**Hbase Real Time Questions & Answers**

**Important Points:**

Apache HBase is an open-source, non-relational, distributed database modeled after Google's Bigtable and is written in Java.

It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System), providing Bigtable-like capabilities for Hadoop.   
  
**1)When Should I Use HBase?**   
**Answer:** HBase isn't suitable for every problem.  
First, make sure you have enough data. If you have hundreds of millions or billions of rows, then HBase is a good candidate. If you only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of your data might wind up on a single node (or two) and the rest of the cluster may be sitting idle.  
Second, make sure you can live without all the extra features that an RDBMS provides (e.g., typed columns, secondary indexes, transactions, advanced query languages, etc.) An application built against an RDBMS cannot be "ported" to HBase by simply changing a JDBC driver, for example. Consider moving from an RDBMS to HBase as a complete redesign as opposed to a port.  
Third, make sure you have enough hardware. Even HDFS doesn't do well with anything less than 5 DataNodes (due to things such as HDFS block replication which has a default of 3), plus a NameNode.  
HBase can run quite well stand-alone on a laptop - but this should be considered a development configuration only.  
  
**2)Can you create HBase table without assigning column family?**   
**Answer:** No, Column family also impact how the data should be stored physically in the HDFS file system, hence there is a mandate that you should always have at least one column family. We can also alter the column families once the table is created.   
  
**3)Does HBase support SQL?**   
**Answer:** Not really. SQL-ish support for HBase via Hive is in development, however Hive is based on MapReduce which is not generally suitable for low-latency requests.   
  
**4)Why are the cells above 10MB not recommended for HBase?**   
**Answer:** Large cells don’t fit well into HBase’s approach to buffering data. First, the large cells bypass the MemStoreLAB when they are written. Then, they cannot be cached in the L2 block cache during read operations. Instead, HBase has to allocate on-heap memory for them each time. This can have a significant impact on the garbage collector within the RegionServer process.   
**5)How should I design my schema in HBase?**   
**Answer:** A good introduction on the strength and weaknesses modelling on the various non-rdbms datastores is to be found in Ian Varley’s Master thesis, No Relation: The Mixed Blessings of Non-Relational Databases. It is a little dated now but a good background read if you have a moment on how HBase schema modeling differs from how it is done in an RDBMS. Also, read keyvalue for how HBase stores data internally, and the section on schema.casestudies.  
The documentation on the Cloud Bigtable website, Designing Your Schema, is pertinent and nicely done and lessons learned there equally apply here in HBase land; just divide any quoted values by ~10 to get what works for HBase: e.g. where it says individual values can be ~10MBs in size, HBase can do similar   perhaps best to go smaller if you can   and where it says a maximum of 100 column families in Cloud Bigtable, think ~10 when modeling on HBase.  
  
**6)Can you please provide an example of "good de-normalization" in HBase and how its held consistent (in your friends example in a relational db, there would be a cascadingDelete)? As I think of the users table: if I delete an user with the userid='123', do I have to walk through all of the other users column-family "friends" to guaranty consistency?! Is de-normalization in HBase only used to avoid joins? Our webapp doesn't use joins at the moment anyway.**   
**Answer:** You lose any concept of foreign keys. You have a primary key, that's it. No secondary keys/indexes, no foreign keys.  
It's the responsibility of your application to handle something like deleting a friend and cascading to the friendships. Again, typical small web apps are far simpler to write using SQL, you become responsible for some of the things that were once handled for you.  
Another example of "good denormalization" would be something like storing a users "favorite pages". If we want to query this data in two ways: for a given user, all of his favorites. Or, for a given favorite, all of the users who have it as a favorite. Relational database would probably have tables for users, favorites, and userfavorites. Each link would be stored in one row in the userfavorites table. We would have indexes on both 'userid' and 'favoriteid' and could thus query it in both ways described above. In HBase we'd probably put a column in both the users table and the favorites table, there would be no link table.  
That would be a very efficient query in both architectures, with relational performing better much better with small datasets but less so with a large dataset.  
Now asking for the favorites of these 10 users. That starts to get tricky in HBase and will undoubtedly suffer worse from random reading. The flexibility of SQL allows us to just ask the database for the answer to that question. In a small dataset it will come up with a decent solution, and return the results to you in a matter of milliseconds. Now let's make that userfavorites table a few billion rows, and the number of users you're asking for a couple thousand. The query planner will come up with something but things will fall down and it will end up taking forever. The worst problem will be in the index bloat. Insertions to this link table will start to take a very long time. HBase will perform virtually the same as it did on the small table, if not better because of superior region distribution.  
  
