**Table of Contents**

Lab: Starting an HDP 2.6 Cluster(D)

About This Lab

Lab Steps

Result

Demonstration: Understanding Block Storage(D)

About this Demonstration

Demonstration Steps

Lab: Using HDFS Commands(D)

About this Lab

Lab Steps

Result

Lab: Using WebHDFS(D)

About this Lab

Lab Steps

Result

Demonstration: Putting Files in HDFS with Java (Optional)(D)

About this Demonstration

Demonstration Steps

Lab: Importing RDBMS Data into HDFS(D)

About this Lab

Lab Steps

Result

Lab: Exporting HDFS Data to an RDBMS(D)

About this Lab

Lab Steps

Result

Demonstration: Understanding MapReduce(D)

About this Demonstration

Demonstration Steps

Result

Lab: Running a MapReduce Job(D)

About this Lab

Lab Steps

Result

Demonstration: Understanding Pig(D)

About this Demonstration

Demonstration Steps

Lab: Getting Started with Pig(D)

About this Lab

Lab Steps

Result

Lab: Preparing Data for Hive(D)

About this Lab

Lab Steps

Result

Lab: Understanding Hive Tables(D)

About this Lab

Lab steps

Result

Demonstration: Understanding Partitions and Skew

About this Demonstration

Demonstration Steps

Lab: Analyzing Big Data with Hive

About this Lab

Lab Steps

Result

Demo: Writing a Hive UDF(Optional) (D)

About this Lab

Lab Steps

Result

Demonstration: Computing ngrams (D)

About this Demonstration

Demonstration Steps

Lab: Joining Datasets in Hive

About this Lab

Lab Steps

Result

Lab: Computing ngrams of Emails in Avro Format(D)

About this Lab

Lab Steps

Result



Lab: Advanced Hive Programming

About this Lab

Lab Steps

Result

Lab: Running a YARN Application(Optional)

About this Lab

Lab Steps

Result

Lab: Getting Started with Apache Spark(D)

About this Lab

Lab Steps

Result

Lab: Exploring Spark SQL(D)

About this Lab

Lab Steps

Result

**Lab: Starting an HDP 2.6 Cluster**

**About This Lab**

**Objective:**

To start an HDP cluster in your VM.

**File locations:**

NA

**Successful**

The (local) VM will be running HDP 2.6.

**outcome:**

**Before you begin:**

VMWare should be installed on your machine and the classroom VM should be imported, unless you are using a cloud-hosted VM.

**Related lesson:**

***Introduction to Hadoop and its eco system***

**Lab Steps**

1 ) Start the HDP Sandbox

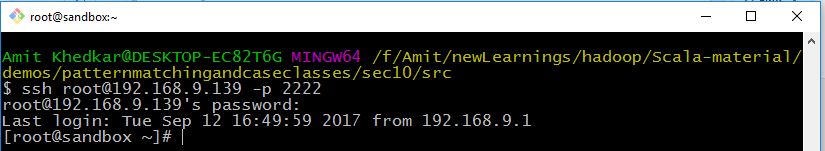
1. Start the sandbox:





b. Connect to the lab environment (Using Putty or git-bash)

**$ ssh root@192.168.9.139 - p 2222**



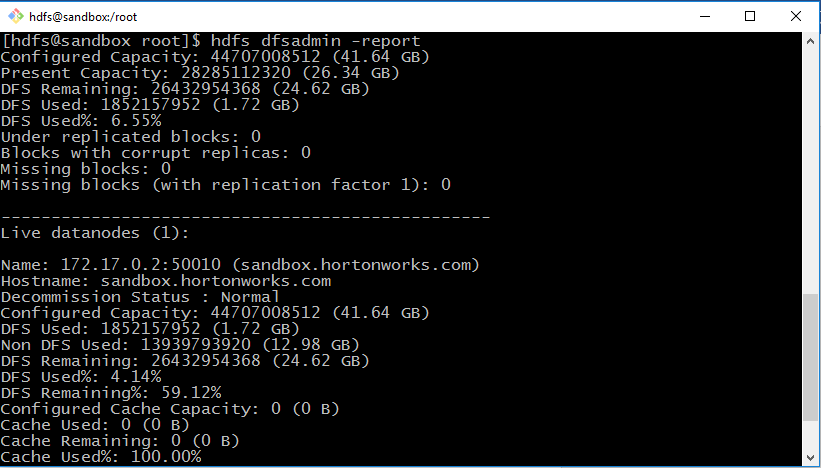
2 ) Verify that the cluster is running

1. To verify HDFS connectivity, run the hdfs dfsadmin -report command. Verify that it provides output similar to the screenshot provided.
2. Switch user as hdfs using command

*$ su hdfs*

Then give following hdfs dfsadmin command





**Note**

The “dfs” in dfsadmin stands for distributed filesystem, and the dfsadmin utility contains administrative commands for communicating with the Hadoop Distributed File System.

* 1. Notice the dfsadmin utility has a -report option, which outputs the current health of your cluster. Enter the following command to view this report:
  2. What is the configured capacity of your distributed filesystem?

\_\_\_\_\_\_\_\_\_\_\_

***Answer:*** Look for the value of “Configured Capacity” at the start of the output.e) What is the present capacity? \_\_\_\_54.51GB\_\_\_\_\_\_\_

***Answer:*** Look for the value of “Present Capacity” at the start of the output.

1. How much of your distributed filesystem is used right now? \_\_\_\_\_43.44GB\_\_\_\_\_\_

***Answer:*** Look for the value of “DFS Used.”

1. What do you think an “Under-replicated block” is?

\_\_\_\_\_\_\_\_\_\_\_1.51%\_\_\_\_\_\_\_\_\_\_\_

***Answer:*** Data in HDFS is chunked into blocks and copied to various nodes in thecluster. If a particular block does not have enough copies, it is referred to as “under replicated.”

1. How many available DataNodes does your cluster have? \_\_\_\_\_1\_\_\_\_\_\_

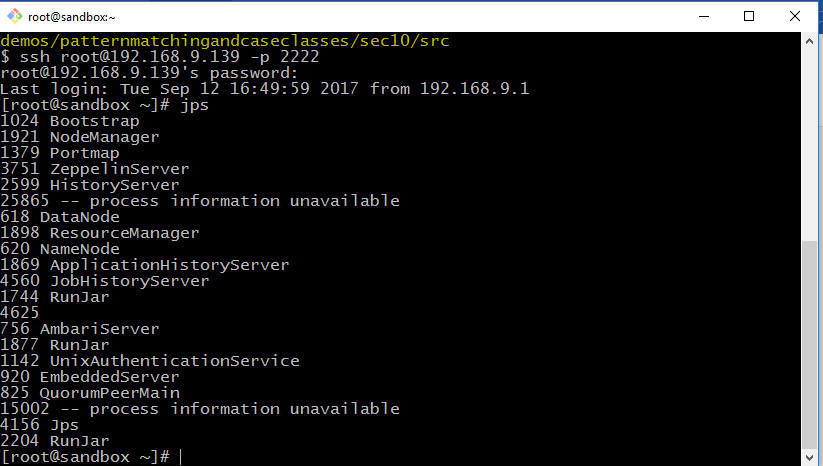
***Answer:*** 1

3 ) View the Processes on the Cluster Nodes

Enter the jps command, which lists all Java processes running on this machine.



While your specific processes and they order they are presented in may look slightly different than the list below, you should still see the NameNode process running:



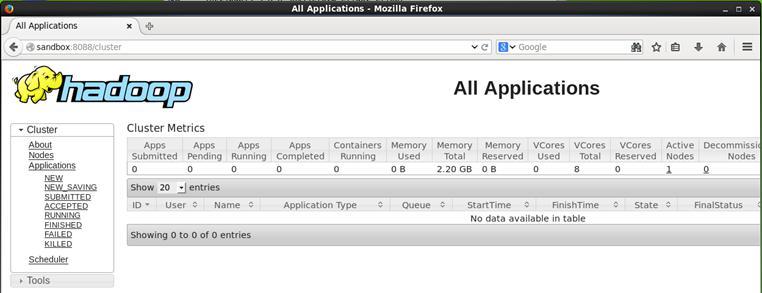
4 ) View the

1. Open Firefox in the VM by double-clicking on the Firefox icon.
2. Enter the following URL:

http:// IP address>:8088/



c) Notice that the URL shows the ResourceManager Web UI:

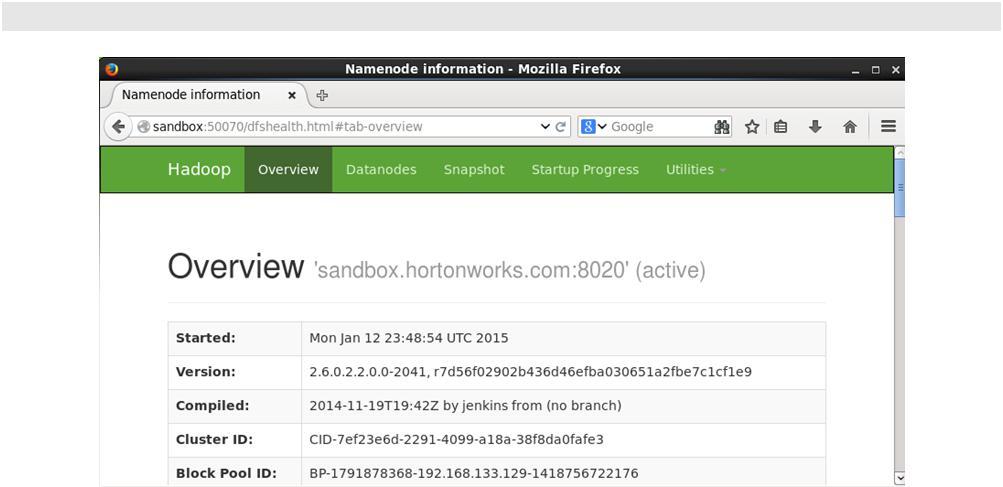


The ResourceManager UI displays information about the applications that have been executed on your Hadoop cluster.



5 ) View the NameNode UI

1. Point your browser to the NameNode UI: [http://<sandbox](about:blank) IP address>:50070/

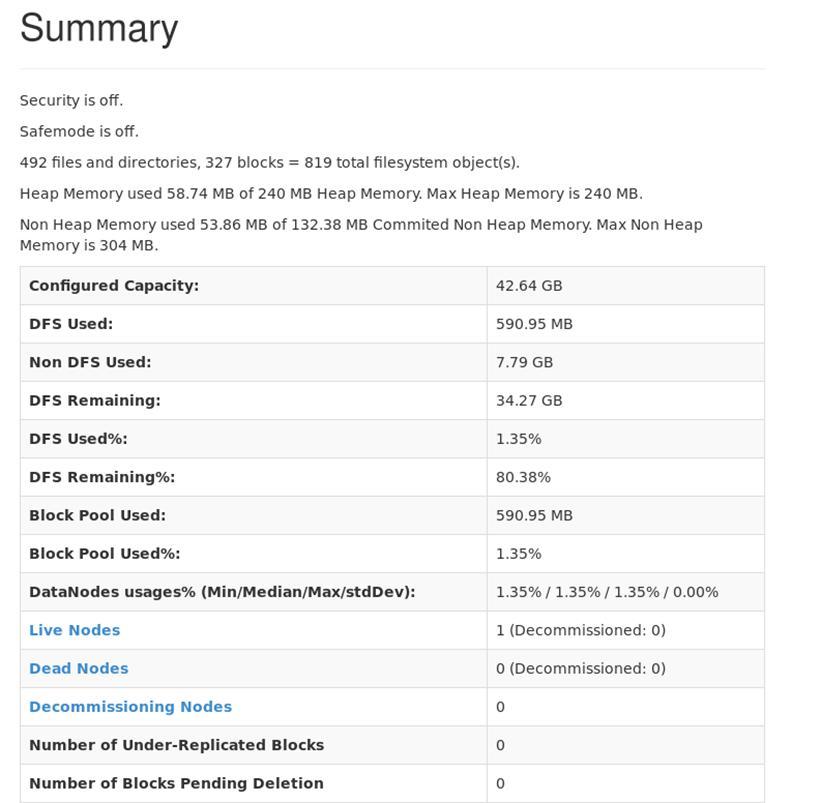


Notice the NameNode UI contains a lot of information about the cluster. The Overview page shows the version of Hadoop and other details.

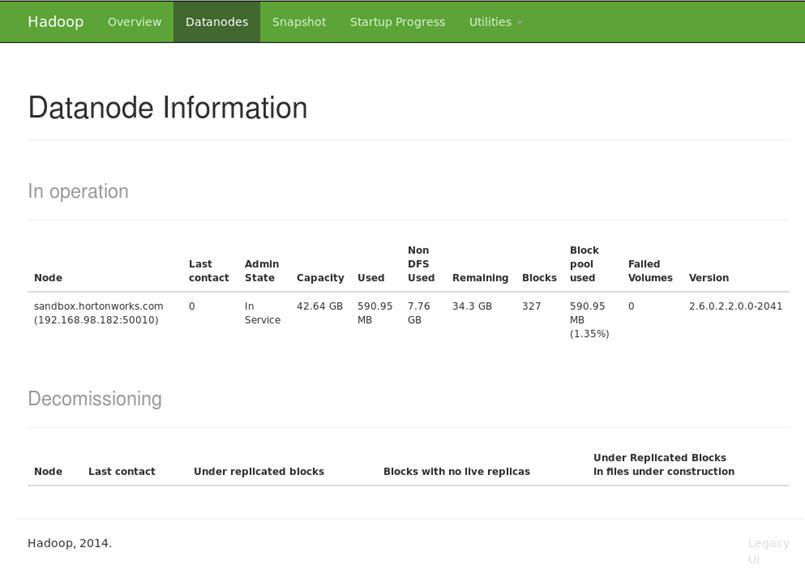
b) Scroll down to the Summary section on the Overview page.



You will see a table that looks similar to the hdfs dfsadmin -report output:



c) Click on the Datanodes tab of the NameNode UI:



You should see one DataNode in your cluster.

6 ) View the JobHistory UI

1. The JobHistory UI is at:

http:// [http://<sandbox](about:blank) IP address>:19888/



As it sounds, the JobHistory UI shows the jobs that have executed on your cluster. You have not submitted any jobs to your cluster yet, but this page comes in handy as you work on the labs throughout this course.

**Result**

You now have a single-node Hortonworks Data Platform 2.6 cluster running in a virtual machine. You will use this cluster to perform the labs in this course.



**Demonstration: Understanding Block Storage**

**About this Demonstration**

|  |  |
| --- | --- |
| **Objective:** | To understand how data is partitioned into blocks and |
|  | stored in HDFS. |
| **During this** | Watch as your instructor performs the following steps. |
| **Demonstration:** | ***The Hadoop Distributed File System (HDFS)*** |
| **Related lesson:** |

**Demonstration Steps**

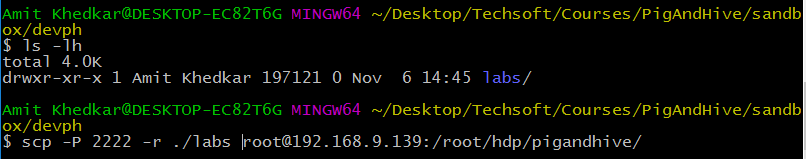
* 1. ) Put the File into HDFS

1. Create a directory pig and hive inside /root/hdp on sandbox

# cd hdp

# mkdir pigandhive

1. Copy a *devph* folder containing the data sets and scripts of this program from local system to /root/hdp/pigandhive on sandbox



2 . **Create directories in HDFS .**

* 1. Create a /user/root directory on HDFS.

# hdfs dfs -mkdir /user/root

* 1. Confirm the creation on HDFS.

# hdfs dfs -ls /user

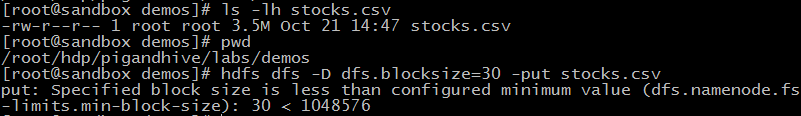
This directory will be your home on hdfs

3. Put the dataset on HDFS

1. Use less to view the contents of the stocks.csv file. Press q when you are finished to exit less.



1. Try putting the file into HDFS with a block size of 30 bytes:



Notice that a size of 30 bytes is not a valid blocksize. The blocksize needs to be at least 1,048,576 according to the dfs.namenode.fs-limits.min-block-size property

1. Try the put again, but use a block size of 2,000,000:



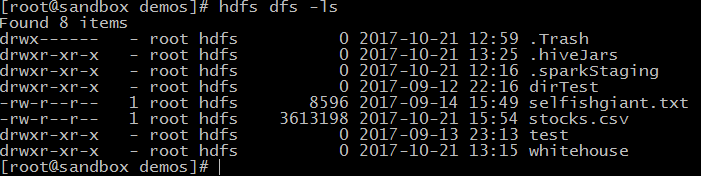
Notice that 2,000,000 is not a valid blocksize because it is not a multiple of 512 (the checksum size)

1. Try the put again, but this time use 1,048,576 for the blocksize



This time the put command should have worked.

