# Explain the below concepts with an example in brief.

# Nosql databases:

A *NoSQL* (originally referring to "non SQL" or "non relational") *database* provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational *databases.* These databases donot use SQL like queries. Generally, NoSQL databases are structured in a key-value pair, graph database, document-oriented or column-oriented structure.

Applications such as Facebook, Google, Amazon, Watsapp, etc. gave rise to an entire new era of database management which follows approach of simple design, speed and faster scaling than the traditional databases. Such databases are used in big data, massive real time applications and analytics.

As an example, consider that you have a blogging application that stores user blogs. Now suppose that you have to incorporate some new features in your application such as users liking these blog posts or commenting on them or liking these comments. With a typical RDBMS implementation, this will need a complete overhaul to your existing database design. However, if you use NoSQL in such scenarios, you can easily modify your data structure to match these agile requirements. With NoSQL you can directly start inserting this new data in your existing structure without creating any new pre-defined columns or pre-defined structure.

### Types of NoSQL Databases

**Document Oriented Databases**   
Document oriented databases treat a document as a whole and avoid splitting a document in its constituent name/value pairs. At a collection level, this allows for putting together a diverse set of documents into a single collection. Document databases allow indexing of documents on the basis of not only its primary identifier but also its properties. Different open-source document databases are available today but the most prominent among the available options are MongoDB and CouchDB. In fact, MongoDB has become one of the most popular NoSQL databases.

**Graph Based Databases**  
A graph database uses graph structures with nodes, edges, and properties to represent and store data. By definition, a graph database is any storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent element and no index lookups are necessary. General graph databases that can store any graph are distinct from specialized graph databases such as triple-stores and network databases. Indexes are used for traversing the graph. For example-Neo4j, OrientDB, Facebook Open Graph, FlockDB, etc.

**Column Based Databases**  
The column-oriented storage allows data to be stored effectively. It avoids consuming space when storing nulls by simply not storing a column when a value doesn’t exist for that column. Each unit of data can be thought of as a set of key/value pairs, where the unit itself is identified with the help of a primary identifier, often referred to as the primary key. Bigtable and its clones tend to call this primary key the row-key. For example-CouchDB, OrientDB, etc.

**Key Value Databases**  
The key of a key/value pair is a unique value in the set and can be easily looked up to access the data. Key/value pairs are of varied types: some keep the data in memory and some provide the capability to persist the data to disk. A simple, yet powerful, key/value store is Oracle’s Berkeley DB. For example-Membase, Redis, MemcacheDB, etc.

## CAP Theorem

Credit: http://architects.dzone.com/articles/better-explaining-cap-theorem
 

**Consistency** – All the servers in the system will have the same data so anyone using the system will get the same copy regardless of which server answers their request.

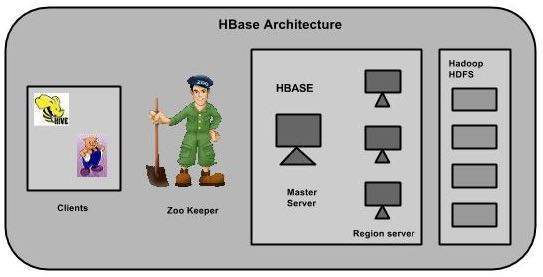
**Availability –** The system will always respond to a request (even if it's not the latest data or consistent across the system or just a message saying the system isn't working)

**Partition Tolerance** – The system continues to operate as a whole even if individual servers fail or can't be reached.

# Hbase Architecture:

In HBase, tables are split into regions and are served by the region servers. Regions are vertically divided by column families into “Stores”. Stores are saved as files in HDFS. Shown below is the architecture of HBase.

**Note:** The term ‘store’ is used for regions to explain the storage structure.



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.

When we take a deeper look into the region server, it contain regions and stores as shown below:



The store contains memory store and HFiles. Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

## 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 and HDFS

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | **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. | |