## ! STOP!

## You are required to watch the walkthrough video for this week's assignment. In the video, I guide you through each command, showing both the input and the expected output. Since you are working with new technology for the first time, it’s important not to just run a command

## Week 6 Assignment: Hands-on with HBase

**Objective: Mastering HBase for NoSQL Data Storage and Manipulation**

In this assignment, you will gain hands-on experience with **Apache HBase**, a distributed, scalable, and NoSQL database designed for large-scale data storage. HBase allows you to manage unstructured and semi-structured data, and it excels at handling sparse data with variable schema. You will explore core functionalities such as table creation, data manipulation, and querying in HBase’s interactive shell. You’ll also generate, modify, and retrieve data using HBase’s powerful features like composite row keys and efficient scanning mechanisms.

By the end of this assignment, you will:

* Understand how to create and manage HBase tables.
* Gain experience with basic and advanced data manipulation using HBase commands.
* Explore the concept of composite row keys for optimized data access.
* Generate and modify large datasets programmatically, making use of HBase’s distributed storage.

#### **1. Environment Initialization**

* Start by navigating to the required directory and initiating the Docker containers:

cd dsc650-infra/bellevue-bigdata/hadoop-hive-spark-hbase

docker-compose up -d

* Access the master container:

docker-compose exec master bash

#### **2. Introduction to HBase**

You will begin by accessing the **HBase interactive shell**, where you can issue commands to create and manage HBase tables. The HBase shell provides a command-line interface for interacting with the HBase database.

* Enter the HBase interactive shell:

hbase shell

#### **3. Table Creation and Management**

In this section, you will create an HBase table named students, which will store information about students. This introduces you to the basics of HBase table creation, where each table contains **column families**, such as details in this case.

**Exercise 1:** Create a table named ‘students’ with a column family ‘details’.

create 'students', 'details'

**Deliverable:** Screenshot of the table creation command and its output.

**Exercise 2:** Verify that the table has been created.

list

**Deliverable:** Screenshot of the tables listed in HBase.

#### **4. Data Manipulation in HBase**

You will now add data to the students table using the put command, which inserts rows into an HBase table. This introduces the concept of storing data in key-value pairs, with each student being identified by a unique row key (their ID).

**Exercise 3:** Add data to the ‘students’ table. Let’s assume each student has a unique ID, a first name, and a last name.

put 'students', '1', 'details:firstName', 'John'  
put 'students', '1', 'details:lastName', 'Doe'

**Deliverable:** Screenshot of the commands used to add data and their outputs.

**Exercise 4:** Query the data from the ‘students’ table to retrieve the details of the student with ID ‘1’.

get 'students', '1'

**Deliverable:** Screenshot of the query and its output.

#### **5. Advanced HBase Features: Composite Row Key**

In this section, you’ll create a new table named orders that uses **composite row keys**. Composite row keys allow you to combine multiple fields (like customer ID and order date) to uniquely identify each row, optimizing your ability to query data efficiently.

**Exercise 5:** Create a table named ‘orders’ to store data about customer orders. Assume each order is uniquely identified by a composite key formed by combining the customer ID and order date (in the format YYYYMMDD).

create 'orders', 'orderDetails'

**Exercise 6:** Add sample data to the ‘orders’ table using the composite key:

put 'orders', '101:20230806', 'orderDetails:item', 'Laptop'  
put 'orders', '102:20230806', 'orderDetails:item', 'Smartphone'

**Exercise 7:** Query the ‘orders’ table to retrieve details of all orders placed by the customer with ID ‘101’.

scan 'orders', {STARTROW => '101:', ENDROW => '101:~'}

This command will scan rows starting from ‘101:’ to before ‘101:~’ (tilde ‘~’ is the next ASCII character after colon ‘:’).

**Deliverable:** Screenshot of the commands used to query the data with composite key and their outputs.

#### **6. Data Generation for HBase**

You will now automate the insertion of data into the students table using a Ruby loop. This demonstrates how HBase can handle large amounts of data efficiently, and how programmatic data generation can be applied.

**Exercise 8:** Generate random data for the ‘students’ table.

|  |
| --- |
| (2..100).each do |i|  first\_name = "Student#{i}"  last\_name = "LastName#{i}"  put 'students', "#{i}", 'details:firstName', first\_name  put 'students', "#{i}", 'details:lastName', last\_name  end |

**Exercise 9:** Scan the ‘students’ table to verify data insertion.

scan 'students'

**Deliverable:** Screenshot of the commands used for data generation and their outputs.

**Exercise 10: HBase Data Manipulation**

In this final section, you will modify the data in the students table and practice using various data manipulation techniques, including updating, adding columns, and bulk deletion.

**Tasks:**

1. **Update First Names:**
   * For students with IDs from 2 to 50, change the first name prefix from Student to Scholar. For instance, Student3 should become Scholar3.
2. **Add a Middle Name:**
   * For students with IDs from 51 to 75, add a middle name column under the details column family. The middle name should follow the pattern MidName#{i}.
3. **Modify Last Names:**
   * For students with IDs from 76 to 100, append \_Modified to the last name. So, LastName76 should be updated to LastName76\_Modified.
4. **Bulk Delete:**
   * Delete all the details for students with IDs from 90 to 100.
5. **Data Retrieval:**
   * After all modifications, retrieve and display the details for students with IDs 40, 60, 80, and 90 to verify changes.

**Deliverable:**

* Screenshot of the commands used to implement the tasks above.
* Screenshots of the resulting output from a scan.

## Shutting Down

Ensure all Docker containers are turned off with docker-compose down for each directory. If you’re using google cloud, please shut down your virtual machine to preserve cloud costs.