Data Model in HBase

The agenda of this discussion would be to try to understand the HBase data model that is how HBase actually stores the data and then the operations that are permitted on HBase. HBase uses the following data model that is in HBase data is stored in tables which have rows and columns. This might look very similar to that of a table in traditional RDBMS. However, as we proceed further we would understand that there is a huge difference.

Every entry in a table is actually going to be indexed by means of a RowKey. Here the rows are going to be sorted numerically by the RowKey. For this reason, the design of the RowKey is very important. The most important thing is that as a when the data is going to be stored in the table, the RowKeys will be sorted. so the sorting keeps happening dynamically.

The goal here is to store the data in such a way that the related rows are going to be next to each other. For every RowKey, an unlimited number of attributes or simply columns can actually be stored. It can also be called as columns or column qualifier. There is no strict schema with respect to the columns. New columns can be added during runtime and the columns are grouped into column families.

The column families have to be defined during the table creation time and they cannot be changed afterwards. Column families physically store a set of columns and their values together, mostly for performance reasons and each column family has a set of storage property such as how its data is compressed, etc. Each row in the table has the same column families. Though a given row might not store anything particular in a particular column family.

Column families are going to be specified when the tables are going to be created. These column families actually influenced the way your data is going to be stored in the underlying system. Therefore, these column family should be considered carefully during the schema design. Notice here that, the column families are simply the rows are going to be sparsely populated. This is what actually is meant by a flexible schema when it comes to HBase. This is not possible in the case of your RDBMS tables.

HBase tables are sparsed that is not all rows need to have values for all columns. So here you can notice that RowKey 4 has just one value that is for column 6 and the rest of here is empty. This is what we actually mean by sparse. I can add a new column dynamically and the remaining values can also be empty like how we have seen here for column number 6.

A cell is a combination of row, column family and the column. It contains a value at a timestamp which represents the values version. The timestamp is written alongside each value. So what you see here in column one which belongs to the column family one, also has a RowKey of one. So this is actually a cell. The intersection of a RowKey, column family, column and also the timestamp, when it was updated into a table is actually belonging to a cell, basically the timestamp is always been alongside each value and is the identifier for a given version of a value and by default, the timestamp is the time on which the server when the data was written but also you can specify a different timestamp value when you put the data into a cell.

Once a value is written into a HBase cell, it cannot be changed. Instead another version with the more recent timestamp can be added. You can specify the maximum number of values that it can actually store. This is actually called as versions that is HBase retains per column family and when the maximum number of versions is reached, the oldest version are eventually deleted and by default only the newest version actually is kept.

Even though, HBase is a distributed system, it is guaranteed that all the values belonging to the same RowKey and the column family will always be stored together. This is the data model of HBase. HBase supports the following operations.

Get: Get basically returns the values for a given RowKey. Here filters can be used to restrict the results of specific column families. Columns or even versions but you can only get one row at a time.

Put: Put basically adds a new entry into the table. Here the timestamp can be set automatically or it can also be set manually like just we mention some time ago.

Scan: Scan basically returns the values for a range of RowKey. Scans a very efficient in HBase because it is very fast and easy to fetch large range of rows from a billion row table because the way HBase stores the data internally. This is something that we will look at in the video when we discuss the architecture.

Delete: Delete basically deletes a cell value. Let us quickly setup a HBase on virtual machine and explore some of the features we talked about. We will create a sample table, load some data and then play around with HBase just to get a feel of it.