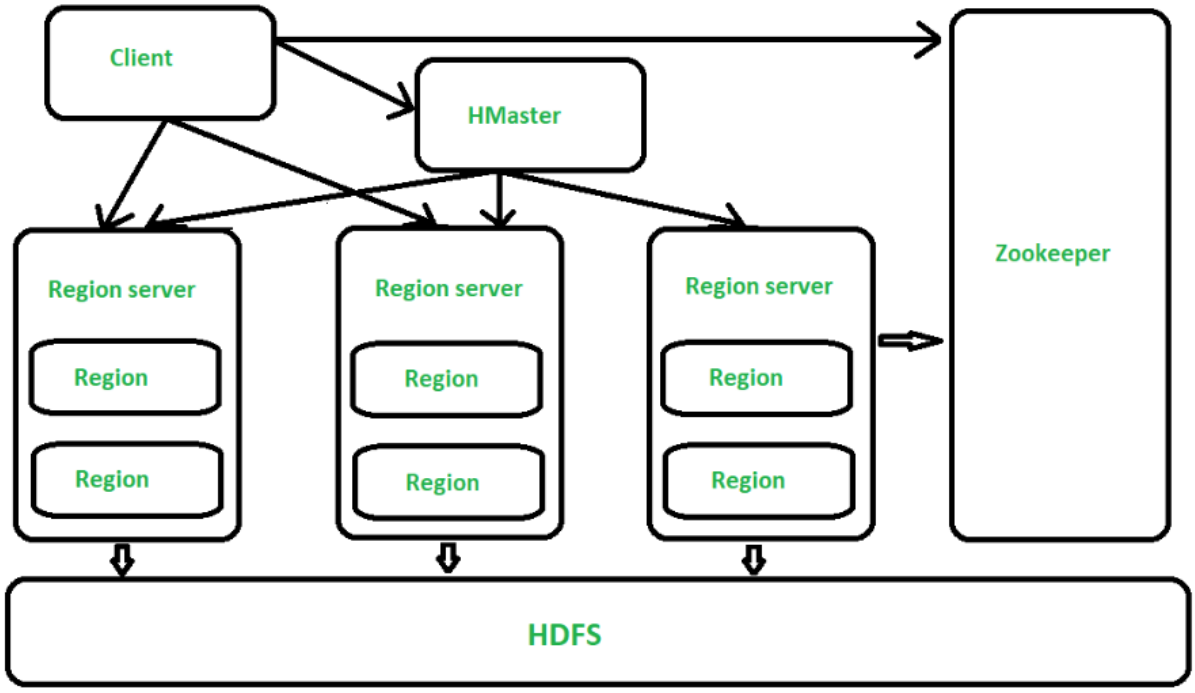
HBase can run quite well stand-alone on a laptop - but this should be considered a development configuration only.

What Is The Difference Between HBase and Hadoop/HDFS?

HDFS is a distributed file system that is well suited for the storage of large files. Its documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed lookups. See the Data Model and the rest of this chapter for more information on how HBase achieves its goals.

HBase Architecture

HBase architecture has 3 main components: HMaster, Region Server, Zookeeper.



All the 3 components are described below:

HBase has three major components: the client library, a master server, and region servers. Region servers can be added or removed as per requirement.

MasterServer

The master server -

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Regions

Regions are nothing but tables that are split up and spread across the region servers.