**7)How would you design an Hbase table for many-to-many association between two entities, for example Student and Course?  
I would define two tables:  
Student: student id student data (name, address, ...) courses (use course ids as column qualifiers here) Course: course id course data (name, syllabus, ...) students (use student ids as column qualifiers here)  
Does it make sense?**  
**Answer:** Your design does make sense.  
As you said, you'd probably have two column-families in each of the Student and Course tables. One for the data, another with a column per student or course. For example, a student row might look like: Student : id/row/key = 1001 data:name = Student Name data:address = 123 ABC St courses:2001 = (If you need more information about this association, for example, if they are on the waiting list) courses:2002 = .  
This schema gives you fast access to the queries, show all classes for a student (student table, courses family), or all students for a class (courses table, students family).  
  
**8)What is the maximum recommended cell size?**   
**Answer:** A rough rule of thumb, with little empirical validation, is to keep the data in HDFS and store pointers to the data in HBase if you expect the cell size to be consistently above 10 MB. If you do expect large cell values and you still plan to use HBase for the storage of cell contents, you'll want to increase the block size and the maximum region size for the table to keep the index size reasonable and the split frequency acceptable.   
  
**9)Why can't I iterate through the rows of a table in reverse order?**   
**Answer:** Because of the way HFile works: for efficiency, column values are put on disk with the length of the value written first and then the bytes of the actual value written second. To navigate through these values in reverse order, these length values would need to be stored twice (at the end as well) or in a side file. A robust secondary index implementation is the likely solution here to ensure the primary use case remains fast.   
  
**10)Can I fix OutOfMemoryExceptions in hbase?**   
**Answer:** Out-of-the-box, hbase uses a default of 1G heap size. Set the HBASE\_HEAPSIZE environment variable in ${HBASE\_HOME}/conf/hbase-env.sh if your install needs to run with a larger heap. HBASE\_HEAPSIZE is like HADOOP\_HEAPSIZE in that its value is the desired heap size in MB. The surrounding '-Xmx' and 'm' needed to make up the maximum heap size java option are added by the hbase start script (See how HBASE\_HEAPSIZE is used in the ${HBASE\_HOME}/bin/hbase script for clarification). 

**11)How do I enable hbase DEBUG-level logging?**   
**Answer:** Either add the following line to your log4j.properties file log4j.logger.org.apache.hadoop.hbase=DEBUG and restart your cluster or, if running a post-0.15.x version, you can set DEBUG via the UI by clicking on the 'Log Level' link (but you need set 'org.apache.hadoop.hbase' to DEBUG without the 'log4j.logger' prefix).   
  
**12)What ports does HBase use?**   
**Answer:** Not counting the ports used by hadoop - hdfs and mapreduce - by default, hbase runs the master and its informational http server at 60000 and 60010 respectively and regionservers at 60020 and their informational http server at 60030. ${HBASE\_HOME}/conf/hbase-default.xml lists the default values of all ports used. Also check ${HBASE\_HOME}/conf/hbase-site.xml for site-specific overrides.   
  
**13)Why is HBase ignoring HDFS client configuration such as dfs.replication?**   
**Answer:** If you have made HDFS client configuration on your hadoop cluster, HBase will not see this configuration unless you do one of the following:  
Add a pointer to your HADOOP\_CONF\_DIR to CLASSPATH in hbase-env.sh or symlink your hadoop-site.xml from the hbase conf directory.  
Add a copy of hadoop-site.xml to ${HBASE\_HOME}/conf, or  
If only a small set of HDFS client configurations, add them to hbase-site.xml  
The first option is the better of the three since it avoids duplication.  
  
**14)Can I safely move the master from node A to node B?**   
**Answer:** Yes. HBase must be shutdown. Edit your hbase-site.xml configuration across the cluster setting hbase.master to point at the new location.   
  
**15)Can I safely move the hbase rootdir in hdfs?**   
**Answer:** Yes. HBase must be down for the move. After the move, update the hbase-site.xml across the cluster and restart.   
  
**16)How do I add/remove a node?**   
**Answer:** For removing nodes, see the section on decommissioning nodes in the HBase  
Adding and removing nodes works the same way in HBase and Hadoop. To add a new node, do the following steps:  
Edit $HBASE\_HOME/conf/regionservers on the Master node and add the new address.  
Setup the new node with needed software, permissions.  
On that node run $HBASE\_HOME/bin/hbase-daemon.sh start regionserver  
Confirm it worked by looking at the Master's web UI or in that region server's log.  
Removing a node is as easy, first issue "stop" instead of start then remove the address from the regionservers file.  
For Hadoop, use the same kind of script (starts with hadoop-\*), their process names (datanode, tasktracker), and edit the slaves file. Removing datanodes is tricky, please review the dfsadmin command before doing it.  
  