1. Use ls to verify that the file is in HDFS in the /user/root folder



.



|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |

2 ) View the Number of Blocks

* 1. Run the following command to view the number of blocks that were created for stocks.csv:



* 1. Notice there are four blocks. Look for the following line in the output:



Total blocks (validated):4 (avg. block size 903299 B)

3 ) Find the Actual Blocks

* 1. Enter the same fsck command as before, but add the -files and -blocks options:

1. hdfs fsck /user/root/stocks.csv -files -blocks

Notice the output contains the block IDs, which are coincidentally the names of the files on the DataNodes.

* 1. Run the command again, but this time add the -locations flag:



1. hdfs fsck /user/root/stocks.csv -files -blocks -locations

Notice in the output that the IP address of the DataNode appear next to each block.

* 1. Change directories to the following:

Try and find the folder that contains the blocks you are looking for and change directories into that folder. The easiest way is to

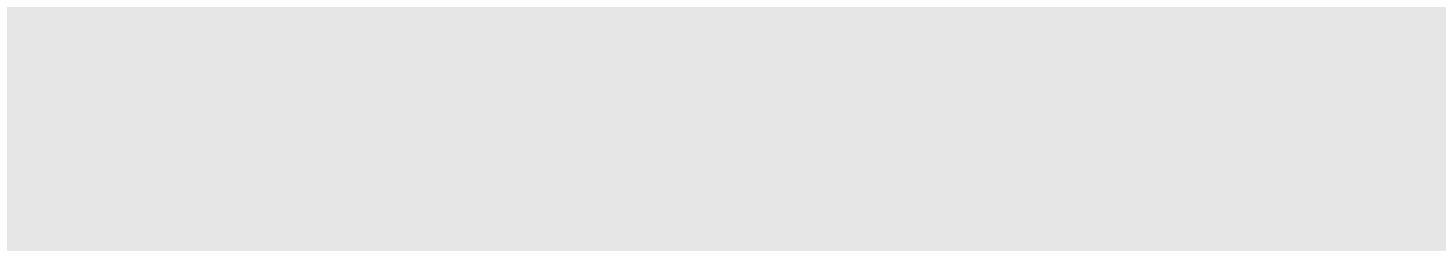




**Important**

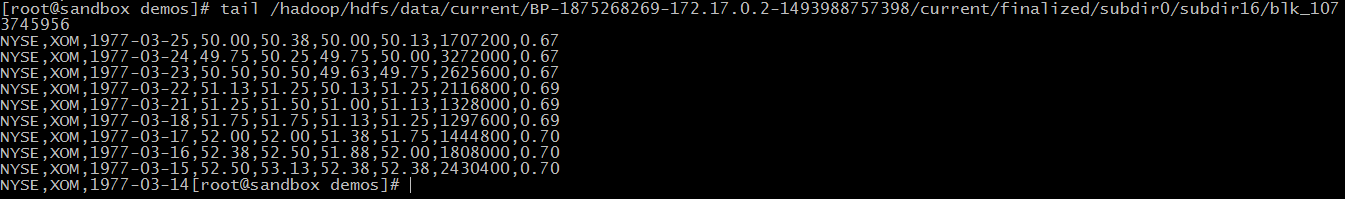
1. Notice that the actual blocks appear in this folder. Look for files that are exactly 1,048,576 bytes. These are three of the blocks.

Notice that the fourth block is smaller: 467,470 bytes.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| -rw-r-- | r-- | 1 | hdfs hdfs 1048576 | | blk\_1073742090 |
| -rw-r-- | r-- | 1 | hdfs hdfs | 8199 | blk\_1073742090\_1266.meta |
| -rw-r-- | r-- | 1 | hdfs hdfs 1048576 | | blk\_1073742091 |
| -rw-r-- | r-- | 1 | hdfs hdfs | 8199 | blk\_1073742091\_1267.meta |
| -rw-r-- | r-- | 1 | hdfs hdfs | 467470 | blk\_1073742093 |
| -rw-r-- | r-- | 1 | hdfs hdfs | 3663 | blk\_1073742093\_1269.meta |

* 1. You can view the contents of a block (although this is not a typical task in Hadoop). Here is the tail of the second block:



Notice the last record in this file is not complete and spills over to the next block, a common occurrence in HDFS.

g. Go back to the home directory.



# cd ~



**Lab: Using HDFS Commands**

**About this Lab**

**Objective:**

To become familiar with how files are added to and removed from HDFS and how to view files in HDFS.

**File locations:**

/root/hdp/pigandhive/devph/labs/Lab2.1

**Successful outcome:**

You will have added and deleted several files and folders in HDFS.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:**

***The Hadoop Distributed File System (HDFS)***

**Lab Steps**

1 ) View the hdfs dfs Command

* 1. Open a Terminal in your VM .
  2. Enter the following command to view the usage of hdfs dfs:

1. hdfs dfs
   1. Notice that the usage contains options for performing filesystem tasks in HDFS, like copying files from a local folder into HDFS, retrieving a file from HDFS, copying and moving files around, and making and removing directories. In this lab, you will perform these commands, and many others, to help you become comfortable with working with HDFS.



2 ) Create a Directory in HDFS

* 1. Enter the following -ls command to view the contents of the user’s root directory in HDFS, which is /user/root

1. hdfs dfs -ls



You do not have any files in /user/root yet, so no output is displayed.

Run the -ls command again, but this time specify the root HDFS folder:



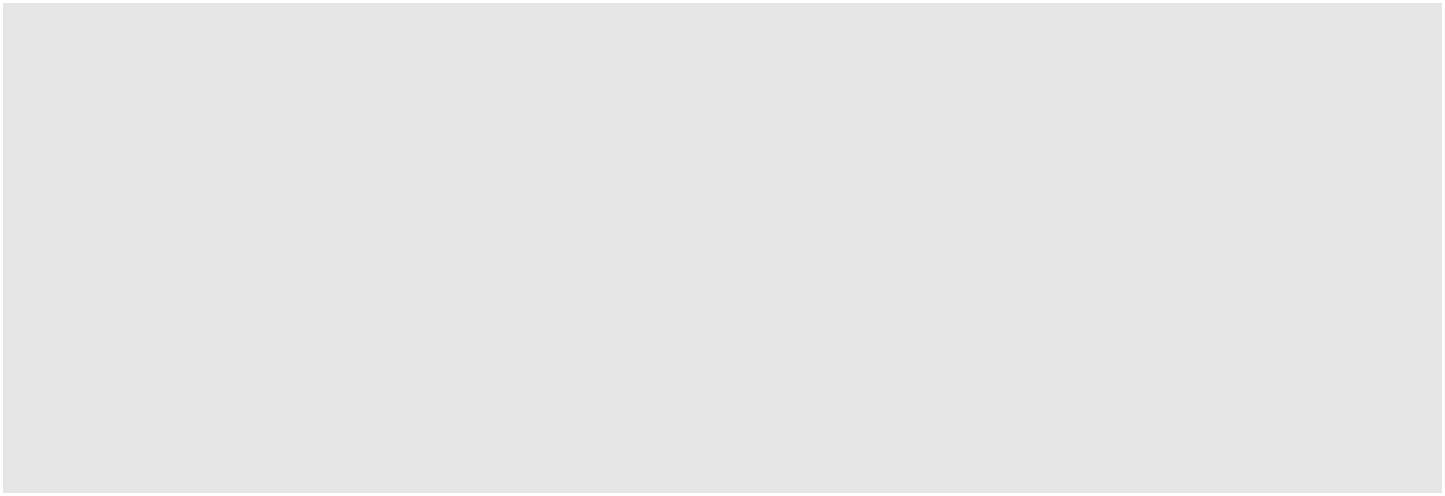
# hdfs dfs -ls /



|  |  |
| --- | --- |
|  |  |

The output should look similar to:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Found 10 items | | hadoop | 0 | 2014-12-16 19:06 /app-logs | |
| drwxrwxrwx | - yarn |
| drwxr-xr-x | - hdfs | hdfs | 0 | 2014-12-16 19:13 /apps | |
| drwxr-xr-x | - hdfs | hdfs | 0 | 2014-12-16 19:48 /demo | |
| drwxr-xr-x | - hdfs | hdfs | 0 | 2014-12-16 19:07 /hdp | |
| drwxr-xr-x | - mapred | hdfs | 0 | 2014-12-16 19:06 /mapred | |
| drwxr-xr-x | - hdfs | hdfs | 0 | 2014-12-16 19:06 /mr- | |
| history | - hdfs | hdfs | 0 | 2014-12-16 | 19:37 /ranger |
| drwxr-xr-x |
| drwxr-xr-x | - hdfs | hdfs | 0 | 2014-12-16 | 19:08 /system |
| drwxrwxrwx | - hdfs | hdfs | 0 | 2014-12-16 | 19:29 /tmp |
| drwxr-xr-x | - hdfs | hdfs | 0 | 2015-01-12 | 05:34 /user |



**Important**

Notice how adding the / in the -ls command caused the contents of the root folder to display, but leaving off the / showed the contents of /user/root, which is the default prefix if you leave off the leading / on any of the hadoop commands (assuming the command is run by the “root” user).

* 1. Enter the following command to create a directory named test in HDFS:

1. hdfs dfs -mkdir test
   1. Verify that the folder was created successfully:
2. hdfs dfs -ls



Found 1 items

drwxr-xr-x - root root 0 test

* 1. Create a couple of subdirectories for test:

1. hdfs dfs -mkdir test/test1
2. hdfs dfs -mkdir -p test/test2/test3



Notice how the -p command can be used to create multiple directories.

The second command above will fail if you omit the -p.

* 1. Use the -ls command to view the contents of /user/root:

1. hdfs dfs -ls



Notice you only see the test directory. To recursively view the contents of a folder, use -ls -R:



# hdfs dfs -ls -R



The output should look like:

|  |  |  |  |
| --- | --- | --- | --- |
| drwxr-xr-x | - root root | 0 | test |
| drwxr-xr-x | - root root | 0 | test/test1 |
| drwxr-xr-x | - root root | 0 | test/test2 |
| drwxr-xr-x | - root root | 0 | test/test2/test3 |



3 ) Delete a Directory

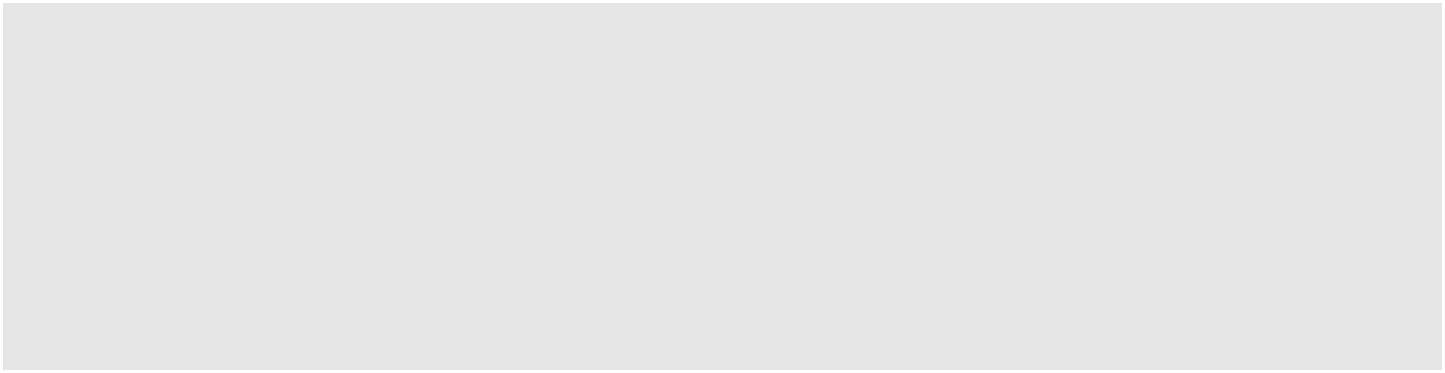
* 1. Delete the test2 folder (and recursively, its subcontents) using the -rm -R command:

1. hdfs dfs -rm -R test/test2
   1. Now run the -ls -R command:
2. hdfs dfs -ls -R



The directory structure of the output should look like:

.Trash



.Trash/Current

.Trash/Current/user

.Trash/Current/user/root

.Trash/Current/user/root/test

.Trash/Current/user/root/test/test2

.Trash/Current/user/root/test/test2/test3 test

test/test1

**Note**

Notice Hadoop created a .Trash folder for the root user and moved the deleted content there. The .Trash folder empties automatically after a configured amount of time.

4 ) Upload a File to HDFS

a. Now let’s put a file into the test folder. Change directories to

/root/hdp/pigandhive/labs/Lab2.1:



1. cd /root/hdp/pigandhive/devph/labs/Lab2.1/
   1. Notice this folder contains a file named data.txt:
2. tail data.txt
   1. Run the following -put command to copy data.txt into the test folder in HDFS:
3. hdfs dfs -put data.txt test/



* 1. Verify that the file is in HDFS by listing the contents of test:

1. hdfs dfs -ls test



The output should look like the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Found 2 items | 1 | root root | 1529355 | test/data.txt |
| -rw-r--r-- |
| drwxr-xr-x | - | root root | 0 | test/test1 |



5 ) Copy a File in HDFS

* 1. Now copy the data.txt file in test to another folder in HDFS using the -cp command:

1. hdfs dfs -cp test/data.txt test/test1/data2.txt
   1. Verify that the file is in both places by using the -ls -R command on test. The output should look like the following:
2. hdfs dfs -ls -R test



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -rw-r--r-- | 1 | root root | 1529355 | test/data.txt |
| drwxr-xr-x | - root root | | 0 | test/test1 |
| -rw-r--r-- | 1 | root root | 1529355 | test/test1/data2.txt |

* 1. Now delete the data2.txt file using the -rm command:

1. hdfs dfs -rm test/test1/data2.txt
   1. Verify that the data2.txt file is in the .Trash folder.



6 ) View the Contents of a File in HDFS

* 1. You can use the -cat command to view text files in HDFS. Enter the following command to view the contents of data.txt:

1. hdfs dfs -cat test/data.txt
   1. You can also use the -tail command to view the end of a file:
2. hdfs dfs -tail test/data.txt



Notice the output this time is only the last 20 rows of data.txt. 7 ) Getting a File from HDFS

* 1. See if you can figure out how to use the get command to copy test/data.txt from HDFS into your local /tmp folder.

***Answer*:**

1. hdfs dfs -get test/data.txt /tmp/
2. cd /tmp
3. ls data\*



8 ) The getmerge Command

1. Put the file /root/hdp/pigandhive/labs/demos/small\_blocks.txt into the test folder

in HDFS. You should now have two files in test: data.txt and small\_blocks.txt.

***Answer:***



1. hdfs dfs -put /root/hdp/pigandhive/labs/demos/small\_blocks.txt test/
   1. Run the following getmerge command:
2. hdfs dfs -getmerge test /tmp/merged.txt
   1. What did the previous command do? Did you open the file merged.txt to see what happened?



***Answer:*** The two files that were in the test folder in HDFS were mergedinto a single file and stored on the local file system.

9 ) Specify the Block Size and Replication Factor

1. Put /root/hdp/pigandhive/labs/Lab2.1/data.txt into /user/root in HDFS, giving it

a blocksize of 1,048,576 bytes.

**Hint**

The blocksize is defined using the dfs.blocksize property on the command line.

***Answer*:**



1. hdfs dfs -D dfs.blocksize=1048576 -put data.txt data.txt
   1. Run the following fsck command on data.txt:
2. hdfs fsck /user/root/data.txt
   1. How many blocks are there for this file?



***Answer:*** The file should be broken down into two blocks.

**Result**

You should now be comfortable with executing the various HDFS commands, including creating directories, putting files into HDFS, copying files out of HDFS, and deleting files and folders.



