HBASE INTERVIEW QUESTIONS

By HadoopExam Learning Resources

Q1. Can you create HBase table without assigning column family.
Ans : 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.

Q2. In which file the default configuration of HBase is stored.
Ans : hbase-site.xml

Q3. What is the RowKey.
Ans: 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.*

Q4. Please specify the command (Java API Class) which you will be using to interact with HBase table.
Ans: `Get`, `Put`, `Delete`, `Scan`, and `Increment`

Q5. Which data type is used to store the data in HBase table column?
Ans: 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.

Q6. To locate the HBase data cell which three co-ordinate is used?
Ans: 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)

Co-ordinates for the John Smith Name Cell.
["John Smith userID", "info", "name"]

Q7. When you persist the data in HBase Row, in which two places HBase writes the data to make sure the durability.
Ans : 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

Q8. What is MemStore ?

Ans : 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)

Q9. What is HFile ?
Ans : 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.

Q10. How HBase handles the write failure.
Ans: 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.

Q11. Which of the API command you will use to read data from HBase.
Ans : Get
exmaple
Get g = new Get(Bytes.toBytes("John Smith"));
Result r = usersTable.get(g);

Q12. What is the BlcokCache?

Ans : HBase also use the cache where it keeps the most used data in JVM Heap, along side Memstore. d. 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. f. This means reading a block from HBase requires only looking up that block's 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.

Q13. BlcokSize is configured on which level ?

Ans : 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.

Q14. If your requirement is to read the data randomly from HBase User table. Then what would be your preference to keep block size.

Ans : Smaller, i. 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.

Q15. What is a block, in a BlockCache ?

Ans : The "Bock" 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 block's 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.

Q16. While reading the data from HBase, from which three places data will be reconciled before returning the value?
Ans:
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.

Q17. Once you delete the data in HBase, when exactly they are physically removed?
Ans : During Major compaction, b. 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.

Q18. Please describe minor compaction
Ans : Minor : A minor compaction folds HFiles together, creating a larger HFile from multiple smaller HFiles.

Q19. Please describe major compaction?
Ans : 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

Q20. What is tombstone record?
Ans : 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.

Q21. Can major compaction manually triggered?
Ans : 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.

Q22. Can you explain data versioning?
Ans : In addition to being a schema-less database, HBase is also versioned.
Every time you perform an operation on a cell, HBase implicitly stores a new version. Creating, modifying, and deleting a cell are all treated identically; they're all new versions. When a cell exceeds the maximum number of versions, the extra records are dropped during the next major compaction. Instead of deleting an entire cell, you can operate on a specific version or versions within that cell. Values within a cell are versioned. Versions are identified by their timestamp, a long. When a version isn't specified, the current timestamp is used as the basis for the operation. The number of cell value versions retained by HBase is configured via the column family. The default number of cell versions is three.

Q23: Which process or component is responsible for managing HBase RegionServer?
Ans : `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.

Q24. Which component is responsible for managing and monitoring of Regions?
Ans : `HRegionServer` is the RegionServer implementation. It is responsible for serving and managing regions. In a distributed cluster, a RegionServer runs on a DataNode.

Q25. Why Would You Need HBase?
Ans : Use HBase when you need fault-tolerant, random, real time read/write access to data stored in HDFS. Use HBase when you need strong data consistency. HBase provides Bigtable-like capabilities on top of Hadoop. HBase's goal is the hosting of very large tables — billions of rows times millions of columns — atop clusters of commodity hardware. HBase manages structured data on top of HDFS for you, efficiently using the underlying replicated storage as backing store to gain the benefits of its fault tolerance and data availability and locality.

Q26. When Would You Not Want To Use HBase?
Ans: A. When your data access patterns are largely sequential over immutable data. Use plain MapReduce
B. When your data is not large
C. When the large overheads of the extract-transform-load (ETL) of your data into alternatives such as Hive is not an issue because you are purely operating on the data in a batching manner and can afford to wait, and some feature of the alternative is simply a must-have.
D. If you need to make a different trade off between consistency and availability. HBase is a strongly consistent system. HBase regions can be temporarily unavailable during fault recovery.
E. If you just can't live without SQL.
F. When you really do require normalized schemas or a relational query engine.

