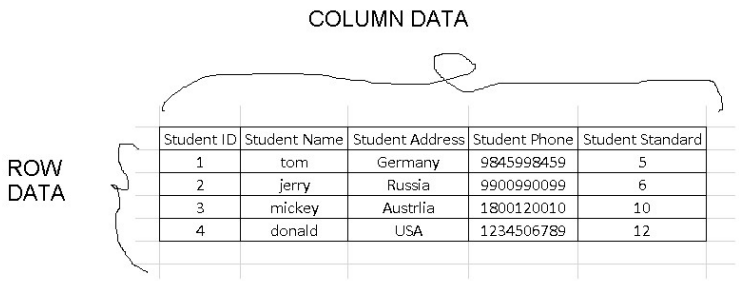
BigData Concepts

29 December 2023 08:35

HBase Architecture: HBase Data Model
As we know, 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.



Student ID	Student Name	Student Address	Student Phone	Student Standard
1	tom	Germany	9845998459	5
2	jerry	Russia	9900990099	6
3	mickey	Austrlia	1800120010	10
4	donald	USA	1234506789	12

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

1 ,tom ,Germany ,9845998459 ,5
2 ,jerry ,Russia ,9900990099 ,6
3 ,mickey ,Austrlia ,1800120010 ,10
4 ,donald ,USA ,1234506789 ,12

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:

1,2,3,4, tom,jerry,mickey,donald, Germany, Russia, Australia,USA, 9845998459, 9900990099, 1800120010, 1234506789 ,5,6,10,12

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 analyze 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.

HBase tables has following components, shown in the image below:

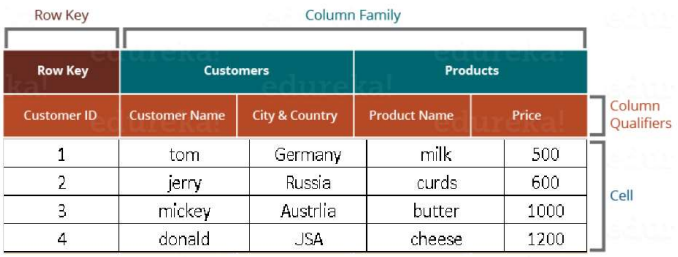


Figure: HBase Table

- **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.

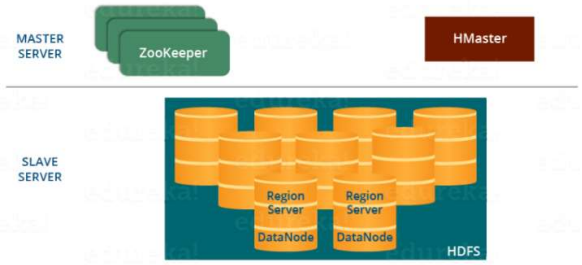
In a more simple and understanding way, we can say HBase consists of:

- 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.

HBase Architecture: Components of HBase Architecture

HBase has three major components i.e., **HMaster Server**, **HBase Region Server, Regions** and **Zookeeper**.

The below figure explains the hierarchy of the HBase Architecture. We will talk about each one of them individually.



HBase Architecture: Region

A region contains all the rows between the start key and the end key assigned to that region. HBase tables can be divided into a number of regions in such a way that all the columns of a column family is stored in one region. Each region contains the rows in a sorted order.
Many regions are assigned to a **Region Server**, which is responsible for handling, managing, executing reads and writes operations on that set of regions.

So, concluding in a simpler way:

- A table can be divided into a number of regions. A Region is a sorted range of rows storing data between a start key and an end key.
- A Region has a default size of 256MB which can be configured according to the need.
- A Group of regions is served to the clients by a Region Server.
- A Region Server can serve approximately 1000 regions to the client.

