**Assignment 9.3**

**Explain the below concepts with an example in brief.**

**● Nosql Databases**

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean “not SQL” or “not only SQL.” As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

* **Types of Nosql Databases**

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

* **Key-value data stores:**[Key-value NoSQL databases](http://basho.com/resources/nosql-databases/) emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned *and replicated*across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.
* **Document stores:**[Document databases](http://basho.com/resources/document-databases/) typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.
* **Wide-column stores:**Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.
* **Graph stores:**A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

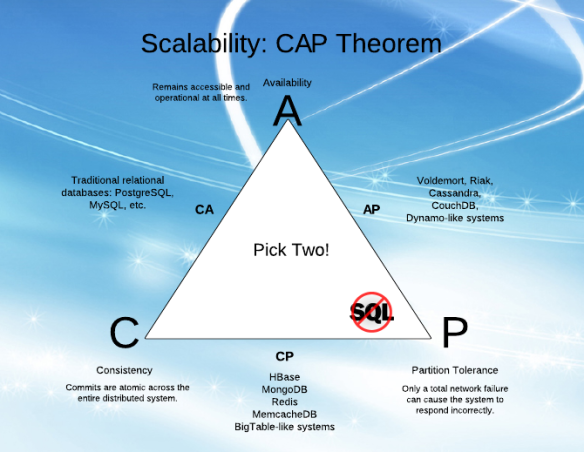
**Multi-modal databases**leverage some combination of the four types described above and therefore can support a wider range of applications

* **CAP Theorem**

[CAP theorem](http://en.wikipedia.org/wiki/CAP_theorem), that - so people say - states that you can only choose 2 out of the 3

* Consistency: every read would get you the most recent write
* Availability: every node (if not failed) always executes queries
* Partition-tolerance: even if the connections between nodes are down, the other two (A & C) promises, are kept.

Usually its depicted in a nicely equilaterl triangle, as this one from [Ofirm](http://ofirm.wordpress.com/2013/01/22/classical-big-data-reading-cap-theorem/" \l "more-65):



There's a nice proof and explanation of it in this 4 minute video  [here](http://www.youtube.com/watch?v=Jw1iFr4v58M). But if we think about it, and also see some of Brewer's (the theorem author) later  [remarks](http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed), we'll see that the 2 out of 3 is really 1 out of 2:

**It's really just A vs C!**

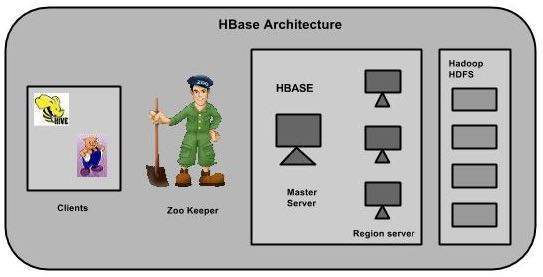
And this is simply because:

1. Availability is achieved by replicating the data across different machines
2. Consistency is achieved by updating several nodes before allowing further reads
3. Total partitioning, meaning failure of part of the system is rare. However, we could look at a delay, a latency, of the update between nodes, as a *temporary partitioning*. It will then cause a temporary decision between A and C:
   1. On systems that allow reads before updating all the nodes, we will get high Availability
   2. On systems that lock all the nodes before allowing reads, we will get Consistency.

* **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 vs RDBMS**

| Relational database | HBase |
| --- | --- |
| This supports scale up. In other words, when more disk and memory processing power is needed, we need to upgrade it to a more powerful server. | This supports scale out. In other words, when more disk and memory processing power is needed, we need not upgrade the server. However, we need to add new servers to the cluster. |
| This uses SQL queries for reading records from tables. | This uses APIs and MapReduce for accessing data from HBase tables. |
| This is row oriented, that is, each row is a contiguous unit of page. | This is column oriented, that is, each column is a contiguous unit of page. |
| The amount of data depends on configuration of server. | The amount of data does not depend on the particular machine but the number of machines. |
| It's Schema is more restrictive. | Its schema is flexible and less... |