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

NoSQL is an approach to [database](http://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](http://searchsqlserver.techtarget.com/definition/SQL)," is an alternative to traditional relational databases in which data is placed in tables and data [schema](http://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.

**2. How does data get stored in NoSQl database?**

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.

Application developers have been frustrated with the impedance mismatch between the relational data structures and the in-memory data structures of the application. Using NoSQL databases allows developers to develop without having to convert in-memory structures to relational structures.

There is also movement away from using databases as integration points in favor of encapsulating databases with applications and integrating using services. The rise of the web as a platform also created a vital factor change in data storage as the need to support large volumes of data by running on clusters. Relational databases were not designed to run efficiently on clusters. The data storage needs of an ERP application are lot more different than the data storage needs of a Facebook or an Etsy.

**3) 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 andcourses:math are both members of the courses column family. The colon character (:) delimits the column family from the . The column family prefix must be composed ofprintable 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 an 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.

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

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.

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.

A row key, column family and column qualifier form a cell that has a value and timestamp that represents the value’s version. Values also do not have data types and they are always treated as arrays of bytes. A timestamp is recorded for each value and it is the time on the region server when the value was written.

All cell’s values are stored in a descending order by its timestamp. When values are retrieved and if the timestamp is not provided then HBase will return the cell value with the latest (the most recent) timestamp. If a timestamp is not specified during the write, the current timestamp is used.

The maximum number of versions (timestamps) for a given column to store is part of the column schema. It is specified at table creation. It can be specified via alter table command as well. The default value is 1. The minimum number of versions can be also set up per column family. You can also globally set up a maximum number of versions per column.

HBase does not overwrite row values. It stores different values per row by time and column qualifier. Extra versions above the current max version setup are removed during major compactions. If it is not necessary it is not recommended to have very high maximum number of versions since it will increase the HFile size significantly.

It is worth to mention that the column metadata is only stored in internal key/value instances for a column family. You have to keep track of the column names since HBase can support very high number of columns per row and columns can differ between the rows as well. If you do not record these column names by yourself and you forget them you will have to retrieve all rows from a column family in order to find out the column names.

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

There is really no such thing as a nested table in hbase ... sometimes it is called nested entity. The main idea is really a column family. A single column family contains one or more columns. Column families must be defined at table creation time but columns can be added dynamically after table creation (if an insert statement states a column that does not exist for a column family it will create it). Column families thus can be seen as holding an array of information that may have different lengths among rows (keys). You do not have to use it that way: you can always use identical columns for each column family. Another feature of column families is that they are written to their own files. Thus queries read only the column families holding the columns in the queries. This allows you to design very wide tables (hundreds of columns) and read only a subset of columns for each query (resulting in faster performance). Also, column families can have different properties, e.g one can be compressed and others not. Thus the general rule is to group columns that will be queried together into the same column family and allow the number of columns in a column family to be dynamic among records if you wish.

Altogether a row key defines a single row or record. All rows of a table have the same number of one or more column families. Each column family can have the same or different numbers of columns among rows because new columns can be added at insert-time for a particular record, and not necessarily at table create-time (for all records).

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

[HBase](http://hbase.apache.org/) is a column-oriented database that’s an open-source implementation of Google’s [Big Table](http://research.google.com/archive/bigtable.html) storage architecture. It can manage structured and semi-structured data and has some built-in features such as scalability, versioning, compression and garbage collection.

Since its uses write-ahead logging and distributed configuration, it can provide fault-tolerance and quick recovery from individual server failures. HBase built on top of Hadoop / HDFS and the data stored in HBase can be manipulated using Hadoop’s MapReduce capabilities.

Let’s now take a look at how HBase (a column-oriented database) is different from some other data structures and concepts that we are familiar with **Row-Oriented vs. Column-Oriented data stores.**As shown below, in a row-oriented data store, a row is a unit of data that is read or written together. In a column-oriented data store, the data in a column is stored together and hence quickly retrieved.

Row-oriented data stores –

* Data is stored and retrieved one row at a time and hence could read unnecessary data if only some of the data in a row is required.
* Easy to read and write records
* Well suited for OLTP systems
* Not efficient in performing operations applicable to the entire dataset and hence aggregation is an expensive operation
* Typical compression mechanisms provide less effective results than those on column-oriented data stores

Column-oriented data stores –

* Data is stored and retrieved in columns and hence can read only relevant data if only some data is required
* Read and Write are typically slower operations
* Well suited for OLAP systems
* Can efficiently perform operations applicable to the entire dataset and hence enables aggregation over many rows and columns
* Permits high compression rates due to few distinct values in columns\

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

When you put data into HBase, a timestamp is required. The timestamp can be generated automatically by the RegionServer 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.

HBase's behavior regarding versions is highly configurable. The maximum number of versions defaults to 1 in CDH 5, and 3 in previous versions. You can change the default value for HBase by configuring hbase.column.max.version in hbase-site.xml, either using an advanced configuration snippet if you use Cloudera Manager, or by editing the file directly otherwise.

You can also configure the maximum and minimum number of versions to keep for a given column, or specify a default time-to-live (TTL), which is the number of seconds before a version is deleted. The following examples all use alter statements in HBase Shell to create new column families with the given characteristics, but you can use the same syntax when creating a new table or to alter an existing column family. This is only a fraction of the options you can specify for a given column family.

hbase> alter ‘t1′, NAME => ‘f1′, VERSIONS => 5

hbase> alter ‘t1′, NAME => ‘f1′, MIN\_VERSIONS => 2

hbase> alter ‘t1′, NAME => ‘f1′, TTL => 15

HBase sorts the versions of a cell from newest to oldest, by sorting the timestamps lexicographically. When a version needs to be deleted because a threshold has been reached, HBase always chooses the "oldest" version, even if it is in fact the most recent version to be inserted. Keep this in mind when designing your timestamps. Consider using the default generated timestamps and storing other version-specific data elsewhere in the row, such as in the row key. If MIN\_VERSIONS and TTL conflict, MIN\_VERSIONS takes precedence.