[**https://www.tutorialspoint.com/hbase/hbase\_overview.htm**](https://www.tutorialspoint.com/hbase/hbase_overview.htm)

**Why HBase?**

1. Hadoop is well suited for batch processing
2. Hadoop always accesses the data sequentially. In other words, irrespective of the task being performed entire data set is always scanned
3. So there was a need of a way when it was required to access the data in a random manner
4. Some of the other example databases which ensures random access of data are HBase, Cassandra, MongoDB etc.
5. RDBMS get exponentially slow as the data becomes large. Expects data to be highly structured, i.e. ability to fit in a well-defined schema. Any change in schema might require a downtime. For sparse datasets, too much of overhead of maintaining NULL values

*“Anybody who wants to keep data within an HDFS environment and wants to do anything other than brute-force reading of the entire file system [with* [*MapReduce*](https://www.dezyre.com/hadoop-course/mapreduce)*] needs to look at HBase. If you need random access, you have to have HBase."- said Gartner analyst Merv Adrian.*

**What is HBase?**

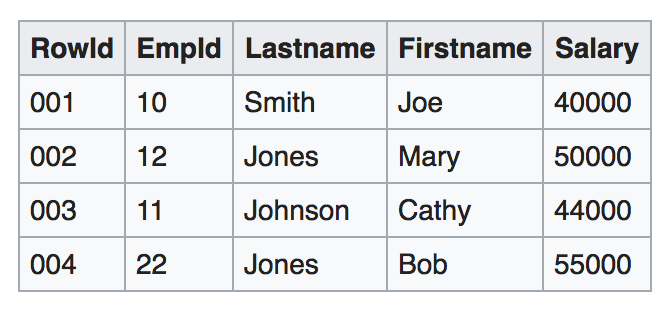
Hbase is a distributed column oriented database built on top of the hadoop file system, hence, it leverages fault tolerant features of HDFS. It provided random real time read/write access to the data present in the HDFS.

1. Hbase is an open source and sorted map data built on Hadoop.
2. horizontally scalable.
3. Hbase is well suited for sparse data sets which are very common in big data use cases

**Features of HBase**

1. Horizontally scalable: You can add any number of columns anytime
2. HBase can be referred to as a data store instead of a database as it misses out on some important features of traditional RDBMs like typed columns, triggers, advanced query languages and secondary indexes.
3. Automatic Failover: Automatic failover is a resource that allows a system administrator to automatically switch data handling to a standby system in the event of system compromise.
4. Integrations with Map/Reduce framework: All the commands and java codes internally implement Map/ Reduce to do the task and it is built over Hadoop Distributed File System.
5. Sparse, distributed, persistent, multidimensional sorted map, which is indexed by **rowkey, column key,and timestamp.**
6. Often referred as a key value store or column family-oriented database, or storing versioned maps of maps.
7. fundamentally, it's a platform for storing and retrieving data with random access. Ensures fast random reads and writes.
8. It doesn't care about datatypes(storing an integer in one row and a string in another for the same column).
9. It doesn't enforce relationships within your data.
10. It is designed to run on a cluster of computers, built using commodity hardware.

**Row oriented db vs Column oriented database:**

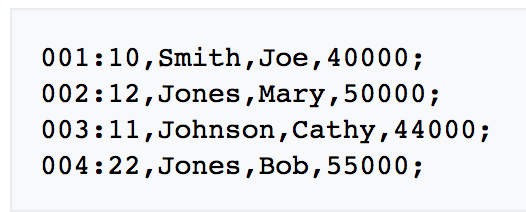
****

[**https://en.wikipedia.org/wiki/Column-oriented\_DBMS**](https://en.wikipedia.org/wiki/Column-oriented_DBMS)

When the data is stored in hard disks, the most expensive operations are **disk seeks.** In order to improve overall performance, related data has to be stored together to minimise the number of disk seeks. Hard disks are divided into blocks which are typically capable of storing multiple rows. So, its required to store related rows in a single block

**Row oriented db:**

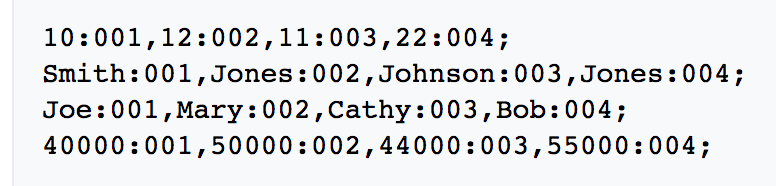
1. In a row oriented table an entire row is serialised and is stored in a block.



