HBASE BASICS

Assignment 10.1

Task 1:

Answer in your own words with example

1. **What is NoSQL database?**

A: A **NoSQL** database provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases.
NoSQL, which stand for "**not only SQL**," is an alternative to traditional relational databases in which data is placed in tables and data schema is carefully designed. **NoSQL** databases are especially useful for working with large sets of distributed data.

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.

Limitations of Traditional RDBMS apprise the introduction of NoSQL databases.
Challenges with Traditional RDBMS were –

a) **Not optimized for horizontal scaling out** – Data size have had been increased tremendously sizing in Petabytes.
b) **Schema-less data** - Majority of the data comes in a semi-structured or unstructured format.
c) **High velocity of data ingestion** - RDBMS lacks in high velocity because it's designed for steady data retention rather than rapid growth.
d) **Cost** - High licensing cost for data analysis.

Characteristics of NoSQL Database-

I. Not using the relational model
II. Running well on clusters
III. Open-source
IV. Schemaless
V. Highly Distributable

One of the most obvious shifts with NoSQL is a move away from the relational model. Each NoSQL solution has a different model that it uses, which we put into four categories widely used in the NoSQL ecosystem i.e

a) key-value
b) document
c) column-family
d) graph

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

A: Data get stored in NoSQL database in different Data Models.
Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

- **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.

3. What is a column family in HBase?

A: In the HBase data model columns are grouped into *column families*, which must be defined up front during table creation. Column families are stored together on disk, *which is why HBase is referred to as a* ***column-oriented data store*****.**

All column members of a column family have the same prefix.

For Example –

In below screenprint, **hits11** is the 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.

Physically, all column family members are stored together on the filesystem.

```
hbase(main):014:0> put 'clicks11', '192.168.0.7', 'hits11:qualifiers16', 'Q6'
0 row(s) in 0.0090 seconds

hbase(main):015:0> put 'clicks11', '192.168.0.7', 'hits11:qualifiers16', 'Q7'
0 row(s) in 0.0050 seconds

hbase(main):016:0> put 'clicks11', '192.168.0.7', 'hits11:qualifiers16', 'Q8'
0 row(s) in 0.0060 seconds

hbase(main):017:0> put 'clicks11', '192.168.0.7', 'hits11:qualifiers16', 'Q9'
0 row(s) in 0.0100 seconds

hbase(main):018:0> scan 'clicks11'
ROW                                   COLUMN+CELL
 192.168.0.7                          column=hits11:qualifiers11, timestamp=1534312648735, value=Q1
 192.168.0.7                          column=hits11:qualifiers12, timestamp=1534312717631, value=Q2
 192.168.0.7                          column=hits11:qualifiers13, timestamp=1534312731327, value=Q3
 192.168.0.7                          column=hits11:qualifiers14, timestamp=1534312738496, value=Q4
 192.168.0.7                          column=hits11:qualifiers15, timestamp=1534312747651, value=Q5
 192.168.0.7                          column=hits11:qualifiers16, timestamp=1534313322556, value=Q9
1 row(s) in 0.0230 seconds
```

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

A: Hbase table does not do good with more number of Column families. Anything more than 2 or 3 column families does not work properly. So it need to be kept low number of column families in the schema.

Where multiple Column Families exist in a single table, in this case the one have to make sure of the cardinality (i.e, number of rows).

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

A: Columns are usually physically co-located in column families.
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.
Columns can be added on demand by the client. A table's column families must be specified up front as part of the table schema definition, but it's member can be added onn demand by the client as part of an update, and its value persisted, as long as the column family info already exists on the table.
Physically, all column family members are stored together on the filesystem. So although earlier we described HBase as a column-oriented store, it would be more accurate if it were described as a column-family-oriented store. Because tuning 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. For the photos table, the image data, which is large (megabytes), is stored in a separate column family from the metadata, which is much smaller in size (kilobytes).

6. How does data get managed in HBase?

A: The HBase Physical Architecture consists of servers in a Master-Slave relationship as shown below. Typically, the HBase cluster has one Master node, called HMaster and multiple Region Servers called HRegionServer. Each Region Server contains multiple Regions – HRegions.

Just like in a Relational Database, data in HBase is stored in Tables and these Tables are stored in Regions. When a Table becomes too big, the Table is partitioned into multiple Regions. These Regions are assigned to Region Servers across the cluster. Each Region Server hosts roughly the same number of Regions.

The layout of the data model makes it easier to partition the data and distribute it across the cluster. The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells and Versions.

Tables – The HBase Tables are more like logical collection of rows stored in separate partitions called Regions. As shown above, every Region is then served by exactly one Region Server. The figure above shows a representation of a Table.

