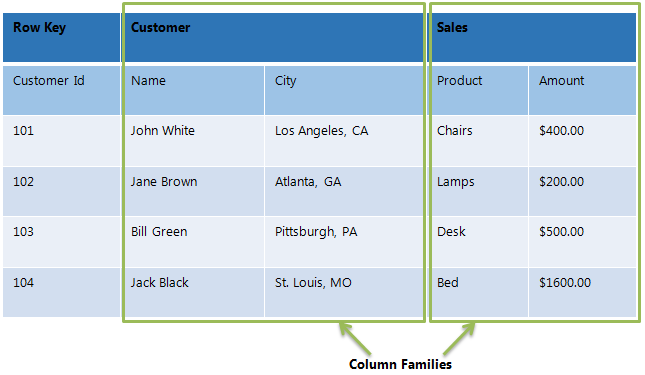
* Hbase is a schema less database, what does it mean?

What this really means is that the "schema" is stored with the record, not the table. In a RDBMS, the schema is defined and that table has the schema. In HBase data is labeled with its types. An analogy is CSV : RDBMS is to XML : HBase. In CSV, you have an assumption on what a column is. In XML, you specify exactly what it is.

The Data Model in HBase is designed to accommodate semi-structured data that could vary in field size, data type and columns. Additionally, the layout of the data model makes it easier to partition the data and distribute it across the cluster. The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells and Versions.

[](http://www.netwoven.com/wp-content/uploads/2013/10/hbase-2.png)

*Tables* – The HBase Tables are more like logical collection of rows stored in separate partitions called Regions. As shown above, every Region is then served by exactly one Region Server. The figure above shows a representation of a Table.

*Rows* – A row is one instance of data in a table and is identified by a *rowkey*. Rowkeys are unique in a Table and are always treated as a byte[].

*Column Families* – Data in a row are grouped together as Column Families. Each Column Family has one more Columns and these Columns in a family are stored together in a low level storage file known as HFile. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence it’s important that proper care be taken when designing Column Families in table. The table above shows Customer and Sales Column Families. The Customer Column Family is made up 2 columns – Name and City, whereas the Sales Column Families is made up to 2 columns – Product and Amount.

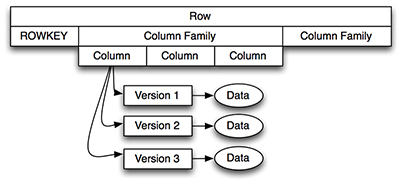
*Columns* – A Column Family is made of one or more columns. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have varied number of Columns.

*Cell* – A Cell stores data and is essentially a unique combination of *rowkey*, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value and the data type is always treated as byte[].

*Version* – The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

HBase is not a relational database and requires a different approach to modeling your data. HBase actually defines a four-dimensional data model and the following four coordinates define each cell (see Figure 1):

* Row Key: Each row has a unique row key; the row key does not have a data type and is treated internally as a byte array.
* Column Family: Data inside a row is organized into column families; each row has the same set of column families, but across rows, the same column families do not need the same column qualifiers. Under-the-hood, HBase stores column families in their own data files, so they need to be defined upfront, and changes to column families are difficult to make.
* Column Qualifier: Column families define actual columns, which are called column qualifiers. You can think of column qualifiers as the columns themselves.
* Version: Each column can have a configurable number of versions, and you can access the data for a specific version of a column qualifier.

[](javascript:popUp('/content/images/art_haines_hbases1_1/elementLinks/haines_hbase_1_1_01.jpg'))

[Figure 1**.**](javascript:popUp('/content/images/art_haines_hbases1_1/elementLinks/haines_hbase_1_1_01.jpg'))  HBase Four-Dimensional Data Model

● What is the minimum number of column family every Hbase table should have?

HBase currently does not do well with anything above two or three column families so keep the number of column families in your schema low. Currently, flushing and compactions are done on a per Region basis so if one column family is carrying the bulk of the data bringing on flushes, the adjacent families will also be flushed even though the amount of data they carry is small. When many column families exist the flushing and compaction interaction can make for a bunch of needless i/o. Where multiple ColumnFamilies exist in a single table, be aware of the cardinality (i.e., number of rows). If ColumnFamilyA has 1 million rows and ColumnFamilyB has 1 billion rows, ColumnFamilyA's data will likely be spread across many, many regions (and RegionServers). This makes mass scans for ColumnFamilyA less efficient.

● What is the benefit of using connection pool in Hbase?

For applications which require high-end multithreaded access (e.g., web-servers or application servers that may serve many application threads in a single JVM), you can pre-create an HConnection, as shown in the following example:

Example Pre-Creating a HConnection

// Create a connection to the cluster.

HConnection connection = HConnectionManager.createConnection(Configuration);

HTableInterface table = connection.getTable("myTable");

// use table as needed, the table returned is lightweight

table.close();

// use the connection for other access to the cluster

connection.close();

A cluster connection encapsulating lower level individual connections to actual servers and a connection to zookeeper. Connections are instantiated through the ConnectionFactory class. The lifecycle of the connection is managed by the caller, who has to close() the connection to release the resources.

The connection object contains logic to find the master, locate regions out on the cluster, keeps a cache of locations and then knows how to re-calibrate after they move. The individual connections to servers, meta cache, zookeeper connection, etc are all shared by the Table and Admin instances obtained from this connection.

Connection creation is a heavy-weight operation. Connection implementations are thread-safe, so that the client can create a connection once, and share it with different threads. Table and Admin instances, on the other hand, are light-weight and are not thread-safe. Typically, a single connection per client application is instantiated and every thread will obtain its own Table instance. Caching or pooling of Table and Admin is not recommended.