Assignment Day 10

Task 1:

Answer in your own words with example.

1.What is NoSQL data base?

2.How does data get stored in NoSQL database?

3.What is a column family in HBase?

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

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

6.How does data get managed in HBase?

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

Answers:

1.What is NoSQL data base?

Ans-

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.

An approach to database design that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats.

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 before the database is built.

NoSQL database provides ability to process very large volumes of data and quickly distribute that data across those distinct computing clusters.

Features: -

•**Generic Data Model** -Heterogeneous containers, including sets, maps, and arrays

•**Dynamic type discovery** and **conversion** –NoSQL analytics systems support runtime type identification and conversion so that custom business logic can be used to dictate analytic treatment of variation.

•**Non-relational** and **De-normalised**- Data is stored in single tables as compared to joining multiple tables.

•**Commodity hardware** - Adding more of the economical servers allows NoSQL databases to scale to handle more data.

•**Highly distributable** - Distributed databases can store and process a set of information on more than one device.

Examples of **NoSQL** are: Cassandra, MongoDB, HBase, Redis etc.

2.How does data get stored in NoSQL database?

Ans-

There are various NoSQL Databases. Each one uses a different method to store data. Some use column store, some document, some graph, etc., Each database has its own unique characteristics.

Different NoSQL Database schemes are as below:

a>Key-value stores

Key-value stores, or key-value databases, implement a simple data model that pairs a unique key with an associated value. Because this model is simple, it can lead to the development of key-value databases, which are extremely performant and highly scalable for session management and caching in web applications. Implementations differ in the way they are oriented to work with RAM, solid-state drives or disk drives.

e.g. Riak, MemchacheDB, Aerospike, Berkeley DB and Redis.

b>Document databases

Document databases, also called document stores, store semi-structured data and descriptions of that data in document format. They allow developers to create and update programs without needing to reference master schema. Use of document databases has increased along with use of JavaScript and the JavaScript Object Notation (JSON), a data interchange format that has gained wide currency among web application developers, although XML and other data formats can be used as well. Document databases are used for content management and mobile application data handling.

e.g. Couchbase Server, MarkLogic, CouchDB, Document DB and MongoDB

c>Wide-column stores

Wide-column stores organize data tables as columns instead of as rows. Wide-column stores can be found both in SQL and NoSQL databases. Wide-column stores can query large data volumes faster than conventional relational databases. A wide-column data store can be used for recommendation engines, CATALOGS, fraud detection and other types of data processing.

e.g. HBase, Google Bigtable and Cassandra.

d>Graph stores

Graph data stores organizes data as nodes, which are like records in a relational database, and edges, which represent connections between nodes. Because the graph system stores the relationship between nodes, it can support richer representations of data relationships. Also, unlike relational models reliant on strict schemas, the graph data model can evolve over time and use. Graph databases are applied in systems that must map relationships, such as reservation systems or customer relationship management.

e.g AllegroGraph, Neo4j, IBM Graph and Titan.

3.What is a column family in HBase?

Ans-

Columns in Apache HBase are grouped into column families. All column members of a column family have the same prefix. For example, the columns **cf1: street** and **cf1: name** is both members of the **cf1** column family. The colon character **(:)** delimits the column family from 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.

e.g.

R1 column=**cf1**:**dept_city**, timestamp=15334595875, value=**NEW YORK**

R1 column=**cf1**:**dept_name**, timestamp=15334425907, value=**ACCOUNTING**

R1 = Row

cf1 = Column Family

dept_city, dept_name = Columns

NEW YORK, ACCOUNTING = Values

Note:-

Column family cannot be updated.

If a table consists of only one column family then that column family cannot be deleted.

Column families are grouped together on disk, so grouping data with similar access patterns reduces overall disk access and increases performance.

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

Ans-

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).

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

Ans-

HBase has a Dynamic Shema. It uses query-first schema design; all possible queries should be identified first, and the schema model designed accordingly. By not defining columns at the time of creation it provides flexibility to add columns on need basis. Traditional RDBMS database, all the schemas are defined at the beginning. This is a constraint when project requirement changes, we need to rework on whole model. By keeping flexibility, any new requirements can be easily done without revisiting models.

