# **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/) - distributed data store.
* Which is based on log-structured merge tree.
* Optimal read performance will be used for only one file per store (Column Family).
* Ideally it is not possible during the time of heavy incoming writes.
* HBase will combine HFiles in order to reduce maximum number of disk specially to read.
* And this process is called as compaction.
* Compactions means it will choose some file from single store in a particular region.
* And will combine those.
* All the process have reading the Key pair Values from input files
* Those input will be written out to any of the KeyValues which is not deleted inside time to live (TTL).
* Number of versions will not be violated.
* New combined file which were created will be replaced by input files.
* When any client questions for data then HBase will know about the data from input files which were held by contiguous file on disk .
* one seek will be needed.
* Previously seek for each file is required.

And there are 2 types of compactions and they are:

1.)**Minor compactions -** will combine configurable files into number of smaller HFiles to larger HFile.

2.)**Major compactions -** seeks will combine all HFiles into one a large HFile.

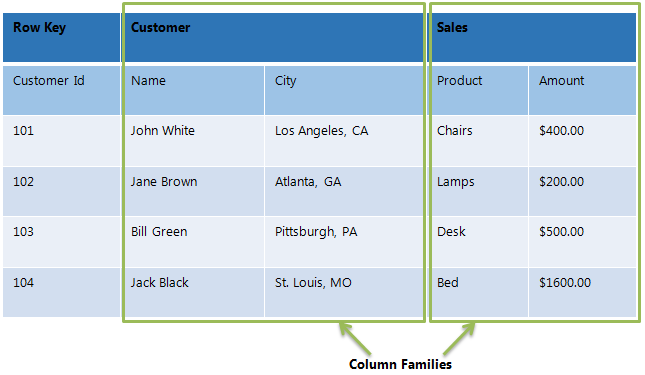
compaction will not cleanup work after a user deletes a record.

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

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.

The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells and Versions.



**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. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence it’s important that proper care be taken when designing Column Families in table. The table above shows Customer and Sales Column Families. The Customer Column Family is made up 2 columns – Name and City, whereas the Sales Column Families is made up to 2 columns – Product and Amount.

**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 – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have varied number of Columns.

**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.

**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.

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.

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

When reading data from HBase using Get or Scan operations, you can use custom filters to return a subset of results to the client.

While this does not reduce server-side IO, it does reduce network bandwidth and reduces the amount of data the client needs to process.

Filters are generally used using the Java API, but can be used from HBase Shell for testing and debugging purposes.

Dynamically loading the custom filter:

CDH 5.5 and higher adds (and enables by default) the ability to dynamically load a custom filter by adding a JAR with your filter to the directory specified by the hbase.dynamic.jars.dir property (which defaults to the lib/ directory under the HBase root directory).

To disable automatic loading of dynamic JARs, set hbase.use.dynamic.jars to false in the advanced configuration snippet for hbase-site.xml if you use Cloudera Manager, or to hbase-site.xml otherwise.

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

**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.

**PageFilter** - takes one argument, a page size. It returns page size number of rows from the table.

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

There are 4 primary data model operations and they are

* 1. Get
  2. Put
  3. Scan
  4. Delete

**Get:**

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

**Put:**

[Put](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Put.html) will either add new row to a table when the provided key is new or it can update with a existing rows when a key already exists.

**Scan:**

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

**Delete:**

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

**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.

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.

**8.What is regionserver?**

* HBase Tables are divided horizontally by means of row key range into “Regions.”
* Region will contain all the rows in a table between region’s start key and end key.
* Regions were assigned to nodes in a cluster, called as “Region Servers,”.
* Serve data will be used for reads and writes.
* Region server will serve about 1,000 regions.
* RegionServers are basically software processes called as daemons.
* One can activate store and retrieve data into a HBase Hadoop Database.
* For production environments each RegionServer will be deployed on its own which is dedicated into compute node.
* One can start HBase, and can create a table
* Which is used to store and retrieve data.
* Table will grow beyond the configurable limit.
* HBase system will automatically splits table and will distributes those load to another RegionServer.