ISIT312 Big Data Management

Session 2, 2022

Exercise 6

Introduction to HBase

In this exercise, you will learn how to create, how to load data, how to perform data manipulations, and how to search HBase tables using Hbase shell commands. You will also learn how to use API to process HBase tables.

Be careful when copying the Linux commands in this document to your working Terminal, because it is error-prone. Maybe you should type those commands by yourself.

Prologue
Login to your system and start VirtualBox.

When ready start a virtual machine `ISIT312-BigDataVM-07-SEP-2020`.

(1) How to start Hadoop and HBase server ?

Open a new Terminal window and start Hadoop in the following way.

```
$HADOOP_HOME/sbin/start-all.sh
```

When ready navigate to a folder where you plan to keep HBase scripts from this lab (you may have to create such folder now) and start HBase server in the following way.

```
$HBASE_HOME/bin/start-hbase.sh
```

(2) How to start HBase command shell ?

To start command line interface to HBase process the following command in a Terminal window.

```
$HBASE_HOME/bin/hbase shell
```

For a good start use a command `help` to get a pretty comprehensive help from HBase command Line Interface (CLI). A complete printout of help is listed at the end of this document.

(3) How to use HBase CLI ?

HBase CLI provide several commands to be processed at `hbase main: ...:0>` command prompt. For example, processing of a command `whoami` at the prompt returns the following message.

```
bigdata (auth:SIMPLE)
```

```
    groups: bigdata, adm, cdrom, sudo, dip, plugdev, lpadmin,
sambashare
```

Processing of a command `version` returns the following messages.

```
1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017
```

Processing of a command `table_help` returns information about table-reference commands (see Appendix B in this document).

Command `status` returns the following message.

```
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000
average load
```

(4) How to create a script file and how to process a script file ?

To create HBase CLI script start `gedit` editor, type in the lines

```
status
whoami
```

and save a file as `script.hb`.

To process a script file type the following command at `hbase` prompt:

```
source('script.hb')
```

and press Enter key.

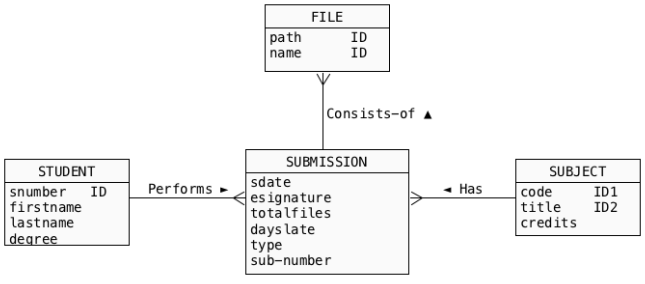
If you would like to process a script and save the results in a file then exit HBase CLI and in a Terminal window process the following command.

```
$HBASE_HOME/bin/hbase shell < script.hb > report.rpt
```

The command starts HBase shall that reads input from a file `script.hb` and outputs the results into a file `report.rpt`.

(5) How to design a HBase table ?

Consider the following conceptual schema.



We would like to implement a conceptual schema given above as a single HBase table. We create four column families: `STUDENT`, `SUBJECT`, `FILE`, and `SUBMISSION` and we distribute the rows over the column families in a way schematically presented below. I would called as "a relational implementation style" ;). We concatenate the names of entity types with the values of identifiers of entity instances and we use it as the keys of rows in HBase table. The submitted files will be represented as column qualifiers. it means that a column family `FILE` will have a variable number of column qualifiers depending on the total number of submitted files.

```
                    STUDENT      SUBJECT        SUBMISSION  FILE
student:snumber1    xxxxxxxxxxx
student:snumber2    xxxxxxxxxxx
…
subject:code1                    xxxxxxxxxxx
subject:code2                    xxxxxxxxxxx
…
submission:skey1    xxx          xxx            xxxxxxxxxxx xxxxxx
submission:skey2    xxx          xxx            xxxxxxxxxxx xxx
…
```

where a submission `skey-n` consists of `snumber-n|code-n|sub-number-n`. The rows that describe submissions spread over all column families. The total number of overlaps determines what we call in the relational model as "denormalization" or simply speaking "redundancies" in HBase table.

