

1.What is NoSQL data base?

NoSQL Database is used to refer a non-SQL or non relational database.

It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

NoSQL refers to all databases and data stores that are not based on the Relational Database Management Systems or RDBMS principles. It relates to large data sets accessed and manipulated on a Web scale. NoSQL does not represent single product or technology. It represents a group of products and a various related data concepts for storage and management.

Database Features of NoSQL

NoSQL Databases can have a common set of features such as:

- Non-relational data model.
- Runs well on clusters.
- Mostly open-source.
- Built for the new generation Web applications.
- Is schema-less.

Why NoSQL?

With the explosion of social media, user-driven content has grown rapidly and has increased the volume and type of data that is produced, managed, analyzed, and archived. In addition, new sources of data, such as sensors, Global Positioning Systems or GPS, automated trackers, and other monitoring systems generate huge volumes of data on a regular basis.

These large volumes of data sets, also called big data, have introduced new challenges and opportunities for data storage, management, analysis, and archival. In addition, data is becoming increasingly semi-structured and sparse. This means that RDBMS databases which require upfront schema definition and relational references are examined.

To resolve the problems related to large-volume and semi-structured data, a class of new database products have emerged. These new classes of database products consist of column-based data stores, key/value pair databases, and document databases.

Together, these are called NoSQL. The NoSQL database consists of diverse products with each product having unique sets of features and value propositions.

2.How does data get stored in NoSQL database?

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.

Types of NoSQL

There are four basic types of NoSQL databases.

Key-Value database

Key-Value database has a big hash table of keys and values. Riak (Pronounce as REE-awk), Tokyo Cabinet, Redis server, Memcached ((Pronounce as mem-cached), and Scalaris are examples of a key-value store.

Document-based database

Document-based database stores documents made up of tagged elements. Examples: MongoDB, CouchDB, OrientDB, and RavenDB .

Column-based database

Each storage block contains data from only one column, Examples: BigTable, Cassandra, Hbase, and Hypertable.

Graph-based database

A graph-based database is a network database that uses nodes to represent and store data. Examples are Neo4J, InfoGrid, Infinite Graph, and FlockDB.

3.What is a column family in HBase?

Column families are the base storage mechanism in HBase.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 rowkey as a clustered primary key index. A HBase table would then be a view which does a full outer join on a set of RDBMS tables which all share the same primary key (thus having a 1:1 relationship). In this analogy, HBase region files map to pages in an RDBMS.

Logical View of Customer Contact Information in HBase

Row Key	Column Family: {Column Qualifier:Version:Value}
1	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'} ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}
2	CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe', ContactInfo: { 'SA': 1383859185577:'7 HBase Ave, CA 22222'}

The table shows two column families: CustomerName and ContactInfo. When creating a table in HBase, the developer or administrator is required to define one or more column families using printable characters.

Generally, column families remain fixed throughout the lifetime of an HBase table but new column families can be added by using administrative commands.

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.

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?

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 from the . 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.

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.

Data Model Operations

The four primary data model operations are Get, Put, Scan, and Delete. Operations are applied via HTable instances.

Get

Get returns attributes for a specified row. Gets are executed via HTable.get.

Put

Put either adds new rows to a table (if the key is new) or can update existing rows (if the key already exists). Puts are executed via HTable.put (writeBuffer) or HTable.batch (non-writeBuffer).

Scans

Scan allow iteration over multiple rows for specified attributes.

Delete

Delete removes a row from a table. Deletes are executed via HTable.delete.

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.

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
```

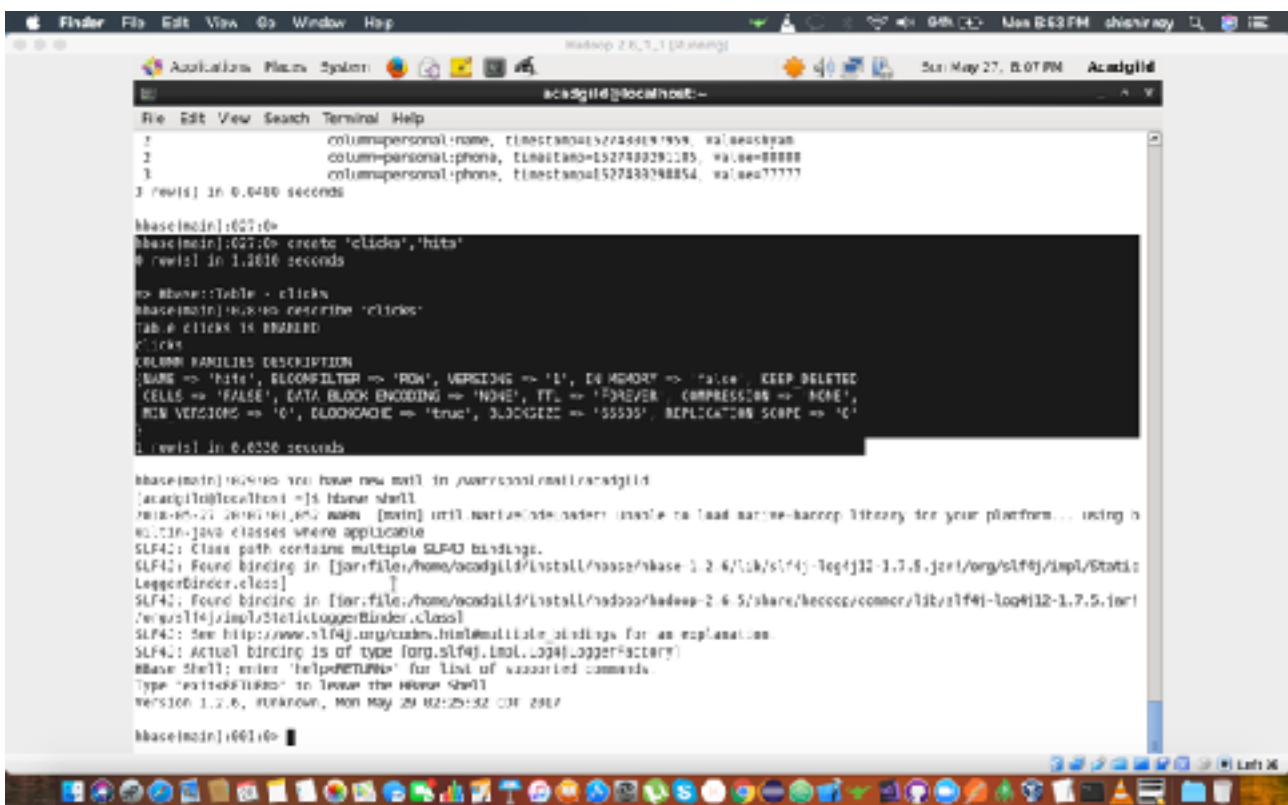
HBase sorts the versions of a cell from newest to oldest, by sorting the timestamps lexicographically. When a version needs to be deleted because a threshold has been reached, HBase always chooses the "oldest" version, even if it is in fact the most recent version to be inserted. Keep this in mind when 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.

Task 2

1. Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.
2. Add few records in the table and update some of them. Use IP Address as row-key. Scan the table to view if all the previous versions are getting displayed.