Q27. What is the use of "HColumnDescriptor " ?
Ans : 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.

Q28.In HBase what is the problem with "Time Series Data" and can you explain the Hotspot ?
Ans : When dealing with stream processing of events, the most common use case is time series data. Such data could be coming from a sensor in a power grid, a stock exchange, or a monitoring system for computer systems. Its salient feature is that its row key represents the event time. This imposes a problem with the way HBase is arranging its rows: they are all stored sorted in a distinct range, namely regions with specific start and stop keys. The sequential, monotonously increasing nature of time series data causes all incoming data to be written to the same region. And since this region is hosted by a single server, all the updates will only tax this one machine. This can cause regions to really run hot with the number of accesses, and in the process slow down the perceived overall performance of the cluster,

because inserting data is now bound to the performance of a single machine.

Q29: What is salting and How it helps the "TimeSeries HotSpot" problem?

Answer : It is easy to overcome this problem by ensuring that data is spread over all region servers instead. This can be done, for example, by prefixing the row key with a nonsequential prefix. Common choices include: *Salting:* You can use a *salting* prefix to the key that guarantees a spread of all rows across all region servers. For example:

```
byte prefix = (byte) (Long.hashCode(timestamp) % <number of region servers>);
byte[] rowkey = Bytes.add(Bytes.toBytes(prefix), Bytes.toBytes(timestamp);
```

This formula will generate enough *prefix* numbers to ensure that rows are sent to all region servers. Of course, the formula assumes a specific number of servers, and if you are planning to grow your cluster you should set this number to a multiple instead. The generated row keys might look like this:

0myrowkey-1, 1myrowkey-2, 2myrowkey-3, 0myrowkey-4, 1myrowkey-5, 2myrowkey-6, ...

When these keys are sorted and sent to the various regions the order would be:

```
0myrowkey-1
0myrowkey-4
1myrowkey-2
1myrowkey-5
...
```

In other words, the updates for row keys 0myrowkey-1 and 0myrowkey-4 would be sent to one region (assuming they do not overlap two regions, in which case there would be an even broader spread), and1myrowkey-2 and 1myrowkey-5 are sent to another.

The drawback of this approach is that access to a range of rows must be *fanned out* in your own code and read with <number of region servers> get or scan calls. On the upside, you could use multiple threads to read this data from distinct servers, therefore parallelizing read access. This is akin to a small *map-only*MapReduce job, and should result in increased I/O performance.

Q30: 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.

Q31: How "*Randomization" helps in Time series data ?*

Answer: A totally different approach is to randomize the row key using, for example:

```
byte[] rowkey = MD5(timestamp)
```

Using a hash function like MD5 will give you a random distribution of the key across all available region servers. For time series data, this approach is obviously less than ideal, since there is no way to scan entire ranges of consecutive timestamps. On the other hand, since you can re-create the row key by hashing the timestamp requested, it still is very suitable for random lookups of single rows. When your data is not scanned in ranges but accessed randomly, you can use this strategy.Summarizing the various approaches, you can see that it is not trivial to find the right balance between optimizing for read and write performance. It depends on your access pattern, which ultimately drives the decision on how to structure your row keys.

Q32: List the main component of HBase?

Answer: Zookeeper, Catalog Tables , Master , RegionServer , Region

Q33. 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

Q34. 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");

Q35. Please let us know the Difference between HBase and Hadoop/HDFS?
Answer: HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS is a distributed file system that is well suited for the storage of large files. Its documentation states that it is not, however, a general purpose file system, and does not provide fast individual record lookups in files. HBase, on the other hand, is built on top of HDFS and provides fast record lookups (and updates) for large tables. This can sometimes be a point of conceptual confusion. HBase internally puts your data in indexed StoreFiles that exist on HDFS for high-speed lookups. Assumptions and Goals of HDFS
1. Hardware Failure
2. Streaming Data Access
3. Large Data Sets
4. Simple Coherency Model
5. Moving Computation is Cheaper than Moving Data
6. Portability Across Heterogeneous Hardware and Software Platforms

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications.

Q36. 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.

Q37. What happens if we change the block size of a column family on an already populated database?
When we change the block size of the column family, the new data takes the new block size while the old data is within the old block size. When the compaction occurs, old data will take the new block size. "New files, as they are flushed, will have the new block size, whereas existing data will continue to be read correctly. After the next major compaction, all data should be converted to the new block size."