Hbase Commands

1. Basic commands
    a. Status
    b. Version
    c. Table_help
    d. Whoami
    e. Process list

2. DDL commands
    a. Create table
    b. List
    c. Describe
    d. Disable
    e. Disable_all
    f. Drop table
    g. Drop_all
    h. Is_enabled
    i. Alter
    j. Show_filters

3. DML Commands
    a. Put
    b. Count
    c. Get
    d. Scan
    e. Delete
    f. Delte all
    g. Truncate

HBASE COMMANDS

BASIC COMMANDS
>status
>status 'summary'
>status 'simple'
>status 'detailed'
>version
>table_help
> whoami
hadoop (auth:SIMPLE)
    groups: hadoop
Took 0.0966 seconds

> processlist

DDL COMMAND
> list

```
> scan 'mytable'

> disable 'mytable'
hbase:010:0> drop 'mytable'
Took 0.8233 seconds

> create 'Students','Student ID','Student Name','Student Address','Student Phone','Student Standard'
> list
> describe 'Students'
> show_filters
> is_enabled 'Students'

> alter 'Students', NAME='Student DOB'
>describe 'Students'
> alter 'Students','Student ID','Student Name','Student Address','Student Phone','Student Standard', {NAME => 'Student Name',IN_MEMORY => true }

> alter 'Students',{'NAME' => 'Student Name','IN_MEMORY' => 'true' }
> alter 'Students',{'NAME' => 'Student Standard','IN_MEMORY' => 'true' }
> describe 'Students'
> put 'Students','Student Address:germany','Student DOB:12122012','Student ID:1','Student Name:tom','Student Phone:9856471236','Student Standard:5'

>put 'Students','1','Student Address:SAddr','Germany'
put 'Students','2','Student Address:SAddr','Russia'
put 'Students','3','Student Address:SAddr','Australia'
put 'Students','4','Student Address:SAddr','USA'

put 'Students','1','Student DOB:Sdob','12122012'
put 'Students','2','Student DOB:Sdob','19102010'
put 'Students','3','Student DOB:Sdob','01012020'
put 'Students','4','Student DOB:Sdob','02062011'

put 'Students','1','Student ID:SID','1'
put 'Students','2','Student ID:SID','2'
put 'Students','3','Student ID:SID','3'
put 'Students','4','Student ID:SID','4'

put 'Students','1','Student Name:SName','Tom'
put 'Students','2','Student Name:SName','Jerry'
put 'Students','3','Student Name:SName','mickey'
put 'Students','4','Student Name:SName','donald'

>alter 'Students','delete' =>'Student Standard'
>alter 'Students','delete' =>'Student Phone'


>scan 'Students'
>count 'Students'

>count 'Students',CACHE=>10

> get 'Students','1'
> get 'Students','1',{ 'COLUMN'=>'Student Name:SName'}
> get 'Students','2',{ 'COLUMN'=>'Student Name:SName'}
> get 'Students','3',{ 'COLUMN'=>'Student Name:SName'}
> get 'Students','4',{ 'COLUMN'=>'Student Name:SName'}

> delete 'Students','4','Student Name:SName'

SECURITY RELATED COMMANDS
> grant 'hadoop','RWXCA'


CREATING NEW USER JAMES
bigadmin@mainserver1:~$ su - root
Password:
root@mainserver1:~# useradd -d /home/james -m -s /bin/bash james
root@mainserver1:~# passwd james
New password:
Retype new password:
passwd: password updated successfully
root@mainserver1:~# su - james
james@mainserver1:~$ hbase shell


CHECKING FOR MSSQL JDBC

hadoop@mainserver1:/usr/local/hbase/conf$ jar -tf /usr/local/hbase/lib/mssql-jdbc-6.2.1.jre7.jar | grep -i version


hadoop@mainserver1:/usr/local/hbase/conf$ cd
hadoop@mainserver1:~$ mkdir hbase-scripts
hadoop@mainserver1:~$ cd hbase-scripts/
hadoop@mainserver1:~/hbase-scripts$
```

```
hadoop@mainserver1:~/hbase-scripts$ nano Create_Table.java
hadoop@mainserver1:~/hbase-scripts$ cd
hadoop@mainserver1:~$ su
Password:
root@mainserver1:/home/hadoop# wget https://mirrors.estointernet.in/apache/ma     ven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz
root@mainserver1:/home/hadoop# tar -xvf apache-maven-3.6.3-bin.tar.gz
root@mainserver1:/home/hadoop# mv apache-maven-3.6.3 /opt/

root@mainserver1:/home/hadoop# exit
hadoop@mainserver1:~$ nano .profile
hadoop@mainserver1:~$ source .profile
hadoop@mainserver1:~$ mvn -version
```

09. Hive

The Hive Data-ware House

Lecture

- Introduction to Hive,
- Hive architecture and Installation,
- Comparison with Traditional Database,
- Basics of Hive Query Language.

