# **ASSIGNMENT 32.1**

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

# **MEMSTORE:**

# The MemStore is a write buffer where HBase accumulates data in memory before a permanent write. Its contents are flushed to disk to form an HFile when the MemStore fills up. It doesn’t write to an existing HFile but instead forms a new file on every flush.  There is one MemStore per column family. (The size of the MemStore is defined by the system-wide property in  hbase-site.xml called hbase.hregion.memstore.flush.size)

# **HFILE:**

# The HFile is the underlying storage format for HBase. HFiles belong to a column family and a column family can have multiple HFiles. But a single HFile can’t have data for multiple column families.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*2****)* Describe compactions in HBase.**

[Apache HBase](http://hbase.apache.org/) 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.

Compactions choose some files from a single store in a region and combine them. This process involves reading KeyValues in the input files and writing out any KeyValues that are not deleted, are inside of the time to live (TTL), and don’t violate the number of versions. The newly created combined file then replaces the input files in the region.

Now, whenever a client asks for data, HBase knows the data from the input files are held in one contiguous file on disk — hence only one seek is needed, whereas previously one for each file could be required.

There are two types of compactions. They are

* Minor compactions - combine a configurable number of smaller HFiles into one larger HFile.
* Major compactions - seeks to combine all HFiles into one large HFile.

In addition, a major compaction does the cleanup work after a user deletes a record.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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.

So we can say that row key is very important aspect of the hbase. It is used to denote the lowest unit of storage that is record.

Without the row key no operations on the particular record will be possible.

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

HBase includes several filter types, as well as the ability to group filters together and create your own custom filters.

* **KeyOnlyFilter** - takes no arguments. Returns the key portion of each key-value pair.
* **FirstKeyOnlyFilter** - takes no arguments. Returns the key portion of the first key-value pair.
* **PrefixFilter** - takes a single argument, a prefix of a row key. It returns only those key-values present in a row that start with the specified row prefix
* **ColumnPrefixFilter** - takes a single argument, a column prefix. It returns only those key-values present in a column that starts with the specified column prefix.
* **MultipleColumnPrefixFilter** - takes a list of column prefixes. It returns key-values that are present in a column that starts with *any* of the specified column prefixes.
* **ColumnCountGetFilter** - takes one argument, a limit. It returns the first limit number of columns in the table.
* **PageFilter** - takes one argument, a page size. It returns page size number of rows from the table.
* **ColumnPaginationFilter** - takes two arguments, a limit and offset. It returns limit number of columns after offset number of columns. It does this for all the rows.
* **InclusiveStopFilter** - takes one argument, a row key on which to stop scanning. It returns all key-values present in rows *up to and including* the specified row.
* **TimeStampsFilter** - takes a list of timestamps. It returns those key-values whose timestamps matches *any* of the specified timestamps.
* **RowFilter** - takes a compare operator and a comparator. It compares each row key with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that row.
* **FamilyFilter** - takes a compare operator and a comparator. It compares each family name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that family.
* **QualifierFilter** - takes a compare operator and a comparator. It compares each qualifier name with the comparator using the compare operator and if the comparison returns true, it returns all the key-values in that column.
* **ValueFilter** - takes a compare operator and a comparator. It compares each value with the comparator using the compare operator and if the comparison returns true, it returns that key-value.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

The four primary data model operations are Get, Put, Scan, and Delete.

**Get:**

[Get](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Get.html) returns attributes for a specified row.

**Put:**

[Put](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Put.html) either adds new rows to a table (if the key is new) or can update existing rows (if the key already exists).

**Scan:**

[Scan](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Scan.html) allow iteration over multiple rows for specified attributes.

**Delete:**

[Delete](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Delete.html) removes a row from a table.

**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.

Finally, the results are stored in HBase by the HBase MapReduce infrastructure. (You do not need to execute the Put commands.)

Map reduce can be used to process the data stored in hbase. For processing the data there is a special implementation called tableinputformatbase whose subclass is tableinputformat. The former implements the majority of the functionality but remains abstract. The subclass is a lightweight concrete version of tableinputformat and is used by many supplied samples and real mapreduce classes.

Hbase provides the tablemapper class that enforces key class 1 to be an immutablebyteswritable, and value class 1 to be a Result type—since that is what the tablerecordreader is returning.

**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.

However, at some point — and perhaps quite quickly in big data use cases — the table grows beyond a configurable limit. At this point, the HBase system automatically splits the table and distributes the load to another RegionServer.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_