1 Overview

Welcome to HBase Machine Practice. It is highly recommended that you practice the **Tutorial: Introduction to HBase** before beginning this assignment. This MP has both java and python templates. Please use whichever you prefer.

2 General Requirements

Please note that our grader runs on a docker container and is **NOT connected to the internet**. Therefore, **no additional libraries are allowed** for this assignment. Also, you will **NOT be allowed to create any file or folder outside the current folder** (that is, you can only create files and folders in the folder that your solutions are in).

3 Set Up HBase

Step 1: Start the "default" Docker machine that you created when following the "Tutorial: Docker installation" in week 4, run:

```
docker-machine env
# follow the instruction to configure your shell: eval $(...)
docker-machine start default
```

Step 2: Download the Dockerfile and related files for this MP, change the current folder, build, and run the docker image, run:

```
git clone https://github.com/UIUC-public/Docker_MP3.git
cd Docker_MP3
docker build -t mp3 .
docker run -it mp3 bin/bash
```

Step 3: start HBase, run

```
start-hbase.sh
```

Java submission

If you choose to do this assignment in Python, skip this part and go to "Python submission**" part below.

1 Requirements

This assignment will be graded based on **JDK 8**

2 Procedures

Step 1: Download the Java templates and change the current folder, run:

```
git clone https://github.com/UIUC-public/MP3_java.git
cd MP3_java
```

Step 2: Set the environment variable CLASSPATH

```
export CLASSPATH=$(hbase classpath)
export CLASSPATH="/MP3_java:${CLASSPATH}"
```

Step 3: Finish the exercises by editing the provided templates files. All you need to do is complete the parts marked with **TODO**. Please note that you are **NOT allowed to import any additional libraries.**

- Each exercise has a Java code template. All you must do is edit this file.
- Each exercise should be implemented in one file only. Multiple file implementation is not allowed.
- The code should be compiled and run on the sample Docker image.
- You should run the parts in the order described below. Otherwise, you might not get the correct results. For example, you should create the tables first and then list them.
- Remember that the output is case sensitive. Furthermore, your application should not remove the table and its values at the end of execution.

step 4: [updated]After you are done with the assignments, please check out the submission instructions under week 11 to submit your files..

Exercise A: **Create HBase Tables**

In this exercise, you will create 2 tables as shown below. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartA.java** .

Table 1: a table named "powers" in HBase with three column families as given below. For simplicity, treat all data as **String** (i.e., don't use any numerical data type).

row key	personal	professional	custom

Table 2: a table named "food" in HBase with two column families as given below. For simplicity, treat all data as **String** (i.e., don't use any numerical data type).

row key	nutrition	taste

Following are the commands we will use to run your file:

```
1  javac TablePartA.java
2  java TablePartA
```

Your output should contain:

```
INFO  [main] client.HBaseAdmin: Created powers
INFO  [main] client.HBaseAdmin: Created food
```

Exercise B: **List HBase Tables**

In this exercise, you will list all the tables created in the previous part. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartB.java** .

Following are the commands we will use to run your file:

```
1  javac TablePartB.java
2  java TablePartB
```

Your output should contain:

```
food
powers
```

Exercise C: **Populate HBase Table with Data**

In this exercise, you will insert data in the “powers“ table created in the previous part A with following schema according to **input.csv**. You can assume that **input.csv** is in the same folder as the java files. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartC.java** .

row key	personal		professional		custom
	hero	power	name	xp	color
row1					

Following are the commands we will use to run your file:

```
javac TablePartC.java
java TablePartC
```

Exercise D: **Read Data**

In this exercise, you will read some of the data in the populated “powers “ table created in part C. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartD.java** .

Read the values for the following row ids in order and the given attributes:

Id: "row1", Values for (hero, power, name, xp, color)

Id: "row19", Values for (hero, color)

Id: "row1", Values for (hero, name, color)

Following is an example of output lines, please use spaces between the words. Your output should contain three lines for this exercise.

```
hero: no, color: green
```

Following are the commands we will use to run your file:

```
javac TablePartD.java
java TablePartD
```

Exercise E: **Scan Data**

In this exercise, you will scan the data in the populated “powers “ table created in part C. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartE.java** .