1. Row based systems are designed to efficiently return data for an entire row, or record, in as few operations as possible
2. Row based systems are not good at performing set wide operations on the whole table, suppose need to find all the records in a table between sal 40000 to 50000 because the dbms fully scans the tables to fetch the relevant records.
3. So as all the data will not fit in a single block, lot of disk seek are required to scan the entire table which is costly in terms of time
4. In row oriented databases this problem is solved using indexes.
5. Row-oriented databases are well-suited for [OLTP](https://en.wikipedia.org/wiki/OLTP)-like workloads which are more heavily loaded with interactive transactions. For example, retrieving all data from a single row is more efficient when that data is located in a single location (minimising disk seeks), as in row-oriented architectures.

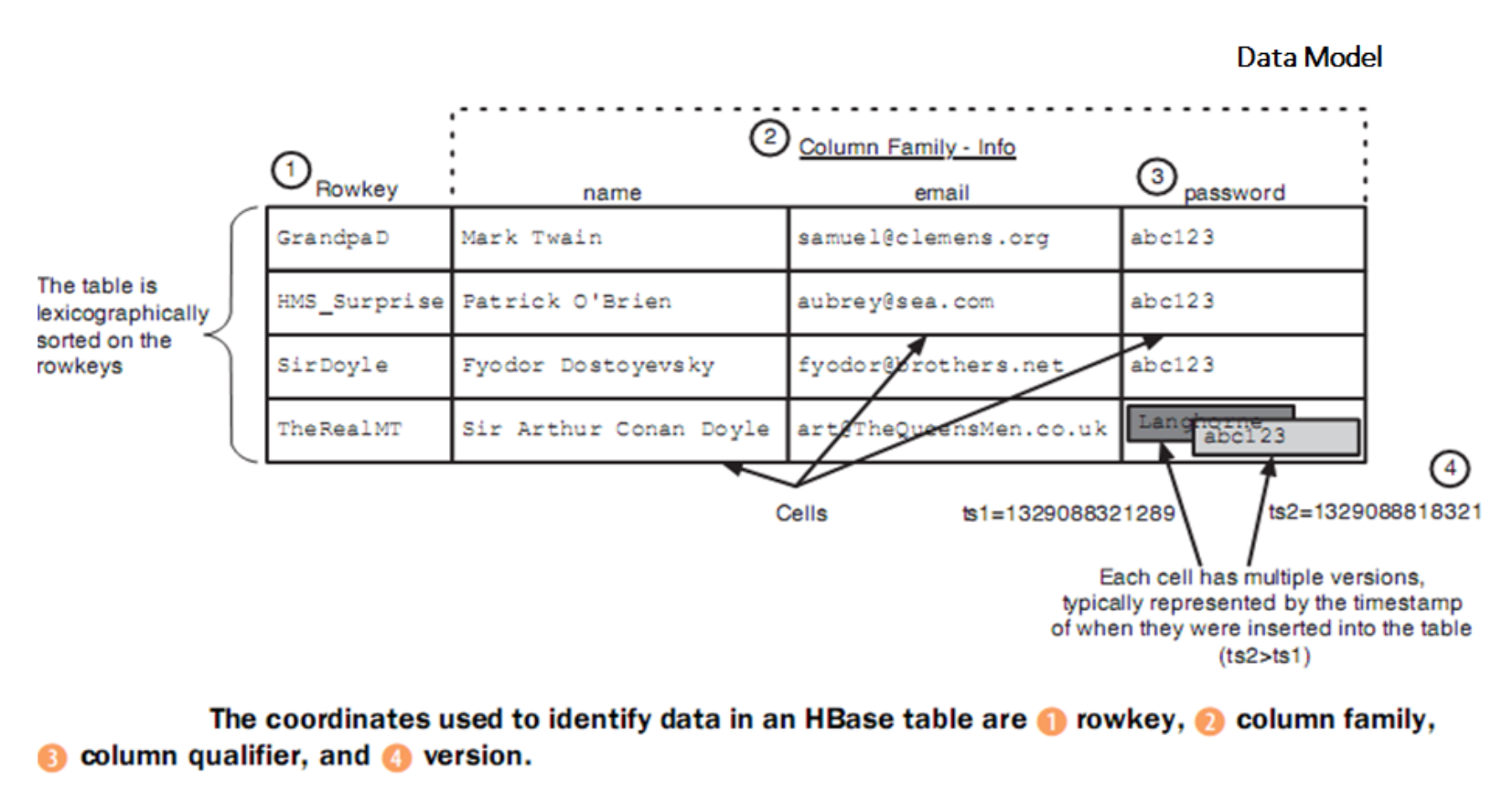
**Column oriented db**

1. Serialised all the values in a single column together



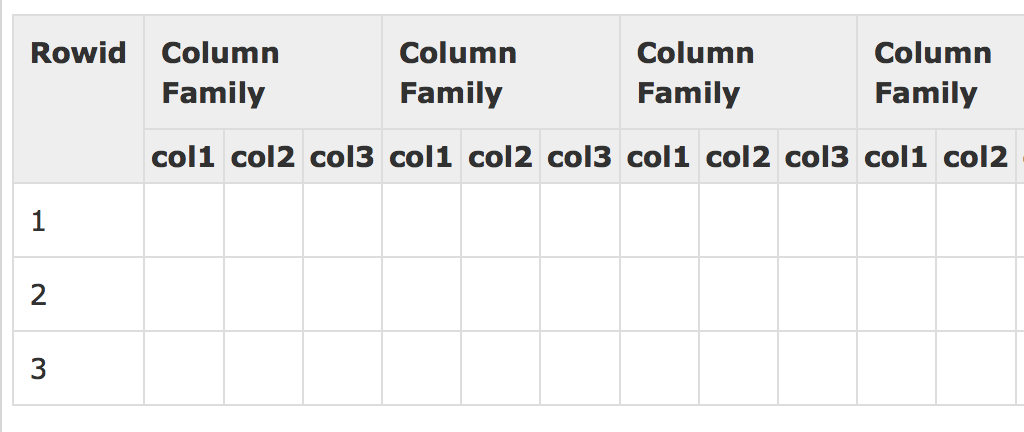
1. In a row-oriented indexed system, the primary key is the rowid that is mapped from indexed data. In the column-oriented system, the primary key is the data, which is mapped from rowids.
2. A row-based system can retrieve the row in a single disk read, whereas numerous disk operations to collect data from multiple columns are required from a columnar database. However, these whole-row operations are generally rare.
3. Columnar databases boost performance by reducing the amount of data that needs to be read from disk, both by efficiently compressing the similar columnar data and by reading only the data necessary to answer the query.
4. Columnar databases are well-suited for [OLAP](https://en.wikipedia.org/wiki/OLAP)-like workloads (e.g., [data warehouses](https://en.wikipedia.org/wiki/Data_warehouse)) which typically involve highly complex queries over all data (possibly petabytes)
5. Column data is of uniform type; therefore, there are some opportunities for storage size optimizations available in column-oriented data that are not available in row-oriented data. For example, many popular modern compression schemes, such as [LZW](https://en.wikipedia.org/wiki/Lempel-Ziv-Welch) or [run-length encoding](https://en.wikipedia.org/wiki/Run-length_encoding), make use of the similarity of adjacent data to compress.