(6) How to create a HBase table ?

To create a HBase table we must provide a name of a table and a name of at least one column family. Process the following command to create HBase table `COURSEWORK` with a column family `STUDENT`.

```
create 'COURSEWORK', 'STUDENT'
```

To verify a structure of HBase table use `describe` command.

```
describe 'COURSEWORK'
```

(7) How to add the new column families to an existing HBase table ?

Next, we use `alter` command to add a column family `SUBJECT` to HBase table `COURSEWORK`.

```
alter 'COURSEWORK', {NAME=>'SUBJECT', VERSIONS=>'1'}
```

In our design we plan to represent each entity type as a separate column family. Process the following commands to add the column families `FILE` and `SUBMISSION`.

```
alter 'COURSEWORK', {NAME=>'FILE', VERSIONS=>'2'}
alter 'COURSEWORK', {NAME=>'SUBMISSION', VERSIONS=>'1'}
```

Use `describe` command to verify the structures of HBase table `COURSEWORK`.

```
describe COURSEWORK
```

(8) How to enter data into HBase table ?

First, we shall add few rows to a column family `STUDENT`. We shall use the values of attribute `snumber` prefixed with a string `'student'` as a key of rows in HBase table. The following commands add to a column family a row with key `'student:007'`, and the column qualifiers `snumber`, `fname`, `lname`, `degree`, and `put` the values into cells of HBase table.

```
put 'COURSEWORK','student:007','STUDENT:snumber','007'
put 'COURSEWORK','student:007','STUDENT:first-name','James'
put 'COURSEWORK','student:007','STUDENT:last-name','Bond'
put 'COURSEWORK','student:007','STUDENT:degree','MIT'
```

To verify the insertions use `scan` command.

```
scan 'COURSEWORK'
```

A sequence of `put` commands implements an insertion of a single row with a key `'student:007'` into HBase table `COURSEWORK`.

Now, add all remaining data to a table `COURSEWORK`. The next sequence of put commands inserts the next row into column family `STUDENT`.

```
put 'COURSEWORK','student:666','STUDENT:snumber','666'
put 'COURSEWORK','student:666','STUDENT:firstname','Harry'
put 'COURSEWORK','student:666','STUDENT:lastname','Potter'
put 'COURSEWORK','student:666','STUDENT:degree','BCS'
```

Next, we insert two rows into a column family `SUBJECT`.

```
put 'COURSEWORK','subject:312','SUBJECT:code','312'
put 'COURSEWORK','subject:312','SUBJECT:title','Big Data'
put 'COURSEWORK','subject:312','SUBJECT:credits','6'
put 'COURSEWORK','subject:313','SUBJECT:code','313'
put 'COURSEWORK','subject:313','SUBJECT:title','Very Big Data'
put 'COURSEWORK','subject:313','SUBJECT:credits','12'
```

Next, we insert information about a submission performed by one of the students enrolled in one of the subjects.

```
put 'COURSEWORK','submission:007|312|1','SUBMISSION:sdate','01-APR-2017'
put 'COURSEWORK','submission:007|312|1','SUBMISSION:esignature','jb'
put 'COURSEWORK','submission:007|312|1','SUBMISSION:totalfiles','2'
put 'COURSEWORK','submission:007|312|1','SUBMISSION:dayslate','0'
put 'COURSEWORK','submission:007|312|1','STUDENT:snumbner','007'
put 'COURSEWORK','submission:007|312|1','SUBJECT:code','312'
put 'COURSEWORK','submission:007|312|1','FILE:fnumber1','path/file-name1-1'
put 'COURSEWORK','submission:007|312|1','FILE:fnumber2','path/file-name1-1'
```

Note, that a row with a key `'submission:007|312|1'` contributes to column families `SUBMISSION`, `STUDENT`, `SUBJECT`, `FILE`.