Rows – A row is one instance of data in a table and is identified by a rowkey. Rowkeys are unique in a Table and are always treated as a byte[].

Column Families – Data in a row are grouped together as Column Families. Each Column Family has one more Columns and these Columns in a family are stored together in a low level storage file known as HFile. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence it's important that proper care be taken when designing Column Families in table.

The table above shows Customer and Sales Column Families. The Customer Column Family is made up 2 columns – Name and City, whereas the Sales Column Families is made up to 2 columns – Product and Amount.

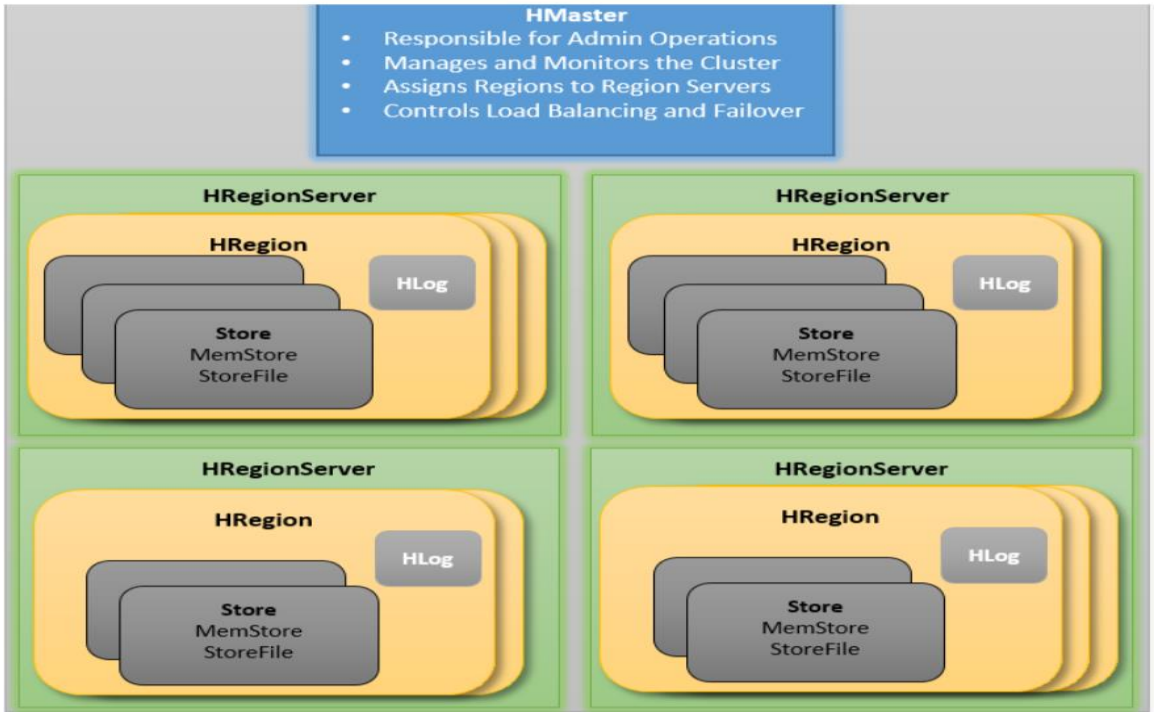
Columns – A Column Family is made of one or more columns. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have varied number of Columns.

Cell – A Cell stores data and is essentially a unique combination of rowkey, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value and the data type is always treated as byte[].

Version – The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

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

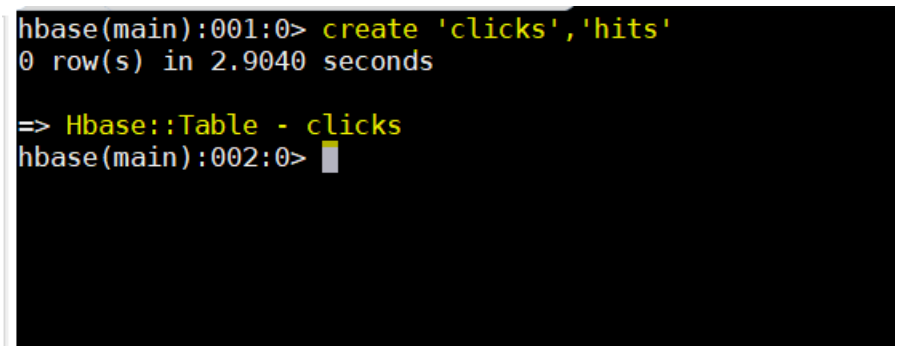
A: Tables are automatically partitioned horizontally by HBase into *regions*. Each region comprises a subset of a table's rows. A region is denoted by the table it belongs to, its first row (inclusive), and its last row (exclusive). Initially, a table comprises a single region, but as the region grows it eventually crosses a configurable size threshold, at which point it splits at a row boundary into two new regions of approximately equal size. Until this first split happens, all loading will be against the single server hosting the original region. As the table grows, the number of its regions grows. Regions are the units that get distributed over an HBase cluster. In this way, a table that is too big for any one server can be carried by a cluster of servers, with each node hosting a subset of the table's total regions. This is also the means by which the loading on a table gets distributed. The online set of sorted regions comprises the table's total content.