Q38. **What is the difference between HBASE and RDBMS?**

HBASE	**RDBMS**
It is distributed, column oriented, versioned data storage system.	It is designed to follow FIXED schema. It is row-oriented databases and doesn't natively scale to distributed storage.
HDFS is underlying layer of HBase and provides fault tolerance and linear scalability. It doesn't support secondary indexes and support data in key-value pair.	It supports secondary indexes and improvises data retrieval through SQL language.
It supports dynamic addition of column in table	It has slow learning curve and support complex joins

schema. It is not relational database like RDBMS.	and aggregate functions.
HBASE helps Hadoop overcome the challenges in random read and write.	

Q39. Explain what is WAL and Hlog in Hbase?
Answer: WAL (Write Ahead Log) is similar to MySQL BIN log; it records all the changes occur in data. It is a standard sequence file by Hadoop and it stores HLogkey's. These keys consist of a sequential number as well as actual data and are used to replay not yet persisted data after a server crash. So, in cash of server failure WAL work as a life-line and retrieves the lost data's.

Q40. In Hbase what is column families?
Answer: Column families comprise the basic unit of physical storage in Hbase to which features like compressions are applied.

Q41. 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.

Q42. Explain deletion in Hbase? Mention what are the three types of tombstone markers in Hbase?
Answer: When you delete the cell in Hbase, the data is not actually deleted but a tombstone marker is set, making the deleted cells invisible. Hbase deleted are actually removed during compactions. Three types of tombstone markers are there:

- Version delete marker: For deletion, it marks a single version of a column
- Column delete marker: For deletion, it marks all the versions of a column
- Family delete marker: For deletion, it marks of all column for a column family

Q43. Explain how does Hbase actually delete a row?
Answer: In Hbase, whatever you write will be stored from RAM to disk, these disk writes are immutable barring compaction. During deletion process in Hbase, major compaction process delete marker while minor compactions don't. In normal deletes, it results in a delete tombstone marker- these delete data they represent are removed during compaction.
Also, if you delete data and add more data, but with an earlier timestamp than the tombstone timestamp, further Gets may be masked by the delete/tombstone marker and hence you will not receive the inserted value until after the major compaction.

Q44. Explain what happens if you alter the block size of a column family on an already occupied database?
Answer: When you alter the block size of the column family, the new data occupies the new block size while the old data remains within the old block size. During data compaction, old data will take the new block size. New files as they are flushed, have a new block size whereas existing data will continue to be read correctly. All data should be transformed to the new block size, after the next major compaction.

Q45. What is Bloom filter and how it helps ?
Answer: HBase supports Bloom Filter to improve the overall throughput of the cluster. A HBase Bloom Filter is a space-efficient mechanism to test whether a StoreFile contains a specific row or row-col cell. Without Bloom Filter, the only way to decide if a row key is contained in a StoreFile is to check the StoreFile's block

index, which stores the start row key of each block in the StoreFile. It is very likely that the row key we are finding will drop in between two block start keys; if it does then HBase has to load the block and scan from the block's start key to figure out if that row key actually exists.

Q46. How scan caching helps in HBase ?
Answer: If HBase is used as an input source for a MapReduce job, for example, make sure that the input Scan instance to the MapReduce job has setCaching set to something greater than the default (which is 1). Using the default value means that the map-task will make call back to the region-server for every record processed. Setting this value to 500, for example, will transfer 500 rows at a time to the client to be processed. There is a cost/benefit to have the cache value be large because it costs more in memory for both client and RegionServer, so bigger isn't always better.

Q47. What is the impact of Scan Caching in MapReduce Jobs?
Answer: Scan settings in MapReduce jobs deserve special attention. Timeouts can result (e.g., UnknownScannerException) in Map tasks if it takes longer to process a batch of records before the client goes back to the RegionServer for the next set of data. This problem can occur because there is non-trivial processing occuring per row. If you process rows quickly, set caching higher. If you process rows more slowly (e.g., lots of transformations per row, writes), then set caching lower.
Timeouts can also happen in a non-MapReduce use case (i.e., single threaded HBase client doing a Scan), but the processing that is often performed in MapReduce jobs tends to exacerbate this issue.