**Lab: Using WebHDFS**

**About This Lab**

**Objective:** To gain familiarity with some basic operation of WebHDFS

**File locations:** /root/hdp/pigandhive/devph/labs/demos

**Successful outcome:** **You will:** Use Ambari Web UI to verify that WebHDFS is enabled in your

cluster and used WebHDFS API commands to create an HDFS directory, list an HDFS directory, and read a file from HDFS.

**Before you begin** Start and connect to your classroom lab environment

**Related lesson:** ***Inputting data into HDFS***

**Verifying WebHDFS is Enabled**

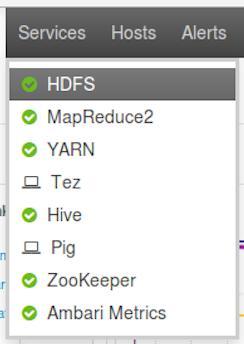
**Use the Ambari Web UI to verify that WebHDFS is enabled in your cluster.**

1 . Open the Firefox browser on the Ubuntu desktop and connect to the Ambari Server at the URL http://<sandbox\_IP>:8080.



2 . Log in to the Ambari Web UI using the user name admin and the password admin.

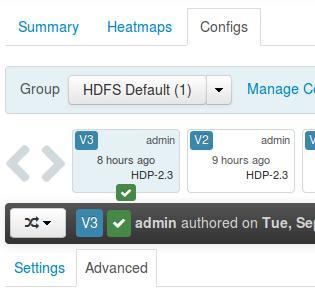
3 . In the Ambari Web UI, click **Services** and select **HDFS** .



Lab: Using WebHDFS

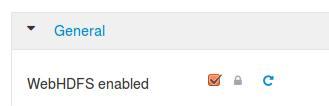
4 . Click **Configs** and then click the **Advanced** tab.

Ensure that the **HFDS Default** configuration group is selected.



5 . Scroll down to find and expand the **General** section and look for the **WebHDFS enabled** check box.

It should already be selected.



6 . You are finished using the Ambari Web UI.

**Using WebHDFS API Commands**

**Use WebHDFS API commands as arguments to the Linux curl command .**

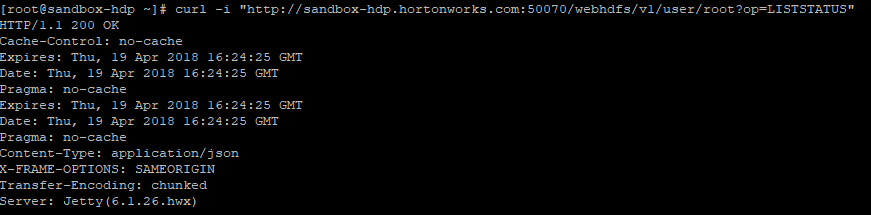
(This can be useful to ensure that WebHDFS is functioning properly.)

Open a Terminal in your VM .

* 1. Use the *curl* command with WebHDFS API arguments to list the contents of the HDFS /user/root directory.

curl -i “http://<NameNode>:50070/webhdfs/v1/user/root?op=LISTSTATUS”





You should see a 200 OK response (you might have to scroll up in the terminal window), along with a JSON object containing the list of files and directories in the /user/root directory.



* 1. Create a new subdirectory named history in /user/root. *The following command is entered on a single line.*

1. curl -i X PUT “http://<NameNode>:50070/webhdfs/v1/user/root/history?op=MKDIRS&user.name=root”

The user.name=root argument is required to have the necessary permissions to write to the /user/root directory.

* 1. Use the HDFS Shell *–ls* command to verify that the history subdirectory was created successfully.

1. hdfs dfs -ls



You should see a history directory in the output.

Read the contents of the stocks.csv file using the following *curl* command and WebHDFS API.

1. curl -i -L “http://node1:50070/webhdfs/v1/user/root/stocks.csv?op=OPEN”



The contents of the file may scroll out of the terminal window.

Create a new file named as small\_blocks.txt on /user/root on HDFS

Change to a directory /root/hdp/pigandhive/devph/labs/demos



cd /root/hdp/pigandhive/devph/labs/demos



curl -i -X PUT small\_blocks.txt -L "http:<Namenode>:50070/webhdfs/v1/user/root/small\_blocks.txt?op=CREATE&user.name=root"

Use the HDFS Shell *–ls* command to verify that the small\_blocks.txt get created successfully under /user/root

**R e s u l t**

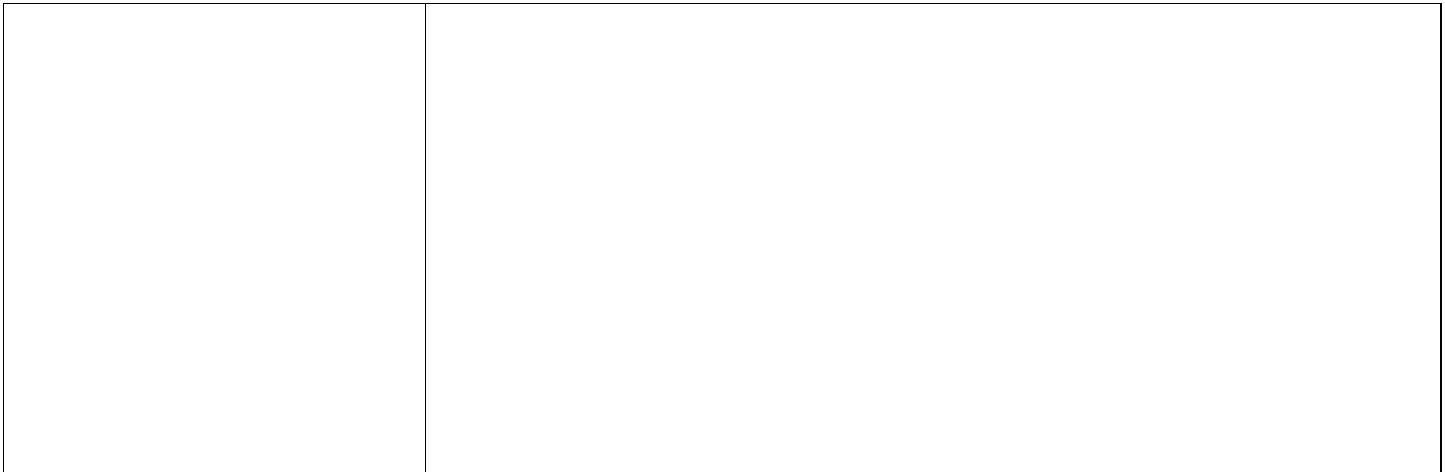
**You have now:**

Used Ambari Web UI to verify that WebHDFS is enabled in your cluster and used WebHDFS API commands to create an HDFS directory, list an HDFS directory, and read a file from HDFS.

**Demo: Putting Files in HDFS with Java**

**Use the HDFS API to put files into HDFS**

*Table 3. About this Lab*



|  |  |  |
| --- | --- | --- |
| **Objective:** |  | Use the HDFS API to put files into HDFS. |
|  |  |  |
| **File locations:** |  | /root/hdp/Java/labs/workspace/Lab1.2 |
|  |  |  |
| **Successful outcome:** |  | Four text files copied from your local file system to a new directory |
|  |  | in HDFS named counties. |
|  |  |  |
| **Before you begin** |  | Open Eclipse in your classroom VM if you do not have it open |
|  |  | already. Make sure that build.gradle is at F:/Amit |
|  |  |  |
| **Related lesson:** |  | Understanding Hadoop and HDFS |
|  |  |  |

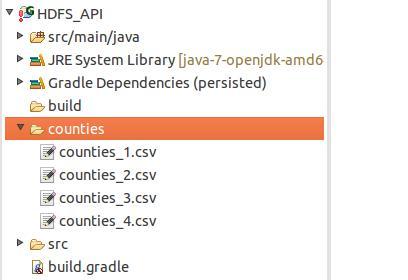
**Perform the following steps:**

**Step 1:** View the Project

1. Copy Techsoft\Courses\Java\sandbox\workspace\HDFS\_API to Techsoft\Courses\Java\eclipse-workspace

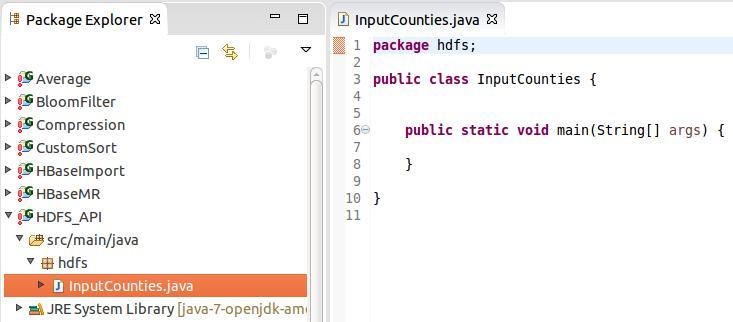
**1.1.** In Eclipse, import a gradle project HDFS\_API and expand theHDFS\_APIproject folder.

**1.2.** Notice the project has a subfolder namedcountiesthat contains four text files:



These files contain population information from U.S. censuses dating back to 1930. In this lab, you are going to put these files into HDFS in a new folder named counties.

**1.3.** Notice there is a package insrc/main/javanamedhdfsthat contains a classnamed InputCounties. Open this file and you will notice it simply contains the main method:



**Step 2:** Create a New Directory in HDFS

**2.1.** Withinmain, instantiate a newConfigurationobject (from theorg.apache.hadoop.conf package).

**2.2.** Use thestatic getmethod ofFileSystemto obtain a reference to the HDFSfilesystem. Pass in the Configuration object from the previous step. You can add a try/catch block or simply have main declare IOException.

**2.3.** Instantiate a newPathobject for a path named “counties”:

**Path counties = new Path("counties");**

**2.4.** If the counties path does not exist already, create it using themkdirsmethod ofFileSystem.

**Step 3:** Put the Text Files in HDFS

**3.1.** The four files on your local file system that you need to put into HDFS are namedcounties\_x.csv, where x is 1, 2, 3 and 4. Start by writing a for loop that iterates four times.

**3.2.** Create aStringto represent the filename:

**String filename = "counties\_" + x + ".csv";**

**3.3.** Create aPathobject for the local filename. You are going run this program from theproject folder, so make the path relative to there:

**Path localSrc = new Path("counties/" + filename);**

(You can alternately use the full path to counties, which would make more sense but requires more typing.)

**3.4.** Similarly, create aPathobject for the destination filename. Keep the filenames thesame in HDFS as they are on the local filesystem:

**Path dest = new Path("counties/" + filename);**

**3.5.** Use thecopyFromLocalFilemethod ofFileSystemto copy the file into HDFS.

**3.6.** Save your changes to InputCounties.java.

**Step 4:** Build the Application

**4.1.** Select theHDFS\_APIproject name in the Project Explorer window and Run Gradle Task, *clean* under build Gradle Tasks.

**4.2.** Select theHDFS\_APIproject name in the Project Explorer window and Run Gradle Task, *build* under build Gradle Tasks.

**Step 5:** Run the Program

**5.1.** Create a folder *workspace* under /root/hdp/Java/labs

**5.2.** Create a folder *Lab1.2* under hdp/Java/labs

**5.3.** SCP the jar and counties folder from development workspace to Lab1.2 folder on sandbox

**5.4.** Run theInputCountiesprogram using the following command:

1. **yarn jar inputcounties.jar hdfs.InputCounties**

**Step 6:** Verify the Results

**6.1.** Let’s look in HDFS and see if the files are in there. View the contents ofcounties:

1. **hdfs dfs -ls counties**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Found 4 items** | | **1** | **root root** | | **62890** | **counties/counties\_1.csv** |
| **-rw-r--** | **r--** |
| **-rw-r--** | **r--** | **1** | **root root** | | **58423** | **counties/counties\_2.csv** |
| **-rw-r--** | **r--** | **1** | **root** | **root** | **44294** | **counties/counties\_3.csv** |
| **-rw-r--** | **r--** | **1** | **root** | **root** | **54237** | **counties/counties\_4.csv** |



You should see that /user/root/counties folder has four files in it.

**Results:** You have now seen two ways to get data into HDFS: the- putcommand from thecommand line, or by using the HDFS API in a Java program. You will use both the dividends data and the counties data in upcoming labs later in the course.

**Lab: Importing RDBMS Data into HDFS**

**About this Lab**

**Objective:**

Import data from a database into HDFS.

**File locations:**

/root/hdp/pigandhive/devph/labs/Lab3.1/

**Successful outcome:**

You will have imported data from MySQL into folders in HDFS.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:**

***Inputting Data into HDFS***

**Lab Steps**

1 ) Create a Table in MySQL

1. If not already done, open a Terminal in your VM and type "ssh sandbox".
2. From the command prompt, change directories to

/root/devph/labs/Lab3.1/:



1. cd ~/devph/labs/Lab3.1/

View the contents of salaries.txt:

1. tail salaries.txt



copy salaries.txt to /tmp salaries.txt:



1. Lab3.1> cp salaries.txt /tmp

The comma-separated fields represent a gender, age, salary, and zip code.

Open mysql prompt by giving following command

Lab3.1> mysql -u root -p

A prompt for password appear on the console, give password as *hadoop*

Create database test in mysql

mysql>CREATE DATABASE test;

Give following commands on above sql prompt (Please refer salaries.sql)

* 1. Notice that there is a salaries.sql script that defines a new table in MySQL named salaries. For this script to work, you need to copy salaries.txt to the /var/lib/mysql-files/ directory:

1. cp salaries.txt /var/lib/mysql-files/
   1. Now run the my-sql commands in salaries.sql script on sql prompt by copying each statement on mysql prompt: The commands are as given below using the following command:





Mysql>use test;

Mysql>drop table if exists salaries;

Mysql>create table salaries (

gender varchar(1),

age int,

salary double,

zipcode int);

Mysql>load data infile '/var/lib/mysql-files/salaries.txt' into table salaries fields terminated by ',';

Mysql>alter table salaries add column `id` int(10) unsigned primary KEY AUTO\_INCREMENT;

2 ) View the Table

a. To verify that the table is populated in MySQL, open the mysql prompt:



# mysql -u root -p



A prompt for password appear on the console give password as *hadoop*

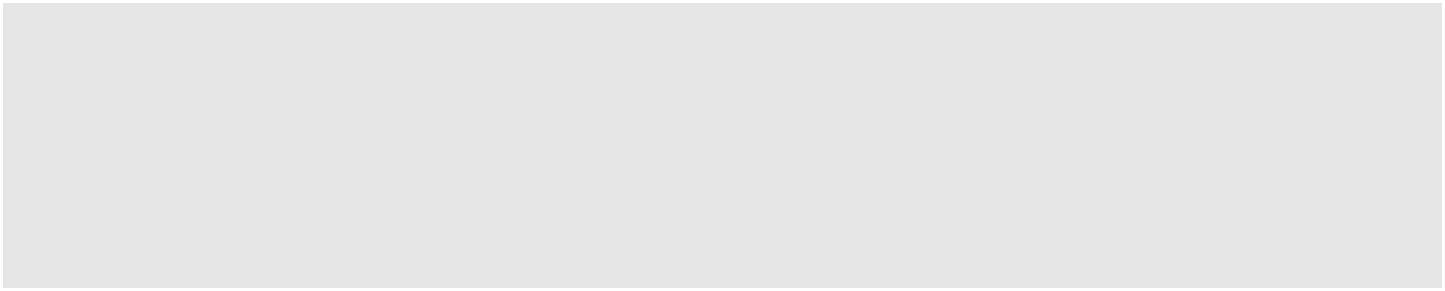
1. Switch to the test database, which is where the salaries table was created:

mysql> use test;

