1. What is NoSQL database?

A NoSQL (originally referring to "non-SQL", "non-relational" or "not only SQL") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century, triggered by the needs of Web 2.0 companies such as Facebook, Google, and Amazon.com. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages.

Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

1. How does data gets stored in NoSQL databases ?

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. Since there is no schema limitations you can easily archive your events including all the relevant data as their properties and have them serialized for you automatically.

NoSQL Database Types

**Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

**Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.

**Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as 'integer', which adds functionality.

**Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows

**In the in-memory databases** like Redis/CouchBase/Tarantool/Aerospike everything is stored in RAM in balanced trees like RB-Tree or in hash tables. All the writes are applied on both RAM and disk, but on disk it goes in an append-only way. A file append can be done as fast as 100Mbytes per second on a normal magnetic disk. If a record size is, say, 1K, then the data will be written at 100krps.

**In the on-disk NoSQL databases and db-engines** like Cassandra/HBase/RocksDB/LevelDB/Sophia the main idea is that you have a snapshot file and a write ahead log (WAL) file. Snapshot contains already prepared data in a form of B-Tree with upper levels of that tree being permanently in RAM, that can be accesses for reading by doing only one disk seek. A WAL contains all the new changes on top of a current snapshot. A snapshot file is being totally rebuilt on a regular basis using current snapshot and a WAL. All the writes are done nearly as fast as with in-memory databases. "Nearly" because disk is partially busy by doing regular snapshot converting that was described earlier. Reads are significantly slower than that are in in-memory databases, because they take at least one disk seek, but good news is that they can be cached in optimized in-memory structures like RB-Trees/hash tables.

1. What is a column Family in HBase ?

Columns in Apache HBase are grouped into column families. All column members of a column family have the same prefix.

For example, the columns **courses**: history and **courses**: math

are both members of the courses column family. The colon character (:) delimits the column family from the column family qualifier. The column family prefix must be composed of printable characters. The qualifying tail, the column family qualifier, can be made of any arbitrary bytes. Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up and running.

Physically, all column family members are stored together on the filesystem. Because tunings and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics.

1. What is the maximum number of columns that can be added to HBase table?

A column family can have an arbitrary number of columns denoted by a column qualifier which is like a column’s label. For example:

{

"row1": {"1": {"color": "green",

"size": 25},

"2": {"weight": 52,

"size": 18}

},

"row2": {"1": {"color": "blue"},

"2": {"height": 192,

"size": 43}

}

}

As you can see in the example above, the same column family (e.g., “1”) in two rows can have different columns. In row “row1”, it has columns “color” and “size”, while in row “row2”, it has only “color” column. It can also have a column that is none of the above. Since rows can have different columns in column families there is no a single way to query for a list of all columns in all column families. This means that you have to do a full table scan.

There is no specific limit on the number of columns in a column family. You can have millions of columns in the single column family.

1. why columns are not defined at the time of table creation in HBase?

Columns are usually physically co-located in column families. A column is identified by column family and column qualifier separated by a colon character (:). For example, courses:math. The column family prefix must be composed of printable characters. The column qualifiers (columns) do not have to be defined at schema definition time and they can be added on the fly while the database is up and running.

A column qualifier is an index for a given data and it is added to a column family. Data within a column family is addressed via the column qualifier. Column qualifiers are mutable and they may vary between rows. They do not have data types and they are always treated as arrays of bytes.

Since columns doesn’t have fixed data type and they will change with every row. So they won’t be defined at the time of table creation.

1. How does data get managed in HBase ?

NoSQL databases are designed for scalability where unstructured data is spread across multiple nodes. When data volumes increase you just need to add another node to accommodate the growth. The lack of structure in NoSQL databases relaxes stringent requirements of consistency enforced in relational databases to improve speed and agility. Hbase, MongoDB and Cassandra are the three major options that provide NoSQL capabilities. The options differ in the features they provide, so the decision on which to use is informed by the workload that will be handled. The main difference between Hbase and Cassandra databases is the consistency model they implement. Cassandra implements eventual consistency which guarantees writes are available. This provides excellent write scaling but suffers a penalty when reading because for consistency in reads you have to read from many copies of data. On the other hand HBase provides a strong consistency model that excels at scaling reads but does not scale on writes as well as Cassandra does.

Hbase is natively supported on Hadoop and it is the subject of this tutorial. The main characteristics that make Hbase an excellent data management platform are fault tolerance, speed and usability. Fault tolerance is provided by automatic fail-over, automatically sharded and load balanced tables, strong consistency in row level operations and replication. Speed is provided by almost real time lookups, in memory caching and server side processing. Usability is provided by a flexible data model that allows many uses, a simple Java API and ability to export metrics.

1. what happens when new data gets inserted into the HBase table ?

When you issue a Put, the coordinates of the data are the row, the column, and the timestamp. The timestamp is unique per version of the cell, and can be generated automatically or specified programmatically by your application, and must be a long integer.

When you put data into HBase, a timestamp is required. The timestamp can be generated automatically by the Region Server or can be supplied by you. The timestamp must be unique per version of a given cell, because the timestamp identifies the **version**. To modify a previous version of a cell, for instance, you would issue a Put with a different value for the data itself, but the same timestamp.

To insert the new data there are 2 cases. They are :

i) So if you want to insert the new data that is not overriding the previous one there will be no issues. The timestamp varies.

ii) Another case is if you want to override the existing data the you need to override it with put command with different value but the same time stamp of the previous value. Versioning plays major role here.