**Assignment 9.3:**

Explain the below concepts with an example in brief.

1. **Nosql Databases:**

A NoSQL database provides a mechanism for [storage](https://en.wikipedia.org/wiki/Computer_data_storage) and [retrieval](https://en.wikipedia.org/wiki/Data_retrieval) of data that is modeled in means other than the tabular relations used in [relational databases](https://en.wikipedia.org/wiki/Relational_database).

The various approaches to classify NoSQL databases, each with different categories and subcategories, some of which overlap. What follows is a basic classification by data model, with examples:

* [Column](https://en.wikipedia.org/wiki/Column_(data_store)): [Accumulo](https://en.wikipedia.org/wiki/Accumulo" \o "Accumulo), [Cassandra](https://en.wikipedia.org/wiki/Apache_Cassandra), [Druid](https://en.wikipedia.org/wiki/Druid_(open-source_data_store)), [HBase](https://en.wikipedia.org/wiki/HBase), [Vertica](https://en.wikipedia.org/wiki/Vertica), [SAP HANA](https://en.wikipedia.org/wiki/SAP_HANA)
* [Document](https://en.wikipedia.org/wiki/Document-oriented_database): [Apache CouchDB](https://en.wikipedia.org/wiki/Apache_CouchDB), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), [Clusterpoint](https://en.wikipedia.org/wiki/Clusterpoint" \o "Clusterpoint), [Couchbase](https://en.wikipedia.org/wiki/Couchbase" \o "Couchbase), [Cosmos DB](https://en.wikipedia.org/wiki/Cosmos_DB), [HyperDex](https://en.wikipedia.org/wiki/HyperDex" \o "HyperDex), [IBM Domino](https://en.wikipedia.org/wiki/Lotus_Notes), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic" \o "MarkLogic), [MongoDB](https://en.wikipedia.org/wiki/MongoDB), [OrientDB](https://en.wikipedia.org/wiki/OrientDB" \o "OrientDB), [Qizx](https://en.wikipedia.org/wiki/Qizx" \o "Qizx), [RethinkDB](https://en.wikipedia.org/wiki/RethinkDB" \o "RethinkDB), RavenDB
* [**Key-value**](https://en.wikipedia.org/wiki/Key-value_store): [Aerospike](https://en.wikipedia.org/wiki/Aerospike_database), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), [Couchbase](https://en.wikipedia.org/wiki/Couchbase" \o "Couchbase), [Dynamo](https://en.wikipedia.org/wiki/Dynamo_(storage_system)), FairCom [c-treeACE](https://en.wikipedia.org/wiki/C-treeACE), [FoundationDB](https://en.wikipedia.org/wiki/FoundationDB), [HyperDex](https://en.wikipedia.org/wiki/HyperDex), [InfinityDB](https://en.wikipedia.org/wiki/InfinityDB), [MemcacheDB](https://en.wikipedia.org/wiki/MemcacheDB), [MUMPS](https://en.wikipedia.org/wiki/MUMPS), [Oracle NoSQL Database](https://en.wikipedia.org/wiki/Oracle_NoSQL_Database), [OrientDB](https://en.wikipedia.org/wiki/OrientDB" \o "OrientDB), [Redis](https://en.wikipedia.org/wiki/Redis" \o "Redis), [Riak](https://en.wikipedia.org/wiki/Riak" \o "Riak), [Berkeley DB](https://en.wikipedia.org/wiki/Berkeley_DB), SDBM/Flat File [dbm](https://en.wikipedia.org/wiki/Dbm" \o "Dbm)
* [**Graph**](https://en.wikipedia.org/wiki/Graph_database): [AllegroGraph](https://en.wikipedia.org/wiki/AllegroGraph" \o "AllegroGraph), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), [InfiniteGraph](https://en.wikipedia.org/wiki/InfiniteGraph" \o "InfiniteGraph), [Apache Giraph](https://en.wikipedia.org/wiki/Apache_Giraph), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic" \o "MarkLogic), [Neo4J](https://en.wikipedia.org/wiki/Neo4J), [OrientDB](https://en.wikipedia.org/wiki/OrientDB" \o "OrientDB), [Virtuoso](https://en.wikipedia.org/wiki/Virtuoso_Universal_Server)
* [**Multi-model**](https://en.wikipedia.org/wiki/Multi-model_database): Alchemy Database, [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), CortexDB, [Couchbase](https://en.wikipedia.org/wiki/Couchbase), [FoundationDB](https://en.wikipedia.org/wiki/FoundationDB), [InfinityDB](https://en.wikipedia.org/wiki/InfinityDB), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic), [OrientDB](https://en.wikipedia.org/wiki/OrientDB)

**Key-value store:**

Key-value (KV) stores use the [associative array](https://en.wikipedia.org/wiki/Associative_array) (also known as a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.

The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented as an extension of it. The key-value model can be extended to a discretely ordered model that maintains keys in [lexicographic order](https://en.wikipedia.org/wiki/Lexicographical_order). This extension is computationally powerful, in that it can efficiently retrieve selective key ranges.

Key-value stores can use [consistency models](https://en.wikipedia.org/wiki/Consistency_model) ranging from [eventual consistency](https://en.wikipedia.org/wiki/Eventual_consistency) to [serializability](https://en.wikipedia.org/wiki/Serializability" \o "Serializability). Some databases support ordering of keys. There are various hardware implementations, and some users maintain data in memory (RAM), while others employ [solid-state drives](https://en.wikipedia.org/wiki/Solid-state_drive) or [rotating disks](https://en.wikipedia.org/wiki/Hard_disk_drive).

