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

Apache HBase is a column oriented database which supports dynamic database schema. It mainly runs on top of the HDFS and supports MapReduce jobs. HBase also supports other high level languages for data processing.  
Let us have a look at the different features of HBase:.

**Scalability:** HBase supports scalability in both linear and modular form  
**Sharding**: HBase supports automatic sharding of tables. It is also configurable.  
**Distributed storage:** HBase supports distributed storage like HDFS  
**Consistency:** It supports consistent read and write operations  
**Failover support:** HBase supports automatic failover  
**API support:** HBase supports Java APIs so clients can access it easily  
**MapReduce support**: HBase supports MapReduce for parallel processing of large volume of data  
**Back up support:** HBase supports back up of Hadoop MapReduce jobs in HBase tables  
**Real time processing:** It supports block cache and Bloom filters. So, real time query processing is easy

Apart from the above major features, HBase also supports REST-ful web services, jruby-based shell, Ganglia and JMX. So, HBase has a very strong presence in the NoSQL database world. HBase scales in a linear way, so all the tables should have a primary key. All the key spaces are distributed into sequential blocks and these blocks are allotted to regions. Now, these regions are controlled by RegionServers to distribute the load uniformly in a clustered environment. HBase supports automatic data sharding, so manual intervention is not required. After deploying HBase, Zookeeper and HMaster servers are configured to provide cluster topology information to the HBase clients. Client applications connect to these utilities and get the lists of RegionServers, regions and key ranges information. It helps the client to know the exact data position and connect to the RegionServer directly. RegionServers also provide caching (by using memstore) support for the frequently accessed rows. It helps improve the performance.

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.

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

HBase run modes: Standalone and Distributed

HBase has two run modes: Standalone HBase and [Distributed](http://hbase.apache.org/0.94/book/standalone_dist.html#distributed). Out of the box, HBase runs in standalone mode. To set up a distributed deploy, you will need to configure HBase by editing files in the HBase conf directory.

Whatever your mode, you will need to edit conf/hbase-env.sh to tell HBase which java to use. In this file you set HBase environment variables such as the heapsize and other options for the JVM, the preferred location for log files, etc. Set JAVA\_HOME to point at the root of your java install.

Standalone HBase

This is the default mode. 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.

Distributed

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

Distributed modes require an instance of the *Hadoop Distributed File System* (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 a HDFS. Before proceeding, ensure you have an appropriate, working HDFS.

Pseudo-distributed

A pseudo-distributed mode is simply a distributed mode run on a single host. Use this configuration testing and prototyping on HBase. Do not use this configuration for production nor for evaluating HBase performance.

● Need and working of zookeeper in Hbase?

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.

HBase is a NoSQL datastore that runs on top of your existing Hadoop cluster(HDFS). It provides you capabilities like random, real-time reads/writes, which HDFS being a FS lacks. Since it is a NoSQL datastore it doesn't follow SQL conventions and terminologies. HBase provides a good set of APIs( includes JAVA and Thrift). Along with this HBase also provides seamless integration with MapReduce framework. But, along with all these advantages of HBase you should keep this in mind that random read-write is quick but always has additional overhead. So think well before ye make any decision.

ZooKeeper is a high-performance coordination service for distributed applications(like HBase). It exposes common services like naming, configuration management, synchronization, and group services, in a simple interface so you don't have to write them from scratch. You can use it off-the-shelf to implement consensus, group management, leader election, and presence protocols. And you can build on it for your own, specific needs.

HBase relies completely on Zookeeper. HBase provides you the option to use its built-in Zookeeper which will get started whenever you start HBAse. But it is not good if you are working on a production cluster. In such scenarios it's always good to have a dedicated Zookeeper cluster and integrate it with your HBase cluster.