It is possible to "denormalize" the row by inserting the values into the cells labelled with the column qualifiers in `STUDENT` and `SUBJECT` to the rows that describe submissions. For example, the following commands add the first and the last name of a student who performed a submission.

```
put 'COURSEWORK','submission:007|312|1','STUDENT:firstname','James'
put 'COURSEWORK','submission:007|312|1','STUDENT:lastname','Bond'
```

Verify a structure of HBase table after all additions.

```
describe 'COURSEWORK'
```

Add all remaining data.

```
put 'COURSEWORK','submission:007|313|1','SUBMISSION:sdate','02-APR-2017'
put 'COURSEWORK','submission:007|313|1','SUBMISSION:esignature','jb'
put 'COURSEWORK','submission:007|313|1','SUBMISSION:totalfiles','2'
put 'COURSEWORK','submission:007|313|1','SUBMISSION:dayslate','0'
put 'COURSEWORK','submission:007|313|1','SUBMISSION:type','project'
put 'COURSEWORK','submission:007|313|1','STUDENT:snumbner','007'
put 'COURSEWORK','submission:007|313|1','SUBJECT:code','313'
put 'COURSEWORK','submission:007|313|1','FILE:fnumber1','path/file-name3-1'

put 'COURSEWORK','submission:666|312|3','SUBMISSION:sdate','01-APR-2017'
put 'COURSEWORK','submission:666|312|3','SUBMISSION:esignature','hp'
put 'COURSEWORK','submission:666|312|3','SUBMISSION:totalfiles','2'
put 'COURSEWORK','submission:666|312|3','SUBMISSION:dayslate','0'
put 'COURSEWORK','submission:666|312|3','SUBMISSION:type','assignment'
put 'COURSEWORK','submission:666|312|3','STUDENT:snumbner','666'
put 'COURSEWORK','submission:666|312|3','SUBJECT:code','312'
put 'COURSEWORK','submission:666|312|3','FILE:fnumber1','path/file-name1-1'
put 'COURSEWORK','submission:666|312|3','FILE:fnumber2','path/file-name1-1'

put 'COURSEWORK','submission:666|312|4','SUBMISSION:sdate','02-APR-2017'
put 'COURSEWORK','submission:666|312|4','SUBMISSION:esignature','hp'
put 'COURSEWORK','submission:666|312|4','SUBMISSION:totalfiles','2'
put 'COURSEWORK','submission:666|312|4','SUBMISSION:dayslate','0'
put 'COURSEWORK','submission:666|312|4','SUBMISSION:type','assignment'
put 'COURSEWORK','submission:666|312|4','STUDENT:snumbner','666'
put 'COURSEWORK','submission:666|312|4','SUBJECT:code','312'
put 'COURSEWORK','submission:666|312|4','FILE:fnumber1','path/file-name2-1'
put 'COURSEWORK','submission:666|312|4','FILE:fnumber2','path/file-name2-2'

put 'COURSEWORK','submission:666|313|2','SUBMISSION:sdate','02-APR-2017'
put 'COURSEWORK','submission:666|313|2','SUBMISSION:esignature','hp'
put 'COURSEWORK','submission:666|313|2','SUBMISSION:totalfiles','2'
put 'COURSEWORK','submission:666|313|2','SUBMISSION:dayslate','0'
put 'COURSEWORK','submission:666|313|2','SUBMISSION:type','project'
put 'COURSEWORK','submission:666|313|2','STUDENT:snumbner','666'
put 'COURSEWORK','submission:666|313|2','SUBJECT:code','313'
put 'COURSEWORK','submission:666|313|2','FILE:fnumber1','path/file-name3-1'
```

(9) How to replace a row ?

To replace a row use `put` command in the following way. Note, that the replacements happen only for the cells with one version allowed. If some cells allow for more than one version then instead of the replacement a new version of a value in a cell is created.

```
put 'COURSEWORK','student:007','STUDENT:snumber','008'
scan 'COURSEWORK'
```

(10) How to retrieve a row and a cell ?

