Aims

This exercise aims to get you to:

- Install and configure HBase
- Manage data using HBase Shell
- Manage data using HBase Java API

HBase Installation and Configuration

1. Download HBase 1.2.2

```
$ wget http://apache.uberglobalmirror.com/hbase/1.2.2/hbase-1.2.2-bin.tar.gz
```

Then unpack the package:

```
$ tar xvf hbase-1.2.2-bin.tar.gz
```

2. Define environment variables for HBase

We need to configure the working directory of HBase, i.e., `HBASE_HOME`.

Open the file `~/.bashrc` and add the following lines at the **end** of this file:

```
export HBASE_HOME = ~/hbase-1.2.2
export PATH = $HBASE_HOME/bin:$PATH
```

Save the file, and then run the following command to take these configurations into effect:

```
$ source ~/.bashrc
```

Open the HBase environment file, hbase-env.sh, using:

```
$ gedit $HBASE_HOME/conf/hbase-env.sh
```

Add the following lines at the **end** of this file:

```
export JAVA_HOME = /usr/lib/jvm/java-1.7.0-openjdk-amd64
export HBASE_MANAGES_ZK = true
```

3. Configure HBase as Pseudo-Distributed Mode

Open the HBase configuration file, hbase-site.xml, using:

```
$ gedit $HBASE_HOME/conf/hbase-site.xml
```

Add the following lines in between <configuration> and </configuration>:

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:9000/hbase</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
```

Now you have already done the basic configuration of HBase, and it is ready to use. Start HBase by the following command (start HDFS and YARN first!):

```
$ start-hbase.sh
```

You will see:

```
comp9313@comp9313-VirtualBox:~$ start-hbase.sh
localhost: starting zookeeper, logging to /home/comp9313/hbase/bin/../logs/hbase
-comp9313-zookeeper-comp9313-VirtualBox.out
starting master, logging to /home/comp9313/hbase/logs/hbase-comp9313-master-comp
9313-VirtualBox.out
starting regionserver, logging to /home/comp9313/hbase/logs/hbase-comp9313-1-reg
ionserver-comp9313-VirtualBox.out
```

Type "jps" in the terminal, you can see that more daemons are started.