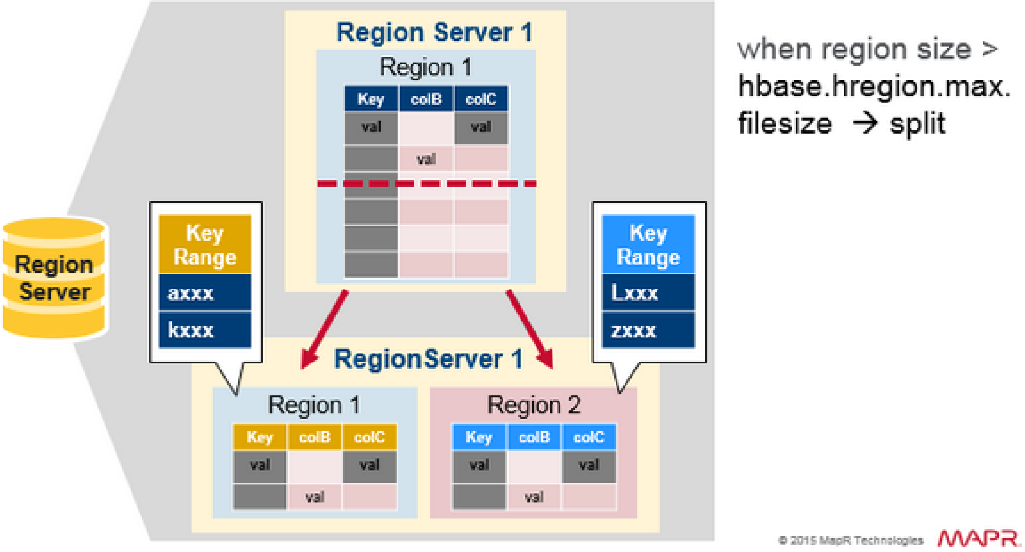
6.How does data get managed in HBase?

Ans-

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.

HBase stores data in a table-like format with the ability to store billions of rows with millions of columns. Columns can be grouped together in “column families” which allows physical distribution of row values onto different cluster nodes.

HBase group rows into “regions” which define how table data is split over multiple nodes in a cluster. If a region gets too large, it is automatically split to share the load across more servers.



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

Ans-

When **Put** is used to store data, it uses 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 the application and must be a long integer.

When a put/insert request is initiated, the value is first written into the WAL and then into the memstore. The values in the memstore is stored in the same sorted manner as in the HFile. Once the memstore is full, it is then flushed into a new HFile.

HFile stores the data in sorted order i.e. the sequential rowkeys will be next to each other.

HBase periodically performs compactions which will merge multiple HFiles and rewrite's them to a single HFile, this new HFile which is a result of compaction is also sorted.

The READ-WRITE data in the HBASE is managed in the following ways:

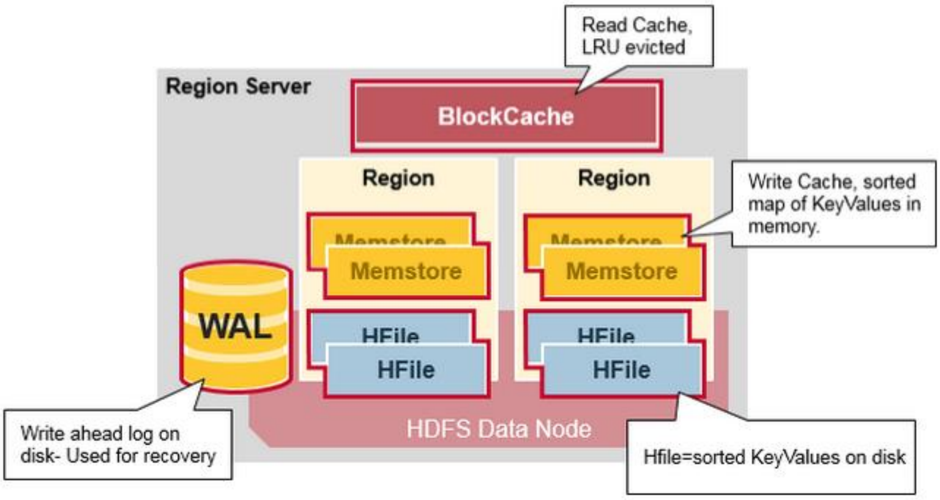
•**WAL**: Write Ahead Log is a file on the distributed file system. The WAL is used to store new data that hasn't yet been persisted to permanent storage; it is used for recovery in the case of failure.

•**BlockCache**: is the read cache. It stores frequently read data in memory. Least Recently Used data is evicted when full.

•**MemStore**: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.

•**Hfiles** store the rows as sorted KeyValues on disk.

All the above are sub components of Region server which manages the data in the HBASE.



Task 2:

a. 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.

b. 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.

