**Problem Statement:**

Explain in brief

● When should we use HBASE, list some of the scenarios for the same in real time.

● What are the different modes in which Hbase can be run?

● Need and working of zookeeper in Hbase?

● When should we use HBASE, list some of the scenarios for the same in real time.

**Solution:**

Following are some of the key areas to be considered before finalizing HBase for your application.

**Data volume:** The volume of data is the most common point to be considered. We should have peta bytes of data to be processed in a distributed environment. Otherwise, for a small amount of data, it will be stored and processed in a single node, keeping other nodes idle. So, it will be a misuse of technology framework.

**Application Types:** HBase is not suitable for transactional applications, large volume MapReduce jobs, relational analytics, etc. It is preferred when you have a variable schema with slightly different rows. It is also suitable when we are going for a key dependent access to your stored data.

**Hardware environment:** HBase runs on top of HDFS. And HDFS works efficiently with a large number of nodes (minimum 5). So, if we have good hardware support, then HBase can be a good selection.

**No requirement of relational features:** Our application should not have any requirement for RDBMS features like transaction, triggers, complex query, complex joins etc. If we can build our application without these features, then we can go for HBase.

**Quick access to data:**If we need a random and real time access to your data, then HBase is a suitable candidate. It is also a perfect fit for storing large tables with multi structured data. It gives ‘flashback’ support to queries, which makes it more suitable for fetching data in a particular instance of time.

HBase is also suitable when you need fault tolerant, fast and usable data management in a non-relational environment

● What are the different modes in which Hbase can be run?

**Solution:**

HBase has two run modes:

1. Standalone mode
2. Distributed mode

**Standalone mode:**

-This is the default mode.

-In standalone mode, HBase does not use HDFS .It uses the local filesystem.

- It runs all HBase daemons and a local ZooKeeper all up in the same JVM. Zookeeper binds to a well known port so clients may talk to HBase.

**Distributed mode:**

**-**Distributed mode can be subdivided into:

1) Pseudo-Distributed mode

2) Fully-Distributed mode

**Pseudo-Distributed mode:**

-All daemons run on a single node.

-A pseudo-distributed mode is simply a distributed mode run on a single host.

-This is used for prototyping HBASE .

- This should not be used for production and evaluating the HBASE performance.

**Pseudo-Distributed configuration file:**

<configuration>

...

<property>

<name>hbase.rootdir</name>

<value>hdfs://h-24-30.sfo.stumble.net:8020/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>h-24-30.sfo.stumble.net</value>

</property>

...

</configuration>

**Fully-Distributed mode:**

-Fully-distributed where the daemons are spread across all nodes in the cluster.

-This is the one which is used in the real time and for production of the Hadoop applications.

**Fully-Distributed configuration file:**

<configuration>

...

<property>

<name>hbase.rootdir</name>

<value>hdfs://namenode.example.org:8020/hbase</value>

<description>The directory shared by RegionServers.

</description>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

<description>The mode the cluster will be in. Possible values are

false: standalone and pseudo-distributed setups with managed Zookeeper

true: fully-distributed with unmanaged Zookeeper Quorum (see hbase-env.sh)

</description>

</property>

...

</configuration>

● Need and working of zookeeper in Hbase?

**Solution:**

-ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

-All of these kinds of services are used in some form or another by distributed applications.

-Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.

-Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them , which make them brittle in the presence of change and difficult to manage.

-Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

ZooKeeper will help you with coordination between Hadoop nodes.  
  
 It makes it easier to:

* **Manage configuration across nodes.** If we have dozens or hundreds of nodes, it becomes hard to keep configuration in sync across nodes and quickly make changes. ZooKeeper helps you quickly push configuration changes.
* **Implement reliable messaging.** With ZooKeeper, we can easily implement a producer/consumer queue that guarantees delivery, even if some consumers or even one of the ZooKeeper servers fails.
* **Implement redundant services.** With ZooKeeper, a group of identical nodes (e.g. database servers) can elect a leader/master and let ZooKeeper refer all clients to that master server. If the master fails, ZooKeeper will assign a new leader and notify all clients.
* **Synchronize process execution.** With ZooKeeper, multiple nodes can coordinate the start and end of a process or calculation. This ensures that any follow-up processing is done only after all nodes have finished their calculations.