ACD\_BDD2.3\_Session\_9\_Assignment\_1

1.What is NoSQL data base?

Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today.

NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

Features:-

* Generic Data Model -Heterogeneous containers, including sets, maps, and arrays
* Dynamic type discovery and conversion –NoSQL analytics systems support runtime type identification and conversion so that custom business logic can be used to dictate analytic treatment of variation.
* Non-relational and De-normalised- Data is stored in single tables as compared to joining multiple tables.
* Commodity hardware - Adding more of the economical servers allows NoSQL databases to scale to handle more data.
* Highly distributable - Distributed databases can store and process a set of information on more than one device.

2. How does data get stored in NoSQl database?

There are various NoSQL Databases. Each one uses a different method to store data. Some might use column store, some document, some graph, etc., Each database has its own unique characteristics.

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), [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), [Clusterpoint](https://en.wikipedia.org/wiki/Clusterpoint), [Couchbase](https://en.wikipedia.org/wiki/Couchbase), [Cosmos DB](https://en.wikipedia.org/wiki/Cosmos_DB), [HyperDex](https://en.wikipedia.org/wiki/HyperDex), [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), [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), [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), [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)
* [**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): [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)

3. What is a column family in HBase?

HBase tables are organized by column, rather than by row. The columns are organized in groups called column families. When creating a HBase table, we must define the column families before inserting any data. Column families should not be changed often, nor should there be too many of them, so it is important to think carefully about what column families will be useful for our particular data. Each column family, however, can contain a very large number of columns. Columns are named using the format family:qualifier.

4. How many maximum number of columns can be added to HBase table?

There is no hard limit to number of columns in HBase , we can have more than 1 million columns but usually three column families are recommended ( not more than three).

5. Why columns are not defined at the time of table creation in HBase?

HBase partitions the key space. Each partition is called a Table. Each table declares one or more column families. Column families define the storage properties for an arbitrary set of columns. Columns are not declared, they are essentially just an additional label for the value.

HBase is a key/value store. HBase maintains maps of Keys to Values (key -> value). Each of these mappings is called a "[KeyValue"](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/KeyValue.html) or a "Cell".

These cells are sorted by the key. This is a very important property as it allows for searching ("give me all values for which the key is between X and Y"), rather than just retrieving a value for a known key.

The key itself has structure. Each key consists of the following parts:  
row-key, column family, column, and time-stamp.

This follows from the fact the HBase stores key -> value mappings and that a "row" is nothing more than a grouping of these mappings

6. How does data get managed in HBase?

However, because HBase is a column-oriented database, no two rows in a table need have the same columns. To complicate matters further, data is versioned in HBase. The actual coordinates of a value (cell) is the tuple {row key, column key, timestamp}. In addition, columns can be grouped into column families,

Write operation in HBase first records the data to a commit log (a "write-ahead log"), then to an internal memory structure called a MemStore.

When the MemStore fills, it is flushed to disk as an entity called an HFile.

HFiles are stored as a sequence of data blocks, with an index appended to the file's end. Another index, kept in memory, speeds searches for data in HFiles.

HFiles are immutable once written. If a key is deleted, HBase records it and the data is deleted when HFiles are periodically compacted.

HBase attempts to satisfy read operations first through the MemStore. Failing that, HBase checks yet another in-memory structure, the BlockStore, which is a read cache designed to deliver frequently read data from memory, rather than from the disk-based HFiles.

HBase shards rows by regions, which are defined by a range of row keys. Every region in an HBase cluster is managed by a RegionServer process. Typically, there is a single RegionServer process per HBase node. As the amount of data grows, HBase splits regions and migrates the associated data to different nodes in the cluster for balancing purposes.



7. What happens internally when new data gets inserted into HBase table?

When new data is inserted it is first written to a commit log, called a write-ahead log (WAL) in HBase, and then stored in the in-memory memstore. Once the data in memory has exceeded a given maximum value, it is flushed as an HFile to disk. After the flush, the commit logs can be discarded up to the last unflushed modification.