Region server

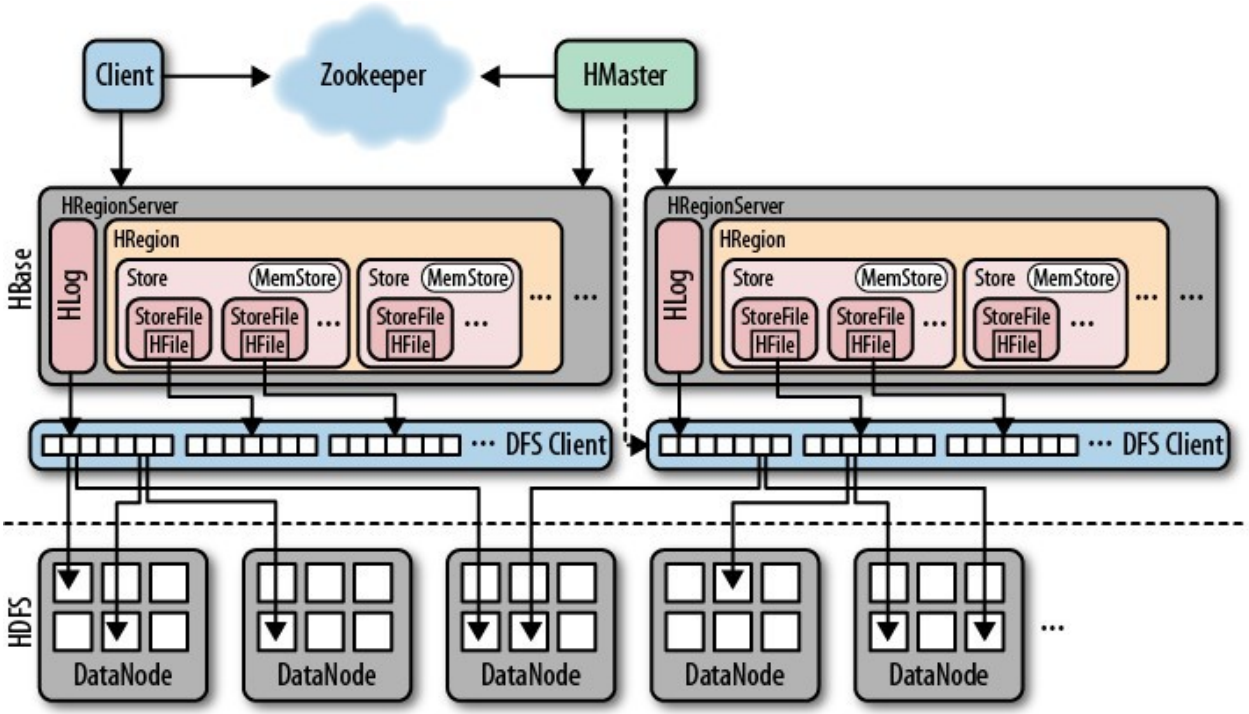
The region servers have regions that -

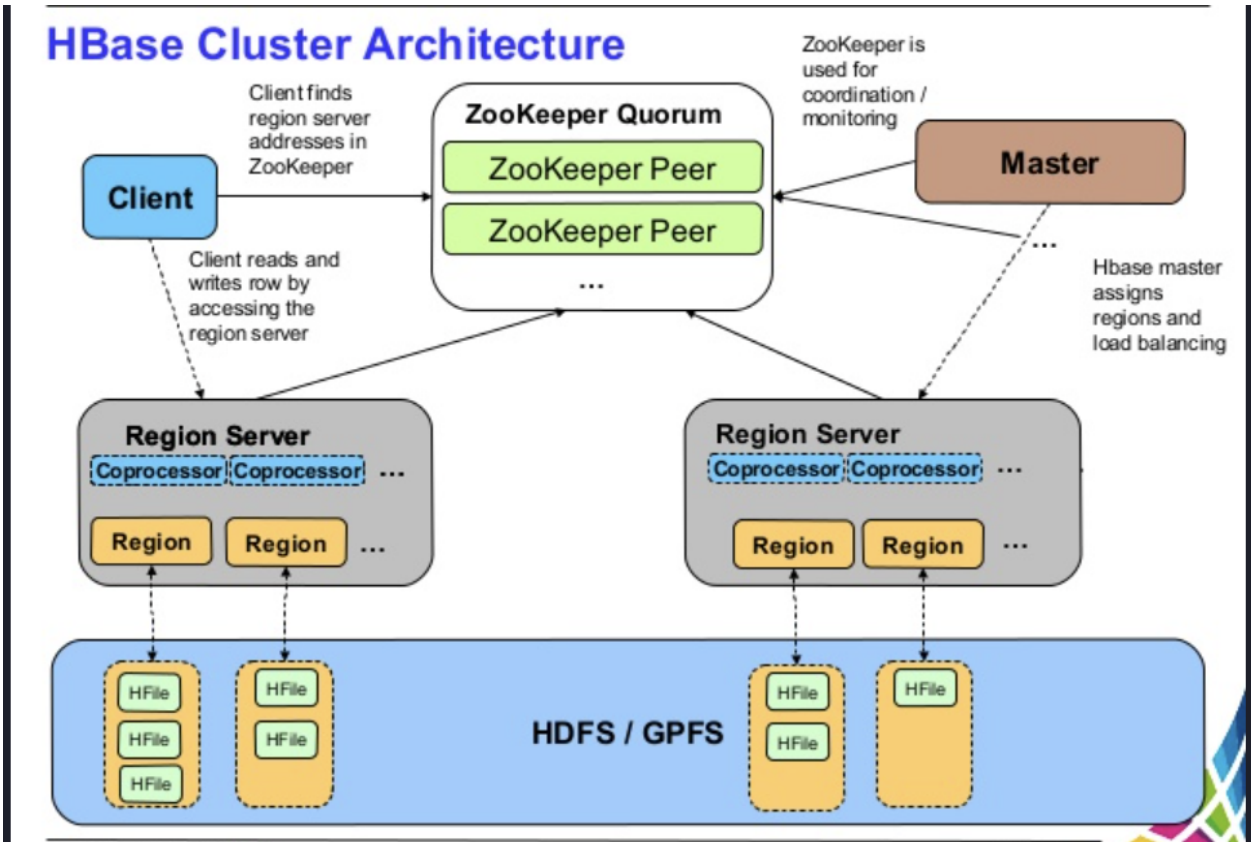
- Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Decide the size of the region by following the region size thresholds.

