1. **What is NoSQL data base?**

* NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (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
* Log data
* Time series data such as IoT and device data
* Large objects such as video and images.

1. **How does data get stored in NoSQl database**

Unlike a RDBMS, NoSQL databases do not always implement all parts of the ACID model. Many of them adhere to a principle known as BASE (basically available, soft state, and eventually consistent). The system allows for horizontal scaling across multiple servers that work in the background to ensure that each instance is eventually consistent, but there is no guarantee that every instance is a mirror copy of the others at any given moment.

**MongoDB:** Mongo is a document-store database designed to rapidly develop Web applications and Internet infrastructure. Its model and persistence mechanisms are built for high read-and-write throughput and to scale easily with automatic failover. MongoDB does not rely on relationships to manage the data but instead stores JSON documents.

These JSON documents can have embedded structure and the attributes of data can be queried from within the structure. Unlike relational databases, which have structure on definition, document store databases such as MongoDB embed the structure in the documents and rely on the front end to store the structure.

This establishes a structure-on-query paradigm and allows for diverse documents to be stored together and the front end to determine what gets queried on read. Thus, MongoDB is flexible, capable of organic growth, and useful in situations when the data structure is not completely known at the beginning of the analytics process.

**Redis:** Redis is an in-memory key-value store. Each record is a combination of a key and a value, similar to a hash table or a table with two columns (ID and value). Because it stores the key in memory, Redis is very fast at retrieving data based on a specified key.

This data can be simple, as in the case of a single data value associated with a single key, or the key can point to a complex structure that is retrieved in its entirety and parsed by the client. A common use of Redis is to store post-processed and aggregated data that needs to be recalled very quickly.

**ElasticSearch:** ElasticSearch was developed to optimize search capabilities within a database. JSON documents are stored in ElasticSearch and indexed for fast retrieval. ElasticSearch is built on top of Lucene, making it very flexible for finding data using pattern matching and combinations of searches. When search across attributes and within attributes is important, ElasticSearch can provide a database/search engine hybrid.

**Cassandra:** Cassandra is a decentralized and distributed column store database. Created at Facebook and built on a combination of Google's BigData architecture and Amazon's Dynamo distribution design, Cassandra is built for scalability. Instead of storing rows together, Cassandra focuses on columns of data and optimizing the storage of a column.

Its model allows for sparse data, meaning that not every row will have every column. If a row does not have a column, no data is stored, making data storage compact and efficient. This makes it ideal for data that has many attributes where some instances have values for these attributes and others do not.

**Neo4J:** Neo4J is a memory-bound NoSQL database based on relationship data. Data is entered as nodes with attributes and relationships, also called edges, between those nodes. This allows for complex network analysis using Cypher Query Language (CQL). If data relationships are important in your data, Neo4J can be a strong platform.

**3) What is a column family in HBase**

Column families are specified when a table is created. They should be carefully designed before a table is created since it would be either impossible or difficult to change them later.

Column families’ names are strings that are composed of characters that are safe to use in file system paths.

All columns in a column family are stored and sorted together in the same HFile.

Column families group columns together physically and logically and they are usually used for a performance reason. A column family has a set of parameters that specify its storage (e.g., caching, compression, etc.). All tuning and storage specifications are done at the column family level. It is important that all column family members have the same or similar access pattern and sizes.

Some shortcomings in the current HBase implementation do not properly support large number of column families in a single table. That number should be in low tens. Most of the time up to three column families should work fine without any significant performance drawback. Ideally you should go with a single column family. The column family names should be as small as possible, preferably one character.

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. Actually you can have millions of columns in the single column family.

**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**

 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 an running.

set of columns that are not dependent on any other columns of other families.

**6) How does data get managed in HBase?**

The Hbase data model is different from the model provided by relational databases. Hbase is referred to by many terms like a key-value store, column oriented database and versioned map of maps which are correct. The easiest way of visualizing a Hbase data model is a table that has rows and tables. This is the only similarity shared by Hbase model and the relational model.

Data in Hbase is organized into tables. Any characters that are legal in file paths are used to name tables. Tables are further organized into rows that store data. Each row is identified by a unique row key which does not belong to any data type but is stored as a bytearray. Column families are further used to group data in rows. Column families define the physical structure of data so they are defined upfront and their modification is difficult. Each row in a table has same column families. Data in a column family is addressed using a column qualifier. It is not necessary to specify column qualifiers in advance and there is no consistency requirement between rows. No data types are specified for column qualifiers, as such they are just stored as bytearrays. A unique combination of row key, column family and column qualifier forms a cell. Data contained in a cell is referred to as cell value. There is no concept of data type when referring to cell values and they are stored as bytearrays. Versioning happens to cell values using a timestamp of when the cell was written

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

To write data to HBase, you use methods of the HTableInterface class. You can use the Java API directly, or use HBase Shell, Thrift API, REST API, or another client which uses the Java API indirectly. 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.

**Variations on Put**

There are several different ways to write data into HBase. Some of them are listed below.

* A Put operation writes data into HBase.
* A Delete operation deletes data from HBase. What actually happens during a Delete depends upon several factors.
* A CheckAndPut operation performs a Scan before attempting the Put, and only does the Put if a value matches what is expected, and provides row-level atomicity.
* A CheckAndDelete operation performs a Scan before attempting the Delete, and only does the Delete if a value matches what is expected.
* An Increment operation increments values of one or more columns within a single row, and provides row-level atomicity.