To retrieve a row we use a row key and a command `get` in the following way.

```
get 'COURSEWORK','student:007'
```

To retrieve the contents of a cell we have to provide a full address of a cell that consists of `row key, column family: column qualifier`.

```
get 'COURSEWORK','student:007','STUDENT:snumber'
```

In the following way we can list up to 5 versions in a cell.

```
get 'COURSEWORK','student:007',{COLUMN=>'STUDENT:snumber',VERSIONS=>5}
```

(11) How to create a new version of a value in a cell ?

Assume that in one of existing submissions a student would like to submit a new version of a file `filenumber1`. A new version can be created because a column family `FILE` was created with value of parameter `VERSIONS` equal to `2` (see below). A new version is created by processing of the following `put` command.

```
alter 'COURSEWORK', {NAME=>'FILE', VERSIONS=>'2'}
```

A new version is created by processing the following `put` command.

```
put 'COURSEWORK','submission:007|313|1','FILE:fnumber1','path/file-name3-3'
```

The results can be verified with `get` command.

```
get 'COURSEWORK','submission:007|313|1',{COLUMN=>'FILE:fnumber1',VERSIONS=>2}
```

(12) How to retrieve many rows from a given column family ?

A `scan` command is equivalent to an operation of projection in the relational data model. It can be used to list the contents of entire HBase table, the contents of select column family, and the contents of cells in a given column family and in a given column qualifier (see below).

```
scan 'COURSEWORK',{COLUMN=>'STUDENT'}
scan 'COURSEWORK',{COLUMN=>'STUDENT:first-name'}
```

(13) How to list the names of created Hbase tables ?

A column list displays the names of Hbase tables created so far.

```
list
```

(14) How to drop Hbase table ?

To drop HBase table we have disable it first with a command disable

```
disable 'COURSEWORK'
```

and then we can drop it with `drop` command.

```
drop 'COURSEWORK'
```

(15) How to truncate Hbase table ?

A command `truncate` can be used to delete the contents of HBASE table without changing its internal structure. The command preserves all column families and column qualifiers within the column families.

```
truncate 'COURSEWORK'
```

(16) How to exit Hbase CLI ?

To exit HBase CLI process at `hbase main: ...:0>` prompt a command `exit`.

```
exit
```

(17) How to use Java API to process HBase tables ?

First we have to set `HBASE_CLASSPATH` variable. Process the following shell commands in a terminal window.

```
HBASE_CLASSPATH="."
for jar in `ls /usr/share/hbase/lib/*.jar`; do
HBASE_CLASSPATH=${HBASE_CLASSPATH}:$jar
done
export HBASE_CLASSPATH
```

Next use `gedit` editor and create a file `ListTables.java` with the following contents.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.HBaseAdmin;

public class ListTables {
    public static void main(String[] args) throws Exception {
        Configuration conf = HBaseConfiguration.create();
        HBaseAdmin admin = new HBaseAdmin(conf);
// Get all the list of tables using HBaseAdmin object
      HTableDescriptor[] tableDescriptor = admin.listTables();

// Display the names of all tables
       System.out.println("HBase tables:");
       for (int i=0; i<tableDescriptor.length;i++ ){
      System.out.println(tableDescriptor[i].getNameAsString());}

    }
}
```

To compile `ListTables.java` use the following command.

```
javac -classpath $HADOOP_CLASSPATH:$HBASE_CLASSPATH ListTables.java
```

To process `ListTables` use the following command.

