**Advanced HBASE Assignment.**

**Task1**

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

**● Nosql Databases**

NoSQL is an approach to [database](https://searchsqlserver.techtarget.com/definition/database) design that can accomodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stand for "not only [SQL](https://searchsqlserver.techtarget.com/definition/SQL)," is an alternative to traditional relational databases in which data is placed in tables and data [schema](https://searchsqlserver.techtarget.com/definition/schema) is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data.

The Benefits of NoSQL

When compared to relational databases, NoSQL databases are [more scalable and provide superior performance,](https://www.mongodb.com/scale) and their data model addresses several issues that the relational model is not designed to address:

* Large volumes of rapidly changing structured, semi-structured, and unstructured data
* Agile sprints, quick schema iteration, and frequent code pushes
* Object-oriented programming that is easy to use and flexible
* Geographically distributed scale-out architecture instead of expensive, monolithic architecture

**● Types of Nosql Databases**

* There have been 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), [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).
* [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), [BaseX](https://en.wikipedia.org/wiki/BaseX), [Clusterpoint](https://en.wikipedia.org/wiki/Clusterpoint), [Couchbase](https://en.wikipedia.org/wiki/Couchbase), [Cosmos DB](https://en.wikipedia.org/wiki/Cosmos_DB), [IBM Domino](https://en.wikipedia.org/wiki/Lotus_Notes), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic), [MongoDB](https://en.wikipedia.org/wiki/MongoDB), [OrientDB](https://en.wikipedia.org/wiki/OrientDB), [Qizx](https://en.wikipedia.org/wiki/Qizx), [RethinkDB](https://en.wikipedia.org/wiki/RethinkDB)
* [Key-value](https://en.wikipedia.org/wiki/Key-value_store): [Aerospike](https://en.wikipedia.org/wiki/Aerospike_database), [Apache Ignite](https://en.wikipedia.org/wiki/Apache_Ignite), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB), [Couchbase](https://en.wikipedia.org/wiki/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), [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), [Redis](https://en.wikipedia.org/wiki/Redis), [Riak](https://en.wikipedia.org/wiki/Riak), [Berkeley DB](https://en.wikipedia.org/wiki/Berkeley_DB), SDBM/Flat File [dbm](https://en.wikipedia.org/wiki/Dbm), [ZooKeeper](https://en.wikipedia.org/wiki/Apache_ZooKeeper)
* [Graph](https://en.wikipedia.org/wiki/Graph_database): [AllegroGraph](https://en.wikipedia.org/wiki/AllegroGraph), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB), [InfiniteGraph](https://en.wikipedia.org/wiki/InfiniteGraph), [Apache Giraph](https://en.wikipedia.org/wiki/Apache_Giraph), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic), [Neo4J](https://en.wikipedia.org/wiki/Neo4J), [OrientDB](https://en.wikipedia.org/wiki/OrientDB), [Virtuoso](https://en.wikipedia.org/wiki/Virtuoso_Universal_Server)
* [Multi-model](https://en.wikipedia.org/wiki/Multi-model_database): [Apache Ignite](https://en.wikipedia.org/wiki/Apache_Ignite), [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB), [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), [Cosmos DB](https://en.wikipedia.org/wiki/Cosmos_DB)
* A more detailed classification is the following, based on one from Stephen Yen:[[20]](https://en.wikipedia.org/wiki/NoSQL#cite_note-20)[[21]](https://en.wikipedia.org/wiki/NoSQL#cite_note-21)

| * **Type** | **Notable examples of this type** |
| --- | --- |
| * Key-Value Cache | [Apache Ignite](https://en.wikipedia.org/wiki/Apache_Ignite), [Coherence](https://en.wikipedia.org/wiki/Oracle_Coherence), [eXtreme Scale](https://en.wikipedia.org/wiki/IBM_WebSphere_eXtreme_Scale), [Hazelcast](https://en.wikipedia.org/wiki/Hazelcast), [Infinispan](https://en.wikipedia.org/wiki/Infinispan), [Memcached](https://en.wikipedia.org/wiki/Memcached), [Velocity](https://en.wikipedia.org/wiki/Velocity_(memory_cache)) |
| * Key-Value Store | [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB), [Aerospike](https://en.wikipedia.org/wiki/Aerospike_database) |
| * Key-Value Store (Eventually-Consistent) | [Oracle NoSQL Database](https://en.wikipedia.org/wiki/Oracle_NoSQL_Database), [Dynamo](https://en.wikipedia.org/wiki/Dynamo_(storage_system)), [Riak](https://en.wikipedia.org/wiki/Riak), [Voldemort](https://en.wikipedia.org/wiki/Voldemort_(distributed_data_store)) |
| * Key-Value Store (Ordered) | [FoundationDB](https://en.wikipedia.org/wiki/FoundationDB), [InfinityDB](https://en.wikipedia.org/wiki/InfinityDB), [LMDB](https://en.wikipedia.org/wiki/Lightning_Memory-Mapped_Database), [MemcacheDB](https://en.wikipedia.org/wiki/MemcacheDB) |
| * Data-Structures Server | [Redis](https://en.wikipedia.org/wiki/Redis) |
| * Tuple Store | [Apache River](https://en.wikipedia.org/wiki/Jini), [GigaSpaces](https://en.wikipedia.org/wiki/GigaSpaces) |
| * Object Database | [Objectivity/DB](https://en.wikipedia.org/wiki/Objectivity/DB), [Perst](https://en.wikipedia.org/wiki/Perst), [ZopeDB](https://en.wikipedia.org/wiki/Zope_Object_Database) |
| * Document Store | [ArangoDB](https://en.wikipedia.org/wiki/ArangoDB), [BaseX](https://en.wikipedia.org/wiki/BaseX), [Clusterpoint](https://en.wikipedia.org/wiki/Clusterpoint), [Couchbase](https://en.wikipedia.org/wiki/Couchbase), [CouchDB](https://en.wikipedia.org/wiki/CouchDB), [DocumentDB](https://en.wikipedia.org/wiki/DocumentDB), [IBM Domino](https://en.wikipedia.org/wiki/Lotus_Notes), [MarkLogic](https://en.wikipedia.org/wiki/MarkLogic), [MongoDB](https://en.wikipedia.org/wiki/MongoDB), [Qizx](https://en.wikipedia.org/wiki/Qizx), [RethinkDB](https://en.wikipedia.org/wiki/RethinkDB) |
| * [Wide Column Store](https://en.wikipedia.org/wiki/Wide_column_store) | [Amazon DynamoDB](https://en.wikipedia.org/wiki/Amazon_DynamoDB), [Bigtable](https://en.wikipedia.org/wiki/Bigtable), [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/Apache_HBase), [Hypertable](https://en.wikipedia.org/wiki/Hypertable) |

