



## 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:

```
1 docker-machine env
2 # follow the instruction to configure your shell: eval $(...)
3 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:

```
1 git clone https://github.com/UIUC-CS498-Cloud/Docker_MP3
2 cd Docker_MP3
3 docker build -t mp3 .
4 docker run -it mp3 bin/bash
```

**Step 3:** start HBase, run

```
1 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:

```
1 git clone https://github.com/UIUC-CS498-Cloud/MP3_java.git
2 cd MP3_java
```



### Step 2: Set the environment variable CLASSPATH

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

```
1 javac TablePartC.java
2 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:

```
1 javac TablePartD.java
2 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:

```
1 javac TablePartE.java
2 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/1520644330258/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/1520644330298/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/1520644330301/Put/vlen=3/seqid=0, row11/professional:name/1520644330301/Put/vlen=6/seqid=0, row11/professional:xp/1520644330301/Put/vlen=2/seqid=0}
```

## Exercise F: A JOIN SQL query in HBase

In this exercise, you will implement a SQL query in Java and run this query in the populated “powers” table created in part C. Of course, You are supposed to write the nested loop that would achieve the same results as if HBase supported SQL and could execute that command. The SQL query is as follows:

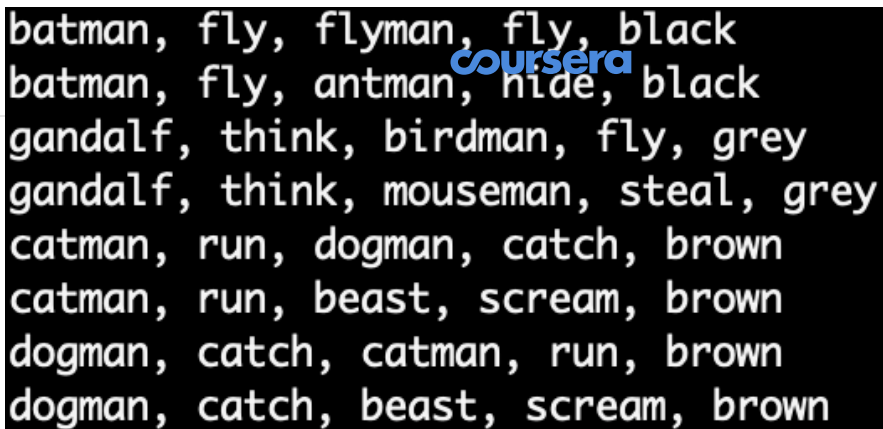
```
SELECT p.name, p.power, p1.name, p1.power, p.color
FROM powers p
INNER JOIN powers p1
    ON p.color = p1.color
    AND p.name <> p1.name;
```

To make the implementation easier, we have provided a boilerplate for this exercise in the following file:

**TablePartF.java**. Don't use any additional libraries. Following are the commands we will use to run your file:

```
1 javac TablePartE.java
2 java TablePartE
```

The output should look similar to below and contains **56** lines in total:



## 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 2.7**

### 2 Procedures

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

```
1 git clone https://github.com/UIUC-CS498-Cloud/MP3_py.git
2 cd MP3_py
3
```

**Step 2:** Start Thrift, run

```
1 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 **python 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@22c76c0f982a:/grader_docker_MP3/MP3Grader# python TablePartB.py
['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.py**.

row key	personal		professional		custom
	hero	power	name	xp	color
row1					

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

```
[root@22c76c0f982a:/grader_docker_MP3/MP3Grader# python TablePartD.py
hero: yes, power: fly, name: batman, xp: 100, color: black
hero: no, color: green
hero: yes, name: batman, color: black
```

## 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 **python TablePartE.py** . The output should look similar to below:

```
root@22c76c0f982a:/grader_docker_MP3/MP3Grader# python TablePartE.py
Found: row1, {'professional:xp': ('100', 1550622365511), 'personal:power': ('fly', 1550622365511), 'personal:hero': ('yes', 1550622365511), 'custom:color': ('black', 1550622365511),
'professional:name': ('batman', 1550622365511)}
Found: row10, {'professional:xp': ('86', 1550622365608), 'personal:power': ('think', 1550622365608), 'personal:hero': ('no', 1550622365608), 'custom:color': ('grey', 1550622365608),
'professional:name': ('gandalf', 1550622365608)}
Found: row11, {'professional:xp': ('84', 1550622365615), 'personal:power': ('run', 1550622365615), 'personal:hero': ('no', 1550622365615), 'custom:color': ('brown', 1550622365615),
'professional:name': ('catman', 1550622365615)}
Found: row12, {'professional:xp': ('47', 1550622365620), 'personal:power': ('catch', 1550622365620), 'personal:hero': ('no', 1550622365620), 'custom:color': ('brown', 1550622365620),
'professional:name': ('dogman', 1550622365620)}
Found: row13, {'professional:xp': ('58', 1550622365626), 'personal:power': ('stink', 1550622365626), 'personal:hero': ('maybe', 1550622365626), 'custom:color': ('white', 1550622365626),
'professional:name': ('eggman', 1550622365626)}
```

## Exercise F: A JOIN SQL query in HBase

In this exercise, you will implement a SQL query in Python and run this query in the populated “powers” table created in part C. Of course HBase does not support SQL queries. Instead, you are supposed to write the nested loop that would achieve the same results as if HBase supported SQL and could execute this command. The SQL query is as follows:

```
SELECT p.name, p.power, p1.name, p1.power, p.color
FROM powers p
INNER JOIN powers p1
    ON p.color = p1.color
    AND p.name <> p1.name;
```

To make the implementation easier, we have provided a boilerplate for this exercise in the following file: **TablePartF.py**. We will run **python TablePartF.py** . **Don't use any additional libraries**. The output should look similar to below and contains **56** lines in total:

```
[root@22c76c0f982a:/grader_docker_MP3/MP3Grader# python TablePartF.py
batman, fly, flyman, fly, black
batman, fly, antman, hide, black
gandalf, think, birdman, fly, grey
gandalf, think, mouseman, steal, grey
catman, run, dogman, catch, brown
catman, run, beast, scream, brown
dogman, catch, catman, run, brown
dogman, catch, beast, scream, brown
eggman, stink, charming, smile, white
eggman, stink, goatman, jump, white
eggman, stink, elisa, freeze, white
fishman, swim, brave, fight, blue
birdman, fly, gandalf, think, grey
birdman, fly, mouseman, steal, grey
flyman, fly, batman, fly, black
flyman, fly, antman, hide, black
sofaman, stay, superman, run, red
sofaman, stay, appleman, eat, red
sofaman, stay, spiderman, fly, red
beast, scream, catman, run, brown
beast, scream, dogman, catch, brown
goblin, hide, shrek, stink, green
goblin, hide, spinachman, grow, green
goblin, hide, robinhood, steal, green
superman, run, sofaman, stay, red
superman, run, appleman, eat, red
superman, run, spiderman, fly, red
```

[Mark as completed](#)