Ans:

Commands:

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

Explanation :

creates a table named '**clicks**' with column family '**hits**' which would keep five version for corresponding columns values of **hits** column family

OR

create 'clicks','hits'

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

Explanation :

creates a table named '**clicks**' with column family '**hits**'

Alter the table/Modify the table attributes to retains five versions for column values of **hits** column family.

```
put 'clicks','IP','hits:hostname','gpu_vdi_01'
put 'clicks','IP','hits:location','India'
put 'clicks','IP','hits:browser','chrome'
```

Explanation :

Inserting values in columns(hostname,location,browser <in purple>) for column family hits with row-key “**IP**” of table ‘**clicks**’

```
scan 'clicks'
```

Explanation :

Reading values of table clicks. To check if values are inserted.

```
put 'clicks','IP','hits:browser','firefox'
put 'clicks','IP','hits:browser','safari'
put 'clicks','IP','hits:browser','IExplorer'
put 'clicks','IP','hits:browser','Opera'
put 'clicks','IP','hits:browser','Chromium'
put 'clicks','IP','hits:browser','Netscape'
```

Explanation :

Changing or **updating** values of **browser** column to check if table retains the last five versions of the updated values.

```
scan 'clicks'
```

Explanation:

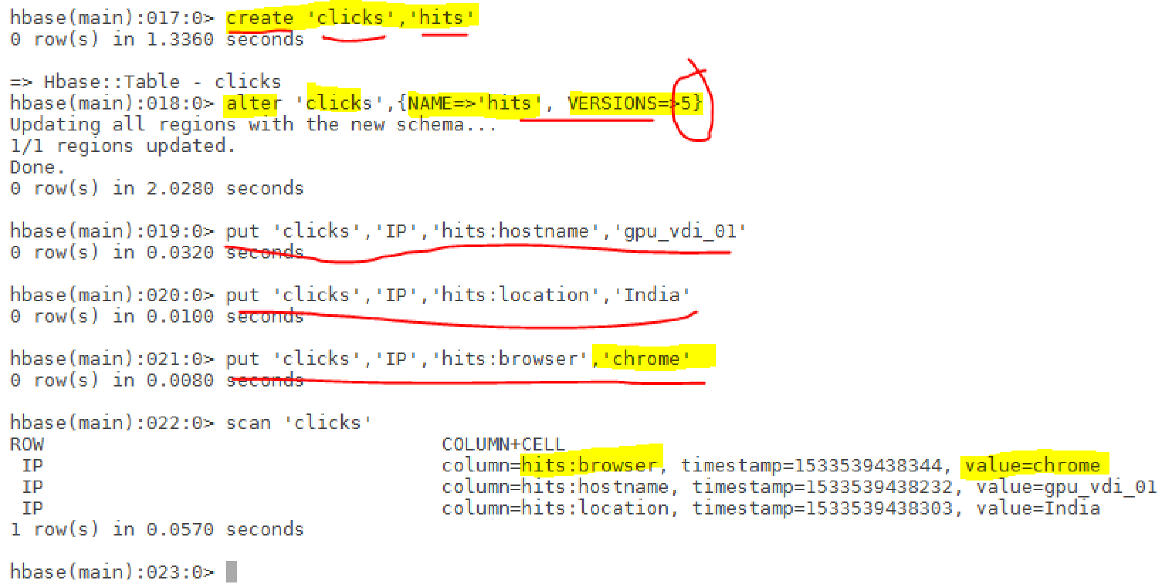
Although table clicks retains the latest updated value for **browser i.e** 'Netscape'

```
scan 'clicks',{COLUMN=>'hits:browser',VERSIONS=>5}
```