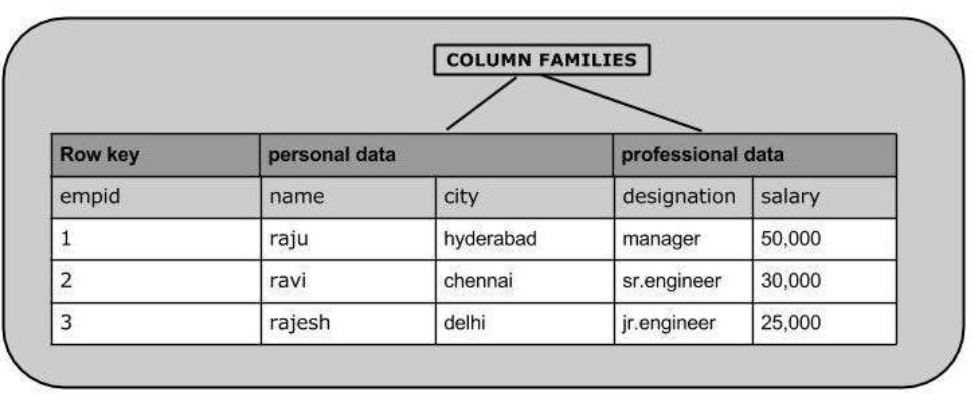
**Hbase data model**

****

[**https://www.javatpoint.com/hbase-data-model**](https://www.javatpoint.com/hbase-data-model)

1. The rows in the Hbase table are sorted by row key
2. The table schema defines only the column families, which are the key value pairs.
3. A table can have multiple column families and each column family can have multiple columns.
4. Column families in HBase are static whereas the columns, by themselves, are dynamic
5. Each cell value in the table has a timestamp
6. In short:
   1. **HBase Tables** are a collection of rows
   2. **HBase Row** is a collection of column families
   3. **RowKey** every entry in an HBase table is identified and indexed by a RowKey.
   4. **Column family** is a collection of column.
   5. **Column** is a collection of key value pairs. For every rowkey an unlimited number of attributes can be stored





**When to use Hbase**

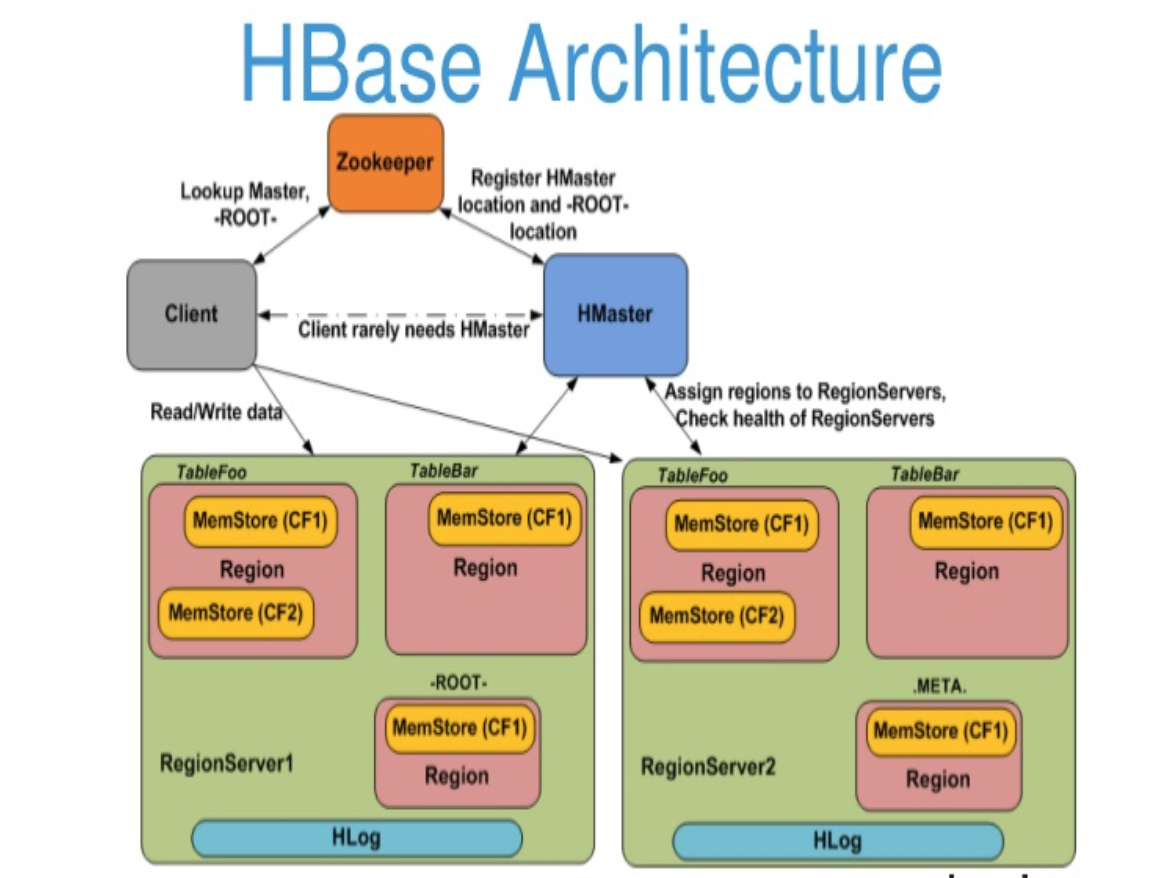
It does not require a fixed schema, so developers have the provision to add new data as and when required without having to conform to a predefined model.