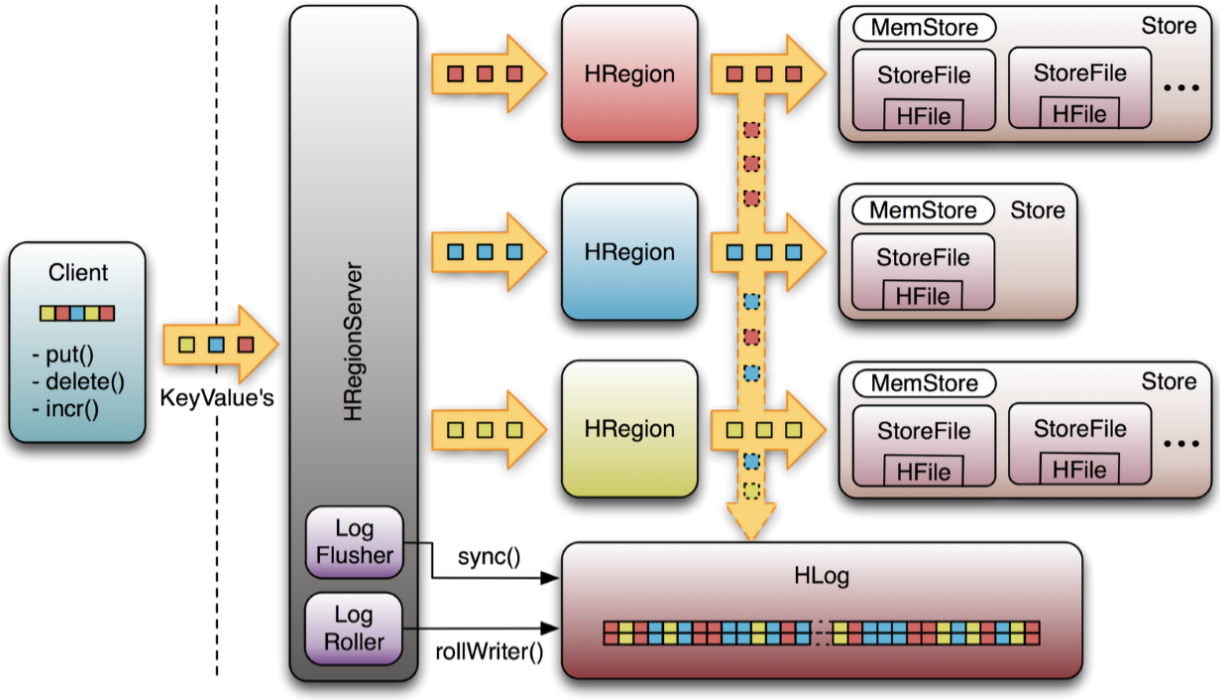
Zookeeper

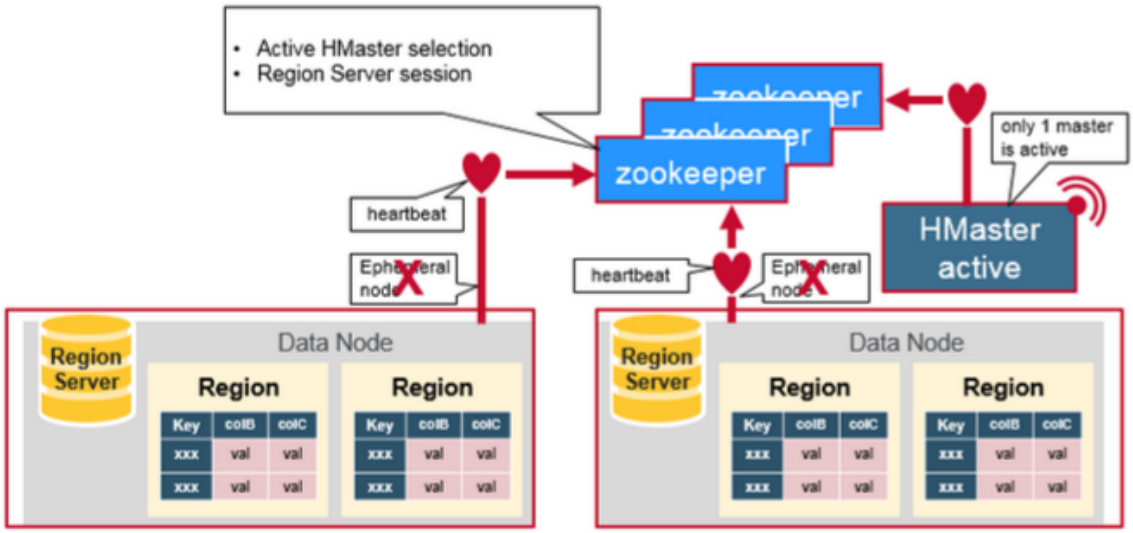
- Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In pseudo and standalone modes, HBase itself will take care of zookeeper.