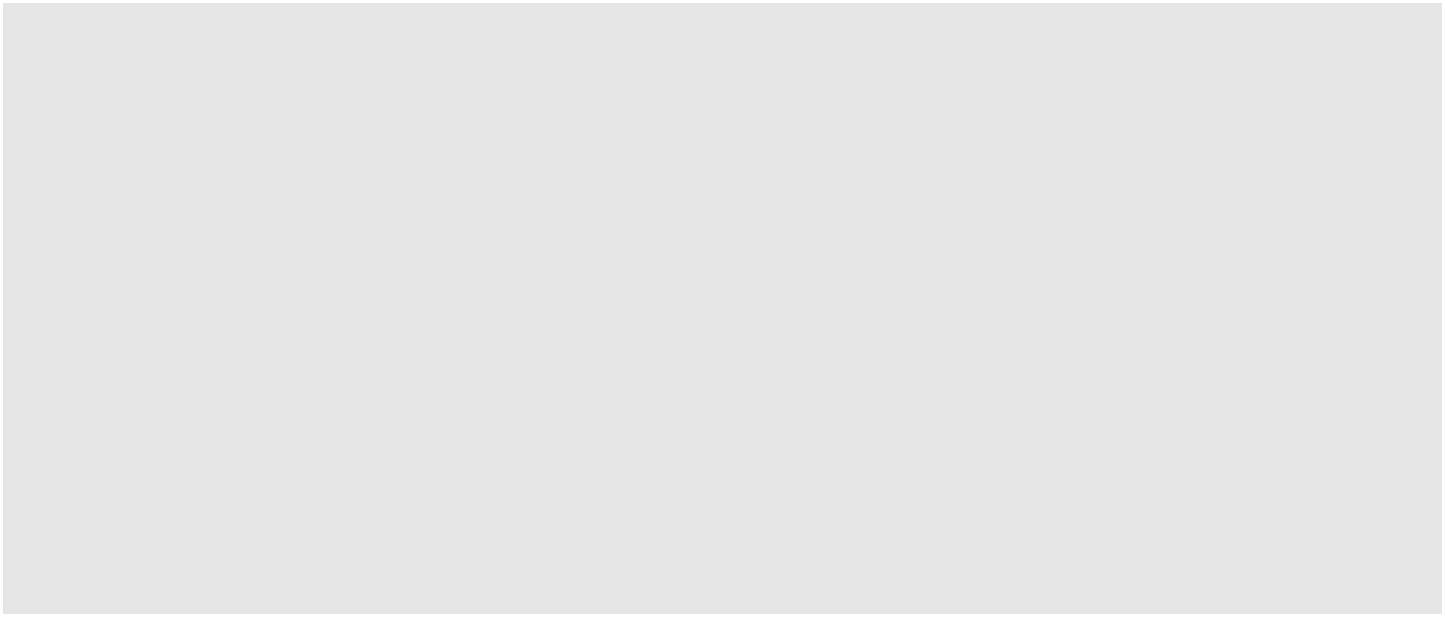


c. Run the show tables command and verify that salaries is defined:

mysql> show tables;



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| + | ---------------- | |  | + |  |  |  |  |  |  |
| | Tables\_in\_test | | | | | |  |  |  |  |  |  |
| + | ---------------- | |  | + |  |  |  |  |  |  |
| | salaries | | |  | | |  |  |  |  |  |  |
| + | ----------------rowinset | | | + | sec) |  |  |  |  |  |
| 1 |  |  | (0.00 | |  |  |  |  |  |
|  |  | d. Select 10 items from the table to verify that it is populated: | | | | | | | | |
| mysql> select \* from salaries limit 10; | | | | | | | | | | + |
| + | -------- | +------ | | +-------- | | + | --------- | + | ---- |
| | gender | age | | | | | salary | zipcode | | | | | id | | | |
| + | -------- | +------ | 66 | +-------- | 41000 | + | ---------95103 | + | ----1 | + |
| | F | | | | | | | | | | | |
| | M | | | | 40 | | | 76000 | | | 95102 | | | 2 | | |
| | F | | | | 58 | | | 95000 | | | 95103 | | | 3 | | |
| | F | | | | 68 | | | 60000 | | | 95105 | | | 4 | | |
| | M | | | | 85 | | | 14000 | | | 95102 | | | 5 | | |
| | M | | | | 14 | | | 0 | | | 95105 | | | 6 | | |
| | M | | | | 52 | | | 2000 | | | 94040 | | | 7 | | |
| | M | | | | 67 | | | 99000 | | | 94040 | | | 8 | | |
| | F | | | | 43 | | | 11000 | | | 94041 | | | 9 | | |
| | F | | | | 37 | | | 65000 | | | 94040 | | 10 | | | |
| + | -------- | +------ | | +-------- | | + | --------- | + | ---- | + |



1. Exit the mysql prompt:

mysql> exit

3 ) Import the Table into HDFS



* 1. Enter the following Sqoop command (all on a single line), which imports the salaries table in the test database into HDFS:

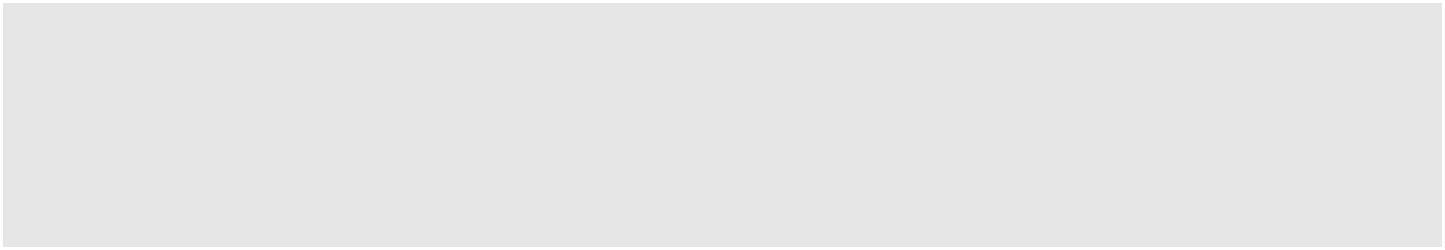
1. sqoop import --connect jdbc:mysql://<hostname>:3306/test --driver com.mysql.jdbc.Driver --username root -password hadoop --table salaries
   1. A MapReduce job should start executing, and it may take a couple of minutes for the job to complete.



4 ) Verify the Import

* 1. View the contents of your HDFS folder:

1. hdfs dfs -ls
   1. You should see a new folder named salaries. View its contents:
2. hdfs dfs -ls salaries



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Found 4 items | | 1 | root hdfs | 272 | salaries/part-m-00000 |
| -rw-r-- | r-- |
| -rw-r-- | r-- | 1 | root hdfs | 241 | salaries/part-m-00001 |
| -rw-r-- | r-- | 1 | root hdfs | 238 | salaries/part-m-00002 |
| -rw-r-- | r-- | 1 | root hdfs | 272 | salaries/part-m-00003 |

* 1. Notice there are four new files in the salaries folder named part-m-0000x. Why are there four of these files?

***Answer*:**The MapReduce job that executed the Sqoop command usedfour mappers, so there are four output files (one from each mapper).

* 1. Use the cat command to view the contents of the files. For example:

1. hdfs dfs -cat salaries/part-m-00000



Notice the contents of these files are the rows from the salaries table in MySQL. You have now successfully imported data from a MySQL database into HDFS. Notice that you imported the entire table with all of its columns. Next, you will import only specific columns of a table.

5 ) Specify Columns to Import

* 1. Using the --columns argument, write a Sqoop command that imports the salary and age columns (in that order) of the salaries table into a directory in HDFS named salaries2. In addition, set the -m argument to 1 so that the result is a single file.

***Solution*:**The command you enter in the command line will look likethis in the terminal window:

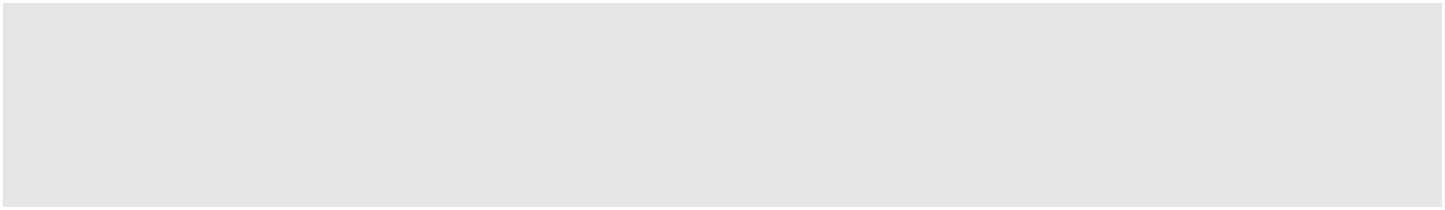


1. sqoop import --connect jdbc:mysql://<hostname>/test --driver com.mysql.jdbc.Driver --username root -password hadoop --table salaries --columns salary,age -m 1 --target-dir salaries2

*Important*

*To make it easier to read, following is the same command as above, however we have broken it down into smaller chunks separated by a "\" at the end of the break point in each line. When you see this formatting in the lab, you should type it out as it appears above, and do not enter the \ characters unless specifically instructed to do so.*





sqoop import --connect\

jdbc:mysql://<hostname>:3306/test –driver \ com.mysql.jdbc.Driver –-username root \

--table salaries \

--columns salary,age \

-m 1 \

--target-dir salaries2

b. After the import, verify you only have one part-m file in salaries2:

1. hdfs dfs -ls salaries2

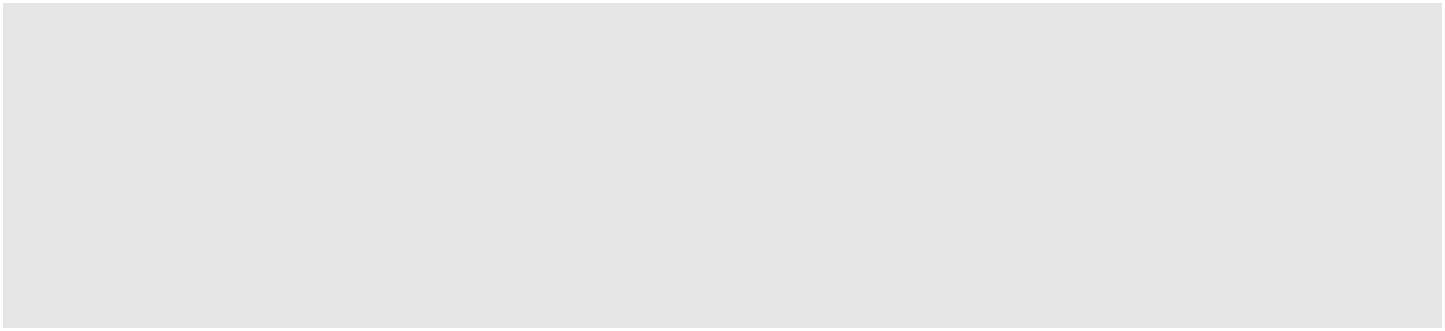


Found 1 items

-rw-r--r-- 1 root hdfs 482 salaries2/part-m-00000

* 1. Verify that the contents of part-m-00000 are only the two columns you specified:

1. hdfs dfs -cat salaries2/part-m-00000



The last few lines should look like the following:

69000.0,97

91000.0,48

0.0,1

48000.0,45

3000.0,39

14000.0,84

6 ) Importing from a Query

Write a Sqoop import command that imports the rows from salaries in MySQL whose salary column is greater than 90,000.00.

***a.*** Use gender as the--split-byvalue, specify only two mappers, andimport the data into the salaries3 folder in HDFS.

**Tip**

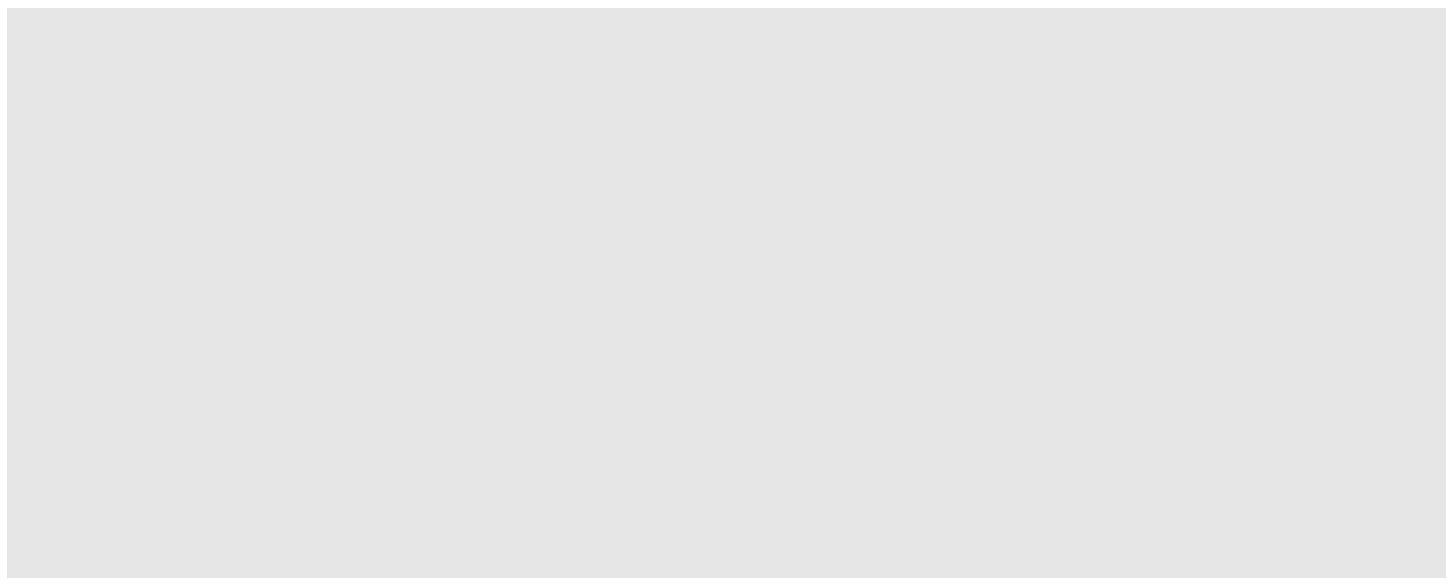
The Sqoop command will look similar to the ones you have been using throughout this lab, except you will use --query instead of -- table. Recall that when you use a -- query command you must also define a -- split-by column, or define -m to be 1.

Also, do not forget to add $CONDITIONS to the WHERE clause of your query, as demonstrated earlier in this unit.



***Solution*:**

In the command below, the "\" at the beginning of line 3 just in front of $CONDITIONS" **is** part of the actual command and is required for it to function properly. All other \ symbols in the command should be ignored in the command line.



sqoop import -Dorg.apache.sqoop.splitter.allow\_text\_splitter=true \

--connect jdbc:mysql://<hostname>:3306/test \

--driver com.mysql.jdbc.Driver --username root --password hadoop \

--query "select \* from salaries s where s.salary > 90000.00 and \$CONDITIONS" \

--split-by gender -m 2 --target-dir salaries3

The actual command on the console will be as shown below

sqoop import -Dorg.apache.sqoop.splitter.allow\_text\_splitter=true --connect jdbc:mysql://sandbox.hortonworks.com:3306/test --driver com.mysql.jdbc.Driver --username root --password hadoop --query "select \* from salaries s where s.salary > 90000.00 and \$CONDITIONS" --split-by gender -m 2 --target-dir salaries3

* 1. To verify the result, view the contents of the files in salaries3. You should have only two output files.

1. hdfs dfs -ls salaries3
   1. View the contents of part-m-00000 and part-m-00001.
2. hdfs dfs -cat salaries3/part-m-00000
3. hdfs dfs -cat salaries3/part-m-00001



Notice that one file contains females, and the other file contains males.

Why?

***Answer:*** You used gender as the split-by column, so all records withthe same gender are sent to the same mapper.

1. Verify that the output files contain only records whose salary is greater than 90,000.00.

**Result**

You have imported the data from MySQL to HDFS using the entire table, specific columns, and also using the result of a query.



