Lab: Exploring HBase 2

Objective: Use the HBase Shell to populate and explore the `hbase:meta` table.

HBase tables: various

In this exercise you will use the HBase Shell to explore the `hbase:meta` table.

> *Note: you may need to alternaitvely create and drop a table like:*

Create the table ‘iot_data’:

```
create 'iot_data','cf1'
```

Now view the table:

```
list
```

<img src="https://user-images.githubusercontent.com/558905/40613898-7a6c70d6-624e-11e8-9178-7bde851ac7bd.png" align=“left” width=“50” height=“50” title=“ToDo Logo”>
<h4>1. Do the Following</h4>

Data Definition Language

This command will display the currently used HBase version:

```
hbase(main):005:0> version
```

Table Help

Guides you what and how to use table-referenced commands. It will give table manipulations commands
like put, get and all other commands information.

```
hbase(main):006:0> table_help
```

whoami

Shows the current hbase user.

```
hbase(main):007:0> whoami
HBase Table Management commands
```

disable ‘[table name]’

disable_all

Will disable all the tables matching the given regex.

disable_all “matching regex”

Disable all tables with names starting with ‘s’

```
hbase(main):003:0> disable_all ‘s.*’
```

enable

Will enable the named table:

```
enable `[table name]`
```

show_filters

Shows all the filters present in HBase:

```
hbase(main):005:0> show_filters
```

drop

Used to drop the named table. Table must first be disabled:

```
disable ‘[table name]’
drop ‘[table name]’
```

Example:

```
hbase(main):007:0> disable 'iot_data'
hbase(main):007:0> drop 'iot_data'
```

> *Note: create the table again*

drop_all

Used to drop all of the tables matching the given regex

drop_all “matching regex”

Example: drop all tables with names starting with ‘s’

```
hbase(main):009:0> drop_all 'iot_data.*'
```

is_enabled

Will check either the table is enabled or not:

```
is_enabled '[table name]'
```

exists

Checks whether the named table exists or not

```
exists '[table name]'
```

alter

Alters the column family schema. You can pass table name and a dictionary specifying new column family schema.

Examples:

To change the column family from col1 to col2 in a table called table1, use this:

```
hbase(main):010:0> alter 'iot_data', NAME=>'para', VERSION
S=>3
```

You can also add more column families lik below:

```
hbase(main):011:0> alter 'iot_data', {NAME=>'col2', VERSIO
NS=>3},{NAME=>'col3', VERSIONS=>5}
```

And if you want to delete the column name ‘col3’, then:

```
hbase(main):011:0> alter 'iot_data', 'delete' =>' col3'
```

alter_status

Get the status of the alter command, which shows the number of regions of the table that have received the updated schema pass table name

alter_status ‘[table name]’

```
alter_status '[table name]'
```

Data manipulation commands

count

Will retrieve the count of a number of rows in a table. Current count is shown every 1000 rows by default.
Count interval may be optionally specified. Count command will work fast when it is configured with right Cache.

Example

```
hbase(main):01:0> count 'iot_data', INTERVAL => 100000
hbase(main):02:0> count 'iot_data', CACHE=> 1000
hbase(main):03:0> count 'iot_data', INTERVAL =>10, CACHE=>
 1000
```

It’s quite fast when configured with the right CACHE.

```
hbase> count '<tablename>', CACHE => 1000
```

The above count fetches 1000 rows at a time. Set CACHE lower if your rows are big. Default is to fetch one row at a time.

Put

Using this command you can put a cell ‘value’ at specified table/row/column and optionally timestamp coordinates.
To put a cell value into table ‘table1’ at row ‘row1’ under column ‘col1’ marked with the time ‘tsp1’, do:

```
hbase(main):04:0> put 'iot_data', 1001, 'cf1:deviceId', 'S
BS05'
hbase(main):04:0> put 'iot_data', 1001, 'cf1:deviceParamet
er', 'Temperature'
hbase(main):04:0> put 'iot_data', 1001, 'cf1:deviceValue',
 84
hbase(main):04:0> put 'iot_data', 1001, 'cf1:dateTime', '2
018-09-12'
hbase(main):04:0> put 'iot_data', 1002, 'cf1:deviceId', 'S
BS05'
hbase(main):04:0> put 'iot_data', 1002, 'cf1:deviceParamet
er', 'Sound'
hbase(main):04:0> put 'iot_data', 1002, 'cf1:deviceValue',
 84
hbase(main):04:0> put 'iot_data', 1002, 'cf1:dateTime', '2
018-09-12'
hbase(main):04:0> put 'iot_data', 1003, 'cf1:deviceId', 'S
```

```
BS05'
hbase(main):04:0> put 'iot_data', 1003, 'cf1:deviceParamet
er', 'Humidity'
hbase(main):04:0> put 'iot_data', 1003, 'cf1:deviceValue',
 84
hbase(main):04:0> put 'iot_data', 1003, 'cf1:dateTime', '2
018-09-12'
hbase(main):04:0> put 'iot_data', 1004, 'cf1:deviceId', 'S
BS05'
hbase(main):04:0> put 'iot_data', 1004, 'cf1:deviceParamet
er', 'Pressure'
hbase(main):04:0> put 'iot_data', 1004, 'cf1:deviceValue',
 84
hbase(main):04:0> put 'iot_data', 1004, 'cf1:dateTime', '2
018-09-12'
```