```
comp9313@comp9313-VirtualBox:~$ jps
4129 HRegionServer
2854 DataNode
4212 Jps
3371 NodeManager
3067 SecondaryNameNode
3221 ResourceManager
2680 NameNode
4008 HMaster
3942 HQuorumPeer
```

Practice HBase Shell Commands

In this part, you will practice on how to manage data using HBase shell commands. As such, after completing this lab, you'll know how to

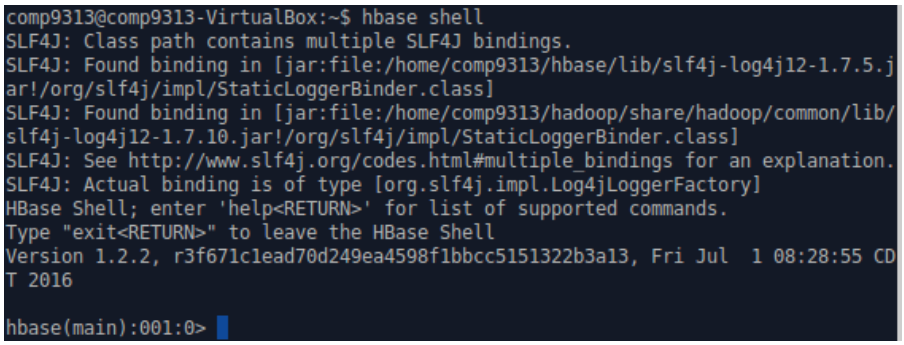
- Launch the HBase shell
- Create an HBase table
- Inspect the characteristics of a table
- Alter properties associated with a table
- Populate a table with data
- Retrieve data from a table

- Use HBase Web interfaces to explore information about your environment

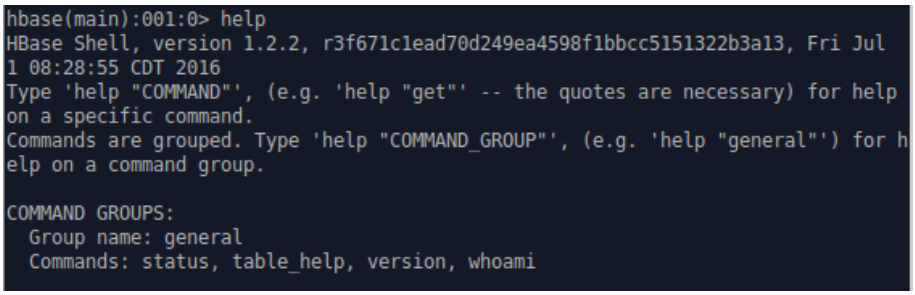
Launch the HBase shell

1. After HBase is started, use the following command to launch the shell:

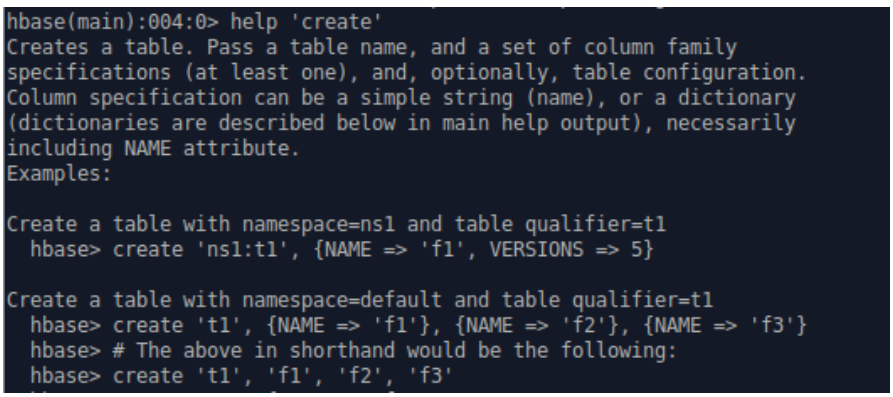
```
$ hbase shell
```



2. Once started, you can type in help, and then press Return, to get the help text (shown abbreviated):



You can request help for a specific command by adding the command when invoking help, or print out the help of all commands for a specific group when using the group name with the help command. The optional command or group name has to be enclosed in quotes. For example, type "help 'create'" in the shell, and you will see the usage of this command:



Creating and altering a table

1. Create an HBase table named reviews with 3 column families: summary, reviewer, and details.

```
$ create 'reviews', 'summary', 'reviewer', 'details'
```

2. Inspect the default properties associated with your new table:

```
$ describe 'reviews'
```

```
hbase(main):012:0> describe 'reviews'
Table reviews is ENABLED
reviews
COLUMN FAMILIES DESCRIPTION
{NAME => 'details', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICA
TION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY
 => 'false', BLOCKCACHE => 'true'}
{NAME => 'reviewer', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLIC
ATION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
 TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMOR
Y => 'false', BLOCKCACHE => 'true'}
{NAME => 'summary', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICA
TION_SCOPE => '0', VERSIONS => '1', COMPRESSION => 'NONE', MIN_VERSIONS => '0',
TTL => 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY
 => 'false', BLOCKCACHE => 'true'}
3 row(s) in 0.0210 seconds
```

3. To alter (or drop) a table, you must first disable it:

```
$ disable 'reviews'
```

4. Alter the table to set the IN_MEMORY property of the summary column family to true.

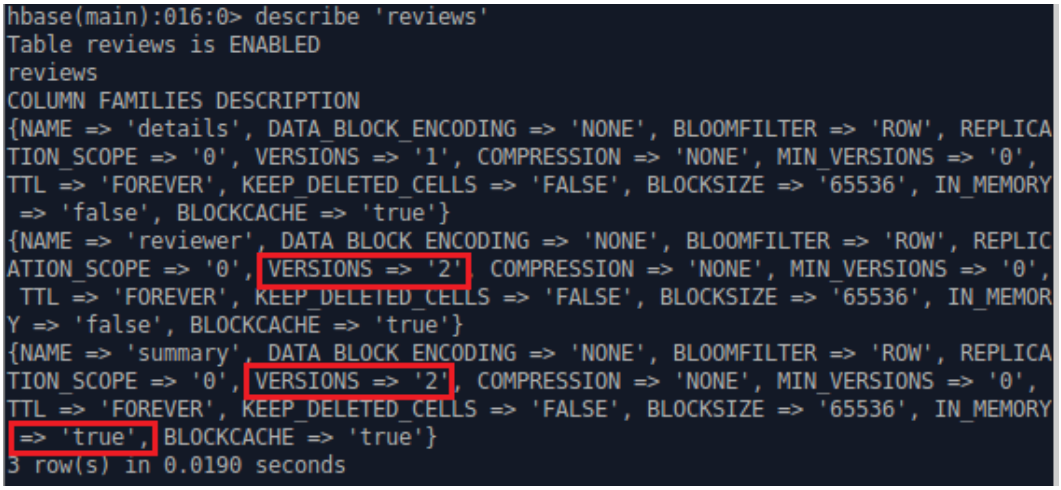
```
$ alter 'reviews', {NAME => 'summary', IN_MEMORY => 'true'}
```

5. Set the number of versions for the summary and reviewer column families to 2. HBase can store multiple versions of data for each column family. By default it is set to 1.