**Lab: Exporting HDFS Data to an RDBMS**

**About this Lab**

**Objective:** Export data from HDFS into a MySQL table using Sqoop.

**File locations:** /root/devph/labs/Lab3.2

**Successful outcome:** The data in salarydata.txt in HDFS will appear in a table inMySQL named salary2.

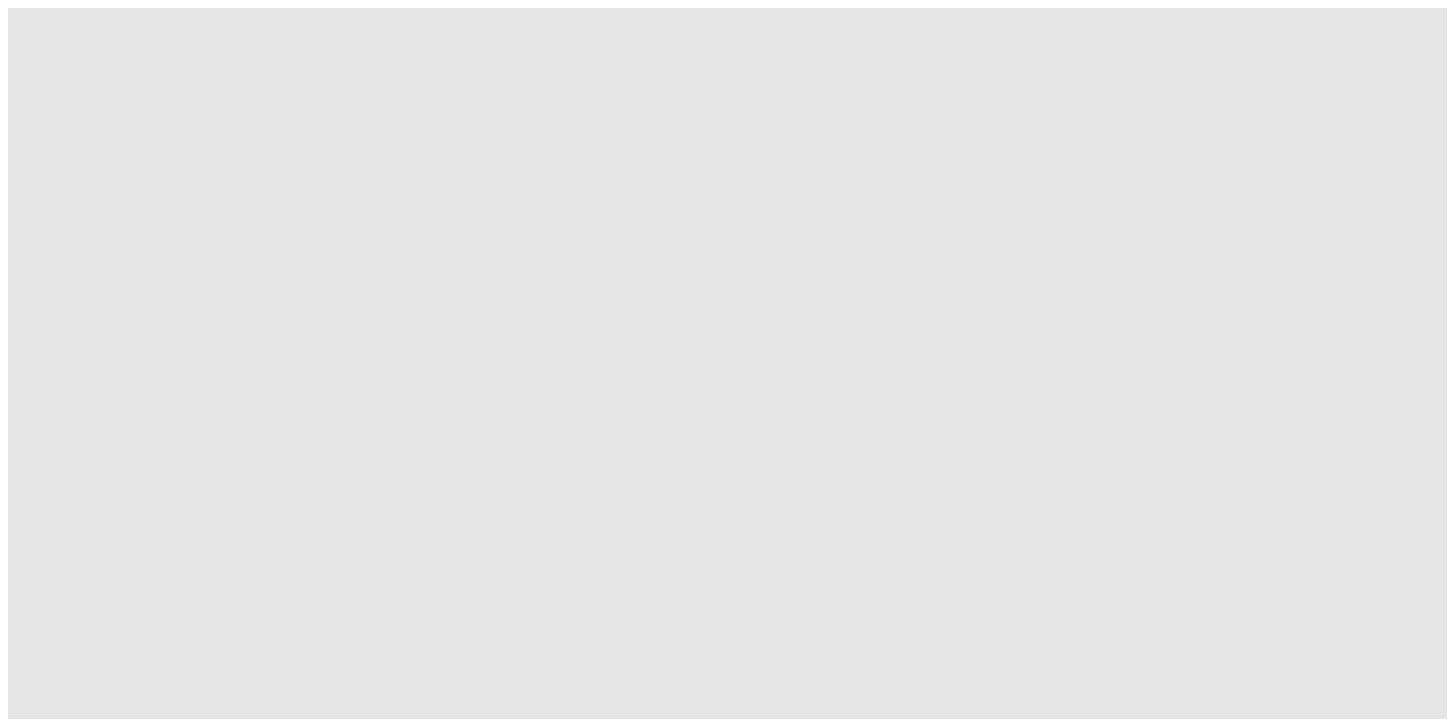
**Before you begin:** Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:** ***Inputting Data into HDFS***

**Lab Steps**

1 ) Put the Data into HDFS

1. Change directories to /root/hdp/pigandhive/labs/Lab3.2:



# cd ~/hdp/pigandhive/labs/Lab3.2

View the contents of salarydata.txt:

1. tail salarydata.txt M,49,29000,95103 M,44,34000,95102 M,99,25000,94041 F,93,96000,95105 F,75,9000,94040 F,14,0,95102 M,68,1000,94040 F,45,78000,94041 M,40,6000,95103 F,82,5000,95050

Notice the records in this file contain four values separated by commas, and the values represent a gender, age, salary, and zip code, respectively.



Create a new directory in HDFS named salarydata.



1. hdfs dfs -mkdir salarydata
   1. Put salarydata.txt into the salarydata directory in HDFS.
2. hdfs dfs –put salarydata.txt salarydata



2 ) Create a Table in the Database

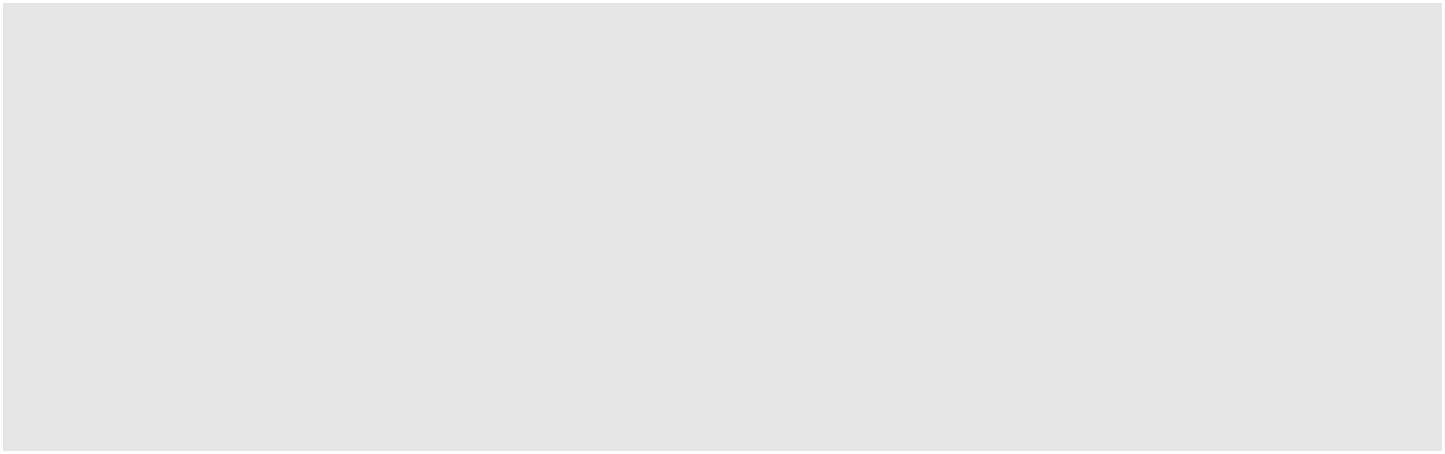
* 1. There is a script in the Exporting HDFS Data to an RDBMS lab folder that creates a table in MySQL that matches the records in salarydata.txt. View the SQL script:



1. more salaries2.sql
   1. Run sql commands in this this script using the following command:



1. # mysql -u root -p
2. # password: hadoop
3. mysql> uset test;
4. mysql> drop table if exists salaries2;
5. mysql> create table salaries2 (
6. gender varchar(1),
7. age int,
8. salary double,
9. zipcode int);
   1. Verify that the table was created successfully in MySQL:



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |
| mysql> describe salaries2; | | | + | +--------- | + | + |
| +--------- | +------------ | +------ |
| | Field | | Type | | Null | | Key | Default | | | Extra | | |
| +--------- | +------------ | +------ | +----- | +--------- | +------- | + |
| | gender | | varchar(1) | | YES | | | | NULL | | | | |
| | age | | int(11) | | YES | | | | NULL | | | | |
| | salary | | double | | YES | | | | NULL | | | | |
| | zipcode | | int(11) | | YES | | | | NULL | | | | |
| +--------- | +------------ | +------ | +----- | +--------- | +------- | + |

1. Exit the mysql prompt:

mysql> exit

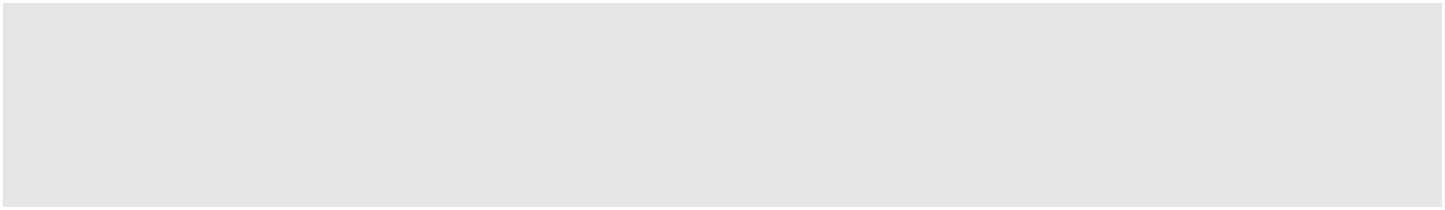
3 ) Export the Data



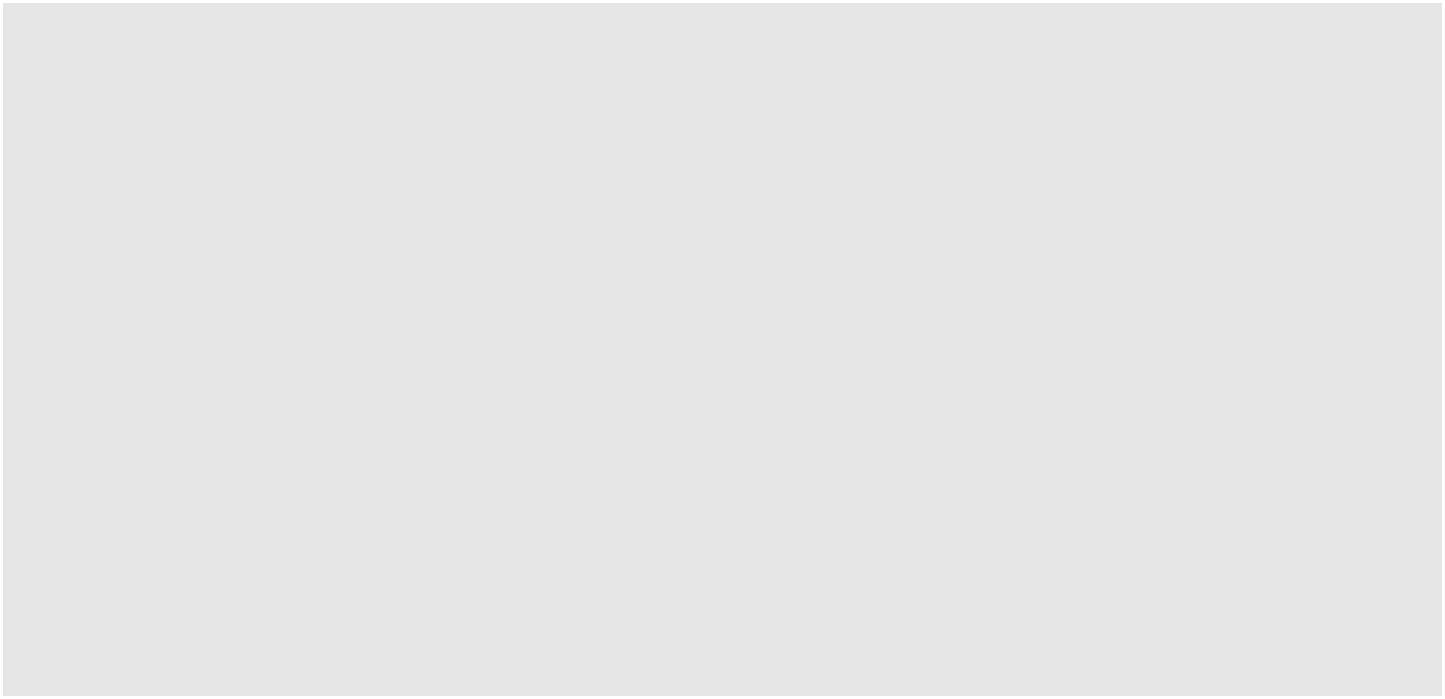
* 1. Run a Sqoop command that exports the salarydata folder in HDFS into the salaries2 table in MySQL. At the end of the MapReduce output, you should see a log event stating that 10,000 records were exported.

"





sqoop export --connect jdbc:mysql://<HOSTNAME>/test --username root --password hadoop --table salaries2 --export-dir salarydata --input-fields-terminated-by ","

1. Verify it worked by viewing the table’s contents from the mysql prompt. The output should look like the following:

# mysql -u root -p

# password: hadoop

mysql> use test;

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| mysql> select \* | | | from salaries2 limit 10; | | | | |
| +-------- | + | ------ | + | -------- | + | --------- | + |
| | gender | age | | | | salary | zipcode | | | | | |
| +-------- | + | ------57 | + | --------39000 | + | ---------95050 | + |
| | M | | | | | | | | |
| | F | | | 63 | | | 41000 | | | 95102 | | |
| | M | | | 55 | | | 99000 | | | 94040 | | |
| | M | | | 51 | | | 58000 | | | 95102 | | |
| | M | | | 75 | | | 43000 | | | 95101 | | |
| | M | | | 94 | | | 11000 | | | 95051 | | |
| | M | | | 28 | | | 6000 | | | 94041 | | |
| | M | | | 14 | | | 0 | | | 95102 | | |
| | M | | | 3 | | | 0 | | | 95101 | | |
| | M | | | 25 | | | 26000 | | | 94040 | | |
| +-------- | + | ------ | + | -------- | + | --------- | + |

c. Exit the mysql prompt.

**Result**

You have now used Sqoop to export data from HDFS into a database table in MySQL.





**Demonstration: Understanding MapReduce**

**About this Demonstration**

|  |  |
| --- | --- |
| **Objective:** | To understand how MapReduce works. |
| **During this** | Watch as your instructor performs the following steps. |
| **Demonstration:** | ***The MapReduce Framework*** |
| **Related lesson:** |

**Demonstration Steps**

1 ) Put the File into HDFS



1. demos#>hdfs dfs -put constitution.txt

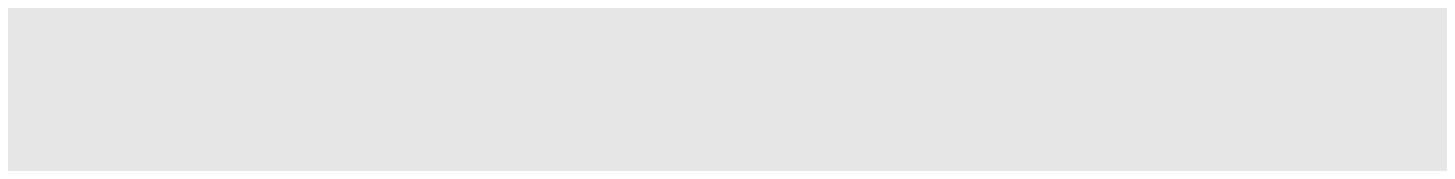
2 ) Run the WordCount Job

* 1. The following command runs a wordcount job on the constitution.txt and writes the output to wordcount\_output:

1. demos#> yarn jar /usr/hdp/ 2.6.4.0-91/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount constitution.txt wordcount\_output
   1. Notice that a MapReduce job gets submitted to the cluster. Wait for the job to complete.

3 ) View the Results

* 1. View the contents of the wordcount\_output folder:



1. hdfs dfs -ls wordcount\_output

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Found 2 items | | root | root | 0 | wordcount\_output/\_SUCCESS |
| -rw-r--r-- | 1 |
| -rw-r--r-- | 1 | root | root | 17049 | wordcount\_output/part-r-00000 |

* 1. Why is there one part-r file in this directory?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer*:**The job only used one reducer.

* 1. What does the “r” in the filename stand for? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer*:**The “r” stands for “reducer.”

* 1. View the contents of part-r-00000:

1. hdfs dfs -cat wordcount\_output/part-r-00000



|  |  |
| --- | --- |
|  |  |

1. Why are the words sorted alphabetically?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer*:**The key in this MapReduce job is the word, and keys aresorted during the shuffle/sort phase.

1. What was the key output by the WordCount reducer? \_\_\_\_\_\_\_\_\_\_\_

***Answer*:**The reducer’s output key was each word.

1. What was the value output by the WordCount reducer? \_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer:*** The value output by the reducer was the sum of the 1s, whichis the number of occurrences of the word in the document.

1. Based on the output of the reducer, what do you think the mapper output would be as key/value pairs? \_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer*:**The mapper outputs each word as a key and the number 1 aseach value.

**Lab: Running a MapReduce Job**

**About this Lab**

**Objective:**

Run a Java MapReduce job.

**File locations:**

/root/hdp/pigandhive/labs/Lab4.1

**Successful**

**outcome:**

You will see the results of the Inverted Index job in the inverted/output folder in HDFS.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Lab Steps**

1 ) Put the Data into HDFS

* 1. If not already done, open a Terminal window.
  2. The MapReduce job you are going to execute is an Inverted Index application, one of the very first use cases for MapReduce. Open a command prompt and change directories to /root/hdp/pigandhive/labs/Lab4.1:

1. cd ~/hdp/pigandhive/labs/Lab4.1
   1. Use more to view the contents of the file hortonworks.txt.
2. more hortonworks.txt



Each line looks like:



http://hortonworks.com/,hadoop,webinars,articles,download,enterpri se,team,reliability

Each line of text consists of a Web page URL, followed by a comma-separated list of keywords found on that page.

* 1. Make a new folder in HDFS named inverted/input:

1. hdfs dfs -mkdir -p inverted/input



Put hortonworks.txt into HDFS into the inverted/input/ folder. This file will be the input to the MapReduce job.

1. hdfs dfs –put hortonworks.txt inverted/input/



2 ) Run the Inverted Index Job

* 1. From the /root/hdp/pigandhive/labs/Lab4.1 folder, enter the following command (all on a single line):

1. yarn jar invertedindex.jar inverted.IndexInverterJob inverted/input inverted/output
   1. Wait for the MapReduce job to execute. The final output should look like:



File Input Format Counters



Bytes Read=1126

File Output Format Counters

Bytes Written=2997

3 ) View the Results

* 1. List the contents of the inverted/output folder.

