HBase Interview Questions and Answers

1. What is HBase?
HBase is Column-Oriented , Open-Source, Multidimensional, Distributed database. It run on the top of HDFS.
2. Why do we use HBase?
HBase provide random read and write, can perform thousand of operation per second on large data set. HBase support record level record level operations on database tables.
3. What are the main component of HBase?
Zookeeper
Catalog Tables
HMaster
HRegionServer
Regions
4. What are the important catalog tables in HBase?
-ROOT- and .META. are special catalog tables in HBase. Within catalog tables it maintains the current list, state, and location of all regions on the cluster. The -ROOT- table holds the list of .META. table regions. The .META. table holds the list of all user space regions.
5. What are the important Operational commands in HBase?
There are five main commands in HBase for record level operations.
1. get
2. put
3. delete
4. scan
5. increment
For Table level operations:
1. list
2. scan
3. describe
4. disable
5. drop

6. How to open a connection in HBase?
To open connection with the help of Java API. The following code provide the connection
```
Configuration conf = HBaseConfiguration.create();
HTableInterface usersTable = new HTable(conf, "users");
```
7. Does HBase support SQL?
Not really. SQL-ish support for HBase via Hive is implemented, however Hive is based on MapReduce which is not generally suitable for low-latency requests.
8. What is the maximum recommended cell size?
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 we expect the cell size to be consistently above 10 MB. If we do expect large cell values and we still plan to use HBase for the storage of cell contents, we'll need 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 cant I iterate through the rows of a table in reverse order?
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. Is NoSQL follow relational DB model?
No

11. Why would NoSQL be better than using a SQL Database? And how much better is it?
It would be better when our site needs to scale so massively that the best RDBMS running on the best hardware we can afford and optimize as much as possible simply can't keep up with the load. How much better it is depends on the specific use case (lots of update activity combined with lots of joins is very hard on "traditional" RDBMSs) – could well be a factor of 1000 in extreme cases
12. What is the difference between HBase and Hive?
Hive doesn't support record level operations but HBase support record level operations.
13. What is the difference between HBase vs RDBMS?
Hbase is a column oriented database whereas RDBMS is row oriented database, the main difference between these two is Hbase can easily scaled up with respect to storage and processing but whereas RDBMS has scalability but limited in storage capacity. when we are working with billions of rows and millions of columns then we need to choose Hbase as our database. RDBMS provides e.g. typed columns, secondary indexes, transactions, advanced query languages, etc.. but HBase doesn't.
14. what is NoSQL?
NoSQL is a general term meaning that the database isn't an RDBMS which supports SQL as its primary access language, but there are many types of NoSQL databases: BerkeleyDB is an example of a local NoSQL database, whereas HBase is very much a distributed database. Technically speaking, HBase is really more a "Data Store" than "Data Base" because it lacks many of the features you find in an RDBMS, such as typed columns, secondary indexes, triggers, and advanced query languages, etc.
15. How Does HBase supports scalability and why RDBMS can't?
HBase has many features which supports both linear and modular scaling. HBase clusters expand by adding RegionServers that are hosted on commodity class servers. If a cluster expands from 10 to 20 RegionServers, for example, it doubles both in terms of storage and as well as processing capacity. RDBMS can scale well, but only up to a point – specifically, the size of a single database server – and for the best performance requires specialized hardware and storage devices.
16. What are the core features of HBase?

- Strongly consistent reads/writes: HBase is not an "eventually consistent" Data Store. This makes it very suitable for tasks such as high-speed counter aggregation.
- Automatic sharding: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- Automatic Region Server failover
- Hadoop/HDFS Integration: HBase supports HDFS out of the box as its distributed file system.
- MapReduce: HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- Java Client API: HBase supports an easy to use Java API for programmatic access.
- Thrift/REST API: HBase also supports Thrift and REST for non-Java front-ends.
- Block Cache and Bloom Filters: HBase supports a Block Cache and Bloom Filters for high volume query optimization.
- Operational Management: HBase provides build-in web-pages for operational insight as well as JMX metrics.

17. What Is The Difference Between HBase and Hadoop/HDFS?
HDFS : is a distributed file system that is well suited for the storage of large files. It's file system, and does not provide fast individual record look-ups in files. HBase: on the other hand, is built on top of HDFS and provides fast record look-ups and updates for large tables. HBase internally puts your data in indexed "StoreFiles" that exist on HDFS for high-speed look-ups.
18. When Should we Use HBase?
HBase isn't suitable for every problem. First, we need make sure we have enough data. If we have hundreds of millions or billions of rows, then HBase is a good candidate. If we only have a few thousand/million rows, then using a traditional RDBMS might be a better choice due to the fact that all of our data might wind up on a single node (or two) and the rest of the cluster may be sitting idle. Second, we need to make sure we 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, we need to make sure we 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.