Add another column family:

```
alter 'iot_data','cf2'
```

Add more data:

```
hbase(main):04:0> put 'iot_data', 1003, 'cf2:deviceId', 'S
BS06'
hbase(main):04:0> put 'iot_data', 1003, 'cf2:deviceParamet
er', 'Solidity'
hbase(main):04:0> put 'iot_data', 1003, 'cf2:deviceValue',
 187
hbase(main):04:0> put 'iot_data', 1003, 'cf2:dateTime', '2
018-09-12'
hbase(main):04:0> put 'iot_data', 1004, 'cf2:deviceId', 'S
BS08'
hbase(main):04:0> put 'iot_data', 1004, 'cf2:deviceParamet
er', 'Accuracy'
hbase(main):04:0> put 'iot_data', 1004, 'cf2:deviceValue',
 90.12
hbase(main):04:0> put 'iot_data', 1004, 'cf2:dateTime', '2
018-09-12'
```

get

Use get command to get row or cell contents. You can pass table name, row, and optionally a dictionary of column(s),
timestamp, timerange and versions to the command. Examples:

```
hbase(main):05:0> get 'iot_data', 1001, 'cf1'
hbase(main):06:0> get 'iot_data', 1003, 'cf1', 'cf2'
hbase(main):07:0> get 'iot_data', 1003, [‘c1’, ‘c2’]
hbase(main):08:0> get 'iot_data', 1001, {TIMERANGE => [153
7829244568, 1537829244569]}
hbase(main):09:0> get 'iot_data', 1001, {COLUMN => 'cf1:de
viceId'}
hbase(main):10:0> get 'iot_data', 1001, {COLUMN => ['cf1:d
eviceId', 'cf1:deviceParameter', 'cf1:deviceValue']}
```

```
hbase(main):10:0> get 'iot_data', 1001, {COLUMN => ['cf1:deviceId', 'cf2:deviceParameter', 'cf2:deviceValue']}
hbase(main):11:0> get 'iot_data', 1001, {COLUMN => ‘c1’, TIMESTAMP => ts1}
hbase(main):12:0> get 'iot_data', 1001, {COLUMN => ‘c1’, TIMERANGE => [ts1, ts2], VERSIONS => 4}
hbase(main):13:0> get 'iot_data', 1001, {COLUMN => ‘c1’, TIMESTAMP => ts1, VERSIONS => 4}
hbase(main):14:0> get 'iot_data', 1001, {FILTER => “ValueFilter(=, ‘binary:abc’)”}
hbase(main):15:0> get 'iot_data', 1001
```

delete

This command can be used to delete cell value at specified table/row/column and optionally timestamp coordinates.
To delete a cell from ‘table1’ at row ‘row1’ under column ‘col1’ marked with the time ‘ts1’, do

```
hbase(main):16:0> delete 'iot_data', ‘r1’, ‘c1’, ts1
```

deleteall

Deletes all cells in a given row. You can pass a table name, row, and optionally a column
and timestamp to the command. Examples:

```
hbase(main):17:0> deleteall 'iot_data', 1001
hbase(main):18:0> deleteall 'iot_data', 1002, 'cf1'
```

truncate

Disables, drops and recreates the specified table.

```
hbase(main):20:0> truncate ‘iot_data’
```

incr

Increments a cell ‘value’ at specified table/row/column coordinates. To increment a cell value in table ‘t1’
at row ‘r1’ under column ‘c1’ by 1 (can be omitted) or 10 do:

```
hbase(main):21:0> incr 'iot_data', 1003, 'cf1:deviceValue'
hbase(main):22:0> incr 'iot_data', 1003, 'cf1:deviceValue'
, 1
hbase(main):23:0> incr 'iot_data', 1004, 'cf1:deviceValue'
, 10
```

scan

This command scans entire table and displays the table contents. You can pass table name and optionally
a dictionary of scanner specifications.

Scanner specifications may include one or more of: `TIMERANGE`, `FILTER`, `LIMIT`, `STARTROW`, `STOPROW`, `TIMESTAMP`, `MAXLENGTH` etc:

```
hbase(main):24:0> scan  'iot_data'
```

Cluster Replication Commands

add_peer

Add peers to cluster to replicate using this command. Example:

```
hbase(main):01:0> add_peer '1', CLUSTER_KEY=>'localhost:21
81:/hbase/unsecure'
```

remove_peer

Stops the specified replication stream and deletes all the meta information kept about it.

Example:

```
hbase(main):02:0> remove_peer '1'
```

list_peers

Lists all the replication peer clusters:

```
hbase(main):03:0> list_peers
```

enable_peer

Restarts the replication to the specified peer cluster:

```
hbase(main):04:0> enable_peer '1'
```

disable_peer

Stops the replication stream to the specified cluster, but still keeps track of new edits to replicate:

```
hbase(main):05:0> disable_peer '1'
```

start_replication

Restarts all the replication features.

```
hbase(main):06:0> start_replication
```

stop_replication

Tops all the replication feature.

```
hbase(main):07:0> stop_replication
```

> *Note: start/stop replication is only meant to be used in critical load situations.*

Security Commands

The HBase shell has been extended to provide simple commands for editing and updating user permissions.
The following commands have been added for access control list management.

grant

Grants specific rights such as read, write, execute, create and admin on a table for a certain user.
The syntax of grant command is as follows:

```
grant [user] [permissions] [table] [ [column family] [ [co
lumn qualifier] ] ]
```

[permissions] is zero or more letters from the set “RWCA”: READ(‘R’), WRITE(‘W’), CREATE(‘C’), ADMIN(‘A’).
Given below is an example that grants all privileges to a user named corejavaguru.

```
hbase(main):01:0> grant 'iot_data', 'RWXCA'
```

revoke

Used to revoke a user’s access rights. Syntax is as below:

revoke <user> <table> [<column family> [<column qualifier>]]

Example:

```
hbase(main):02:0> revoke 'iot_data'
```

user_permission

Shows all access permissions for the current user for a given table:

```
user_permission <table>
```

Results

You’re getting the hang of it! Not too bad!