```
java -cp $HADOOP_CLASSPATH:$HBASE_CLASSPATH ListTables
```

(15) How to use the remaining example of HBase Java API ?

The following simple examples of HBase Java API are available on Moodle.

Creating HBase table	`CreateTable.java`
Inserting a row (cells)	`PutRow.java`
Retrieving a row (cells)	`GetRow.java`
Retrieving entire row with versions	`GetEntireRow.java`
Scanning a table	`ScanTable.java`
Deleting a row	`DelRow.java`
Dropping a table	`DropTable.java`

To compile and to process the remaining example of HBase Java API use `javac` and `java` commands in the same way as for `ListTables.java` example (see yellow highlighted fragments that should be replaced with the appropriate names).

Appendix A

hbase(main):001:0> help
HBase Shell, version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for help on a specific command.
Commands are grouped. Type 'help "COMMAND_GROUP"', (e.g. 'help "general"') for help on a command group.

COMMAND GROUPS:
Group name: general
Commands: status, table_help, version, whoami

Group name: ddl
Commands: alter, alter_async, alter_status, create, describe, disable, disable_all, drop, drop_all, enable, enable_all, exists, get_table, is_disabled, is_enabled, list, locate_region, show_filters

Group name: namespace
Commands: alter_namespace, create_namespace, describe_namespace, drop_namespace, list_namespace, list_namespace_tables

Group name: dml
Commands: append, count, delete, deleteall, get, get_counter, get_splits, incr, put, scan, truncate, truncate_preserve

Group name: tools
Commands: assign, balance_switch, balancer, balancer_enabled, catalogjanitor_enabled, catalogjanitor_run, catalogjanitor_switch, close_region, compact, compact_rs, flush, major_compact, merge_region, move, normalize, normalizer_enabled, normalizer_switch, split, trace, unassign, wal_roll, zk_dump

Group name: replication

Commands: add_peer, append_peer_tableCFs, disable_peer, disable_table_replication, enable_peer, enable_table_replication, list_peers, list_replicated_tables, remove_peer, remove_peer_tableCFs, set_peer_tableCFs, show_peer_tableCFs

Group name: snapshots
Commands: clone_snapshot, delete_all_snapshot, delete_snapshot, list_snapshots, restore_snapshot, snapshot

Group name: configuration
Commands: update_all_config, update_config

Group name: quotas
Commands: list_quotas, set_quota

Group name: security
Commands: grant, list_security_capabilities, revoke, user_permission

Group name: procedures
Commands: abort_procedure, list_procedures

Group name: visibility labels
Commands: add_labels, clear_auths, get_auths, list_labels, set_auths, set_visibility

SHELL USAGE:
Quote all names in HBase Shell such as table and column names. Commas delimit
command parameters. Type <RETURN> after entering a command to run it.
Dictionaries of configuration used in the creation and alteration of tables are
Ruby Hashes. They look like this:

{'key1' => 'value1', 'key2' => 'value2', ...}

and are opened and closed with curley-braces. Key/values are delimited by the
'=>' character combination. Usually keys are predefined constants such as
NAME, VERSIONS, COMPRESSION, etc. Constants do not need to be quoted. Type
'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use
double-quote'd hexadecimal representation. For example:

hbase> get 't1', "key\x03\x3f\xcd"
hbase> get 't1', "key\003\023\011"
hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added.
For more on the HBase Shell, see http://hbase.apache.org/book.html

Appendix B

hbase(main):003:0> table_help
Help for table-reference commands.

You can either create a table via 'create' and then manipulate the table via commands like 'put', 'get', etc.
See the standard help information for how to use each of these commands.

However, as of 0.96, you can also get a reference to a table, on which you can invoke commands.
For instance, you can get create a table and keep around a reference to it via:

```
hbase> t = create 't', 'cf'
```

Or, if you have already created the table, you can get a reference to it:

```
hbase> t = get_table 't'
```

You can do things like call 'put' on the table:

```
hbase> t.put 'r', 'cf:q', 'v'
```

which puts a row 'r' with column family 'cf', qualifier 'q' and value 'v' into table t.

To read the data out, you can scan the table:

```
hbase> t.scan
```

which will read all the rows in table 't'.

Essentially, any command that takes a table name can also be done via table reference.
Other commands include things like: get, delete, deleteall,
get_all_columns, get_counter, count, incr. These functions, along with
the standard JRuby object methods are also available via tab completion.

For more information on how to use each of these commands, you can also just type:

```
hbase> t.help 'scan'
```

which will output more information on how to use that command.

You can also perform the general admin actions directly on a table; things like enable, disable,
flush and drop just by typing:

```
hbase> t.enable
hbase> t.flush
hbase> t.disable
hbase> t.drop
```

Note that after dropping a table, your reference to it becomes useless and further usage
is undefined (and not recommended).

End of exercise 6