**ASSIGNMENT 32.1**

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.

HFile-

* The HFile is the underlying storage format for HBase.
* HFiles belong to a column family(one MemStore per column family). A column family can have multiple HFiles, but the reverse isn't true.
* size of the MemStore is defined in hbase-site.xml called hbase.hregion.memstore.flush.size.
* HFiles are immutable.
* **Data from a single column family for a single row need not be stored in the same HFile.**

**Difference-**

When something is written to HBase, it is first written to an in-memory store (memstore), once this memstore reaches a certain size, it is flushed to disk into a store file (everything is also written immediately to a log file for durability). The store files created on disk are immutable. Sometimes the store files are merged together, this is done by a process called compaction.

**2) Describe compactions in HBase.**

**Compaction**

[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. But disk IO isn’t free, and without careful attention, rewriting data over and over can lead to some serious network and disk over-subscription. In other words, compaction is about trading some disk IO now for fewer seeks later.

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

**Logical entities of HBase**

1. **Normalization**

In a relational database, you normalize the schema to eliminate redundancy by putting repeating information into a table of its own. This has the following benefits:

* You don’t have to update multiple copies when an update happens, which makes writes faster.
* You reduce the storage size by having a single copy instead of multiple copies.

1. **De-normalization**

In a de-normalized datastore, you store in one table what would be multiple indexes in a relational world. De-normalization can be thought of as a replacement for joins. Often with HBase, you de-normalize or duplicate data so that data is accessed and stored together.

1. **Generic Data, Event Data, and Entity-Attribute-Value**

Generic data that is schemaless is often expressed as name value or entity attribute value. In a relational database, this is complicated to represent. A conventional relational table consists of attribute columns that are relevant for every row in the table, because every row represents an instance of a **similar object**. A different set of attributes represents a different type of object, and thus belongs in a different table. The advantage of HBase is that you can define columns on the fly, put attribute names in column qualifiers, and group data by column families.

1. **Self-Join Relationship – HBase**

A self-join is a relationship in which both match fields are defined in the same table.

Consider a schema for twitter relationships, where the queries are: which users does userX follow, and which users follow userX? Here’s a possible solution: The userids are put in a composite row key with the relationship type as a separator. For example, Carol follows Steve Jobs and Carol is followed by BillyBob. This allows for row key scans for everyone carol:follows or carol:followedby

1. **Tree, Graph Data**

Each row shows a node, and the row key is equal to the node id. There is a column family for parent p, and a column family children c. The column qualifiers are equal to the parent or child node ids, and the value is equal to the type to node. This allows to quickly find the parent or children nodes from the row key.

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

**Row Key-**

Every interaction you are going to do in database will start with the RowKey only, so a row key can not be empty.

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

Filters In Hbase Shell,Filter Language was introduced in APache HBase 0.92.

**Filter commands are:**

KeyOnlyFilter

This filter does not take any arguments. It returns only the key component of each key-value.

### FirstKeyOnlyFilter

This filter doesn’t take any arguments. It returns only the first key-value from each row.

### prefixfilter:

This filter takes one argument prefix of a row key. It returns only those key-values present in a row that starts with the specified row prefix

### ColumnPrefixFilter

This filter takes one argument column prefix. It returns only those key-values present in a column that starts with the specified column prefix. The column prefix must be of the form qualifier

### MultipleColumnPrefixFilter

This filter 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. Each of the column prefixes must be of the form qualifier

### ColumnCountGetFilter

This filter takes one argument limit. It returns the first limit number of columns in the table.

### PageFilter

This filter takes one argument page size. It returns page size number of rows from the table.

### InclusiveStopFilter

This filter takes one argument row key on which to stop scanning. It returns all key-values present in rows up to and including the specified row.

### Family Filter (Qualifier Filter)

This filter 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**

This filter 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.

**Benefits of filter:**

Used for debugging and testing purposes.

Filters are a powerful feature of HBase to delegate the selection of rows to the servers rather than moving rows to the Client.

It allows you to perform server-side filtering when accessing [HBase](http://www.hadooptpoint.com/category/nosql-databases/hbase/" \t "_blank)**.**

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

The four primary data model operations are Get, Put, Scan, and Delete. Operations are applied via [HTable](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html" \t "_top) instances.

**Get**

[Get](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Get.html) returns attributes for a specified row. Gets are executed via [HTable.get](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html" \l "get%28org.apache.hadoop.hbase.client.Get%29" \t "_top).

**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). Puts are executed via [HTable.put](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html" \l "put%28org.apache.hadoop.hbase.client.Put%29" \t "_top) (writeBuffer) or [HTable.batch](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html" \l "batch%28java.util.List%29" \t "_top)(non-writeBuffer).

**Scans**

[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. Deletes are executed via [HTable.delete](http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/HTable.html" \l "delete%28org.apache.hadoop.hbase.client.Delete%29" \t "_top).

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.

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

1. 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.
2. 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.
3. The map process outputs its key/value pair based on its business logic in whatever form makes sense to your application.
4. The reduce process builds its results but emits the row key as an ImmutableBytesWritableand a Put command to store the results back to HBase.
5. Finally, the results are stored in HBase by the HBase MapReduce infrastructure. (You do not need to execute the Put commands.)

**8)What is regionserver?**

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, the table grows beyond a configurable limit. At this point, the HBase system automatically splits the table and distributes the load to another RegionServer.