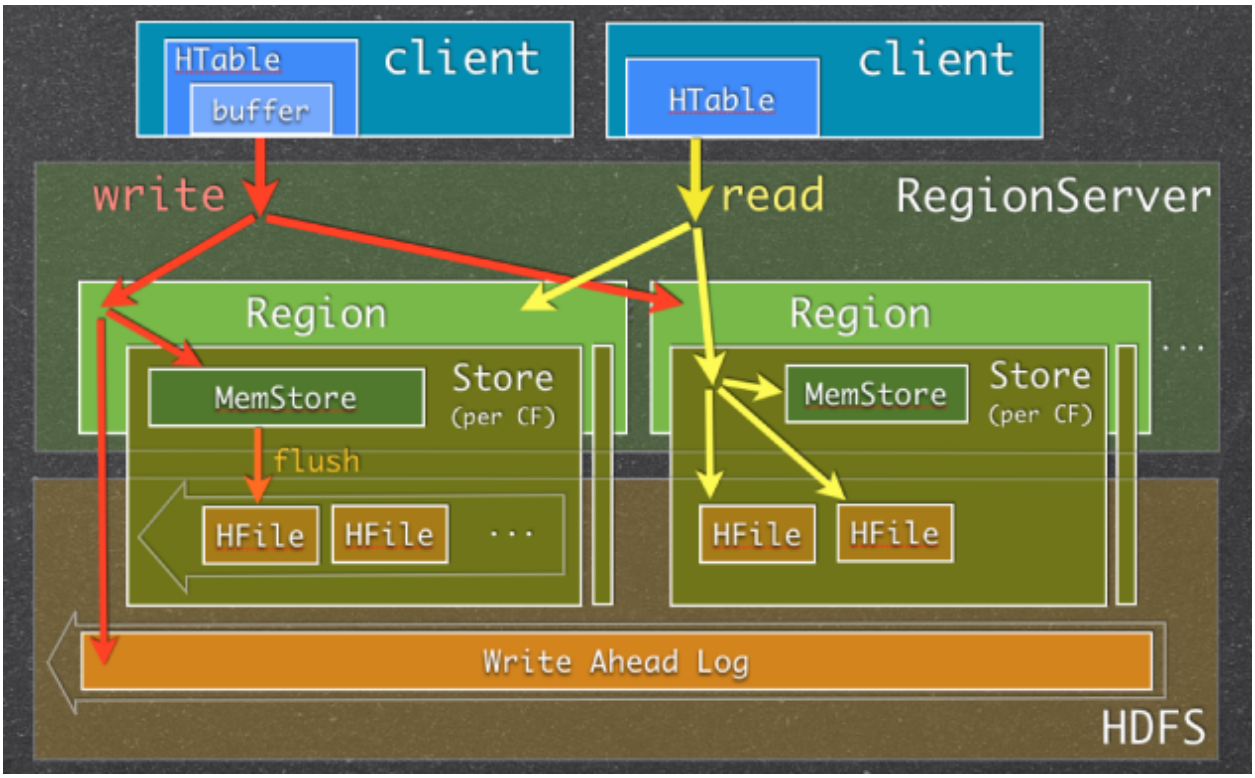
HBase Architecture in detail




HBase Cluster Architecture
Client finds region server addresses in ZooKeeper
ZooKeeper Quorum
ZooKeeper Peer
ZooKeeper Peer
...
ZooKeeper is used for coordination / monitoring
Client
Master
Client reads and writes row by accessing the region server
Hbase master assigns regions and load balancing
Region Server
Coprocessor
Coprocessor
...
Region
Region
...
Region Server
Coprocessor
Coprocessor
...
Region
Region
...
HFile
HFile
HFile
HFile
HFile
HDFS / GPFS
HFile
HFile
HFile


Client
- put()
- delete()
- incr()
KeyValue's
HRegionServer
Log Flusher
Log Roller
sync()
rollWriter()
HRegion
HRegion
HRegion
MemStore
Store
StoreFile
HFile
StoreFile
HFile
...
MemStore
Store
StoreFile
HFile
MemStore
Store
StoreFile
HFile
StoreFile
HFile
...
HLog


Active HMaster selection
Region Server session
zookeeper
zookeeper
zookeeper
only 1 master is active
HMaster active
heartbeat
heartbeat
Ephemeral node
Ephemeral node
Region Server
Data Node
Region
Region
Key
colB
colC
xxx
val
Region Server
Data Node
Region
Region


HTable
buffer
client
HTable
client
write
read
RegionServer
Region
Region
MemStore
Store
(per CF)
flush
HFile
HFile
...
MemStore
Store
(per CF)
HFile
HFile
Write Ahead Log
HDFS

HBase Data Model – LOGICAL VIEW

- **Table**
  - Contains column-families
- **Column family**
  - Logical and physical grouping of columns. Basic storage unit.
- **Column**
  - Exists only when inserted
  - Can have multiple versions
  - Each row can have different set of columns
  - Each column identified by it's key
- **Row key**
  - Implicit primary key
  - Used for storing ordered rows
  - Efficient queries using row key

HBTABLE	
Row key	**Value**
11111	cf_data: {'cq_name': 'name1', 'cq_val': 1111} cf_info: {'cq_desc': 'desc11111'}
22222	cf_data: {'cq_name': 'name2', 'cq_val': 2013 @ ts = 2013, 'cq_val': 2012 @ ts = 2012 }

HFile

11111	cf_data	cq_name	name1	@ ts1
11111	cf_data	cq_val	1111	@ ts1
22222	cf_data	cq_name	name2	@ ts1
22222	cf_data	cq_val	2013	@ ts1
22222	cf_data	cq_val	2012	@ ts2