[Correlation databases](https://en.wikipedia.org/wiki/Correlation_database) are model-independent, and instead of row-based or column-based storage, use value-based storage.

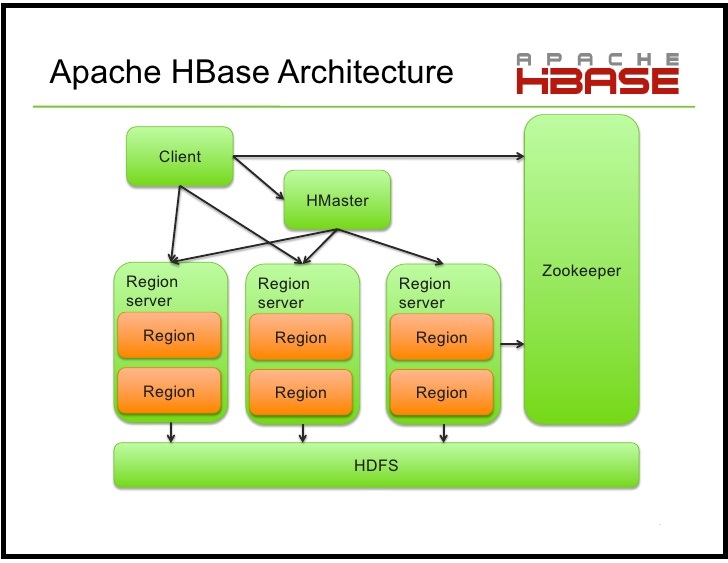
**● CAP Theorem**

For any distributed system, CAP Theorem reiterates the need to find balance between Consistency, Availability and Partition tolerance. Consistency means all the nodes see the same data at the same time. Availability implies that every request receives a response about whether it was successful or failed. It’s more of a handshaking mechanism in computer network methodology.

Coming to partition tolerance, the system continues to operate despite arbitrary message loss or failure of part of the system. Systems with partition tolerance feature works well despite physical network partitions.

According to CAP Theorem distributed systems can satisfy any two features at the same time but not all three features. Traditional systems like RDBMS provide consistency and availability. Column oriented databases like MongoDB, Hbase and Big Table provide features consistency and partition tolerance

**● HBase Architecture**



Hbase architecture consists of mainly HMaster, HRegionserver, HRegions and Zookeeper. Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. If the client wants to communicate with regions servers, client has to approach Zookeeper.

**HMaster**

HMaster in the master server of Hbase and it coordinates the HBase cluster. HMaster is responsible for the administrative operations of the cluster. A region server serves a region at the start of the application. During failure of region server, HMaster assign the region to another Region server. HMaster can also assign a region to another region server as part of load balancing.

**HRegions Servers**

It will perform the following functions in communication with HMaster and Zookeeper.

Hosting and managing regions.

Splitting regions automatically.

Handling read and writes requests.

Communicating with clients directly.

**HRegions**

For each column family, HRegions maintain a store. Main components of HRegions are

Memstore -  Holds in-memory modifications to the store

Hfile

**● HBase vs RDBMS**

|  |  |
| --- | --- |
| **HBASE** | **RDBMS** |
| Schema-less in database. | Having fixed schema in database. |
| Column oriented database. | Row oriented data store. |
| Designed to store De-normalized data. | Designed to store Normalized data. |
| Wide and sparsely populated tables present in Hbase. | Contains thin tables in database. |
| Supports automatic partitioning. | Has no built in support for partitioning. |
| Well suited for OLAP systems. | Well suited for OLTP systems. |
| Read only relevant data from database. | To retrieve one row at a time and hence could read unnecessary data if only some of the data in a row is required. |
| Structured and semi structure data can be stored and processed using Hbase. | Structured data can be stored and processed using an RDBMS. |
| Enables aggregation over many rows and columns. | Aggregation is an expensive operation. |