Examples include [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB" \o "ArangoDB), [InfinityDB](https://en.wikipedia.org/wiki/InfinityDB" \o "InfinityDB), [Oracle NoSQL Database](https://en.wikipedia.org/wiki/Oracle_NoSQL_Database), [Redis](https://en.wikipedia.org/wiki/Redis" \o "Redis), and [dbm](https://en.wikipedia.org/wiki/Dbm" \o "Dbm).

**Document store:**

Main articles: [Document-oriented database](https://en.wikipedia.org/wiki/Document-oriented_database) and [XML database](https://en.wikipedia.org/wiki/XML_database)

The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, [YAML](https://en.wikipedia.org/wiki/YAML), and [JSON](https://en.wikipedia.org/wiki/JSON) as well as binary forms like [BSON](https://en.wikipedia.org/wiki/BSON). Documents are addressed in the database via a unique key that represents that document. One of the other defining characteristics of a document-oriented database is that in addition to the key lookup performed by a key-value store, the database offers an API or query language that retrieves documents based on their contents.

Different implementations offer different ways of organizing and/or grouping documents:

* Collections
* Tags
* Non-visible metadata
* Directory hierarchies

Compared to relational databases, for example, collections could be considered analogous to tables and documents analogous to records. But they are different: every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

**Graph:**

This kind of database is designed for data whose relations are well represented as a [graph](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)) consisting of elements interconnected with a finite number of relations between them. The type of data could be social relations, public transport links, road maps or network topologies

1. **Types of Nosql Databases:**

There are 4 basic types of NoSQL databases:

1. Key-Value Store – It has a Big Hash Table of keys & values {Example- Riak, Amazon S3 (Dynamo)}
2. Document-based Store- It stores documents made up of tagged elements. {Example- CouchDB}
3. Column-based Store- Each storage block contains data from only one column, {Example- HBase, Cassandra}
4. Graph-based-A network database that uses edges and nodes to represent and store data. {Example- Neo4J}
5. **CAP Theorem:**

**Consistency** - This means that the data in the database remains consistent after the

execution of an operation. For example after an update operation, all clients see the

same data.

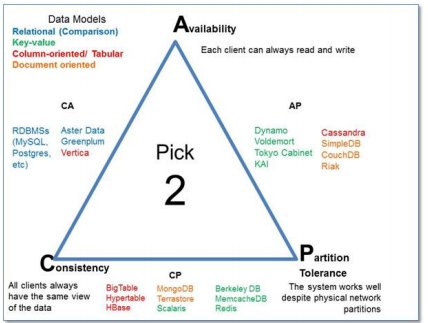
**Availability** - This means that the system is always on (service guarantee availability),

no downtime.

**Partition Tolerance** - This means that the system continues to function even if the

communication among the servers is unreliable, i.e. the servers may be partitioned into

multiple groups that cannot communicate with one another.



1. **HBase Architecture:**

HBase is composed of three types of servers in a master slave type of architecture.

* Region servers serve data for reads and writes.
* HBase Master process handles the Region assignment, DDL (create, delete tables)

Operations.

* Zookeeper maintains a live cluster state.
* The Hadoop DataNode stores the data that the Region Server is managing.
* All HBase data is stored in HDFS files.
* The NameNode maintains metadata information for all the physical data blocks that

comprise the files.

**Regions:**

• HBase Tables are divided horizontally by row key range into “Regions.”

• A region contains all rows in the table between the region’s start key and end key.

• Regions are assigned to the nodes in the cluster, called “Region Servers,” and these serve

data for reads and writes.

• A region server can serve about 1,000 regions.

**HBase HMaster:**

• Region assignment, DDL (create, delete tables) operations are handled by the HBase

Master.

A master is responsible for:

• Coordinating the region servers

• Assigning regions on startup

• Re-assigning regions for recovery or load balancing

• Monitoring all RegionServer instances in the cluster (listens for notifications from

zookeeper)

**Admin functions**

• Interface for creating, deleting, updating tables

**Zookeeper: The Coordinator**

• HBase uses ZooKeeper as a distributed coordination service to maintain server state in

the cluster.

• Zookeeper maintains which servers are alive and available, and provides server failure

notification.

• Zookeeper uses consensus to guarantee common shared state. Note that there should be

three or five machines for consensus.

**HBase Meta Table:**

• This META table is an HBase table that keeps a list of all regions in the system.

• The .META. table is like a b tree.

• The .META. table structure is as follows:

Key: region start key, region id

Values: RegionServer

**Region Server Components:**

• Region Server runs on an HDFS data node and has the following components:

WAL

• Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.

BlockCache

• It is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.

MemStore

• It is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.

Hfiles

• They store the rows as sorted KeyValues on disk.

1. **HBase vs RDBMS**

|  |  |
| --- | --- |
| **RDBMS** | **HBASE** |
| RDBMS is row-oriented databases | HBase is a distributed, column-oriented data storage system |
| RDBMS tables have fixed-schema | Hbase tables do not have fixed-schema  (Flexible schema – add columns on the fly) |
| RDBMS tables guarantee ACID properties | Hbase tables guarantee consistency and partition tolerance |
| RDBMS uses SQL (Structured query Language ) to query the data | Hbase uses Java client API |
| RDBMS are optimized for Joins | In HBase, Joins using MR and are not optimized |
| RDBMS are good for structured data | HBase are good for semi-structured data as well as un-structured data |