Assignment 10.1
HBASE Basics

<u>**Task 1:**</u>

Answer in your own words with example.

1.What is NoSQL database?

- NoSQL stands for Not only SQL. NoSQL are non-relational, open source, schema less and distributed database.
- They are highly popular today because of their ability to scale out or scale horizontally and flexibility to deal with a variety of data : structured, semi-structered and unstructured data.
- There are 4 types of NoSQL :
    1. **Key-Value pair /Key-Value store:**
       It maintains data in key-value pairs. It uses a hash table in which there exists a unique key and a pointer to a particular item of data.
       e.g.

Key	**Value**
First Name	Sachin
Last Name	Gorade
Address	Mumbai, India

       Amazon DynamoDB, LinkedIn uses key-value store NoSQL database.

    2. **Document store:**

       It maintains data in collections constituted of documents.
       It 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.
       e.g.

       ```
       {
         _id: ObjectId(fdf789)
         title: 'MongoDB Overview',
         description: 'MongoDB is no sql database',
         tags: ['mongodb', 'database', 'NoSQL'],
       }
       ```

       MongoDB, Apache CouchDB, Couchbase uses Document store.

Assignment 10.1
HBASE Basics

3. **Column:**

Each storage block has data from only one column. It stores data using a column oriented model.
e.g. Databases : Cassandra, HBase

Facebook, eBay uses Column NoSQL.

4. **Graph :**

A Graph stores data in nodes. They are also called as Network database.

e.g. Databases: Neo4j, hyperGraphDB.

Walmart uses Graph NoSQL.

2.How does data get stored in NoSQL database?

Data gets stored in any one of four mentioned type (format) like key-value pair, Document,Column Or Graph depending on the data you have with you.
In some cases you might be willing to sacrifice full data consistency in order to improve write speed.
In some other you will prioritize read times over write times.

3.What is a column family in HBase?

In HBase, columns are grouped into column families.
Column families are created at the time of table creation. They are stored together on disk, so grouping data with similar access patterns reduces overall disk access and increases performance.
Hence HBase is referred to as a column-oriented data store.

New column families can be added on demand.
Physically all Column families are stored together on the filesystem.

All column members of a column family have the same prefix.
e.g. columns **subjects: geographhy** and **subjects: math** are both members of the **subjects** column family.
The colon character (:) delimits the column family from columns.

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

There is no specific limit on the number of columns in a column family.
Actually you can have millions of columns in the single column family.

But 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.

Assignment 10.1
HBASE Basics

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

In HBase, columns are not defined at the time of table creation.
Because data which HBase supports is schema less and non relational. So for each row, we can have a different set of attributes (columns) and data structure keeps on changing.
Data structure is not fixed as it deals with unstructured data.
As HBase is sparse database, columns are optional.

6.How does data get managed in HBase?

HBase is built on top of the distributed file system (HDFS), which can store large files. HBase provides fast record lookups and updates for large tables.

The ZooKeeper cluster acts as a coordination service for the entire HBase cluster.

HBase contains two primary services:
Master server:
The master server co-ordinates the cluster and performs administrative operations, such as assigning regions and balancing the loads.

Region server:
The region servers do the real work. A subset of the data of each table is handled by each region server. Clients talk to region servers to access data in HBase.

Regions:
Region servers manage a set of regions.

An HBase table is made up of a set of regions. Regions are the basic unit of work in HBase. It is what is used as a split by the map reduce framework. The region contains store objects that correspond to column families. There is one store instance for each column family. Store objects create one or more StoreFiles, which are wrappers around the actual storage file called the HFile.

The region also contains a MemStore, which is in-memory storage and is used as a write cache. Rows are written to the MemStore. The data in the MemStore is ordered. If the MemStore becomes full, it is persisted to an HFile on disk

To improve performance, it is important to get an even distribution of data among regions, which ensures the best parallelism in map tasks.

HFiles:
HFiles are the physical representation of data in HBase. Clients do not read HFiles directly but go through region servers to get to the data.

Everything in HBase is stored as bytes and there are no types. There is no schema since each row in HBase can have a different set of columns.

Assignment 10.1
HBASE Basics

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

- As HBase tables can be large, they are broken up into partitions called **Regions**. Each **Region server** handles one or more of these regions.

- Once client finds Region in HBase to write, the client sends the request to the **Region server** which has that region for changes to be accepted. The region server cannot write the changes to a HFile immediately because the data in a HFile must be sorted by the row key. This allows searching for random rows efficiently when reading the data. Data cannot be randomly inserted into the HFile.

- Instead, the change must be written to a new file. If each update were written to a file, many small files would be created. Such a solution would not be scalable nor efficient to merge or read at a later time. Therefore, changes are not immediately written to a new HFile.

