**Assignment 9.1**

Problem Statement

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?

PS1: What is NoSQL data base?

NoSQL is a very broad term that doesn’t refer to one particular database model. Rather, it refers to a whole variety of different models that don’t fit into the relational model.

NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.

Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

The term NoSQL now generally refers to a particular group of DBMSs that share certain characteristics, such as the following.

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.

* Open Source
* Schema-Less
* Horizontally Scalable
* Lack of Adherence to ACID Principles
* No Standard Query Language
* NoSQL DBs use UnQL (Unstructured Query Language) for querying. The syntax of using UnQL varies from database to database.

**Types of NoSQL DB:**

* **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.

Schemas are dynamic. Each row need not have data for each column.

**PS 2: How does data get stored in NoSQl database?**

Varies based on database type.

key-value stores function similarly to SQL databases, but have only two columns ('key' and 'value'), with more complex information sometimes stored as BLOBs within the 'value' columns.

Document databases do away with the table-and-row model altogether, storing all relevant data together in single 'document' in JSON, XML, or another format, which can nest values hierarchically.

HBASE store columns of data together, instead of rows.

**HBASE USES HDFS TO STORE DATA**

*Apache HBase* is a distributed column-oriented database built on top of the Hadoop file system  and it is horizontally scalable meaning we can add the new nodes to Hbase as data grows.

It is well suited for sparse data sets, which are common in many big data use cases.

PS 3 :What is a column family in HBase?

HBASE data model (COLUMNAR ) i.e. column-oriented

In column-oriented database values of one column is very much similar in nature or even vary only slightly between logical rows and this makes them a very suitable candidate for compression when compared to the heterogeneous values of row oriented record structures.

TABLE consisted of following property.

ROWS--->COLLECTION OF COLUMN FAMILIES

COLUMN FAMILIES--->COLLECTION OF COLUMNS

COLUMNS--->COLECTION OF KEY/VALUE PAIR

Ex:

COLUMN FAMILY-->PERSONAL & PROFESSIONAL

PERSONLA COLUMN FAMILY--->NAME,CITY

PROFESSIONAL COLUMN FAMILY--->ID,COMPANY

PS 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.

PS 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 column family qualifier. 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.

PS 6: How does data get managed in HBase?

We can discuss in two way:

Writing data to HBASE:

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

* A Put operation writes data into HBase.

Reading data to Hbase:

[Get](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Get.html) and [Scan](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Scan.html) are the two ways to read data from HBase, aside from manually parsing HFiles. A Get is simply a Scan limited by the API to one row. A Scan fetches zero or more rows of a table. By default, a Scan reads the entire table from start to end. You can limit your Scan results in several different ways, which affect the Scan's load in terms of IO, network, or both, as well as processing load on the client side.

Hadoop 2.4 introduced a new feature called **hedged reads**. If a read from a block is slow, the HDFS client starts up another parallel, 'hedged' read against a different block replica. The result of whichever read returns first is used, and the outstanding read is cancelled. This feature helps in situations where a read occasionally takes a long time rather than when there is a systemic problem. Hedged reads can be enabled for HBase when the HFiles are stored in HDFS.

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

**HBASE Architecture**

Physically, HBase is composed of three types of servers in a master slave type of architecture. Region servers serve data for reads and writes. When accessing data, clients communicate with HBase RegionServers directly. Region assignment, DDL (create, delete tables) operations are handled by the HBase Master process. Zookeeper, which is part of HDFS, maintains a live cluster state.

The Hadoop DataNode stores the data that the Region Server is managing. All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers. HBase data is local when it is written, but when a region is moved, it is not local until compaction.

The NameNode maintains metadata information for all the physical data blocks that comprise the files.

HBase Tables are divided horizontally by row key range into “Regions.” A region contains all rows in the table between the region’s start key and end key. Regions are assigned to the nodes in the cluster, called “Region Servers,” and these serve data for reads and writes. A region server can serve about 1,000 regions.

Region assignment, DDL (create, delete tables) operations are handled by the HBase Master.

A master is responsible for:

* Coordinating the region servers

- Assigning regions on startup , re-assigning regions for recovery or load balancing

- Monitoring all RegionServer instances in the cluster (listens for notifications from zookeeper)

* Admin functions

- Interface for creating, deleting, updating tables

HBase uses ZooKeeper as a distributed coordination service to maintain server state in the cluster. Zookeeper maintains which servers are alive and available, and provides server failure notification. Zookeeper uses consensus to guarantee common shared state. Note that there should be three or five machines for consensus.

There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster. ZooKeeper stores the location of the META table.

This is what happens the first time a client reads or writes to HBase:

1. The client gets the Region server that hosts the META table from ZooKeeper.
2. The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
3. It will get the Row from the corresponding Region Server.

For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

A Region Server runs on an HDFS data node and has the following components:

* 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.

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

* - Edits are appended to the end of the WAL file that is stored on disk.
* - The WAL is used to recover not-yet-persisted data in case a server crashes.

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.

The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.

When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS. HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances. These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.

Note that this is one reason why there is a limit to the number of column families in HBase. There is one MemStore per CF; when one is full, they all flush. It also saves the last written sequence number so the system knows what was persisted so far.

The highest sequence number is stored as a meta field in each HFile, to reflect where persisting has ended and where to continue. On region startup, the sequence number is read, and the highest is used as the sequence number for new edits.

Data is stored in an HFile which contains sorted key/values. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.

An HFile contains a multi-layered index which allows HBase to seek to the data without having to read the whole file. The multi-level index is like a b+tree:

* Key value pairs are stored in increasing order
* Indexes point by row key to the key value data in 64KB “blocks”
* Each block has its own leaf-index
* The last key of each block is put in the intermediate index
* The root index points to the intermediate index

The trailer points to the meta blocks, and is written at the end of persisting the data to the file. The trailer also has information like bloom filters and time range info. Bloom filters help to skip files that do not contain a certain row key. The time range info is useful for skipping the file if it is not in the time range the read is looking for.

We have seen that the KeyValue cells corresponding to one row can be in multiple places, row cells already persisted are in Hfiles, recently updated cells are in the MemStore, and recently read cells are in the Block cache. So when you read a row, how does the system get the corresponding cells to return? A Read merges Key Values from the block cache, MemStore, and HFiles in the following steps:

1. First, the scanner looks for the Row cells in the Block cache - the read cache. Recently Read Key Values are cached here, and Least Recently Used are evicted when memory is needed.
2. Next, the scanner looks in the MemStore, the write cache in memory containing the most recent writes.
3. If the scanner does not find all of the row cells in the MemStore and Block Cache, then HBase will use the Block Cache indexes and bloom filters to load HFiles into memory, which may contain the target row cells.