```
$ alter 'reviews', {NAME => 'summary', VERSIONS => 2}, {NAME =>
'reviewer', VERSIONS => 2}
```

Verify that your property changes were captured correctly:

```
$ describe 'reviews'
```



6. Enable (or activate) the table so that it's ready for use

```
$ enable 'reviews'
```

Now you can populate your table with data and query it.

Inserting and retrieving data

1. Insert some data into your HBase table. The PUT command enables you to write data into a single cell of an HBase table. This cell may reside in an existing row or may belong to a new row.

```
$ put 'reviews', '101', 'summary:product', 'hat'
```

What happened after executing this command

Executing this command caused HBase to add a row with a row key of 101 to the reviews table and to write the value of hat into the product column of the summary column family. Note that this command dynamically created the summary:product column and that no data type was specified for this column.

What if you have more data for this row? You need to issue additional PUT commands – one for each cell (i.e., each column family:column) in the target row. You'll do that shortly. But before you do, consider what HBase just did behind the scenes . . . .

HBase wrote your data to a Write-Ahead Log (WAL) in your distributed file system to allow for recovery from a server failure. In addition, it cached your data (in a MemStore) of a specific region managed by a specific Region Server. At some point, when the MemStore becomes full, your data will be flushed to disk and stored in files (HFiles) in your distributed file system. Each HFile contains data related to a specific column family.

2. Retrieve the row. To do so, provide the table name and row key value to the GET command:

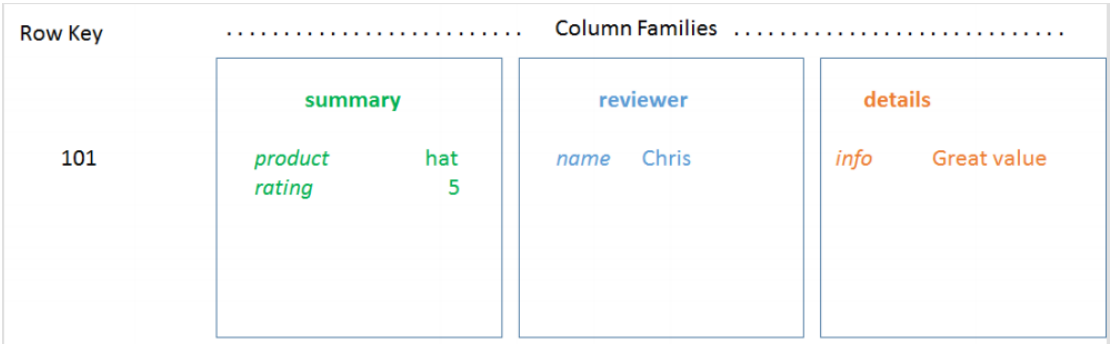
```
$ get 'reviews', '101'
```

```
hbase(main):019:0* get 'reviews', '101'
COLUMN                CELL
 summary:product      timestamp=1472405345388, value=hat
1 row(s) in 0.0350 seconds
```

3. Add more cells (columns and data values) to this row:

```
$ put 'reviews', '101', 'summary:rating', '5'
$ put 'reviews', '101', 'reviewer:name', 'Chris'
$ put 'reviews', '101', 'details:comment', 'Great value'
```

Conceptually, your table looks something like this:



Retrieve the row again:

