1.What is NoSQL data base?
NoSQL is an approach to databases that represents a shift away from traditional relational database management systems(RDMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDMS. Relational databases rely on tables, columns, rows or schemas to organise and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean “not SQL” to “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 and log data, time series data such as IoT and device data; and large objects such as video and images.

2. How does data get stored in NoSQL database?
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.

That's the way it works in a rough approximation.

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?

HBase currently does not do well with anything above two or three column families so keep the number of column families in your schema low. Currently, flushing and compactions are done on a per Region basis so if one column family is carrying the bulk of the data bringing on flushes, the adjacent families will also be flushed though the amount of data they carry is small. When many column families the flushing and compaction interaction can make for a bunch of needless i/o loading (To be addressed by changing flushing and compaction to work on a per column family basis).
Try to make do with one column family if you can in your schemas. Only introduce a second and third column family in the case where data access is usually column scoped; i.e. you query one column family or the other but usually not both at the one time.

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

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.

6. How does data get managed in HBase?

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.

Tables in Hbase have several properties that need to be understood for one to come up with an effective data model. Indexing and sorting only happens on the row key. The concept of data types is absent and everything is stored as bytearray. Only row level atomicity is enforced so multi row transactions are not supported.

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

We can update an existing cell value using the put command. To do so, just follow the same syntax and mention your new value as shown below.

```
1  put 'table name','row ','Column
   family:column name','new value'
```

The newly given value replaces the existing value. The following command will update the city value of the employee named 'Ashok' to Vijayawada.

```
1  hbase>
2  put'emp','row1','personal:city','Vijayaw
   ada'
   0 row(s) in 0.0630 seconds
```