- Instead, each change is stored in a place in memory called the memstore, which cheaply and efficiently supports random writes. Data in the memstore is sorted in the same manner as data in a HFile. When the memstore accumulates enough data, the entire sorted set is written to a new HFile in HDFS.

- Although writing data to the Memstore is efficient, it also introduces an element of risk. Information stored in memstore is stored in volatile memory, so if the system fails, all memstore information is lost. To help mitigate this risk, HBase saves updates in a **write-ahead-log (WAL)** before writing the information to memstore. In this way, if a region server fails, information that was stored in that server's memstore can be recovered from its WAL.

- WAL files contain a list of edits, with one edit representing a single put or delete. The edit includes information about the change and the region to which the change applies. Edits are written chronologically, so, for persistence, additions are appended to the end of the WAL file that is stored on disk.

- As WALs grow, they are eventually closed and a new active WAL file is created to accept additional edits. This is called rolling the WAL file. Once a WAL file is rolled, no additional changes are made to the old file. By default, WAL file is rolled when its size is about 95% of the HDFS block size.

- A region server serves many regions, but does not have a WAL file for each region. Instead, one active WAL file is shared among all regions served by the region server. Because WAL files are rolled periodically, one region server may have many WAL files. Note that there is only one active WAL per region server at a given time.

Assignment 10.1
HBASE Basics

- Summary :

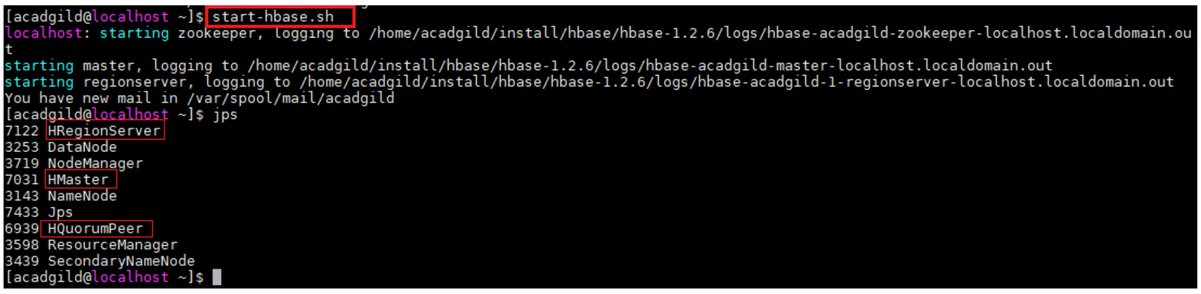
- When the client gives a command to write, instruction is directed to Write Ahead Log.
- Once the log entry is done, the data to be written is forwarded to MemStore which is actually RAM of the data node.
- Data in the memstore is sorted in the same manner as data in a HFile.
- When the memstore accumulates enough data, the entire sorted set is written to a new HFile in HDFS.
- Once writing data is completed, ACK (Acknowledgement) is sent to the client as a confirmation of task completed.

Task 2

1. Create an HBase table named 'clicks' with a column family 'hits' such that it should be

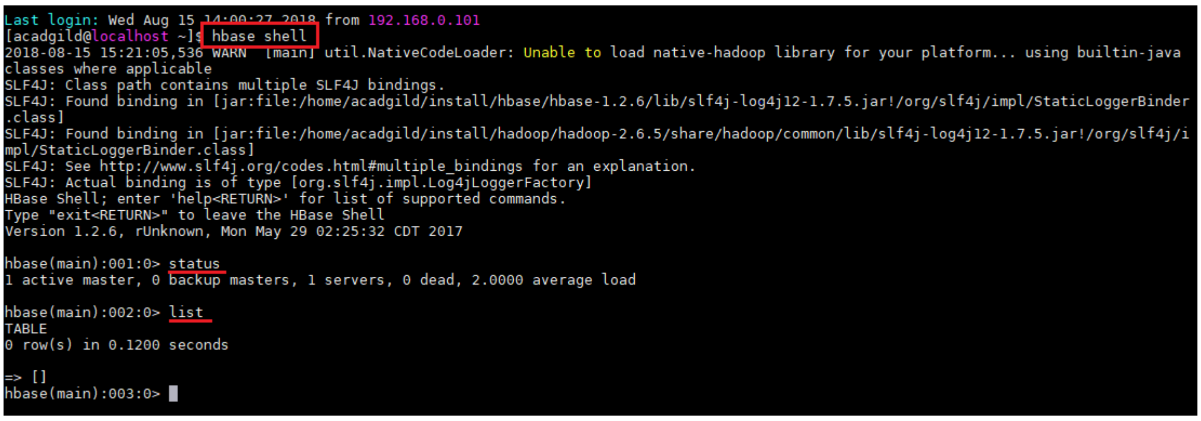
able to store last 5 values of qualifiers inside 'hits' column family.

