

A NoSQL (originally referring to "non SQL" or "non-relational") database provides a mechanism for storage

And retrieval of data that is modelled 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,



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.

You can have a look at the various categories of NoSQL databases here: NOSQL Databases

I have personally played around with MongoDB for a while now and it has always impressed me. MongoDB is an document store, where data is stored as Key: Value pairs in JSON format.



A **HBase** table is comprised of one or more **column families**, each of which is stored in a separate set of region files sharing a common key. To express it in terms of an RDBMS, a **column family** is roughly analogous to a RDBMS table with the row key as a clustered primary key index.



There's not really a limit. Here are some things to consider:  
  
**Lock granularity**  
When you do an operation within a row, the RegionServer code briefly holds a lock on that row while applying the mutation. On the plus side, this means that you can act atomically on several columns - concurrent readers will either see the entire update or won't see the update at all. They shouldn't (barring one or two bugs we're still stomping on) see a partial update. On the minus side, this means that the throughput of write operations within a single row is limited (probably a few hundred per second).  
  
**Region distribution**  
The unit of load balancing and distribution is the region, and a row will never be split across regions. So, no matter how hot a row is, it will always be served by a single server. If the data were split across many rows, you could force a split in between two hot rows to distribute load between two hosts.

**Bugs**  
In prior versions of HBase there were some bugs where we would accidentally load or deserialize an entire row into RAM. So if your row is very large (100s of MBs) you may have run into serious performance issues, OOMEs, etc. I think most of these bugs are since squashed, and the RS does a smart job of only loading the necessary columns, but it's something to be aware of.

**Summary**In summary, if you don't need to do atomic operations across multiple cells, probably better to make a "tall" data layout.



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. 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.



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.



All the data in the HBase is stored as raw byte Array (10101010). Now the put instance is created which can be inserted in the HBase users table.