It provides users with database like access to Hadoop-scale storage, so developers can perform read or write on subset of data efficiently, without having to scan through the complete dataset. HBase is the best choice as a NoSQL database, when your application already has a hadoop cluster running with huge amount of data. HBase helps perform fast read/writes.

**HBase Architecture**

1. In Hbase all the tables are split into regions and are served by the region servers
2. Regions are vertically divided by the column family into stores.
3. **Region:** HBase, tables are dynamically distributed by the system whenever they become too large to handle (Auto Sharding). The simplest and foundational unit of horizontal scalability in HBase is a **Region.** A continuous, sorted set of rows that are stored together is referred to as a region (subset of table data).
4. HBase architecture has a single HBase master node (**HMaster**) and several slaves i.e. **region servers**.
5. Each region server (slave) serves a set of regions, and a region can be served only by a single region server.
6. Whenever a client sends a write request, HMaster receives the request and forwards it to the corresponding region server.
7. HBase tables are partitioned into multiple regions with every region storing multiple table’s rows.

**HBase architecture components**

****

[**https://www.dezyre.com/article/overview-of-hbase-architecture-and-its-components/295**](https://www.dezyre.com/article/overview-of-hbase-architecture-and-its-components/295)

1. **HMaster:** HBase HMaster is a lightweight process that assigns regions to region servers in the Hadoop cluster for load balancing. Its responsibilities are:
   1. Manages and monitors the hadoop cluster
   2. Performs administration. Provides interface for creating, updating and deleting tables.
   3. Controlling the failover
   4. DDL operations are handled by the HMaster like a schema change request raised by client etc.
2. **Region Server:** This process runs in the every node of the hadoop cluster. Handles read, write, update and delete requests from the client. Following are the components of a region server:
   1. **Block cache:** This is the read cache where most frequently read data is stored.
   2. **MemStore:** this is the write cache and stores the new data that is not yet stored to the disk. Every column family in the region has a MemStore. Data is sorted in mem store before storing in HFile
   3. **Write ahead log(WAL):** Stores new data that is not persisted to permanent storage
   4. **HFile** is the actual storage file that stores the rows as sorted key values on a disk
3. **Zookeeper:** HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning.
   1. ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization.
   2. Whenever a client wants to communicate with regions, they have to approach Zookeeper first. HMaster and Region servers are registered with ZooKeeper service.
   3. In case of node failure within an HBase cluster, ZKquoram will trigger error messages and start repairing failed nodes.
   4. ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Some other roles of zookeeper are:
      1. Establishing client communication with region servers.
      2. Tracking server failure and network partitions.
      3. Maintain Configuration Information
      4. Provides ephemeral nodes, which represent different region servers.

<https://www.edureka.co/blog/hbase-architecture/>

<https://www.guru99.com/hbase-architecture-data-flow-usecases.html>