```
Acadgild@localhost:~$ hbase shell
hbase(main):001:0> create 'clicks','hits'
0 row(s) in 1.1016 seconds

hbase(main):002:0> show tables
Table click is ENABLED
clicks
COLUMN FAMILIES DESCRIPTION
NAME => 'hits', BLOCKFILTER => 'ROW', VERSIONS => '5', DELETES => 'false', KEEP DELETED
CELLS => 'FALSE', DATA BLOCK ENCODING => 'NONE', TTL => 'FOREVER', COMPRESSION => 'NONE',
MIN VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65535', REPLICATION SCOPE => '0'
1 row(s) in 0.0236 seconds

hbase(main):003:0> put '1','column:personal:phone','value:88888'
1 row(s) in 0.0480 seconds

hbase(main):004:0> put '2','column:personal:phone','value:88888'
1 row(s) in 0.0480 seconds

hbase(main):005:0> put '3','column:personal:phone','value:77777'
1 row(s) in 0.0480 seconds

hbase(main):006:0> scan 'clicks'
1 row(s) in 0.0480 seconds
```

```
hbase(main):011:0> put 'clicks','1','hits:qualifiers','abc'
0 row(s) in 0.0000 seconds

hbase(main):012:0> put 'clicks','2','hits:qualifiers','xyz'
0 row(s) in 0.0010 seconds

hbase(main):013:0> put 'clicks','3','hits:qualifiers','xyz'
0 row(s) in 0.0100 seconds

hbase(main):014:0> put 'clicks','4','hits:qualifiers','xyz'
0 row(s) in 0.0100 seconds

hbase(main):015:0> put 'clicks','5','hits:qualifiers','def'
0 row(s) in 0.0110 seconds

hbase(main):016:0> scan 'clicks'
row
COLUMN+CELL
column=hits:qualifiers, timestamp=1527432170010, value=abc
column=hits:qualifiers, timestamp=1527432184000, value=xyz
column=hits:qualifiers, timestamp=1527432194000, value=xyz
column=hits:qualifiers, timestamp=1527432204750, value=xyz
column=hits:qualifiers, timestamp=1527432220100, value=def
5 row(s) in 0.0010 seconds

hbase(main):017:0>
```

```
hbase(main):019:0> put 'clicks','1','hits:ipaddress','abc.com'
0 row(s) in 0.0210 seconds

hbase(main):020:0> put 'clicks','2','hits:ipaddress','xyz.com'
0 row(s) in 0.0110 seconds

hbase(main):021:0> put 'clicks','3','hits:ipaddress','xyz.com'
0 row(s) in 0.0130 seconds

hbase(main):022:0> put 'clicks','4','hits:ipaddress','xyz.com'
0 row(s) in 0.0130 seconds

hbase(main):023:0> put 'clicks','5','hits:ipaddress','lax.com'
0 row(s) in 0.0140 seconds

hbase(main):024:0> scan 'clicks'
row
COLUMN+CELL
column=hits:ipaddress, timestamp=1527432209100, value=abc.com
column=hits:qualifiers, timestamp=1527432170010, value=abc
column=hits:ipaddress, timestamp=1527432184000, value=xyz.com
column=hits:qualifiers, timestamp=1527432184000, value=xyz
column=hits:ipaddress, timestamp=1527432203000, value=xyz.com
column=hits:qualifiers, timestamp=1527432194000, value=xyz
column=hits:ipaddress, timestamp=1527432450000, value=xyz.com
column=hits:qualifiers, timestamp=1527432204750, value=xyz
column=hits:ipaddress, timestamp=1527432657000, value=lax.com
column=hits:qualifiers, timestamp=1527432220100, value=def
0 row(s) in 0.0400 seconds

hbase(main):025:0>
```