**17)Why do servers have start codes?**   
**Answer:** If a region server crashes and recovers, it cannot be given work until its lease times out. If the lease is identified only by an IP address and port number, then that server can't do any progress until the lease times out. A start code is added so that the restarted server can begin doing work immediately upon recovery   
  
**18)How do I monitor my HBase Cluster?**  
**Answer:** HBase emits performance metrics that you can monitor with Ganglia. Alternatively, you could use SPM for HBase   
  
**19)In which file the default configuration of HBase is stored.**   
**Answer:** hbase-site.xml   
  
**20)What is the RowKey.**   
**Answer:** Every row in an HBase table has a unique identifier called its rowkey (Which is equivalent to Primary key in RDBMS, which would be distinct throughout the table). Every interaction you are going to do in database will start with the RowKey only   
  
**21)Please specify the command (Java API Class) which you will be using to interact with HBase table.**   
**Answer:** Get, Put, Delete, Scan, and Increment   
  
**22)Which data type is used to store the data in HBase table column.**   
**Answer:** Byte Array,  
Put p = new Put(Bytes.toBytes("John Smith"));  
All the data in the HBase is stored as raw byte Array (10101010). Now the put instance is created which can be inserted in the HBase users table. © HadoopExam Leaning Resource  
  
**23)To locate the HBase data cell which three co-ordinate is used ?**   
**Answer:** HBase uses the coordinates to locate a piece of data within a table. The RowKey is the first coordinate. Following three co-ordinates define the location of the cell.  
1.RowKey  
2.Column Family (Group of columns)  
3.Column Qualifier (Name of the columns or column itself e.g. Name, Email, Address) © HadoopExam Leaning Resource  
Co-ordinates for the John Smith Name Cell.  
["John Smith userID", “info”, “name”]

**24)When you persist the data in HBase Row, In which tow places HBase writes the data to make sure the durability.**   
**Answer:** HBase receives the command and persists the change, or throws an exception if the write fails.  
When a write is made, by default, it goes into two places:  
a. the write-ahead log (WAL), also referred to as the HLog  
b. and the MemStore  
The default behavior of HBase recording the write in both places is in order to maintain data durability. Only after the change is written to and confirmed in both places is the write considered complete. © HadoopExam Leaning Resource  
  
**25)What is MemStore?**   
**Answer:** 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)   
  
**26)What is HFile ?**   
**Answer:** 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. © HadoopExam.com Leaning Resource  
  
**27)How HBase Handles the write failure?**   
**Answer:** Failures are common in large distributed systems, and HBase is no exception.  
Imagine that the server hosting a MemStore that has not yet been flushed crashes. You’ll lose the data that was in memory but not yet persisted. HBase safeguards against that by writing to the WAL before the write completes. Every server that’s part of the.  
HBase cluster keeps a WAL to record changes as they happen. The WAL is a file on the underlying file system. A write isn’t considered successful until the new WAL entry is successfully written. This guarantee makes HBase as durable as the file system backing it. Most of the time, HBase is backed by the Hadoop Distributed Filesystem (HDFS). If HBase goes down, the data that was not yet flushed from the MemStore to the HFile can be recovered by replaying the WAL  
  
**28)Which of the API command you will use to read data from HBase.**   
**Answer:** Get   
Get g = new Get(Bytes.toBytes("John Smith"));  
Result r = usersTable.get(g);  
**29)What is the BlockCache?**   
**Answer:** HBase also use the cache where it keeps the most used data in JVM Heap, along side Memstore.The BlockCache is designed to keep frequently accessed data from the HFiles in memory so as to avoid disk reads. Each column family has its own BlockCache  
The Block in BlockCache is the unit of data that HBase reads from disk in a single pass. The HFile is physically laid out as a sequence of blocks plus an index over those blocks.This means reading a block from HBase requires only looking up that blocks location in the index and retrieving it from disk. The block is the smallest indexed unit of data and is the smallest unit of data that can be read from disk.  
  
**30)BlockSize is configured on which level?**   
**Answer:** The block size is configured per column family, and the default value is 64 KB. You may want to tweak this value larger or smaller depending on your use case.   
  
**31)If your requirement is to read the data randomly from HBase User table. Then what would be your preference to keep blcok size.**   
**Answer:** Having smaller blocks creates a larger index and thereby consumes more memory. If you frequently perform sequential scans, reading many blocks at a time, you can afford a larger block size. This allows you to save on memory because larger blocks mean fewer index entries and thus a smaller index.   
  
**32)What is a block, in a BlockCache ?**   
**Answer:** The Block in BlockCache is the unit of data that HBase reads from disk in a single pass. The HFile is physically laid out as a sequence of blocks plus an index over those blocks.  
This means reading a block from HBase requires only looking up that blocks location in the index and retrieving it from disk. The block is the smallest indexed unit of data and is the smallest unit of data that can be read from disk.  
The block size is configured per column family, and the default value is 64 KB. You may want to tweak this value larger or smaller depending on your use case.  
  
