**Assignment-32.1:**

**Question 1:**

# **What is the difference between memstore and hfile in HBase?**

|  |  |
| --- | --- |
| **Memstore** | **HFILE** |
| The MemStore is a write buffer where HBase accumulates data in memory before a permanent write. | The HFile is the underlying storage format for HBase. |
| Memstore is kept in RegionServer (RS) main memory. | HFiles are written to HDFS. |
| When write request is processed, data is first written into the Memstore. | When certain thresholds are met Memstore data gets flushed into HFile. |

**Question 2:**

**Describe compactions in HBase.**

\* Apache HBase is a distributed data store based upon a log-structured merge tree, so optimal read performance would come from having only one file per store (Column Family).

\* However, that ideal isn’t possible during periods of heavy incoming writes.

\* Instead, HBase will try to combine HFiles to reduce the maximum number of disk seeks needed for a read. This process is called compaction.

Compaction, the process by which HBase cleans up after itself, comes in two flavors: major and minor.

**Minor compactions:**

1. Minor compactions combine a configurable number of smaller HFiles into one larger HFile.

2. You can tune the number of HFiles to compact and the frequency of a minor compaction.

3. Minor compactions are important because without them, reading a particular row can require many disk reads and cause slow overall performance.

4. HBase designers took special care to give the HBase administrator as much tuning control as possible to make any system impact “minor.”

**Major compaction:**

1. As its name implies, a major compaction is different from the perspective of a system impact. However, the compaction is quite important to the overall functionality of the HBase system.

2. A major compaction seeks to combine all HFiles into one large HFile.

3. In addition, a major compaction does the cleanup work after a user deletes a record. When a user issues a Delete call, the HBase system places a marker in the key-value pair so that it can be permanently removed during the next major compaction.

4. Users who are trying to add, retrieve, or manipulate data in the system during a major compaction, they may see poor system response time

**Question 3:**

**List and explain the logical entities in HBase.**

**HBase Data Model**

The Data Model in HBase is designed to accommodate semi-structured data that could vary in field size, data type and columns. Additionally, the layout of the data model makes it easier to partition the data and distribute it across the cluster.

**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.

**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

**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.

**Question 4:**

**What will happen if we do not create a row key while inserting the data?**

\* Hbase is a Nosql database and hence has a columnar structure.

\* When we consider the storage of the data in hbase it is divide into regions. These regions are horizontal partitions of table. Each row in the table should be given to specify start and end key of the region.

\* So in order specify region row key is must.

\* Hbase provides us to random access to particular record and modify it. This is also done with the help of the specifying the row key of that record.

\* Even in case of the deleting the record markers are set rather than directly deleting the data. Then when the major compaction is done after a period of time the record is deleted.

**Question 5:**

**How can filters be applied in HBase and what are the benefits?**

\* HBase can query data very quickly on demand but specific use cases may require to only return a subset of the scan results.

\* Instead of scanning the entire dataset only to return a subset to the client, we can use Filters to get the data closer to what we need in less amount of time.

\* Thus, HBase has a set of predefined Filters as well as custom filters that we can use to scan and get filtered results from the HBase database.

\* There are two prominent ways to read data from HBase.

\* When reading data from HBase using Get or Scan operations, we can use custom filters to return a subset of results to the client. It does reduce network bandwidth and reduces the amount of data the client needs to process.

**Question 6:**

**What are the data model operations in hBase?**

**Data Model Operations**

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

**1. Get**

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

**2. 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).

**3. Scans**

Scan allow iteration over multiple rows for specified attributes.

**4. Delete**

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

HBase does not modify data in place, and so deletes are handled by creating new markers called tombstones. These tombstones, along with the dead values, are cleaned up on major compactions.

**Question 7:**

**How can MapReduce be used with HBase?**

\* HBase provides a TableInputFormat, to which you provided a table scan, that splits the rows resulting from the table scan into the regions in which those rows reside.

\* The map process is passed an ImmutableBytesWritable that contains the row key for a row and a Result that contains the columns for that row.

\* The map process outputs its key/value pair based on its business logic in whatever form makes sense to your application.

\* The reduce process builds its results but emits the row key as an ImmutableBytesWritableand a Put command to store the results back to HBase.

**Question 8:**

**What is regionserver?**

\* 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. RegionServers are the software processes (often called daemons) you activate to store and retrieve data in HBase (Hadoop Database).

\* In production environments, each RegionServer is deployed on its own dedicated compute node. When you start using HBase, you create a table and then begin storing and retrieving your data.