HFile

11111	cf_info	cq_desc	desc11111	@ ts1

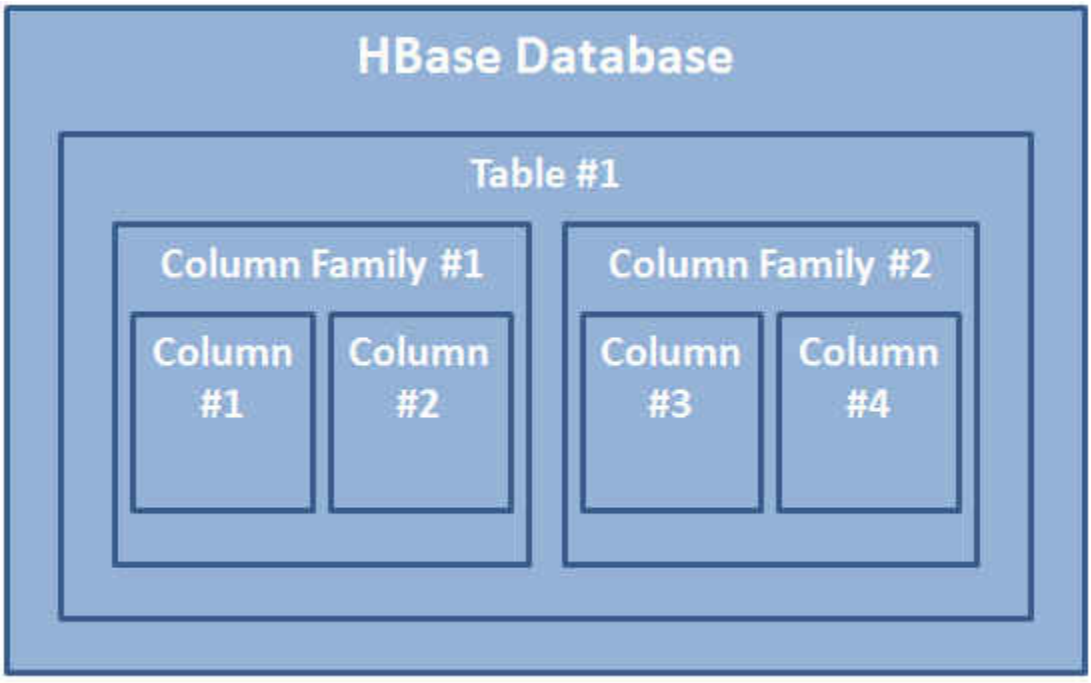
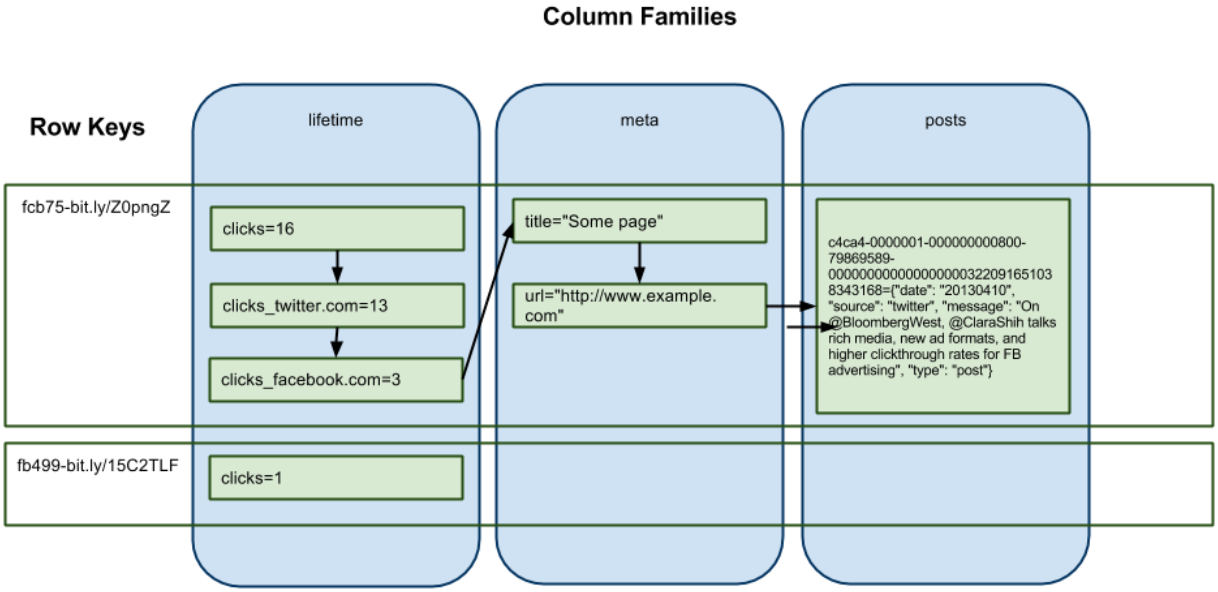
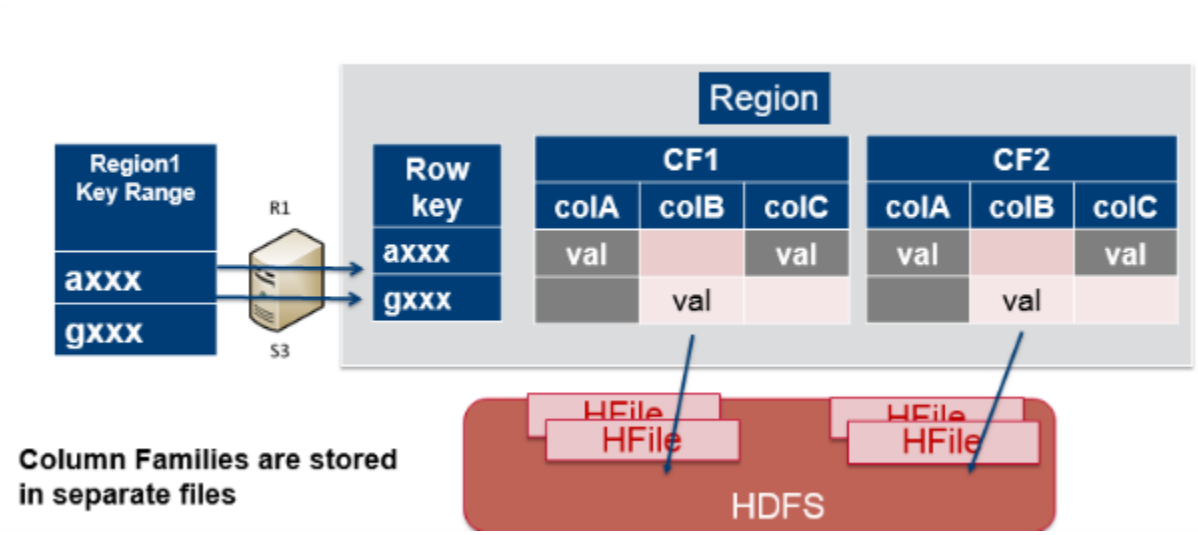