Working with Hive QL

Lecture

- Datatypes,
- Operators and Functions,
- Hive Tables (Managed Tables and Extended Tables),
- Partitions and Buckets,
- Storage Formats,
- Importing data,
- Altering and Dropping Tables

Lab-Assignment:

- Creative a hive DB and table (internal and external)
- Load the data into hive table (using local inpath and HSFS inpath)

10. Querying with Hive QL

Lecture

- Querying Data-Sorting,
- Aggregating,
- Map Reduce Scripts,
- Joins and Sub queries,
- Views,
- Map and Reduce side joins to optimize query.

Lab-Assignment:

- Run all the types of joins in Hive
- Execute the data to be partitioned

More on Hive QL

Lecture

- Data manipulation with Hive,
- UDFs,
- Appending data into existing Hive table,
- custom map/reduce in Hive
- Writing HQL scripts

Apache Hive

IT is a data warehousing and SQL-like query language system built on top of Hadoop.
Provides the feature to manage and query large datasets stored in HDFS or other compatible distributed storage systems.
Hive allows users to write querys using a SQL-like language ---> HiveQL, and later translated itno mapreduce jobs for execution
Hive is a data warehouse infrastructure tool to process structured data in hadoop

Before intalling Apache Hive verify the following

1. Java is installed -->
   a. $ java -version
2. Hadoop is installed, configured and working fine -->
   a. $ hadoop version
   b. $ jps

   This command should list all the services such as one shown below
   $

```
hadoop@mainserver1:~$ jps
12082 HRegionServer
2114 NodeManager
1363 NameNode
1524 DataNode
1944 ResourceManager
1786 SecondaryNameNode
11851 HQuorumPeer
16204 Jps
11964 HMaster
hadoop@mainserver1:~$
```

$ cd
$ wget https://archive.apache.org/dist/hive/hive-2.3.9/apache-hive-2.3.9-bin.tar.gz
$ su
$ tar zxvf apache-hive-2.3.9-bin.tar.gz
$ su
# mv apache-hive-2.3.9-bin.tar.gz /usr/local/hive

We you require an external database server to configure Metastore. We use Apache Derby database.

$ cd
$ wget https://archive.apache.org/dist/db/derby/db-derby-10.15.2.0/db-derby-10.15.2.0-bin.tar.gz
$ tar zxvf db-derby-10.15.2.0-bin.tar.gz
$ cd
# mv db-derby-10.15.2.0-bin /usr/local/derby

nano /etc/bash.bashrc

{ add the below details in to this file }

```
# HIVE  CONFIGS
export HIVE_HOME=/usr/local/hive
export CLASSPATH=`hadoop classpath`
export CLASSPATH=$CLASSPATH:/usr/local/Hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/hive/lib/*:.
export PDSH_RCMD_TYPE=ssh
#HBASE CONFIGS
export HBASE_HOME="/usr/local/hbase"
export PATH="$PATH:$HBASE_HOME/bin:$HIVE_HOME/bin"
#DERBY CONFIG
export DERBY_HOME=/usr/local/derby
export PATH=$PATH:$DERBY_HOME/bin
export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

{save and exit}

# source /etc/bash.bashc
# exit
$ source /etc/bash.bashrc

$ cd $HIVE_HOME/conf

$ cp hive-env.sh.template hive-env.sh

$ nano hive-env.sh

```
export HADOOP_HOME=/usr/local/hadoop
```

<save and exit>

$ mkdir $DERBY_HOME/data

{ if the directory ownerships are different i.e, other then hadoop use the following command to rectify the same

$su
chown -R hadoop.hadoop /usr/local/hive
chown -R hadoop.hadoop /usr/local/derby

}

$ cd $HIVE_HOME/conf

$ cp   hive-default.xml.template   hive-site.xml

$ nano hive-site.xml

{ append the lines to the end of the file but should be placed in between the <cofiguration> and </configuration> tags }

```
 </property>
  <property>
     <name>javax.jdo.option.ConnectionURL</name>
     <value>jdbc:derby://localhost:1527/metastore_db;create=true </value>
     <description>JDBC connect string for a JDBC metastore </description>
  </property>
```

$ nano jpox.properties

```
javax.jdo.PersistenceManagerFactoryClass = org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL = jdbc:derby://localhost:1527/metastore_db;create = true
javax.jdo.option.ConnectionUserName = hadoop
javax.jdo.option.ConnectionPassword = hadoop
```

<save and exit>