1. hdfs dfs -ls inverted/output



How many reducers did this job use?

How can you determine this from the contents of inverted/output?

***Answer* :**The job used one reducer, which you can determine by the existence ofonly one part-r-n file in the output directory.

* 1. Use the cat command to view the contents of inverted/output/part-r-

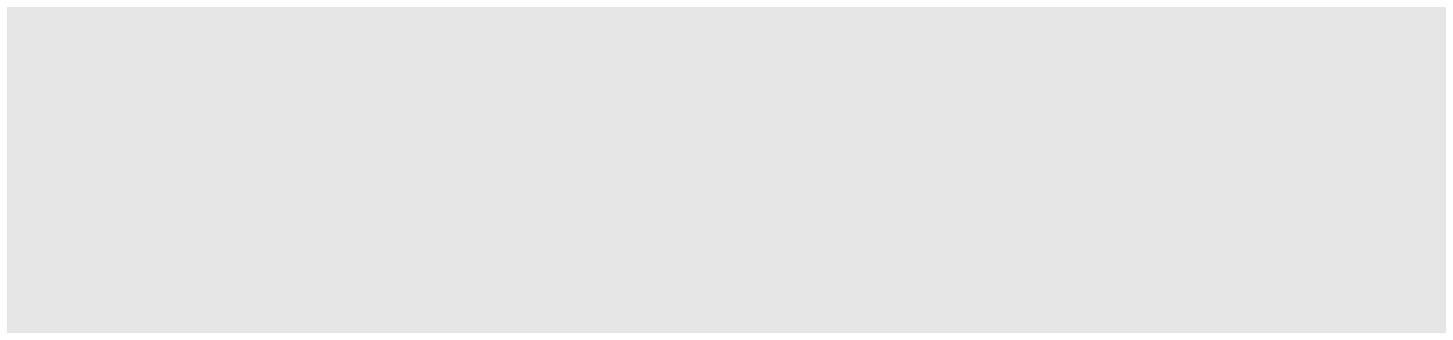
00000. The file should look like:

1. hdfs dfs -cat inverted/output/part-r-00000 abouthttp://hortonworks.com/about-us/, apache

http://hortonworks.com/products/hortonworksdataplatform/, http://hortonworks.com/about-us/,

articles http://hortonworks.com/community/,http://hortonworks.com/,

...



4 ) Specify the Number of Reducers

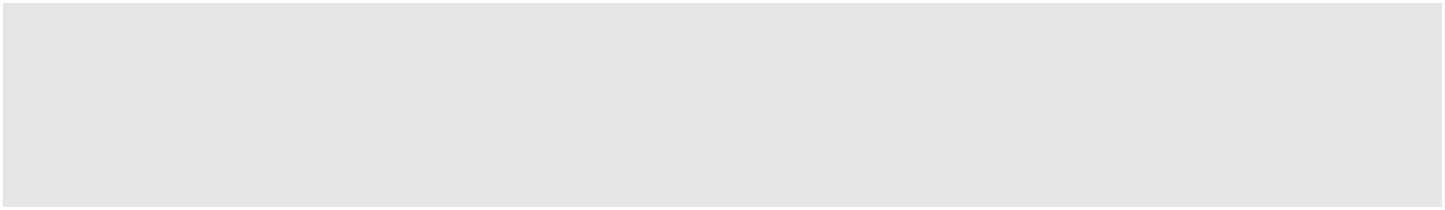
* 1. Try running the job again, but this time specify the number of reducers to be three:

1. yarn jar invertedindex.jar inverted.IndexInverterJob

-D mapreduce.job.reduces=3 inverted/input inverted/output

* 1. View the contents of inverted/output. Notice there are three part-r files:

1. hdfs dfs -ls inverted/output



|  |  |  |  |
| --- | --- | --- | --- |
| Found 3 items | | 1221 | inverted/output/part-r-00000 |
| 1 | root hdfs |
| 1 | root hdfs | 977 | inverted/output/part-r-00001 |
| 1 | root hdfs | 799 | inverted/output/part-r-00002 |

1. View the contents of the three files. How did the MapReduce framework determine which <key,value> pair to send to which reducer?

***Answer*:**<key,value>pairs are sent to the reducer based on the hashingof the key and using the remainder of dividing by the number of reducers.

**Result**

You have now executed a Java MapReduce job from the command line that takes an input text file and outputs the inverted indexes of the lines of text. This common task is what Web search engines like Google and Yahoo! use to determine the pages associated with search terms.

**Demonstration: Understanding Pig**

**About this Demonstration**

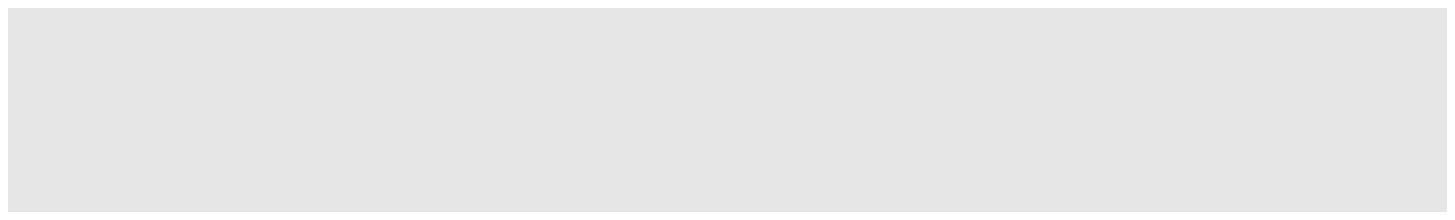
|  |  |
| --- | --- |
| **Objective:** | To understand Pig scripts and relations. |
| **During this** | Watch as your instructor performs the following steps. |
| **Demonstration:** | ***Introduction to Pig*** |
| **Related lesson:** |

**Demonstration Steps**

1 ) Start the Grunt Shell

* 1. If not already done, open a Terminal in your VM.
  2. Review the contents of the file pigdemo.txt located in /root/devph/labs/demos.

1. more /root/hdp/pigandhive/labs/demos/pigdemo.txt
   1. Start the Grunt shell:
2. pig
   1. Notice that the output includes where the logging for your Pig session will go as well as a statement about connecting to your Hadoop filesystem:
   2. Note - Run grunt shell with following command
   3. Use the -x command-line option: pig -x mr
   4. Note – Some Pig or hive labs are not running properly on Tez, there set the execution engine to mr. It works



[main] INFO org.apache.pig.Main - Logging error messages to:

/root/devph/labs/demos/pig\_1377892197767.log

[main] INFO org.apache.pig.backend.hadoop.executionengine. HExecutionEngine - Connecting to hadoop file system at: hdfs://sandbox.hortonworks.com:8020

2 ) Make a New Directory

1. Notice you can run HDFS commands easily from the Grunt shell. For example, run the ls command:



grunt> ls

1. Make a new directory named demos: grunt> mkdir demos



|  |  |
| --- | --- |
|  |  |

Demonstration: Understanding Pig

1. Use copyFromLocal to copy the pigdemo.txt file into the demos folder: grunt> copyFromLocal /root/hdp/pigandhive/labs/demos/pigdemo.txt demos/
2. Verify the file was uploaded successfully:

grunt> ls demos

hdfs://sandbox.hortonworks.com:8020/user/root/demos/pigdemo.txt<r 1>89

e. Change the present working directory to demos:

grunt> cd demos

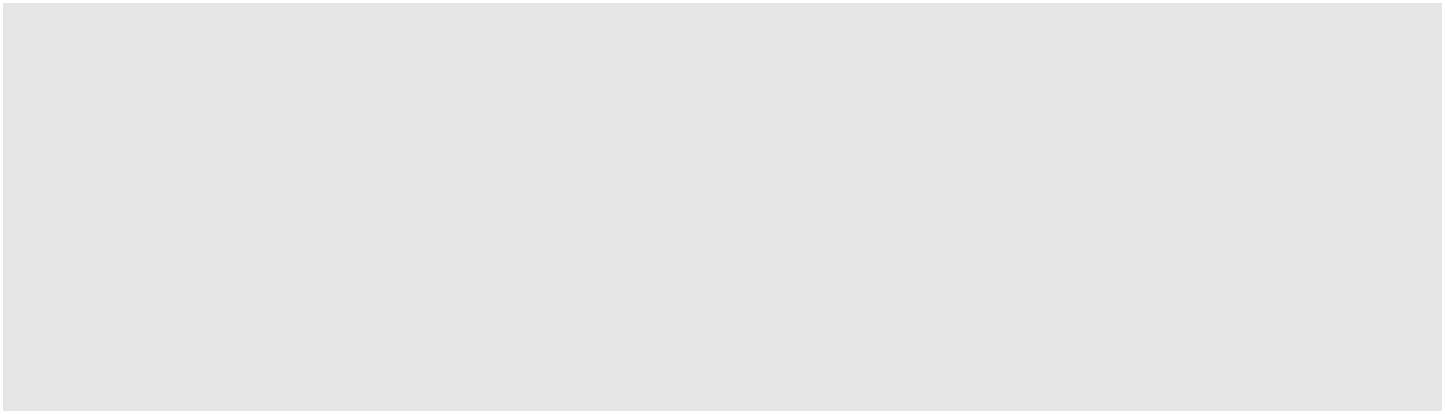


grunt> pwd

hdfs://sandbox.hortonworks.com:8020/user/root/demos

f. View the contents using the cat command:

grunt> cat pigdemo.txt



SD Rich

NV Barry

CO George

CA Ulf

IL Danielle

OH Tom

CA manish

CA Brian

CO Mark

3 ) Define a Relation

a. Define the employees relation, using a schema:

grunt> employees = LOAD 'pigdemo.txt' AS (state, name);



1. Demonstrate the describe command, which describes what a relation looks like:

grunt> describe employees;

employees: {state: bytearray,name: bytearray}

**Note**

Fields have a data type, and we will discuss data types later in this unit. Notice that the default data type of a field (if you do not specify one) is bytearray.

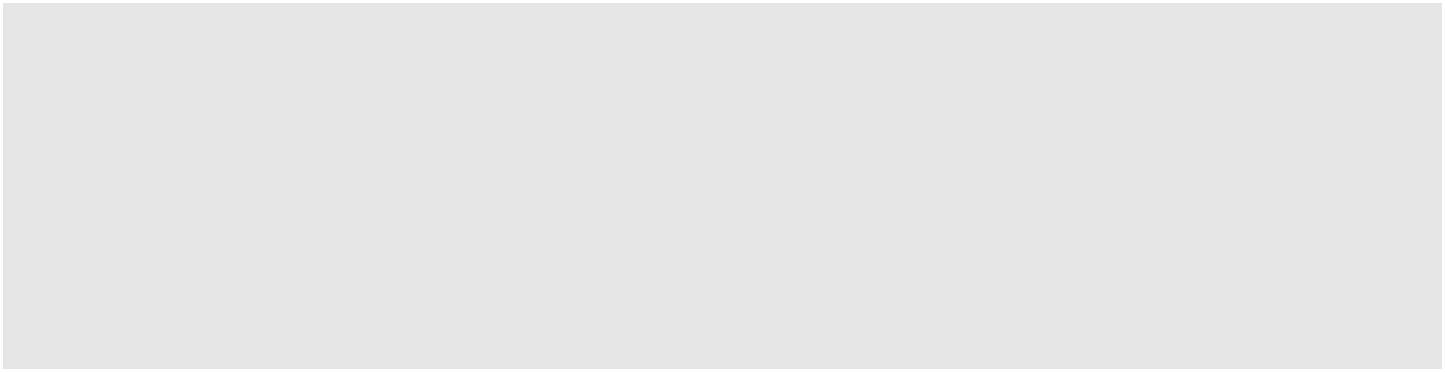


1. Let’s view the records in the employees relation: grunt> DUMP employees;



Notice this requires a MapReduce job to execute, and the result is a collection of tuples:

(SD,Rich)



(NV,Barry)

(CO,George)

(CA,Ulf)

(IL,Danielle)

(OH,Tom)

(CA,manish)

(CA,Brian)

(CO,Mark)

4 ) Filter the Relation by a Field

a. Let’s filter the employees whose state field equals CA:



grunt> ca\_only = FILTER employees BY (state=='CA');

grunt> DUMP ca\_only;

1. The output is still tuples, but only the records that match the filter appear:

(CA,Ulf)



(CA,manish)

(CA,Brian)

5 ) Create a Group

1. Define a relation that groups the employees by the state field: grunt> emp\_group = GROUP employees BY state;
2. Bags represent groups in Pig. A bag is an unordered collection of tuples:



grunt> describe emp\_group;



emp\_group: {group: bytearray,employees: {(state: bytearray,name:

bytearray)}}

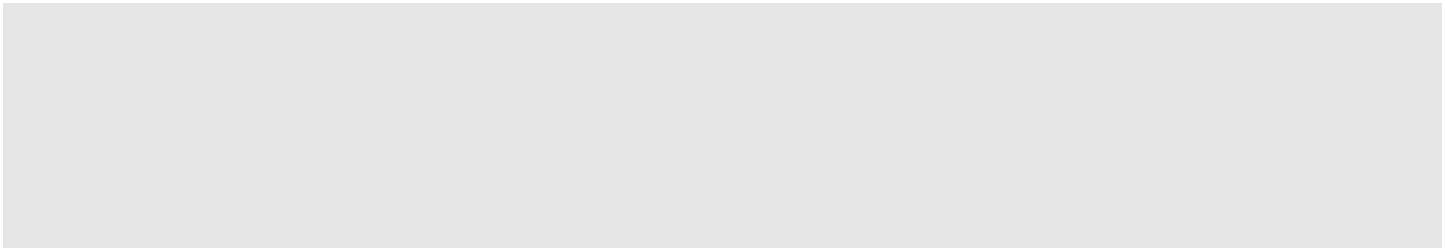
1. All records with the same state will be grouped together, as shown by the output of the emp\_group relation:

grunt> DUMP emp\_group;



The output is:

(CA,{(CA,Ulf),(CA,manish),(CA,Brian)})



(CO,{(CO,George),(CO,Mark)})

(IL,{(IL,Danielle)})

(NV,{(NV,Barry)})

(OH,{(OH,Tom)})

(SD,{(SD,Rich)})

**Note**

Tuples are displayed in parentheses. Curly braces represent bags.

6 ) The STORE Command

1. The DUMP command dumps the contents of a relation to the console. The STORE command sends the output to a folder in HDFS. For example:



grunt> STORE emp\_group INTO 'emp\_group';

Notice at the end of the MapReduce job that no records are output to the console.

b. Verify that a new folder is created:

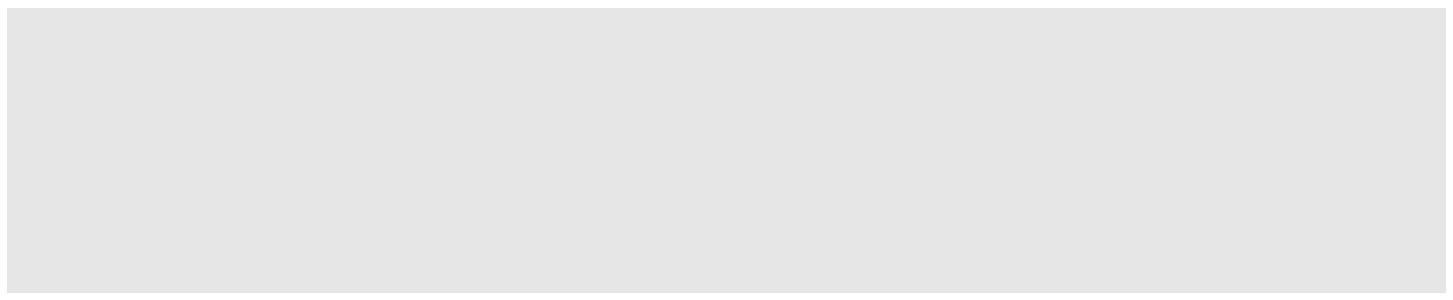


grunt> ls

hdfs://sandbox.hortonworks.com:8020/user/root/demos/emp\_group<dir> hdfs://sandbox.hortonworks.com:8020/user/root/demos/pigdemo.txt<r 1>89



c. View the contents of the output file:



grunt> cat emp\_group/part-v001-o000-r-00000

CA {(CA,Ulf),(CA,manish),(CA,Brian)}

CO {(CO,George),(CO,Mark)}

IL {(IL,Danielle)}

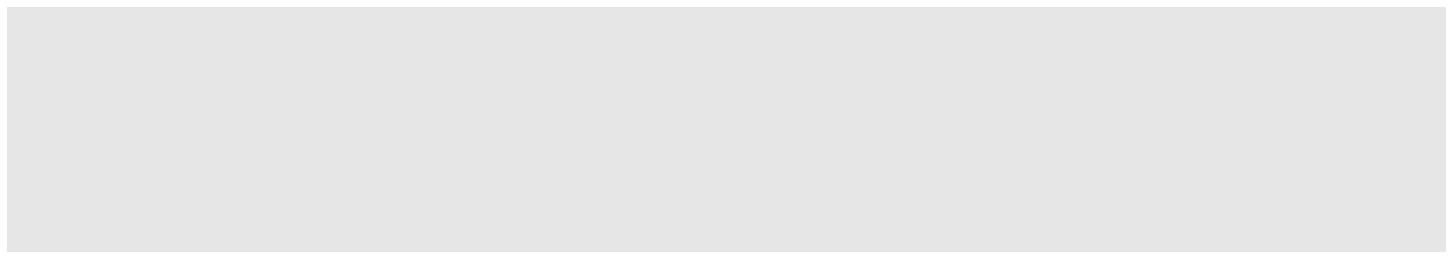
NV {(NV,Barry)}

OH {(OH,Tom)}

SD {(SD,Rich)}

Notice that the fields of the records (which in this case is the state field followed by a bag) are separated by a tab character, which is the default delimiter in Pig. Use the PigStorage object to specify a different delimiter:

grunt> STORE emp\_group INTO 'emp\_group\_csv' USING PigStorage(',');



To view the results:

grunt > ls

grunt > cat emp\_group\_csv/part-r-00000



7 ) Show All Aliases

a. The aliases command shows a list of currently defined aliases:

grunt> aliases;



aliases: [ca\_only, emp\_group, employees]

There will be a couple of additional numeric aliases created by the system for internal use. Please ignore them.