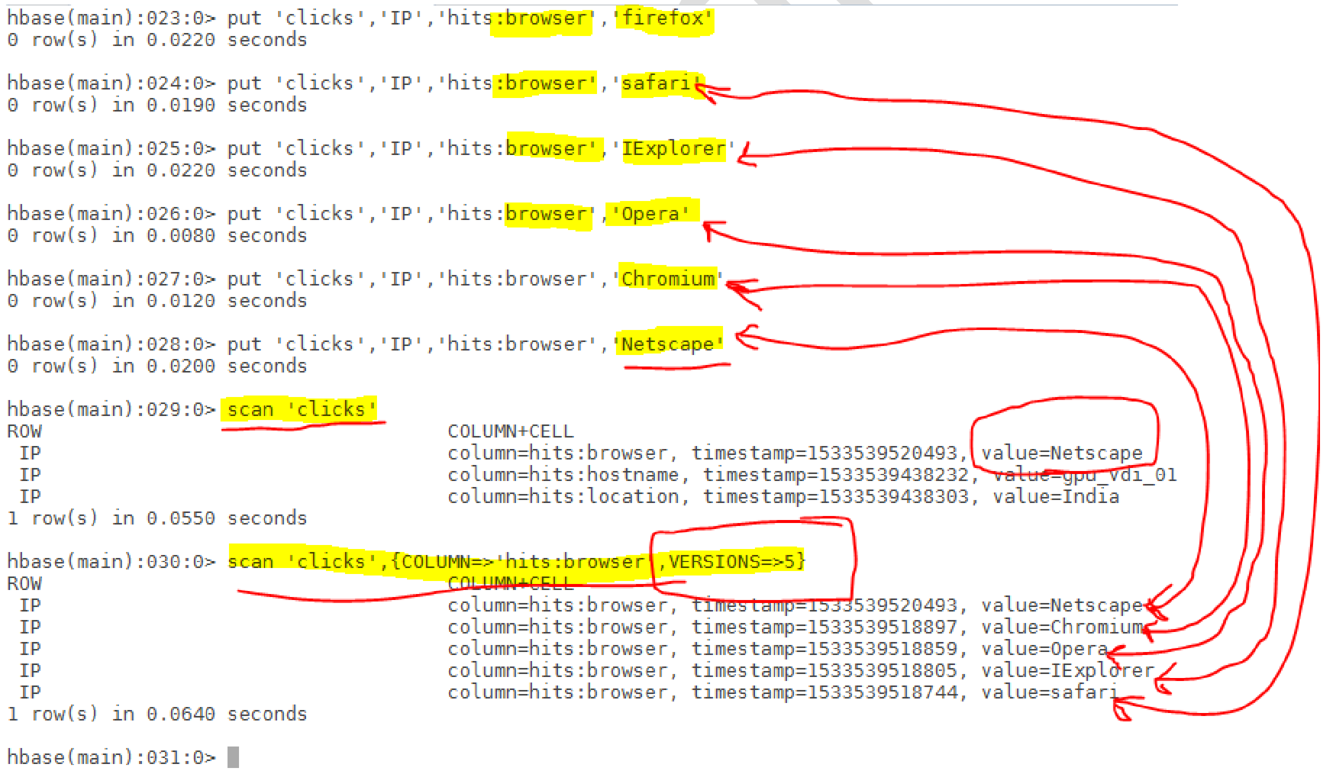
Explanation:

Versions=>5 shows the last five values that were updated/changed for column **browser** in column family **'hits'**

ScreenShot:



```
hbase(main):017:0> create 'clicks','hits'
0 row(s) in 1.3360 seconds

=> Hbase::Table - clicks
hbase(main):018:0> alter 'clicks',{NAME=>'hits', VERSIONS=>5}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 2.0280 seconds

hbase(main):019:0> put 'clicks','IP','hits:hostname','gpu_vdi_01'
0 row(s) in 0.0320 seconds

hbase(main):020:0> put 'clicks','IP','hits:location','India'
0 row(s) in 0.0100 seconds

hbase(main):021:0> put 'clicks','IP','hits:browser','chrome'
0 row(s) in 0.0080 seconds

hbase(main):022:0> scan 'clicks'
ROW                                  COLUMN+CELL
 IP                                  column=hits:browser, timestamp=1533539438344, value=chrome
 IP                                  column=hits:hostname, timestamp=1533539438232, value=gpu_vdi_01
 IP                                  column=hits:location, timestamp=1533539438303, value=India
1 row(s) in 0.0570 seconds

hbase(main):023:0>
```



```
hbase(main):023:0> put 'clicks','IP','hits:browser','firefox'
0 row(s) in 0.0220 seconds

hbase(main):024:0> put 'clicks','IP','hits:browser','safari'
0 row(s) in 0.0190 seconds

hbase(main):025:0> put 'clicks','IP','hits:browser','IExplorer'
0 row(s) in 0.0220 seconds

hbase(main):026:0> put 'clicks','IP','hits:browser','Opera'
0 row(s) in 0.0080 seconds

hbase(main):027:0> put 'clicks','IP','hits:browser','Chromium'
0 row(s) in 0.0120 seconds

hbase(main):028:0> put 'clicks','IP','hits:browser','Netscape'
0 row(s) in 0.0200 seconds

hbase(main):029:0> scan 'clicks'
ROW                                  COLUMN+CELL
 IP                                  column=hits:browser, timestamp=1533539520493, value=Netscape
 IP                                  column=hits:hostname, timestamp=1533539438232, value=gpu_vdi_01
 IP                                  column=hits:location, timestamp=1533539438303, value=India
1 row(s) in 0.0550 seconds

hbase(main):030:0> scan 'clicks',{COLUMN=>'hits:browser',VERSIONS=>5}
ROW                                  COLUMN+CELL
 IP                                  column=hits:browser, timestamp=1533539520493, value=Netscape
 IP                                  column=hits:browser, timestamp=1533539518897, value=Chromium
 IP                                  column=hits:browser, timestamp=1533539518859, value=Opera
 IP                                  column=hits:browser, timestamp=1533539518805, value=IExplorer
 IP                                  column=hits:browser, timestamp=1533539518744, value=safari
1 row(s) in 0.0640 seconds

hbase(main):031:0>
```

******************************** **End** ********************************