To start HBase, we need to use **start-hbase.sh** command. After using **jps** command, you could see below three processes gets started:
HMaster
HRegionServer
HQuorumPeer



```
[acadgild@localhost ~]$ start-hbase.sh
localhost: starting zookeeper, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-acadgild-zookeeper-localhost.localdomain.ou
t
starting master, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-acadgild-master-localhost.localdomain.out
starting regionserver, logging to /home/acadgild/install/hbase/hbase-1.2.6/logs/hbase-acadgild-1-regionserver-localhost.localdomain.out
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost ~]$ jps
7122 HRegionServer
3253 DataNode
3719 NodeManager
7031 HMaster
3143 NameNode
7433 Jps
6939 HQuorumPeer
3598 ResourceManager
3439 SecondaryNameNode
[acadgild@localhost ~]$
```

Then to use hbase shell, we have used ‘hbase shell’ command :
We have verified it’s working fine by using commands **status** and **list**



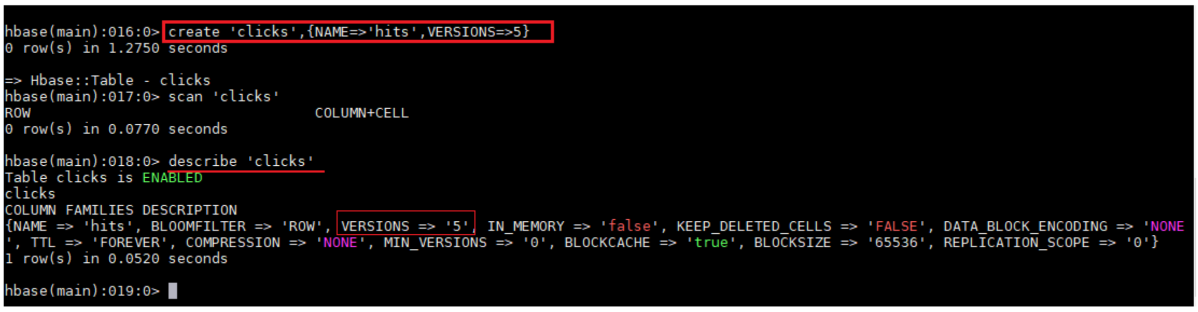
```
Last login: Wed Aug 15 14:00:27 2018 from 192.168.0.101
[acadgild@localhost ~]$ hbase shell
2018-08-15 15:21:05,536 WARN  [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/acadgild/install/hbase/hbase-1.2.6/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder
.class]
SLF4J: Found binding in [jar:file:/home/acadgild/install/hadoop/hadoop-2.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/i
mpl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000 average load

hbase(main):002:0> list
TABLE
0 row(s) in 0.1200 seconds

=> []
hbase(main):003:0>
```

Assignment 10.1
HBASE Basics

We have created clicks table **clicks** with column family **hits** and versions as **5** so that it can store at maximum 5 values (versions) for a column cell by using below command.

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

Then by using command **describe 'clicks'**, we could see that version is set as 5 :

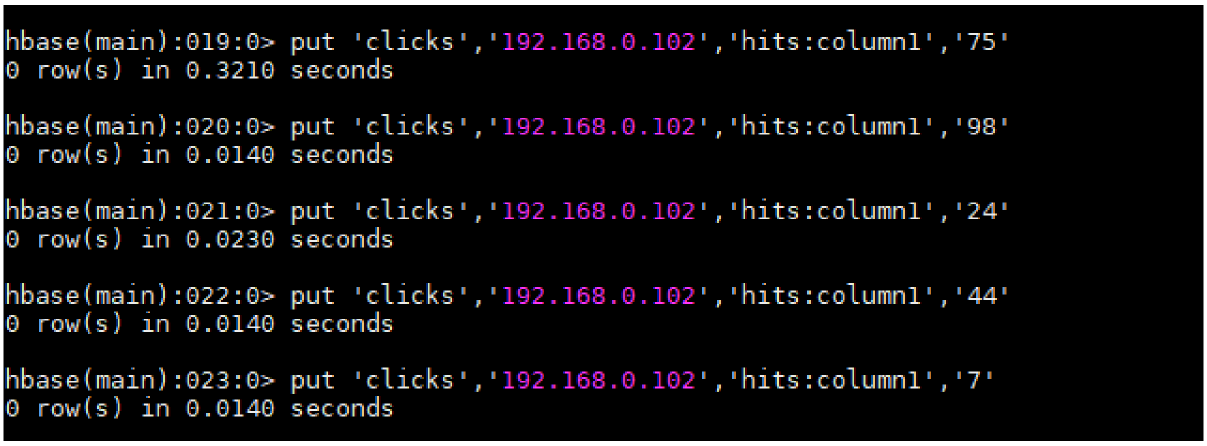


```
hbase(main):016:0> create 'clicks',{NAME=>'hits',VERSIONS=>5}
0 row(s) in 1.2750 seconds

=> Hbase::Table - clicks
hbase(main):017:0> scan 'clicks'
ROW                                COLUMN+CELL
0 row(s) in 0.0770 seconds

hbase(main):018:0> describe 'clicks'
Table clicks is ENABLED
clicks
COLUMN FAMILIES DESCRIPTION
{NAME => 'hits', BLOOMFILTER => 'ROW', VERSIONS => '5', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE
', TTL => 'FOREVER', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.0520 seconds

hbase(main):019:0>
```