```
hbase(main):023:0> get 'reviews', '101'
COLUMN                CELL
 details:comment      timestamp=1472405491873, value=Great value
 reviewer:name        timestamp=1472405487241, value=Chris
 summary:product      timestamp=1472405345388, value=hat
 summary:rating       timestamp=1472405481127, value=5
4 row(s) in 0.0230 seconds
```

This output can be a little confusing at first, because it's showing that 4 rows are returned. This row count refers to the number of lines (rows) displayed on the screen. Since information about each cell is displayed on a separate line and there are 4 cells in row 101, the GET command reports 4 rows.

4. Count the number of rows in the entire table and verify that there is only 1 row:

```
$ count 'reviews'
```

5. Add 2 more rows to your table using these commands:

```
$ put 'reviews', '112', 'summary:product', 'vest'
$ put 'reviews', '112', 'summary:rating', '5'
$ put 'reviews', '112', 'reviewer:name', 'Tina'
$ put 'reviews', '133', 'summary:product', 'vest'
$ put 'reviews', '133', 'summary:rating', '4'
$ put 'reviews', '133', 'reviewer:name', 'Helen'
$ put 'reviews', '133', 'reviewer:location', 'USA'
$ put 'reviews', '133', 'details:tip', 'Sizes run small. Order 1 size
up.'
```

Note that review 112 lacks any detailed information (e.g., a comment), while review 133 contains a tip in its details. Note also that review 133 includes the reviewer's location, which is not present in the other rows.

6. Retrieve the entire contents of the table using this SCAN command:

```
$ scan 'reviews'
```

```
hbase(main):034:0> scan 'reviews'
ROW                     COLUMN+CELL
 101                    column=details:comment, timestamp=1472405491873, value=Great value
 101                    column=reviewer:name, timestamp=1472405487241, value=Chris
 101                    column=summary:product, timestamp=1472405345388, value=hat
 101                    column=summary:rating, timestamp=1472405481127, value=5
 112                    column=reviewer:name, timestamp=1472405576782, value=Tina
 112                    column=summary:product, timestamp=1472405565373, value=vest
 112                    column=summary:rating, timestamp=1472405570311, value=5
 133                    column=details:tip, timestamp=1472405611320, value=Sizes run small. Ord
                        er 1 size up.
 133                    column=reviewer:location, timestamp=1472405607022, value=USA
 133                    column=reviewer:name, timestamp=1472405601078, value=Helen
 133                    column=summary:product, timestamp=1472405591649, value=vest
 133                    column=summary:rating, timestamp=1472405596526, value=4
3 row(s) in 0.0180 seconds
```

Note that SCAN correctly reports that the table contains 3 rows. The display contains more than 3 lines, because each line includes information for a single cell in a row. Note also that each row in your table has a different schema and that missing information is simply omitted.

Furthermore, each displayed line includes not only the value of a particular cell in the table but also its associated row key (e.g., 101), column family name (e.g., details), column name (e.g., comment), and timestamp. As you learned earlier, HBase is a key-value store. Together, these four attributes (row key, column family name, column qualifier, and timestamp) form the key.

Consider the implications of storing this key information with each cell value. Having a large number of columns with values for all rows (in other words, dense data) means that a lot of key information is repeated. Also, large row key values and long column family / column names increase the table's storage requirements.

7. Finally, restrict the scan results to retrieve only the contents of the summary column family and the reviewer:name column for row keys starting at '120' and ending at '150'.

```
$ scan 'reviews', {COLUMNS => ['summary', 'reviewer:name'], STARTROW
=> '120', STOPROW => '150'}
```

Given your sample data, only row '133' qualifies. Note that the reviewer's location (reviewer:location) and all the review details (details:tip) were omitted from the results due to the scan parameters you specified.

Updating data

1. Update Tina's review (row key 112) to change the rating to '4':

```
$ put 'reviews', '112', 'summary:rating', '4'
```

2. Scan the table to inspect the change.