Figure 1 - HBase Data Organization


Column Families
Row Keys
lifetime
meta
posts
fcb75-bit.ly/Z0pngZ
clicks=16
clicks_twitter.com=13
clicks_facebook.com=3
title="Some page"
url="http://www.example.com"
c4ca4-0000001-000000000800-79869589-000000000000000000032209165103 8343168={"date": "20130410", "source": "twitter", "message": "On @BloombergWest, @ClaraShih talks rich media, new ad formats, and higher clickthrough rates for FB advertising", "type": "post"}
fb499-bit.ly/15C2TLF
clicks=1


Region
Region1
Key Range
axxx
gxxx
R1
S3
Row
key
axxx
gxxx
CF1
colA
colB
colC
val
val
val
CF2
colA
colB
colC
val
val
val
HFile
HFile
HFile
HFile
HDFS
Column Families are stored
in separate files

Row Oriented
(RDBMS Model)

Multi-valued

id	Name	Age	Interests
1	Ricky		Soccer, Movies, Baseball
2	Ankur	20	
3	Sam	25	Music

null

Column Oriented
(Multi-value sorted map)

id	Name
1	Ricky
2	Ankur
3	Sam

id	Age
2	20
3	25

id	Interests
1	Soccer
1	Movies
1	Baseball
3	Music

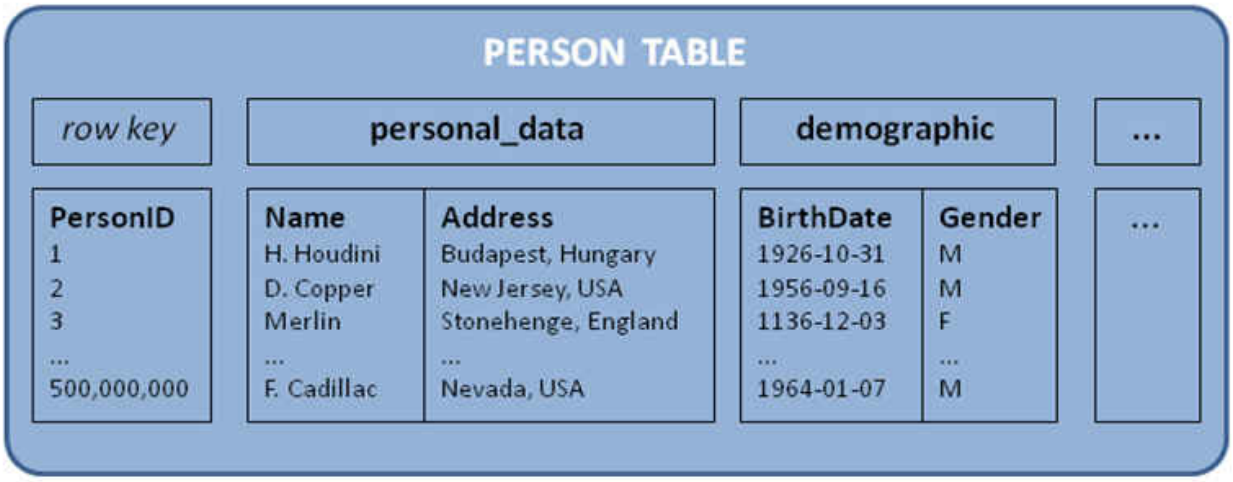


Figure 2 - Census Data in Column Families

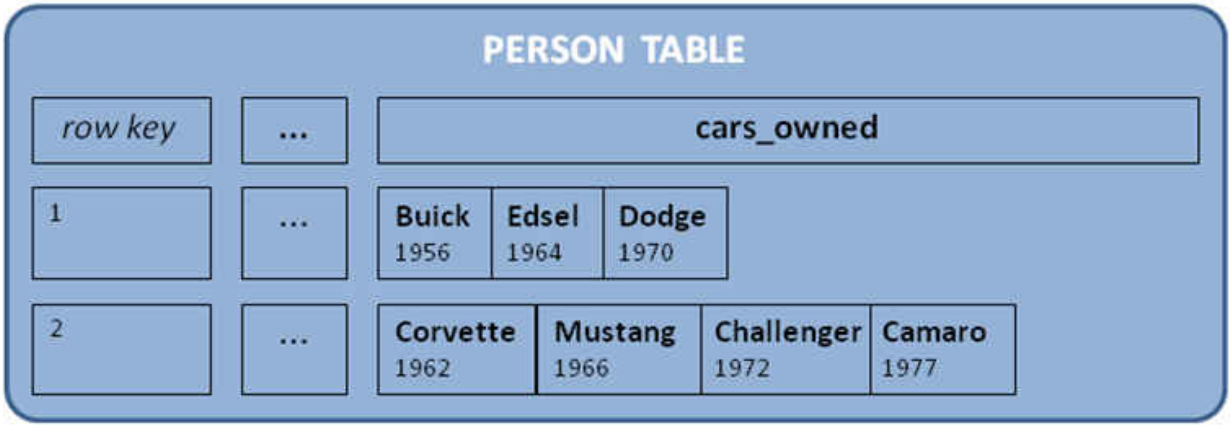
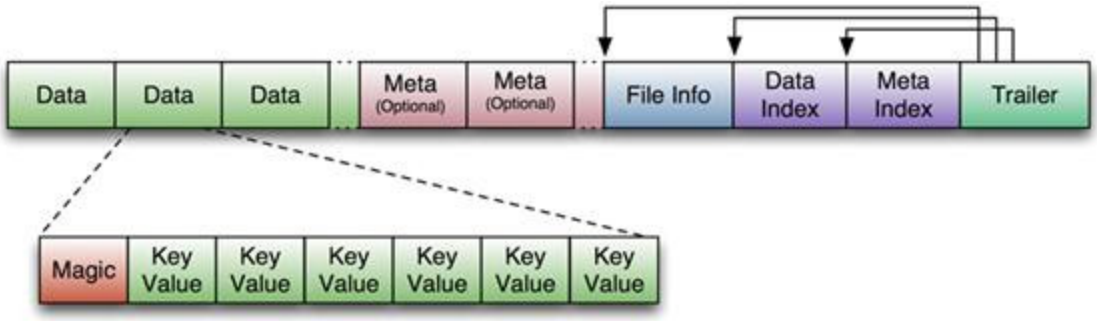


Figure 3 - Data Column Names



Data Model

In HBase, data is stored in tables, which have rows and columns. This is a terminology overlap with relational databases (RDBMSs), but this is not a helpful analogy. Instead, it can be helpful to think of an HBase table as a multi-dimensional map.

- **Table**

An HBase table consists of multiple rows.

- **Row**

A row in HBase consists of a row key and one or more columns with values associated with them. Rows are sorted alphabetically by the row key as they are stored. For this reason, the design of the row key is very important. The goal is to store data in such a way that related rows are near each other. A common row key pattern is a website domain. If your row keys are domains, you should probably store them in reverse (org.apache.www, org.apache.mail, org.apache.jira). This way, all of the Apache

domains are near each other in the table, rather than being spread out based on the first letter of the subdomain.

- **Column**

A column in HBase consists of a column family and a column qualifier, which are delimited by a `:` (colon) character.

- **Column Family**

Column families physically colocate a set of columns and their values, often for performance reasons. Each column family has a set of storage properties, such as whether its values should be cached in memory, how its data is compressed or its row keys are encoded, and others. Each row in a table has the same column families, though a given row might not store anything in a given column family.