Task 2:

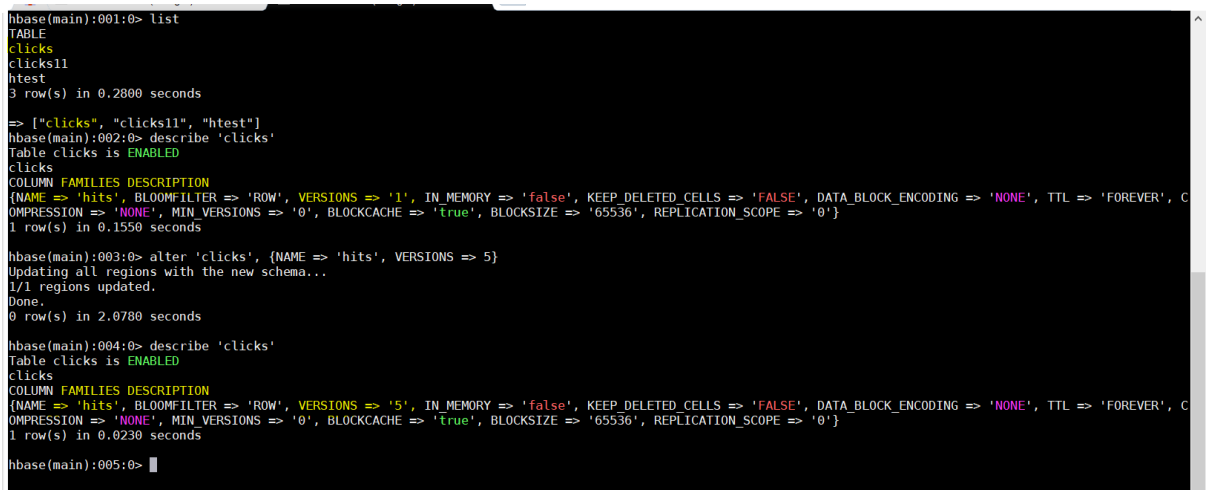
1. **Create an HBase table named 'clicks' with a column family 'hits' such that it should be able to store last 5 values of qualifiers inside 'hits' column family.**

Creating Hbase table **clicks –**



By default after creating table it creates VERSIONS as 1, so we have make it to 5 as in below screenprint.

alter 'clicks', {NAME => 'hits', VERSIONS => 5}



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.**

```
hbase(main):005:0> put 'clicks', '192.168.43.128', 'hits:name', 'developer'
0 row(s) in 0.1090 seconds

hbase(main):006:0> put 'clicks', '192.168.43.128', 'hits:age', '25'
0 row(s) in 0.0100 seconds

hbase(main):007:0> put 'clicks', '192.168.43.128', 'hits:age', '26'
0 row(s) in 0.0090 seconds

hbase(main):008:0> put 'clicks', '192.168.43.128', 'hits:age', '27'
0 row(s) in 0.0080 seconds

hbase(main):009:0> put 'clicks', '192.168.43.128', 'hits:age', '28'
0 row(s) in 0.0090 seconds

hbase(main):010:0> put 'clicks', '192.168.43.128', 'hits:age', '29'
0 row(s) in 0.0080 seconds

hbase(main):011:0> scan 'clicks'
ROW                                   COLUMN+CELL
 192.168.43.128                       column=hits:age, timestamp=1534340384079, value=29
 192.168.43.128                       column=hits:name, timestamp=1534340315499, value=developer
1 row(s) in 0.0430 seconds

hbase(main):012:0>
```

As you can see in above screenprint that we have entered 5 different values for "**age"** column under **"hits"** column family, and only the last value is showing up in the scan result of the table **"clicks".**

Now we will see the versions of the column family "**hits**" for the column member "**age**". We have added **5** entries for the column member.

```
hbase(main):012:0> get 'clicks', '192.168.43.128', {COLUMN=>'hits:age',VERSIONS=>5}
COLUMN                                CELL
 hits:age                             timestamp=1534340384079, value=29
 hits:age                             timestamp=1534340379533, value=28
 hits:age                             timestamp=1534340375345, value=27
 hits:age                             timestamp=1534340369843, value=26
 hits:age                             timestamp=1534340341913, value=25
5 row(s) in 0.0390 seconds

hbase(main):013:0>
```