```
hbase(main):037:0> scan 'reviews'
ROW                     COLUMN+CELL
 101                    column=details:comment, timestamp=1472405491873, value=Great value
 101                    column=reviewer:name, timestamp=1472405487241, value=Chris
 101                    column=summary:product, timestamp=1472405345388, value=hat
 101                    column=summary:rating, timestamp=1472405481127, value=5
 112                    column=reviewer:name, timestamp=1472405576782, value=Tina
 112                    column=summary:product, timestamp=1472405565373, value=vest
 112                    column=summary:rating, timestamp=1472405950015, value=4
 133                    column=details:tip, timestamp=1472405611320, value=Sizes run small. Ord
                        er 1 size up.
 133                    column=reviewer:location, timestamp=1472405607022, value=USA
 133                    column=reviewer:name, timestamp=1472405601078, value=Helen
 133                    column=summary:product, timestamp=1472405591649, value=vest
 133                    column=summary:rating, timestamp=1472405596526, value=4
3 row(s) in 0.0470 seconds
```

By default, HBase returns the most recent version of data for each cell. Value 5 is not shown in the results.

3. To see multiple versions of your data, issue this command:

```
$ scan 'reviews', {VERSIONS => 2}
```

4. You can also GET the original rating value from row 112 by explicitly specifying the timestamp value. This value will differ on your system, so you will need to substitute the value appropriate for your environment for the timestamp shown below. Consult the output from the previous step to obtain this value.

```
$ get 'reviews', '112', {COLUMN => 'summary:rating', TIMESTAMP =>
1421878110712}
```

Deleting data

1. Delete Tina's name from her review (row 112)

```
$ delete 'reviews', '112', 'reviewer:name'
```

Scan the table to inspect the change.

2. Delete all cells associated with Tina's review (i.e., all data for row 112) and scan the table to inspect the change.

```
$ deleteall 'reviews', '112'
```

Scan the table again to see the results.

About DELETE

DELETE doesn't remove data from the table immediately. Instead, it marks the data for deletion, which prevents the data from being included in any subsequent data retrieval operations. Because the underlying files that form an HBase table (HFiles) are immutable, storage for deleted data will not be recovered until an administrator initiates a major compaction operation. This operation consolidates data and reconciles deletions by removing both the deleted data and the delete indicator.

Browse the Web UI of HBase

You can explore some of the meta data available to you about your table as well as your overall HBase environment using the HBase Web UI. The HBase Master Service Web interface port is 16010. Open the URL http://localhost:16010 in a browser. The port information can be configured in the hbase-site.xml file within the installation directory of HBase, by setting the hbase.master.info.port property.

Dropping a table

Disable the table first, and then drop the table.

```
$ disable 'reviews'
$ drop 'reviews'
```

Try more commands by yourself.

You can find more commands at https://hbase.apache.org/book.html#shell. Try them using the 'reviews' table.

Practice HBase Java API

In this section, we will develop Java programs to interact with HBase databases.

Setting up the Eclipse environment.

1. Open Eclipse, Create a project “Lab5” and create a package “comp9313.lab5” in this project.

2. Right click the project -> Properties -> Java Build Path -> Libraries -> Add Externals JARs -> go to the folder “comp9313/base-1.2.2/lib”, and add all the jar files to the project.

3. Download the “HBaseClient.java” file from the course homepage, and add it into the project. Run the program and see if the correct output can be generated.

```
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/comp9313/hadoop/share/hadoop/common/lib/slf4j-log
SLF4J: Found binding in [jar:file:/home/comp9313/hbase/lib/slf4j-log4j12-1.7.5.jar!/org/s
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Get: keyvalues={row1/data:1/1472409699222/Put/vlen=6/seqid=0}
Scan: keyvalues={row1/data:1/1472409699222/Put/vlen=6/seqid=0}
Scan: keyvalues={row2/data:2/1472409699234/Put/vlen=6/seqid=0}
Scan: keyvalues={row3/data:3/1472409699237/Put/vlen=6/seqid=0}
```

Practice HBase Java API

Use the “HBaseClient.java” as reference to finish the tasks in this section. You will use the HBase Java API to create the table “reviews” again, and to put/get/scan/describe/alter/update the table as you’ve done using the HBase shell. Create a java file “HBasePractice.java” in the package “comp9313.lab5”.

All the HBase classes are described at: https://hbase.apache.org/apidocs/index.html. You can also read https://hbase.apache.org/book.html#datamodel if you meet some problems.

1. Create the table ‘reviews’ with column families: summary, reviewer, and details.

2. Write a function DescribeTable(HTableDescriptor htd) to describe the column families of a table. Call the function after you create the table. You should generate the output like:

```
{NAME => 'details', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', COMPF
{NAME => 'reviewer', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', COMF
{NAME => 'summary', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', COMPF
```

Hint: Use htd.getColumnFamilies() to get an array of HColumnDescriptor objects, and use the toString() function of HColumnDescriptor to print the information.

3. Alter the table to set the IN_MEMORY property of the summary column family to true. Set the number of versions for the summary and reviewer column families to 2. Call DescribeTable() again to see if the changes have been taken into effect.

Hints:

1. Disable the table before altering it, and enable it after altering it.
2. Use the setInMemory() and setMaxVersions() functions of HColumnDescriptor to alter the table.

4. Insert the following records into the table:

```
put 'reviews', '101', 'summary:product', 'hat'
put 'reviews', '101', 'summary:rating', '5'
put 'reviews', '101', 'reviewer:name', 'Chris'
put 'reviews', '101', 'details:comment', 'Great value'
put 'reviews', '112', 'summary:product', 'vest'
put 'reviews', '112', 'summary:rating', '5'
put 'reviews', '112', 'reviewer:name', 'Tina'
put 'reviews', '133', 'summary:product', 'vest'
put 'reviews', '133', 'summary:rating', '4'
put 'reviews', '133', 'reviewer:name', 'Helen'
put 'reviews', '133', 'reviewer:location', 'USA'
put 'reviews', '133', 'details:tip', 'Sizes run small. Order 1 size
up.'
```

5. Write a ScanTable(Table table) function to show the data in the table. You may observe that in HBaseClient.java, the table scan results do not show the value. Your task is to display the information in format of:

Row key/Column Family: Qualifier /Value.

Your code will be like:

```
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
try {
    for (Result scannerResult: scanner) {
        //process scannerResult to print the information
    }
} finally {
    scanner.close();
}
```

You should generate the output like:

```
101/details:comment/Great value
101/reviewer:name/Chris
101/summary:product/hat
101/summary:rating/5
112/reviewer:name/Tina
112/summary:product/vest
112/summary:rating/5
133/details:tip/Sizes run small. Order 1 size up.
133/reviewer:location/USA
133/reviewer:name/Helen
133/summary:product/vest
133/summary:rating/4
```

Hints:

1. Use scannerResult.getNoVersionMap() to get a list of key-value pairs. The data is stored in a NavigableMap<byte[] (key), NavigableMap<byte[] (first), byte[] (second)>> object, in which the key is the column family, first is the qualifier, and second is the value.
   Use the following code to iterate this map object:

```
for (Entry<byte[], NavigableMap<byte[], byte[]>> entry :
map.entrySet()) {
    for (Entry<byte[], byte[]> value :
entry.getValue().entrySet()) {
        //print the data
    }
}
```

2. You can also use Result.listCells() function to do this task.
3. Use Bytes.toString(byte[]) to convert the byte[] array to a string.

6. Update Tina's review (row key 112) to change the rating to '4'. Scan the table again to check if the data is updated successfully.

7. Retrieve the row with row key “101”. Show the result in format of column family:qualifier /value (refer to the example in HBaseClient.java). You should generate output like:

```
details:comment/Great value
reviewer:name/Chris
summary:product/hat
summary:rating/5
```

8. Retrieve the data with row key “112” and column family “summary”. Your output should be like:

```
product/vest
rating/4
```

Hints:

1. After creating the Get object “get” using the row key “112”, add the column family name “summary” to get by: get.addFamily(Bytes.toBytes(“summary”)).
2. Use function Result.getFamilyMap() to obtain a NavigableMap<byte[], byte[]> object. Iterate this object to print the information.

9. Retrieve the cell with row key “133” and column “reviewer:name”.

Hints:

1. After creating the Get object “get” using the row key “112”, add the column family name “reviewer” and qualifier “name” to get by: get.addColumn(Bytes.toBytes(“reviewer”), Bytes.toBytes(“name”)).
2. Use function Result.value() to obtain the value in the cell.

10. Restrict the scan results to retrieve only the contents of the summary column family and the reviewer:name column for row keys starting at '120' and ending at '150'.

11. Delete Tina's name from her review (row 112), and then scan the table.

Hint: use the Delete class. First create a Delete object using the row key “112”, and then specify the column family and qualifier by using the function addColumn(family, qualifier).

Keep working on the problems in Lab 4

The codes of problems in Lab 4 were not published in Week 5 because most of you did not finish Lab 4. Please keep working on the problems in this week.