- **Column Qualifier**

A column qualifier is added to a column family to provide the index for a given piece of data. Given a column family `content`, a column qualifier might be `content:html`, and another might be `content:pdf`. Though column families are fixed at table creation, column qualifiers are mutable and may differ greatly between rows.

- **Cell**

A cell is a combination of row, column family, and column qualifier, and contains a value and a timestamp, which represents the value's version.

- **Timestamp**

A timestamp is written alongside each value, and is the identifier for a given version of a value. By default, the timestamp represents the time on the RegionServer when the data was written, but you can specify a different timestamp value when you put data into the cell.

HBase commands

- To open HBase shell

```
hbase shell
```

- List all the tables in HBase

```
list
```

- Return the status of the system including the details of the servers running on the system

```
status
```

- Return the version of HBase used in your system

```
version
```

- Get help with the HBase commands

```
table_help
```

- Return the user details of the user

```
whoami
```

- Creating table in HBase

You can create a table using the **create**
command, here you must specify the table name and the Column Family name.
The **syntax**
to create a table in HBase shell is shown below.

```
create ‘<table name>’,’<column family>’

create 'emp', 'personal_data', 'professional_data'
```

This code will create a table like this

Row key	personal data	professional data

Let’s verify if we successfully create table or not

```
list
```

- To delete a table or change its settings, you need to first disable the table using the disable command.

```
disable 'emp'
```

Even after disabling if we do

```
list
```

we can get the table. So to verify we will have to check via

```
scan 'emp'
```

- Check is a table is disabled or not

```
is_disabled 'emp'
```

- Disabling all the tables

```
disable_all 'c.*'
```

- Enabling a table

```
enable 'emp'
```

Let’s verify

```
is_disabled 'emp'
```

or

```
scan 'emp'
```

- Check if enabled or not

```
is_enabled 'emp'
```

- Get the description of table

```
describe 'emp'
```

- Altering table

Alter is the command used to make changes to an existing table. Using this command, you can change the maximum number of cells of a column family, set and delete table scope operators, and delete a column family from a table.

```
alter 'emp', NAME => "Personal_Data", VERSIONS => 5
```

