**Explain in brief**

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

As a NoSQL DB, HBase offers lot of good functionalities, but it is still not a **‘Fit for All’** 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. You 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 you 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 you have good hardware support, then HBase can be a good selection.

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

**Quick access to data:**If you 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.

Apart from the above points, HBase is also suitable when you need fault tolerant, fast and usable data management in a non-relational environment.

**Other real time examples are:**

1. **Google big table**

HBase is developed for  different purpose and solve different set of problems.  
HBase is implementation based on **Google's BigTable paper** and built to access a small set of data from billions of records real quick, It is good for lookup use cases but not good for queries/use cases that needs to access all the data. It also needs a careful key/ schema design based on access patterns and use cases.

1. **Facebook**

Hbase can be used in the same system. **Facebook** has been using Hbase to query/store their huge amount of data. FB used Hbase to implement their messaging platform

1. **Amazon**   
   Hbase**, Amazon S3** or even plain text. The query is converted to MapReduce job internally. Hbase is more similar to the Database ( open source, non-relational, distributed). Hbase works in different layer of a Data framework. The right comparable for Hbase are HDFS, Cassandra, RDBMS etc.

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

HBase has two run modes: standalone and distributed. Out of the box, HBase runs in standalone mode. Whatever your mode, you will need to configure HBase by editing files in the HBase conf directory.

1. **Standalone HBase**

This is the default mode. Standalone mode is Quick Start - Standalone Hbase section. In standalone mode, HBase does not use HDFS -- it uses the local filesystem instead and 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.

1. **Distributed**

Distributed mode can be subdivided into distributed but all daemons run on a single node i.e. *Pseudo-distributed* and *fully-distributed* where the daemons are spread across all nodes in the cluster.

**Pseudo-distributed mode** can run against the local file system or it can run against an instance of the *Hadoop Distributed File System* (HDFS). Fully-distributed mode can ONLY run on HDFS. See the Hadoop [requirements and instructions](http://hadoop.apache.org/common/docs/r1.1.1/api/overview-summary.html#overview_description) for how to set up HDFS for Hadoop 1.x. A good walk-through for setting up HDFS on Hadoop 2.

**Fully-distributed**

In distributed mode, multiple instances of HBase daemons run on multiple servers in the cluster.

Just as in pseudo-distributed mode, a fully distributed configuration requires that you set the hbase-cluster. distributed property to true. Typically, the hbase.rootdir is configured to point to a highly-available HDFS filesystem.

In addition, the cluster is configured so that multiple cluster nodes enlist as RegionServers, ZooKeeper QuorumPeers, and backup HMaster servers.

**● Need and working of zookeeper in Hbase?**

**Need:**

ZooKeeper is a distributed, open-source coordination service for distributed applications. It exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, and groups and naming. It is designed to be easy to program to, and uses a data model styled after the familiar directory tree structure of file systems. It runs in Java and has bindings for both Java and C.

**Working:**

 Zookeeper as a replicated synchronization service with eventual consistency. It is robust, since the persisted data is distributed between multiple nodes (this set of nodes is called an "ensemble") and one client connects to any of them (i.e., a specific "server"), migrating if one node fails; as long as a strict majority of nodes are working, the ensemble of Zookeeper nodes is alive. In particular, a master node is dynamically chosen by consensus within the ensemble; if the master node fails, the role of master migrates to another node.

### How writes are handled

The master is the authority for writes: in this way writes can be guaranteed to be persisted in-order, i.e., writes are **linear**. Each time a client writes to the ensemble, a majority of nodes persist the information: these nodes include the server for the client, and obviously the master. This means that each write makes the server up-to-date with the master. It also means, however, that you cannot have concurrent writes.

The guarantee of linear writes is the reason for the fact that ZooKeeper does not perform well for write-dominant workloads. In particular, it should not be used for interchange of large data, such as media. As long as your communication involves shared data, ZooKeeper helps you. When data could be written concurrently, ZooKeeper actually gets in the way, because it imposes a strict ordering of operations even if not strictly necessary from the perspective of the writers. Its ideal use is for coordination, where messages are exchanged between the clients.

### How reads are handled

This is where ZooKeeper excels: reads are concurrent since they are served by the specific server that the client connects to. However, this is also the reason for the eventual consistency: the "view" of a client may be outdated, since the master updates the corresponding server with a bounded but undefined delay.