Q48. What is the Impact of "Turn off WAL on Puts"?
Answer: A frequently discussed option for increasing throughput on Puts is to call writeToWAL(false). Turning this off means that the RegionServer will not write the Put to the Write Ahead Log, only into the memstore, HOWEVER the consequence is that if there is a RegionServer failure there will be data loss. If writeToWAL(false) is used, do so with extreme caution. You may find in actuality that it makes little difference if your load is well distributed across the cluster. In general, it is best to use WAL for Puts, and where loading throughput is a concern to use bulk loading techniques instead.

Q49. Why to "Pre-Creating Regions"?
Answer: Tables in HBase are initially created with one region by default. For bulk imports, this means that all clients will write to the same region until it is large enough to split and become distributed across the cluster. A useful pattern to speed up the bulk import process is to pre-create empty regions. Be somewhat conservative in this, because too-many regions can actually degrade performance.

There are two different approaches to pre-creating splits.

The first approach is to rely on the default HBaseAdmin strategy (which is implemented in Bytes.split)..

```
byte[] startKey = ...;    // your lowest keuy
byte[] endKey = ...;               // your highest key
int numberOfRegions = ...;         // # of regions to create
admin.createTable(table, startKey, endKey, numberOfRegions);
```

And the other approach is to define the splits yourself...

```
byte[][] splits = ...; // create your own splits admin.createTable(table, splits);
```

Q50. What is the Deferred Log Flush in HBase?
Answer: The default behavior for Puts using the Write Ahead Log (WAL) is that HLog edits 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.

Q51. 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.

hadoop 430 Q & A Click Here Cloudera Hadoop Developer Certification CCD-410	hadoop 200 + Q & A Click Here Cloudera Hadoop Administrator Certification CCA-500	hadoop 262 Q & A Click Here Cloudera Hadoop HBase Certification CCB-400	amazon webservices 266 Q & A Click Here AWS Developer Certification Associate Level
hadoop 235 Q & A Click Here Cloudera Data Science Certification DS-200	hadoop Click Here Hadoop Training With HandsOn	hadoop Click Here Package Deal	amazon webservices 144 Q & A Click Here AWS Certified Solutions Architect Professional Level

Data, BigData. Cloud.
Anaytics & Virtualizaton
HadoopExam.com | QuickTechie.com

amazon web services 300 + Q & A Click Here AWS Certified SysOps Administrator Associate Level	amazon web services 474 + Q & A Click Here AWS Certified Solutions Architect Associate Level	amazon web services Click Here Click Here for AWS Package Deal	sas 490 Q & A Click Here SAS Base Certification A00-211
sas 365 Q & A Click Here SAS Advance Certification A00-212	sas 86+ Q & A Click Here SAS Certified Statistical Business Analyst A00-240	sas 85 Q & A Click Here SAS Certified Platform Administrator 9 A00-250	sas Click Here SAS Packaged Deal

EMC²
234 Q & A
Click Here
EMC Data Scientist Associate Certification
E20-007 (EMCDSA)

Data Science certification really needs a good and in depth knowledge of statistics cum BigData Hadoop knowledge. It also require you to have good knowledge in like the main phases of the Data Analytics Lifecycle, analyzing and exploring data with R, statistics for model building and evaluation, the theory and methods of advanced analytics and statistical modeling, the technology and tools that can be used for advanced analytics, operationalizing an analytics project, and data visualization techniques. Successful candidates will achieve the EMC Proven Professional – Data Science Associate credential. Hence to clear the real exam it realy needs very well preparation. So HadoopExam Learning Resources brings Data Science Certification Simulator with 234 Practice Questions, which can help you to prepare for this exam in lesser time. **Practice - practice - practice**! The EMC:DS E20-007 Exam Simulator offers you the opportunity to take 4 sample Exams before heading out for the real thing. Be ready to succeed on exam day!

Upcoming Releases

1. Apache Spark Training
2. Apache Spark Certification
3. MongoDB Certification Material

Email: mailto:admin@hadoopexam.com mailto:hadoopexam@gmail.com

Phone: 022-42669636 Mobile : +91-8879712614 © HadoopExam Learning Resources