8 ) Monitor the Pig Jobs

1. Point your browser to the JobHistory UI at http://<sandbox-IPAddress>:19888/.
2. View the list of jobs, which should contain the MapReduce jobs that were executed from your Pig Latin code in the Grunt shell.
3. Notice you can view the log files of the ApplicationMaster and also each map and reduce task.

**Note**

Three commands trigger a logical plan to be converted to a physical plan and execute as a MapReduce job: STORE, DUMP, and ILLUSTRATE.

**Lab: Getting Started with Pig**

**About this Lab**

**Objective:**

Use Pig to navigate through HDFS and explore a dataset.

**File locations:**

/root/hdp/pigandhive/hdp/labs/Lab5.1

**Successful**

**outcome:**

You will have a couple of Pig programs that load the White House visitors’ data, with and without a schema, and store the output of a relation into a folder in HDFS.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:**

***Introduction to Pig***

**Lab Steps**

1 ) View the Raw Data

* 1. If not already done, open a Terminal in your VM and type "ssh sandbox".
  2. Change directories to the /root/hdp/pigandhive/labs/Lab5.1 folder:

1. cd ~/hdp/pigandhive/devph/labs/Lab5.1
   1. Unzip the archive in the /root/hdp/pigandhive/labs/Lab5.1 folder, which contains a file named whitehouse\_visits.txt that is quite large:
2. unzip whitehouse\_visits.zip
   1. View the contents of this file:
3. tail whitehouse\_visits.txt



This publicly available data contains records of visitors to the White House in Washington, D.C.

2 ) Load the Data into HDFS

a. Start the Grunt shell:



# pig



|  |  |
| --- | --- |
|  |  |

1. From the Grunt shell, make a new directory in HDFS named whitehouse: grunt> mkdir whitehouse
2. Use the copyFromLocal command in the Grunt shell to copy the whitehouse\_visits.txt file to the whitehouse folder in HDFS, renaming the file visits.txt. (Be sure to enter this command on a single line):



grunt> copyFromLocal /root/hdp/pigandhive/labs/Lab5.1/whitehouse\_visits.txt whitehouse/visits.txt



d. Use the ls command to verify that the file was uploaded successfully:

grunt> ls whitehouse



hdfs://sandbox.hortonworks.com:8020/user/root/whitehouse/visits.tx



t<r 1>183292235

3 ) Define a Relation

1. You will use the TextLoader to load the visits.txt file.

**Note**

TextLoader simply creates a tuple for each line of text, and it uses a single chararray field that contains the entire line. It allows you to load lines of text and not worry about the format or schema yet.

Define the following LOAD relation:



grunt>A = LOAD '/user/root/whitehouse/' USING TextLoader();

b. Use DESCRIBE to notice that A does not have a schema:



grunt> DESCRIBE A;

Schema for A unknown.

1. We want to get a sense of what this data looks like. Use the LIMIT operator to define a new relation named A\_limit that is limited to 10 records of A.



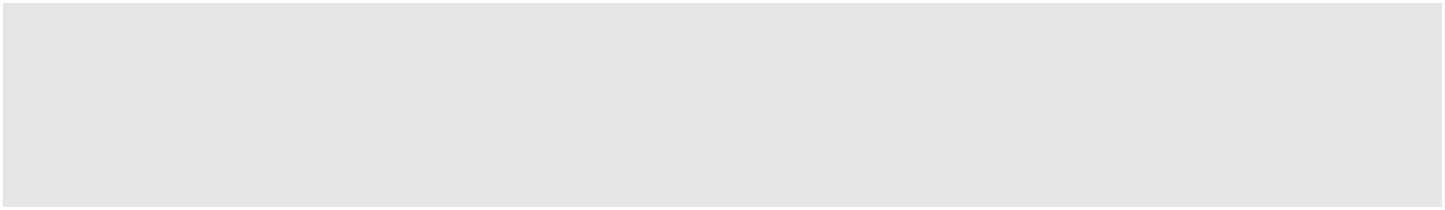
grunt> A\_limit = LIMIT A 10;



d. Use the DUMP operator to view the A\_limit relation.

Each row in the output will look similar to the following and should be 10 arbitrary rows from visits.txt:

grunt> DUMP A\_limit;



(WHITLEY,KRISTY,J,U45880,,VA,,,,,10/7/2010 5:51,10/9/2010

10:30,10/9/2010 23:59,,294,B3,WIN,10/7/2010

5:51,B3,OFFICE,VISITORS,WH,RES,OFFICE,VISITORS,GROUP TOUR

,1/28/2011,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

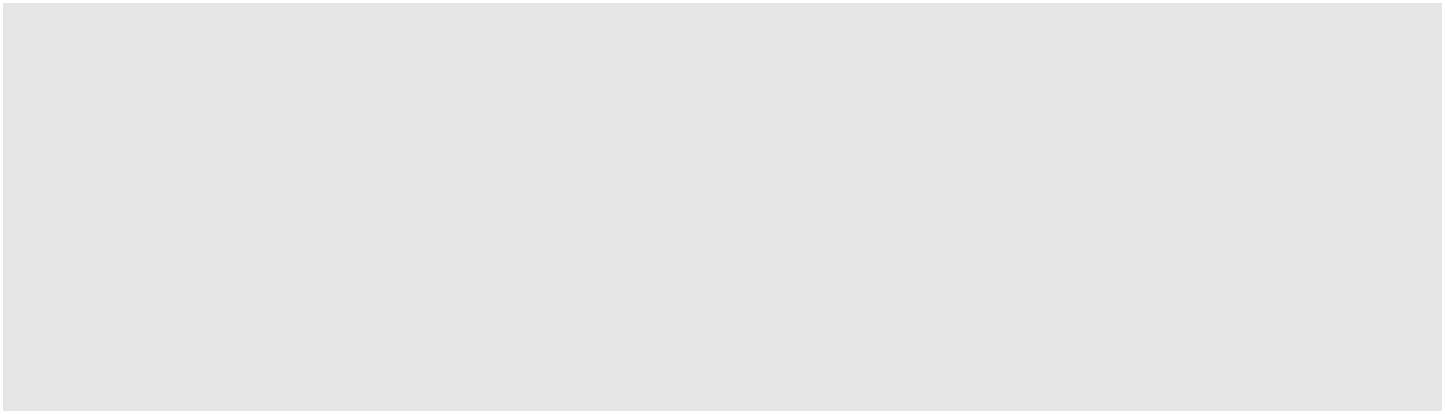


,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,)

4 ) Define a Schema

1. Load the White House data again, but this time use the PigStorage loader and also define a partial schema:

grunt> B = LOAD '/user/root/whitehouse/visits.txt' USING PigStorage(',') AS (



lname:chararray,

fname:chararray,

mname:chararray,

id:chararray,

status:chararray,

state:chararray,

arrival:chararray

);

b. Use the DESCRIBE command to view the schema:

grunt> describe B;



1. {lname: chararray,fname: chararray,mname: chararray,id: chararray,status: chararray,state: chararray,arrival: chararray}

5 ) The STORE Command

1. Enter the following STORE command, which stores the B relation into a folder named whouse\_tab and separates the fields of each record with tabs:

grunt> store B into 'whouse\_tab' using PigStorage('\t');



1. Verify that the whouse\_tab folder was created: grunt> ls whouse\_tab



You should see two map output files.

1. View one of the output files to verify they contain the B relation in a tab-delimited format:

grunt> fs -tail whouse\_tab/ part-v000-o000-r-00000



d. Each record should contain seven fields. What happened to the rest of the fields from the raw data that was loaded from

whitehouse/visits.txt?

***Answer:*** They were simply ignored when each record was read in fromHDFS.

6 ) Use a Different Storer

1. In the previous step, you stored a relation using PigStorage with a tab delimiter. Enter the following command, which stores the same relation but in a JSON format:



grunt> store B into 'whouse\_json' using JsonStorage();

1. Verify that the whouse\_json folder was created: grunt> ls whouse\_json
2. View one of the output files:



grunt> fs -tail whouse\_json/ part-v000-o000-r-00000

Notice that the schema you defined for the B relation was used to create the format of each JSON entry:

**Result**

You have now seen how to execute some basic Pig commands, load data into a relation, and store a relation into a folder in HDFS using different formats.



**Lab: Preparing Data for Hive**

**About this Lab**

**Objective:**

Transform and export a dataset for use with Hive.

**File locations:**

/root/hdp/pigandhive/devph/labs/Lab6.3

**Successful outcome:**

The resulting Pig script stores a projection of visits.txt in a folder in the Hive warehouse named wh\_visits.

**Before you begin:**

You should have visits.txt in a folder named whitehouse in HDFS.

**Related lesson:**

***Advanced Pig Programming***

**Lab Steps**

1 ) Review the Pig Script

* 1. If not already done, open a Terminal window in putty or git-bash by connecting to sandbox.
  2. Change directories to the Preparing Data for Hive lab folder:

1. cd ~/hdp/pigandhive/devph/labs/Lab6.3/
   1. View the contents of wh\_visits.pig:
2. more wh\_visits.pig
   1. Notice that all White House visitors who met with the President are the potus relation.
   2. Notice that the project\_potus relation is a projection of the last name, first name, time of arrival, location, and comments from the visit.



2 ) Store the Projection in the Hive Warehouse

1. Open wh\_visits.pig with the text editor.
2. Add the following command at the bottom of the file, which stores the project\_potus relation into a very specific folder in the Hive warehouse:



STORE project\_potus INTO '/apps/hive/warehouse/wh\_visits/';



|  |  |
| --- | --- |
|  |  |

3 ) Run the Pig Script

* 1. Save your changes to wh\_visits.pig.
  2. Run the script from the command line:

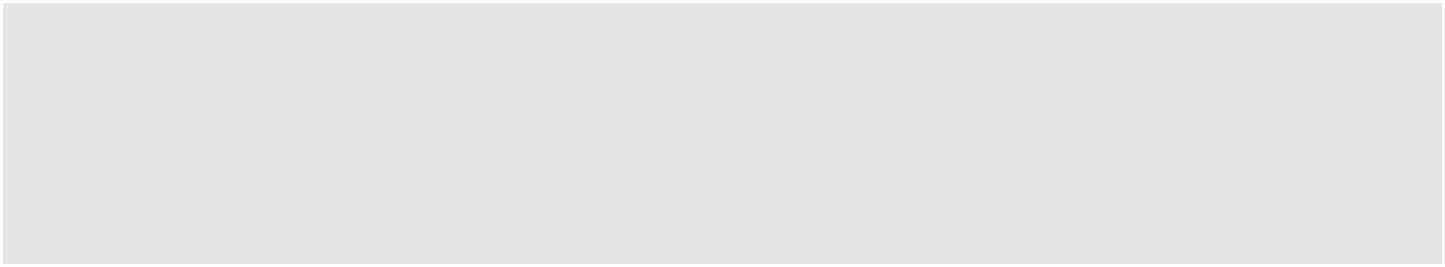
1. pig wh\_visits.pig



4 ) View the Results

* 1. The wh\_visits.pig script creates a directory in the Hive warehouse named wh\_visits. Use ls to view its contents:

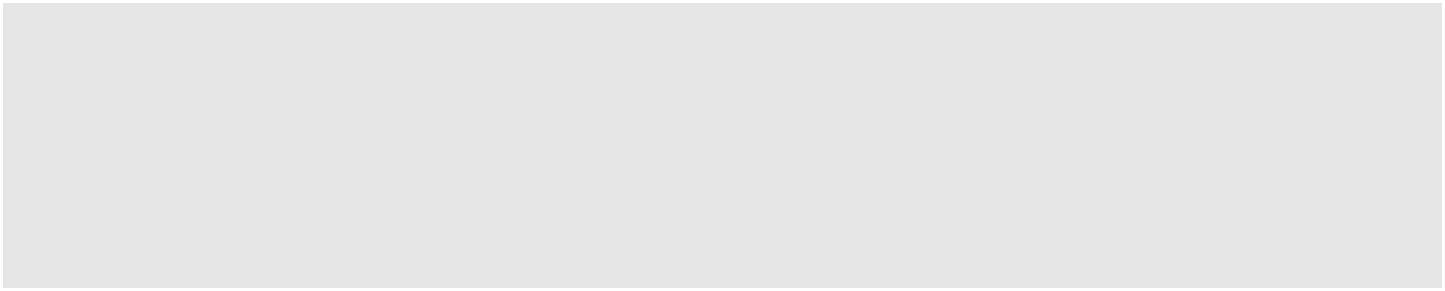
1. hdfs dfs -ls /apps/hive/warehouse/wh\_visits/



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -rw-r--r-- | 1 | root hdfs | 0 |  |
| /apps/hive/warehouse/wh\_visits/\_SUCCESS | | | | |
| -rw-r--r-- | 1 | root hdfs | 971339 | /apps/hive/warehouse/wh\_visits/part- |
| m-00000 | 1 | root hdfs | 142850 | /apps/hive/warehouse/wh\_visits/part- |
| -rw-r--r-- |
| m-00001 |  |  |  |  |

1. View the contents of one of the result files. It should look like the following:

hdfs dfs -cat /apps/hive/warehouse/wh\_visits/part-m-00000 ...



FRIEDMAN THOMAS 10/12/2010 12:08 WH PRIVATE LUNCH BASS EDWIN 10/18/2010 15:01 WH

BLAKE CHARLES 10/18/2010 15:00 WH OGLETREE CHARLES 10/18/2010 15:01 WH RIVERS EUGENE 10/18/2010 15:01 WH

**Result**

You now have a folder in the Hive warehouse named wh\_visits that contains a projection of the data in visits.txt. We will use this file in an upcoming Hive lab.