**33)While reading the data from HBase, from which three places data will be reconciled before returning the value ?**   
**Answer:** A. Reading a row from HBase requires first checking the MemStore for any pending modifications.  
B. Then the BlockCache is examined to see if the block containing this row has been recently accessed.  
C. Finally, the relevant HFiles on disk are accessed.  
D. Note that HFiles contain a snapshot of the MemStore at the point when it was flushed. Data for a complete row can be stored across multiple HFiles.  
E. In order to read a complete row, HBase must read across all HFiles that might contain information for that row in order to compose the complete record.  
**34)Once you delete the data in HBase, when exactly they are physically removed?**   
**Answer:** During Major compaction, Because HFiles are immutable, it’s not until a major compaction runs that these tombstone records are reconciled and space is truly recovered from deleted records.   
  
**35)Please describe minor compaction**  
**Answer:** Minor : A minor compaction folds HFiles together, creating a larger HFile from multiple smaller HFiles.   
  
**36)Please describe major compactation?**   
**Answer:** When a compaction operates over all HFiles in a column family in a given region, it’s called a major compaction. Upon completion of a major compaction, all HFiles in the column family are merged into a single file   
  
**37)What is tombstone record?**   
**Answer:** The Delete command doesn’t delete the value immediately. Instead, it marks the record for deletion. That is, a new tombstone record is written for that value, marking it as deleted. The tombstone is used to indicate that the deleted value should no longer be included in Get or Scan results.   
  
**38)Can major compaction manually triggered?**   
**Answer:** Major compactions can also be triggered (or a particular region) manually from the shell. This is a relatively expensive operation and isn’t done often. Minor compactions, on the other hand, are relatively lightweight and happen more frequently.   
  
**39)Which process or component is responsible for managing HBase RegionServer?**   
**Answer:** HMaster is the implementation of the Master Server.The Master server is responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes. In a distributed cluster, the Master typically runs on the NameNode.   
  
**40)Which component is responsible for managing and monitoring of Regions?**   
**Answer:** HRegionServer is the RegionServer implementation. It is responsible for serving and managing regions. In a distributed cluster, a RegionServer runs on a DataNode.   
  
**41)What is the use of HColumnDescriptor?**  
**Answer:** An HColumnDescriptor contains information about a column family such as the number of versions, compression settings, etc. It is used as input when creating a table or adding a column. It is used as input when creating a table or adding a column. Once set, the parameters that specify a column cannot be changed without deleting the column and recreating it. If there is data stored in the column, it will be deleted when the column is deleted.   
  
**42)What is Field swap/promotion?**   
**Answer:** You can move the timestamp field of the row key or prefix it with another field. This approach uses the composite row key concept to move the sequential, monotonously increasing timestamp to a secondary position in the row key. If you already have a row key with more than one field, you can swap them. If you have only the timestamp as the current row key, you need to promote another field from the column keys, or even the value, into the row key. There is also a drawback to moving the time to the right-hand side in the composite key: you can only access data, especially time ranges, for a given swapped or promoted field.   
  
**43)Please tell us Operational command in Hbase, we you have used?**   
**Answer:** There are five main command in HBase.  
1. Get 2. Put 3. Delete 4. Scan 5. Increment  
  
**44)Write down the Java Code snippet to open a connection in Hbase?**   
**Answer:** If you are going to open connection with the help of Java API.  
The following code provide the connection  
Configuration myConf = HBaseConfiguration.create();  
HTableInterface usersTable = new HTable(myConf, "users");  
  
**45)Explain what is the row key?**   
**Answer:** Row key is defined by the application. As the combined key is pre-fixed by the rowkey, it enables the application to define the desired sort order. It also allows logical grouping of cells and make sure that all cells with the same rowkey are co-located on the same server.   
  
**46)What is the Deferred Log Flush in HBase?**   
**Answer:** will be written immediately. If deferred log flush is used, WAL edits are kept in memory until the flush period. The benefit is aggregated and asynchronous HLog- writes, but the potential downside is that if the RegionServer goes down the yet-to-be-flushed edits are lost. This is safer, however, than not using WAL at all with Puts.  
Deferred log flush can be configured on tables via HTableDescriptor. The default value of hbase.regionserver.optionallogflushinterval is 1000ms.  
  
**47)Can you describe the HBase Client: AutoFlush ?**   
**Answer:** When performing a lot of Puts, make sure that setAutoFlush is set to false on your HTable instance. Otherwise, the Puts will be sent one at a time to the RegionServer. Puts added via htable.add(Put) and htable.add(List Put) wind up in the same write buffer. If autoFlush = false, these messages are not sent until the write-buffer is filled. To explicitly flush the messages, call flushCommits. Calling close on the HTable instance will invoke flushCommits. 