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.

Here we are using IP address **192.168.0.102** as row-key. Then we are adding 5 different values for **column1** under column family **hits** by using below command as:

put 'clicks','192.168.0.102','hits:column1','75'



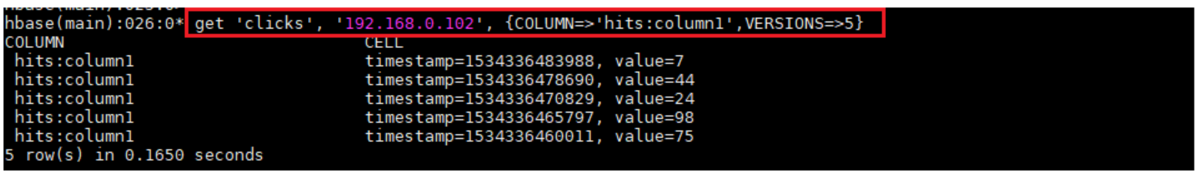
```
hbase(main):019:0> put 'clicks','192.168.0.102','hits:column1','75'
0 row(s) in 0.3210 seconds

hbase(main):020:0> put 'clicks','192.168.0.102','hits:column1','98'
0 row(s) in 0.0140 seconds

hbase(main):021:0> put 'clicks','192.168.0.102','hits:column1','24'
0 row(s) in 0.0230 seconds

hbase(main):022:0> put 'clicks','192.168.0.102','hits:column1','44'
0 row(s) in 0.0140 seconds

hbase(main):023:0> put 'clicks','192.168.0.102','hits:column1','7'
0 row(s) in 0.0140 seconds
```

Assignment 10.1
HBASE Basics

Then to fetch last 5 versions for column1, we have used below command :

get 'clicks', '192.168.0.102', {COLUMN=>'hits:column1',VERSIONS=>5}

This will show last 5 values of column1 as 7,44,24,98,75. As we have written value 7 at the end hence it is showing at the top as it is most recently updated.

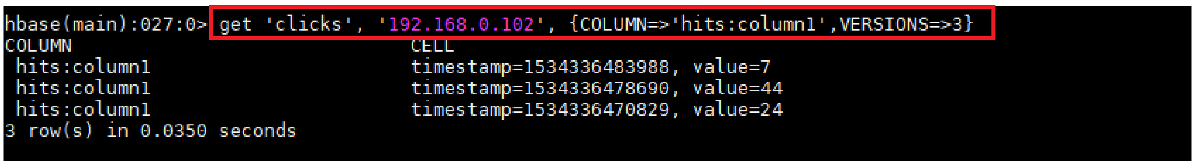


```
hbase(main):026:0* get 'clicks', '192.168.0.102', {COLUMN=>'hits:column1',VERSIONS=>5}
COLUMN                               CELL
 hits:column1                        timestamp=1534336483988, value=7
 hits:column1                        timestamp=1534336478690, value=44
 hits:column1                        timestamp=1534336470829, value=24
 hits:column1                        timestamp=1534336465797, value=98
 hits:column1                        timestamp=1534336460011, value=75
5 row(s) in 0.1650 seconds
```

If we want to fetch last 3 versions for column1, we have used below command :

get 'clicks', '192.168.0.102', {COLUMN=>'hits:column1',VERSIONS=>3}

This will show last 3 values of column1 as 7,44,24. As we have written value 7 at the end hence it is showing at the top as it is most recently updated.



```
hbase(main):027:0> get 'clicks', '192.168.0.102', {COLUMN=>'hits:column1',VERSIONS=>3}
COLUMN                               CELL
 hits:column1                        timestamp=1534336483988, value=7
 hits:column1                        timestamp=1534336478690, value=44
 hits:column1                        timestamp=1534336470829, value=24
3 row(s) in 0.0350 seconds
```