**Assignment 9.1**

**Problem Statements**

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

A **NoSQL** (originally referring to "non SQL" or "non relational")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.

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

Each NoSql Databases have different mechanisms for storing the data:

* **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.

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 .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 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?**

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).

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.

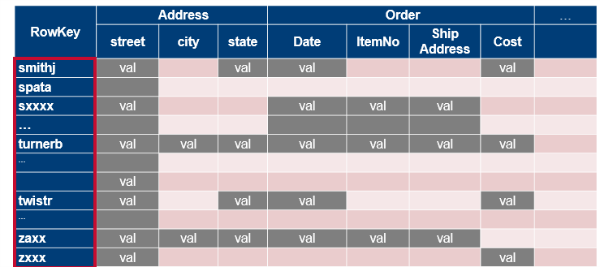
**5)** **Why columns are not defined at the time of table creation in HBase?**

The main motto behind the NOSQL databases , is that we can dynamically add Columns at the runtime. In RDBMS we have to provide schema while creation of table. But in case if HBase we can add Columns at run time.

This gives User flexibility to add as many column they want , so for frequent requirement changes we add Columns at Runtime.

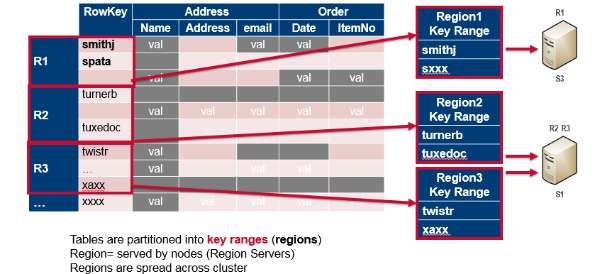
**6)How does data get managed in HBase?**

Data stored in HBase is located by its “rowkey.” This is like a primary key from a relational database. Records in HBase are stored in sorted order, according to rowkey. This is a fundamental tenet of HBase and is also a critical semantic used in HBase schema design.



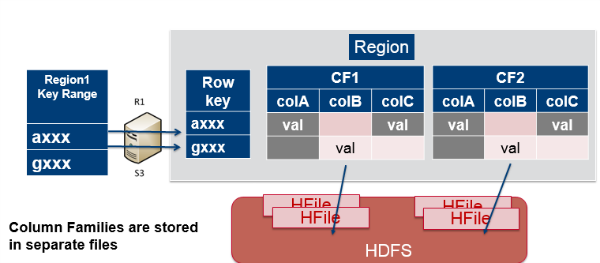
HBase data model – row keys

Tables are divided into sequences of rows, by key range, called regions. These regions are then assigned to the data nodes in the cluster called “RegionServers.” This scales read and write capacity by spreading regions across the cluster. This is done automatically and is how HBase was designed for horizontal sharding.

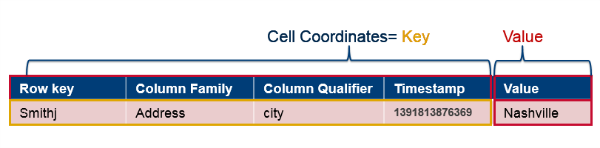


Tables are split into regions = contiguous keys

The image below shows how column families are mapped to storage files. Column families are stored in separate files, which can be accessed separately.

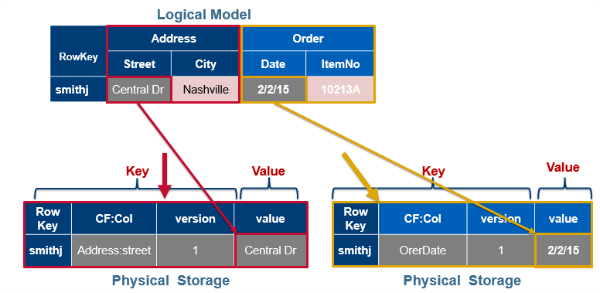


The data is stored in HBase table cells. The entire cell, with the added structural information, is called Key Value. The entire cell, the row key, column family name, column name, timestamp, and value are stored for every cell for which you have set a value. The key consists of the row key, column family name, column name, and timestamp.



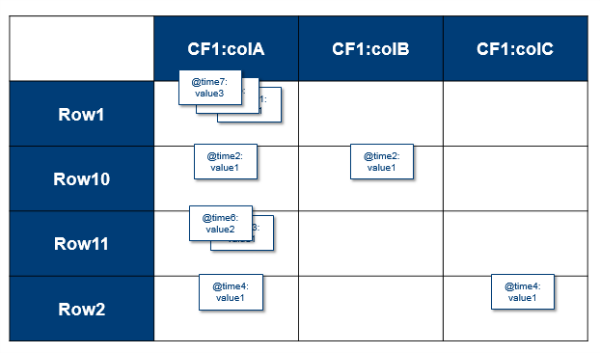
Logically, cells are stored in a table format, but physically, rows are stored as linear sets of cells containing all the key value information inside them.

In the image below, the top left shows the logical layout of the data, while the lower right section shows the physical storage in files. Column families are stored in separate files. The entire cell, the row key, column family name, column name, timestamp, and value are stored for every cell for which you have set a value.



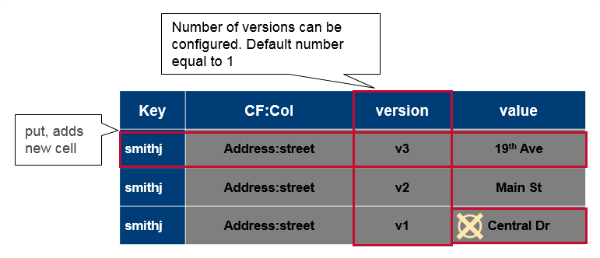
Logical data model vs. physical data storage

As mentioned before, the complete coordinates to a cell's value are: Table:Row:Family:Column:Timestamp ➔ Value. HBase tables are sparsely populated. If data doesn’t exist at a column, it’s not stored. Table cells are versioned uninterpreted arrays of bytes. You can use the timestamp or set up your own versioning system. For every coordinate row:family:column, there can be multiple versions of the value.



Sparse data with cell versions

Versioning is built in. A put is both an insert (create) and an update, and each one gets its own version. Delete gets a tombstone marker. The tombstone marker prevents the data being returned in queries. Get requests return specific version(s) based on parameters. If you do not specify any parameters, the most recent version is returned. You can configure how many versions you want to keep and this is done per column family. The default is to keep up to three versions. When the max number of versions is exceeded, extra records will be eventually removed.



**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.

## Versions

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.