Following are the commands we will use to run your file:

```
javac TablePartE.java
java TablePartE
```

The output should look similar to below:

```
keyvalues={row1/custom:color/1520644330258/Put/vlen=5/seqid=0, row1/personal:hero/1520644330258/Put/vlen=3/seqid=0, row1/personal:power/152064433025
8/Put/vlen=3/seqid=0, row1/professional:name/1520644330258/Put/vlen=6/seqid=0, row1/professional:xp/1520644330258/Put/vlen=3/seqid=0}
keyvalues={row10/custom:color/1520644330298/Put/vlen=4/seqid=0, row10/personal:hero/1520644330298/Put/vlen=2/seqid=0, row10/personal:power/152064433
0298/Put/vlen=5/seqid=0, row10/professional:name/1520644330298/Put/vlen=7/seqid=0, row10/professional:xp/1520644330298/Put/vlen=2/seqid=0}
keyvalues={row11/custom:color/1520644330301/Put/vlen=5/seqid=0, row11/personal:hero/1520644330301/Put/vlen=2/seqid=0, row11/personal:power/152064433
0301/Put/vlen=3/seqid=0, row11/professional:name/1520644330301/Put/vlen=6/seqid=0, row11/professional:xp/1520644330301/Put/vlen=2/seqid=0}
```

Python submission

If you choose to do this assignment in Java, skip this part and go to **"Java submission" part above

1 Requirements

This assignment will be graded based on **Python 3**

2 Procedures

Step 1: Download the Python templates and change the current folder, run,

```
git clone https://github.com/UIUC-public/MP3_py.git
cd MP3_py

```

Step 2: Start Thrift, run

```
hbase thrift start &
```

note that you may want to press "enter" after below INFO to continue

```
INFO  [main] thrift.ThriftServerRunner: starting TBoundedThreadPoolServer on /0.0.0.0:9090 with readTimeout 60000ms; min worker threads=16, max worker threads=1000, max queued requests=1000
```

step 3: Finish the exercises by editing the provided templates files. Please note that you are **NOT allowed to import any additional libraries.**

- Each exercise has one or more code template. All you must do is edit these files.
- The code will be run on the provided Docker image.

step 5: [updated]After you are done with the assignments, please check out the submission instructions under week 11 to submit your files.

Exercise A: **Create HBase Tables**

In this exercise, you will create 2 tables as shown below. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartA.py** .

Table 1: a table named “powers” in HBase with three column families as given below. For simplicity, treat all data as **String** (i.e., don’t use any numerical data type).

row key	personal	professional	custom

Table 2: a table named “food” in HBase with two column families as given below. For simplicity, treat all data as **String** (i.e., don’t use any numerical data type).

row key	nutrition	taste

We will run **python3 TablePartA.py** .

Your output should contain:

```
INFO  [thrift-worker-0] client.HBaseAdmin: Created powers
INFO  [thrift-worker-0] client.HBaseAdmin: Created food
```

Exercise B: **List HBase Tables**

In this exercise, you will list all the tables created in the previous part. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartB.py** . Command line and the expected output are as below

```
root@2a4c5e229080:/MP3_py# python3 TablePartB.py
 [b'food', b'powers']
```

Exercise C: **Populate HBase Table with Data**

In this exercise, you will insert data in the “powers“ table created in the previous part A with following schema according to **input.csv**. You can assume that **input.csv** is in the same folder as the java files. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartC.py** .

row key	personal		professional		custom
	hero	power	name	xp	color
row1					

We will run **python3 TablePartC.py** .

Exercise D: **Read Data**

In this exercise, you will read some of the data in the populated “powers “ table created in part C. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartD.py** .

Read the values for the following row ids in order and the given attributes:

Id: "row1", Values for (hero, power, name, xp, color)

Id: "row19", Values for (hero, color)

Id: "row1", Values for (hero, name, color)

We will run **python3 TablePartD.py** . Following is an example of output lines, please use spaces between the words. Your output should contain three lines for this exercise.

```
hero: b'no', color: b'green'
```

Exercise E: **Scan Data**

In this exercise, you will scan the data in the populated “powers “ table created in part C. To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartE.py** .

We will run **python3 TablePartE.py** . The output should look similar to below:

```
root@2a4c5e229080:/MP3_py# python3 TablePartE.py
Found: b'row1', {b'professional:name': (b'batman', 1520643194084), b'custom:color': (b'black', 1520643194084), b'personal:hero': (b'yes', 1520643194084), b'professional:xp': (b'100', 1520643194084), b'personal:power': (b'fly', 1520643194084)}
Found: b'row10', {b'professional:name': (b'gandalf', 1520643194151), b'custom:color': (b'grey', 1520643194151), b'personal:hero': (b'no', 1520643194151), b'professional:xp': (b'86', 1520643194151), b'personal:power': (b'think', 1520643194151)}
Found: b'row11', {b'professional:name': (b'catman', 1520643194157), b'custom:color': (b'brown', 1520643194157), b'personal:hero': (b'no', 1520643194157), b'professional:xp': (b'84', 1520643194157), b'personal:power': (b'run', 1520643194157)}
```

✓ Complete