This will add a new column family to the “emp” table

- Setting a table to READONLY

```
alter 'emp',{METHOD=>'table_att', READONLY=>true}
```

- Unsetting a table from READONLY

```
alter 'emp',{METHOD=>'table_att_unset', NAME=>READONLY}
```

- Deleting a column family

```
alter 'emp','delete' => 'Personal_Data'
```

- Check if a table exists or not

```
exists 'emp'

exists 'test'
```

- Drop a table

```
drop 'emp'
```

It will throw us an error that the table enable.

```
disable 'emp'

drop 'emp'
```

- Drop multiple tables

```
drop_all 'e.*'
```

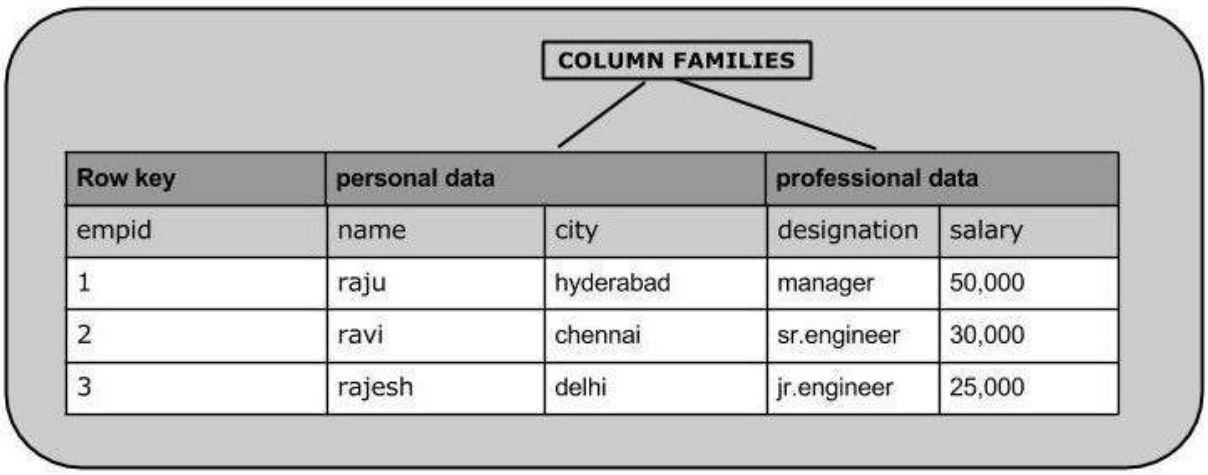
- Exiting of the shell

```
exit

CTRL + c
```

- Inserting data into table

```
put '<table name>','row1','<colfamily:colname>','<value>'
```



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

```
put 'emp','1','personal_data:name','Raju'

put 'emp','1','personal_data:city','hyd'

put 'emp','1','professional_data:designation','DE'

put 'emp','1','professional_data:salary','50,000'

put 'emp','2','personal_data:name','Ravi'

put 'emp','2','personal_data:city','chennai'

put 'emp','2','professional_data:designation','associate DE'

put 'emp','2','professional_data:salary','30,000'

put 'emp','3','personal_data:name','Rajesh'

put 'emp','3','personal_data:city','delhi'

put 'emp','3','professional_data:designation','jr DE'

put 'emp','3','professional_data:salary','25,000'
```

```
scan 'emp'
```

- Reading the data

```
# getting data from a specific row
get 'emp', '1'

# getting data from specific column family
get 'emp', '1', 'personal_data'

# getting data from specific column
get 'emp', '1', 'personal_data:name'
```

- Updating data in table

```
get 'emp', '1'

put 'emp','1','personal_data:city','Delhi'
```

- Deleting data from table

```
# delete a specific column
delete 'emp', '1', 'personal_data:city'

get 'emp', '1'

# delete a specific row id
deleteall 'emp', '1'

scan 'emp'
```

- Count the number of rows of a table

```
count 'emp'
```

- Truncating table

```
truncate 'emp'

scan 'emp'
```

- Grant acess

We can grant and revoke permissions to users in HBase. There are three commands for security purpose: grant, revoke, and user_permission. The **grant** command grants specific rights such as read, write, execute, and admin on a table to a certain user. The syntax of grant command is as follows:

```
grant <user> <permissions> [<table> [<column family> [<column; qualifier>]]
```

We can grant zero or more privileges to a user from the set of RWXCA, where

1. R - represents read privilege.
2. W - represents write privilege.
3. X - represents execute privilege.
4. C - represents create privilege.
5. A - represents admin privilege.

Given below is an example that grants all privileges to a user named ‘vs17’.

```
grant 'vs17', 'RWXCA'
```

- Revoke access

The **revoke** command is used to revoke a user's access rights of a table. Its syntax is as follows:

```
revoke <user>
```

The following code revokes all the permissions from the user named ‘vs17’.

```
revoke 'vs17'
```

user_permission

This command is used to list all the permissions for a particular table. The syntax of **user_permission** is as follows:

```
user_permission ‘tablename’
```

The following code lists all the user permissions of ‘emp’ table.

```
user_permission 'emp'
```