Lab: Column Families

Objective: Create several tables with different column families and explore the physical layout of the table in hdfs.

The components of HBase data model consist of tables, rows, column families, columns, cells and versions. Tables are like logical collection of rows stored in separate partitions. A row is one instance of data in a table and is identified by a rowkey.

Data in a row are grouped together as Column Families. Each Column Family has one or more Columns and these Columns in a family are stored together.

Column Families form the basic unit of physical storage, hence it’s important that proper care be taken when designing Column Families in table. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon.

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 versioned and versions of data are identified by the timestamp.

Versioning in HBase with Column

Families

Now let's create a table and see a little more about how versioning works, this time with a column family.

In the HBase Shell, create a table for trucks that stores more than one version of a column:

```
    hbase> create 'truck', {NAME => 'cf1', VERSIONS => 2}
```

This creates a table with column family `cf1` and any data stored in any columns in that column family will be permitted to have up to 2 versions. Versions beyond 2 will be deleted oldest first.

Insert multiple versions of the same column. We'll call it `cf1:weight`:

```
    hbase> put 'truck','1', 'cf1:weight','58643'
    hbase> put 'truck','1', 'cf1:weight','64532'
```

Now scan the table requesting multiple versions (note different timestamps):

```
    hbase> scan 'truck', {VERSIONS => 2}
    ROW COLUMN+CELL
    1   column=cf1:weight, timestamp=1390167231632, value=
64532
    1   column=cf1:weight, timestamp=1390167226238, value=
```

```
58643
```

Add a third value for the column identifier `cf1:weight`:

```
    hbase> put 'truck','1', 'cf1:weight','42587'
    0 row(s) in 0.0080 seconds
```

And scan again:

```
    hbase> scan 'truck', {VERSIONS => 2}
    ROW COLUMN+CELL
    1   column=cf1:weight, timestamp=1390167445021, value=
42587
    1   column=cf1:weight, timestamp=1390167231632, value=
64532
    1 row(s) in 0.0110 seconds
```

Now, try to scan for three versions:

```
    hbase> scan 'truck', {VERSIONS => 3}
    ROW COLUMN+CELL
    1   column=cf1:weight, timestamp=1390167445021, value=
42587
    1   column=cf1:weight, timestamp=1390167231632, value=
64532
    1 row(s) in 0.0170 seconds
```

Question: what happened?

> *Note: Gets vs. Scans: If the table is large, the scan operation uses a lot of resources. HBase was designed for the optimal lookup to be a single row get.*

Create Table with 2 Column Families

Now create a table called for shipping with two column families, column family `a` will store a single version
of each cell, column family `b` will store up to 3 versions of each cell:

```
    hbase> create 'shipping',{NAME=>'a', VERSIONS =>1},{NA
ME=>'b', VERSIONS=>2}
```

Insert some cells into each column family:

```
    hbase> put 'shipping','1','a:truck','East Lansing'
    0 row(s) in 0.0200 seconds
```

Retrieve all cells for rowkey 1:

```
    hbase> get 'shipping',1
    COLUMN  CELL
    a:truck timestamp=1396035080378, value=East Lansing
```

Put a `temperature` into column family ‘b’:

```
    hbase> put 'shipping','1','b:temperature','87'
```

Retrieve all cells for rowkey 1:

```
    hbase> get 'shipping',1
    COLUMN      CELL
    a:truck     timestamp=1396035080378, value=East Lansin
g
    b:temperature   timestamp=1396035310760, value=87
```

Enter a New Value

Enter a new `temperature` for rowkey ‘1’ into column family b:

```
    hbase> put 'shipping','1','b:temperature','92'
```

Now again:

```
    hbase(main):001:0> get 'shipping',1
    COLUMN            CELL

    a:truck           timestamp=1534219925528, value=East
Lansing
    b:temperature     timestamp=1534220028640, value=92
```

```
    1 row(s)
    Took 0.4605 seconds
```

Question: where did the previous temperature value go?

Retrieve multiple versions of the cells for rowkey 1 by asking for them:

```
    hbase> get 'shipping',1,{COLUMN => ['a','b'],VERSIONS=
>2}
    COLUMN          CELL
    a:truck         timestamp=1396035080378, value=East La
nsing
    b:temperature   timestamp=1396035310760, value=92
    b:temperature   timestamp=1396035206632, value=87
```

Look at the HBase Master UI

Let’s look at the Ambari view of things. Go to the main HBase page on Ambari. Look at the menu at the right:

<img width=“1421” alt=“screen shot 2018-09-01 at 6 03 48 am” src="https://user-images.githubusercontent.com/558905/44944905-033bd380-adad-11e8-9b63-a4a61d87ccc7.png">

Once in the Master UI, then choose the `shipping` table from the list of tables under the `Tables` option:

<img width="1347" alt="screen shot 2018-09-01 at 6 13 11 am" src="https://user-images.githubusercontent.com/558905/44944972-3468d380-adae-11e8-9d50-86d6a0fc8eb9.png">

then scroll down to the `Table Regions` section of the page:

<img width="800" alt="the HBase Master UI" src="https://user-images.githubusercontent.com/558905/44944875-7a249c80-adac-11e8-9033-e3d31349da01.png">

You see here some statistics about your table.

Now Look at the Data - Where is it Stored?

The rows just inserted are in a memory cache. Flush them to hdfs:

```
hbase> flush 'shipping'
0 row(s) in 0.1460 seconds
```

Find the data directories for each column family.

Quit the HBase shell and run this hdfs command:

```
hdfs dfs -ls /apps/hbase/data/data/default/shipping
Found 3 items
```

```
    drwxr-xr-x  - hbase hdfs  0 2017-03-28 12:31  /apps/hb
ase/data/data/default/shipping/.tabledesc
    drwxr-xr-x  - hbase hdfs  0 2017-03-28 12:31  /apps/hb
ase/data/data/default/shipping/.tmp
    drwxr-xr-x  - hbase hdfs  0 2017-03-28 12:43  /apps/hb
ase/data/data/default/shipping/98491258b8d8b1dd5e7d84478a6
f3290
```

This shows that the data for table `shipping` is stored in hdfs under the directory.

The directory with the hex number for a name is the version number for this table; this number will be different.
Use the following command to see the content:

```
    hdfs dfs -ls -r /apps/hbase/data/data/default/shipping
    Found 1 items
    -rw-r--r--  3 hbase hdfs  569 2017-03-28 12:31 /apps/h
base/data/data/default/shipping/.tabledesc/.tableinfo.0000
000001
    Found 4 items
    -rwxr-xr-x  3 hbase hdfs   35 2017-03-28 12:31 /apps/h
base/data/data/default/shipping/98491258b8d8b1dd5e7d84478a
6f3290/.regioninfo
    drwxr-xr-x  - hbase hdfs    0 2017-03-28 12:43 /apps/h
base/data/data/default/shipping/98491258b8d8b1dd5e7d84478a
6f3290/.tmp
```

```
    drwxr-xr-x  - hbase hdfs    0 2017-03-28 12:43 /apps/h
base/data/data/default/shipping/98491258b8d8b1dd5e7d84478a
6f3290/a
    drwxr-xr-x  - hbase hdfs    0 2017-03-28 12:43 /apps/h
base/data/data/default/shipping/98491258b8d8b1dd5e7d84478a
6f3290/b
```

Do the command again recursively:

```
    hdfs dfs -ls -R /apps/hbase/data/data/default/shipping
```

There is a directory `a` for data in column family `a`, and a directory `b` for data in column family `b`.

Summary

You have now created an HBase table with multiple column families, and see how the data is stored as it changes.