Session 9

Assignment 1

|  |  |
| --- | --- |
| **Prepared For:** | AcadGild |
|  |  |
| **Document Approval:** | **AcadGild** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Project Title:** | Session 9 – Assignment 1 |
|  |  |
| **Prepared By:** | Duncan Burgess |
|  |  |
|  | dburgess@duncb.com |
|  |  |
| **Primary Engineer:** | Duncan Burgess |
|  |  |
| **Document Reference:** | **Session 9 – Assignment 1** |
|  |  |
| **Start Date:** | 25/09/2017 |
|  |  |
|  |  |

# 

# Contents

[Contents 2](#_Toc494099462)

[Change History 3](#_Toc494099463)

[1. Problem Statement 4](#_Toc494099464)

[1.1. What is NoSQL data base? 5](#_Toc494099465)

[1.2. How does data get stored in NoSQl database? 5](#_Toc494099466)

[1.3. What is a column family in HBase? 6](#_Toc494099467)

[1.4. How many maximum number of columns can be added to HBase table? 7](#_Toc494099468)

[1.5. Why columns are not defined at the time of table creation in HBase? 7](#_Toc494099469)

[1.6. How does data get managed in HBase? 8](#_Toc494099470)

[1.7. What happens internally when new data gets inserted into HBase table? 8](#_Toc494099471)

# Change History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Document Revision** | **Date** | **Authored By** | **Authorised By** | **Sections Affected** | **Reason for Change** |
| Rev 01 | 25/09/2017 | Duncan Burgess |  | All | Initial release. |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Problem Statement

1. What is NoSQL data base?

2. How does data get stored in NoSQl database?

3. What is a column family in HBase?

4. How many maximum number of columns can be added to HBase table?

5. Why columns are not defined at the time of table creation in HBase?

6. How does data get managed in HBase?

7. What happens internally when new data gets inserted into HBase table?

## What is NoSQL data base?

**What is a NoSQL (Not Only SQL) Database?**

A NoSQL database environment is, simply put, a non-relational and largely distributed database system that enables rapid, ad-hoc organization and analysis of extremely high-volume, disparate data types. NoSQL databases are sometimes referred to as cloud databases, non-relational databases, Big Data databases and a myriad of other terms and were developed in response to the sheer volume of data being generated, stored and analysed by modern users (user-generated data) and their applications (machine-generated data).

In general, NoSQL databases have become the first alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. They go well beyond the more widely understood legacy, relational databases (such as Oracle, SQL Server and DB2 databases) in satisfying the needs of today’s modern business applications. A very flexible and [**schema-less data model**](http://planetcassandra.org/blog/post/schema-vs-schema-less), horizontal scalability, distributed architectures, and the use of languages and interfaces that are “not only” SQL typically characterize this technology.

From a business standpoint, considering a NoSQL or ‘Big Data’ environment has been shown to provide a clear competitive advantage in numerous industries. In the ‘age of data’, this is compelling information as a great saying about the importance of data is summed up with the following “if your data isn’t growing then neither is your business”.

## How does data get stored in NoSQl database?

HBase is a column-oriented NoSQL database. Although it looks similar to a relational database which contains rows and columns, but it is not a relational database. Relational databases are row oriented while HBase is column-oriented. So, let us first understand the difference between Column-oriented and Row-oriented databases:

Row-oriented vs column-oriented Databases:

* Row-oriented databases store table records in a sequence of rows. Whereas column-oriented databases store table records in a sequence of columns, i.e. the entries in a column are stored in contiguous locations on disks.

To better understand it, let us take an example and consider the table below.



If this table is stored in a row-oriented database. It will store the records as shown below:

**1, DuncanBurgess, UK, 242, Fish Food**

**2, Suzanne Smith, US, 487, Dog Meat**

In row-oriented databases data is stored on the basis of rows or tuples as you can see above.

While the column-oriented databases store this data as:

**1,2,Duncan Burgess, Suzanne Smith,UK, US,242, 487,Fish Food, Dog Meat**

In a column-oriented databases, all the column values are stored together like first column values will be stored together, then the second column values will be stored together and data in other columns are stored in a similar manner.

* When the amount of data is very huge, like in terms of petabytes or exabytes we use column-oriented approach, because the data of a single column is stored together and can be accessed faster.
* While row-oriented approach comparatively handles less number of rows and columns efficiently, as row-oriented database stores data is a structured format.
* When we need to process and analyse a large set of semi-structured or unstructured data, we use column oriented approach. Such as applications dealing with **Online Analytical Processing** like data mining, data warehousing, applications including analytics, etc.
* Whereas, **Online Transactional Processing** such as banking and finance domains which handle structured data and require transactional properties (ACID properties) use row-oriented approach.

## What is a column family in HBase?

The following components define a column family in the image and text below:



* **Tables**: Data is stored in a table format in HBase. But here tables are in column-oriented format.
* **Row** **Key**: Row keys are used to search records which make searches fast. You would be curious to know how? I will explain it in the architecture part moving ahead in this blog.
* **Column** **Families**: Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.
* **Column** **Qualifiers**: Each column’s name is known as its column qualifier.
* **Cell**: Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
* **Timestamp**: Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

In a more simple way, we can say HBase consists of:

* Set of tables
* Each table with column families and rows
* Row key acts as a Primary key in HBase.
* Any access to HBase tables uses this Primary Key
* Each column qualifier present in HBase denotes attribute corresponding to the object which resides in the cell.

## How many maximum number of columns can be added to HBase table?

There is no hard/special limit to number of columns in HBase, we can have more than 1 million columns but usually three column families are recommended (not more than three).

**Beware there** is a potential issue where we could have ‘too wide’ rows (many columns) and if you do not specify exact qualifiers any scan will result whole rows so you could get much more data than you actually need.

## Why columns are not defined at the time of table creation in HBase?

HBase has the characteristic of **Schema-Less design:**

Columns in HBase don’t need to be defined up front so they provide a flexible way of managing evolving schemas. However columns can’t be renamed or assigned easily from one column-family to the other.

Making such changes requires creation of new columns, migration of data from existing columns to the new column and then potentially deletion of old columns.

HBase is effectively a map of a map, so it has no schema. The user can define the columns at runtime, and each row can have its own columns. The responsibility is on the application to interpret the values stored in the HBase.

This makes HBase very suitable for applications in which the schema is flexible. Also, unlike a database in which a separate metadata table is needed to describe the page identifiers, each column key describes itself in HBase.

## How does data get managed in HBase?

Data manipulation commands that include **COUNT**, **DELETE**, **DELETEALL** and **SCAN**.

Some table management commands like: **ALTER**, **CREATE**, **DESCRIBE**, **DROP**, and **DROPALL**. Some general commands like **VERSION** and **STATUS** all comprise a very small list of commands available for managing data in Hbase.

The Hbase data model is different from the model provided by relational databases. Hbase is referred to by many terms like a key-value store, column oriented database and versioned map of maps which are correct. The easiest way of visualizing an Hbase data model is a table that has rows and tables.

## What happens internally when new data gets inserted into HBase table?

The following steps are carried out when inserting Data into an HBASE table

**HBase Write Steps**

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

- Edits are appended to the end of the WAL file that is stored on disk.

- The WAL is used to recover not-yet-persisted data in case a server crashes.

**HBase Write Steps**

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.

**HBase MemStore**

The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.

**HBase Region Flush**

When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS. HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances. These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.

**HBase HFile**

Data is stored in an HFile which contains sorted key/values. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.

**HFile Index**

The index, is loaded when the HFile is opened and kept in memory. This allows lookups to be performed with a single disk seek.