**Lab: Understanding Hive Tables**

**About this Lab**

**Objective:** Understand how Hive table data is stored in HDFS.

**File locations:** /root/hdp/pigandhive/devph/labs/Lab7.1

**Successful outcome:** A new Hive table filled with the data from the wh\_visitsfolder.

**Before you begin:** Complete the Preparing Data for Hive lab, or put the data from the solution of that lab into HDFS.

**Related lesson:** Hive Programming

**Lab steps**

1 ) Review the Data

1. If not already done, open a Terminal in your VM and type "ssh sandbox".
2. Use the hdfs dfs -ls command to view the contents of the

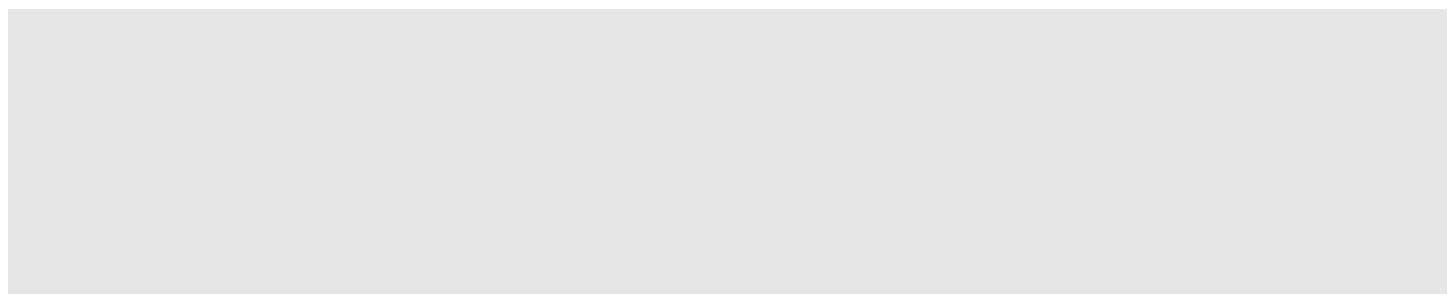
/apps/hive/warehouse/wh\_visits/ folder in HDFS that was created in an

earlier lab. You should see six part-m files:



# hdfs dfs -ls /apps/hive/warehouse/wh\_visits/

1. Recall that the Pig projection to create these files had the following schema (no typing necessary - this is reprinted below for reference only):



project\_potus = FOREACH potus GENERATE $0 AS lname:chararray,

$1 AS fname:chararray,

$6 AS time\_of\_arrival:chararray,

$11 AS appt\_scheduled\_time:chararray, $21 AS location:chararray,

$25 AS comment:chararray ;

In this lab, you will define a Hive table that matches these records and contains the exported data from your Pig script.

2 ) Define a Hive Script

1. In the Understanding /root/hdp/pigandhive/labs/Lab7.1 folder, there is a text file named wh\_visits.hive. View its contents.



|  |  |
| --- | --- |
|  |  |

project\_potus

Notice that it defines a Hive table named wh\_visits with the following

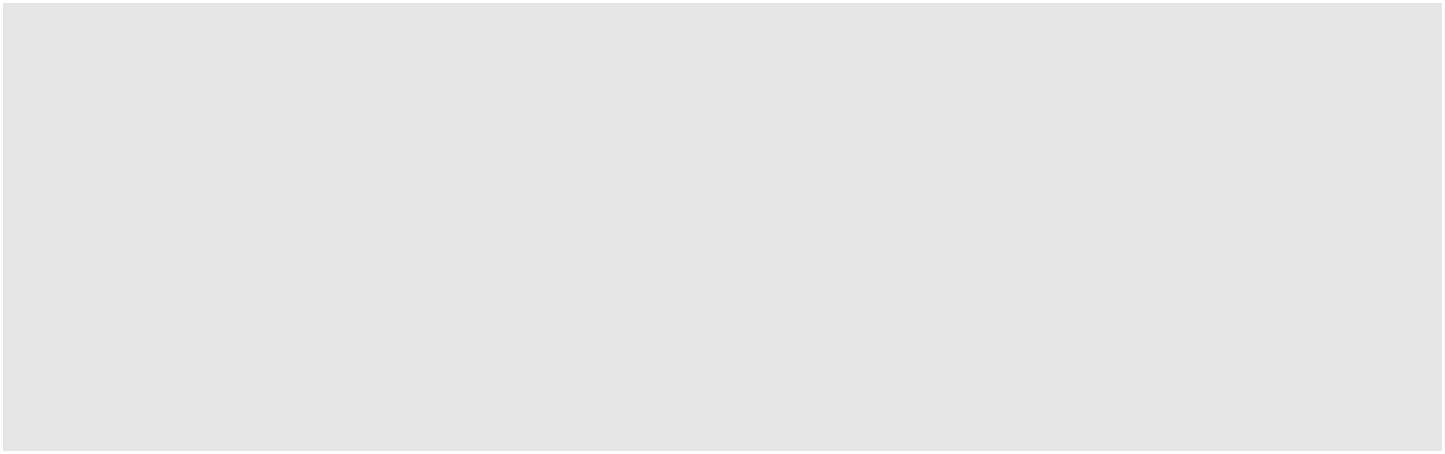
schema that matches the data in your folder:

1. cd ~/devph/labs/Lab7.1
2. more wh\_visits.hive
3. create table wh\_visits (

lname string, fname string,

time\_of\_arrival string, appt\_scheduled\_time string, meeting\_location string, info\_comment string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;



**Note**

You cannot use comment or location as column names because those are reserved Hive keywords, so we changed them slightly.

* 1. Run the script with the following command:

1. hive -f wh\_visits.hive



If successful, you should see “OK” in the output along with the time it took to run the query.

3 ) Verify the Table Creation

a. Start the Hive Shell:



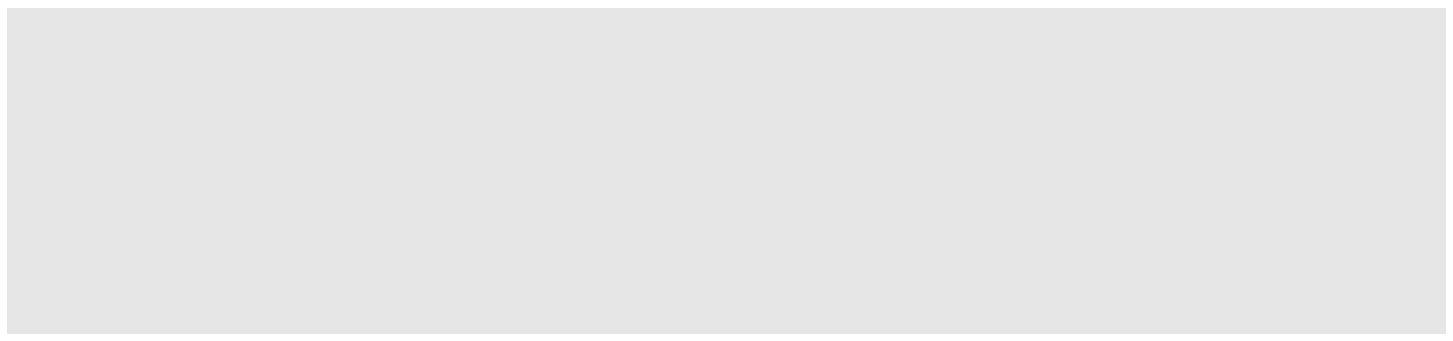
# hive

1. From the hive> prompt, enter the “show tables” command: hive> show tables;



You should see wh\_visits in the list of tables.

c. Use the describe command to view the details of wh\_visits:



hive> describe wh\_visits;

OK

lname string

fname string

time\_of\_arrival string

appt\_scheduled\_time string

meeting\_location string

info\_comment string



1. Try running a query (even though the table is empty): hive> select \* from wh\_visits limit 20;



You should see 20 rows returned. How is this brand new Hive table already populated with records? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer***: In a previous lab, you already populated the

/apps/hive/warehouse/wh\_visits folder with the output of a Pig job.

Why did the previous query not require a Tez or MapReduce job to execute?

***Answer*:**The query selected all columns and did not contain aWHEREclause,the query just needs to read in the data from the file and display it.

4 ) Count the Number of Rows in a Table

a. Enter the following Hive query, which outputs the number of rows in

wh\_visits:



hive> select count(\*) from wh\_visits;

How many rows are currently in wh\_visits? \_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer:*** 21,819

5 ) Selecting the Input File Name

1. Hive has two virtual columns that get created automatically for every

table: INPUT\_\_FILE\_\_NAME and BLOCK\_\_OFFSET\_\_INSIDE\_\_FILE.

Note that between each word in the column name there are two underscore characters, not just one. You must make sure you type both of them when using these columns in a hive command.

You can use these column names in your queries just like any other column of the table. To demonstrate, run the following query:



hive> select INPUT\_\_FILE\_\_NAME, lname, fname FROM wh\_visits WHERE lname LIKE 'Y%';

1. The result of this query is visitors to the White House whose last name starts with “Y.” Notice that the output also contains the particular file that the record was found in:

hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/wh\_visits/



part-m-00000YOUNGMICHELLE

hdfs://sandbox.hortonworks.com:8020/apps/hive/warehouse/wh\_visits/



part-m-00001YOUNGLEDISI



6 ) Drop a Table

1. Let’s see what happens when a managed table is dropped. Start by defining a simple table called names using the Hive Shell:

hive> create table names (id int, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';



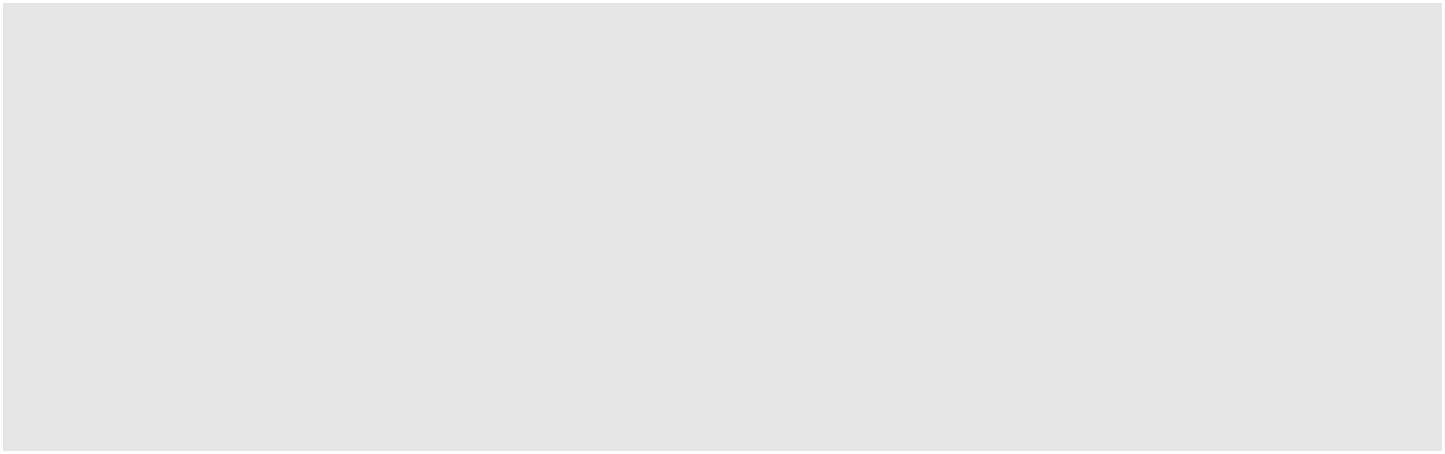
1. Use the Hive dfs command to put Lab7.1/names.txt into the table’s warehouse folder:

.

|  |  |
| --- | --- |
| c. | View the contents of the table’s warehouse folder: |
| hive> dfs | -ls /apps/hive/warehouse/names; |
| Found 1 items | |
| root hdfs | 78 /apps/hive/warehouse/names/names.txt |
| d. | From the Hive Shell, run the following query: |



hive> select \* from names;



OK

1. Rich
2. Barry
3. George
4. Ulf
5. Danielle
6. Tom
7. manish
8. Brian
9. Mark
   1. Now drop the names table:

hive> drop table names;

* 1. View the contents of the table’s warehouse folder again. Notice the names folder is gone:



hive> dfs -ls /apps/hive/warehouse/names;



ls: '/apps/hive/warehouse/names': No such file or directory

**Important**

Be careful when you drop a managed table in Hive. Make sure you either have the data backed up somewhere else or that you no longer want the data.



7 ) Define an External Table

1. In this step you will see how external tables work in Hive. Start by putting names.txt into HDFS:

hive> dfs -put /root/hdp/pigandhive/labs/Lab7.1/names.txt names.txt;



1. Create a folder in HDFS for the external table to store its data in: hive> dfs -mkdir hivedemo;
2. Define the names table as external this time:



hive> create external table names (id int, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/user/root/hivedemo';



d. Load data into the table:

hive> load data inpath '/user/root/names.txt' into table names;



1. Verify that the load worked:

hive> select \* from names;

1. Notice the names.txt file has been moved to /user/root/hivedemo:



|  |  |  |  |
| --- | --- | --- | --- |
| hive> | dfs -ls hivedemo; | |  |
| Found | 1 items | 1 root hdfs | 78 hivedemo/names.txt |
| -rw-r--r-- | |



1. Similarly, verify that names.txt is no longer in your /user/root folder in HDFS.

hive> dfs -ls /user/root/names.txt;



Why is it gone? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer*:**TheLOADcommand moved the file from/user/rootto/user/root/names. The LOAD command does not copy files; it moves them.

1. Use the ls command to verify that the /apps/hive/warehouse folder does not contain a subfolder for the names table.



hive> dfs -ls /apps/hive/warehouse;

1. Now drop the names table:

hive> drop table names;

1. View the contents of /user/root/hivedemo. Notice that names.txt is still there.



hive> dfs -ls /user/root/hivedemo;



**Result**

You have verified that the data for external tables is not deleted when the corresponding table is dropped. Aside from this behavior, managed tables and external tables in Hive are essentially the same. You now have a table in Hive named wh\_visits that was loaded from the result of a Pig job. You also have an external table called names that stores its data in /user/root/hivedemo. At this point, you should have a pretty good understanding of how Hive tables are created and populated.



**Demonstration: Understanding Partitions and Skew**

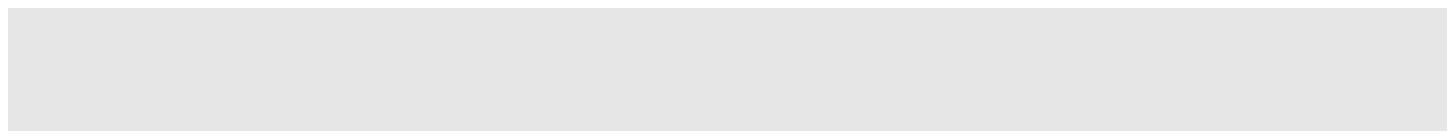
**About this Demonstration**

|  |  |
| --- | --- |
| **Objective:** | To understand how Hive partitioning and skewed tables |
|  | work. |
| **During this** | Watch as your instructor performs the following steps. |
| **Demonstration:** | ***Hive Programming*** |
| **Related lesson:** |

**Demonstration Steps**

1. View the Data
   1. If not already done, open a Terminal to your VM.
   2. Review the hivedata\_<<state>>.txt files in /root/hdp/pigandhive/devph/labs/demos. This will be

the data added to the table.

1. Define the Table in Hive

# hive

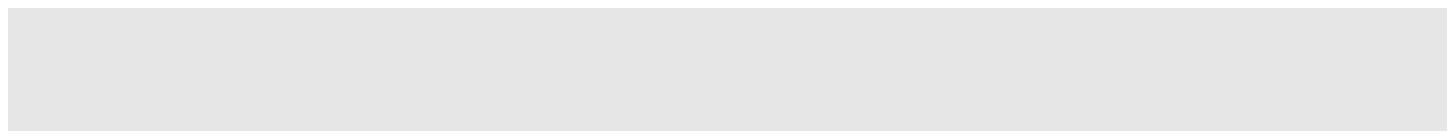
hive> create table names (id int, name string)

partitioned by (state string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' ;

hive> show partitions names;

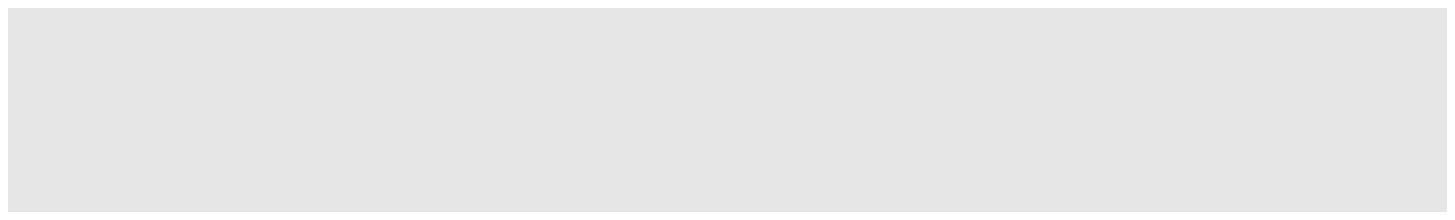
3) Load Data into the Table

1. When you load data into a partitioned table, you specify which partition the data goes into. For example:



hive> load data local inpath '/root/hdp/pigandhive/labs/demos/hivedata\_ca.txt' into table names partition (state = 'CA');

b) Load the CO and SD files also:



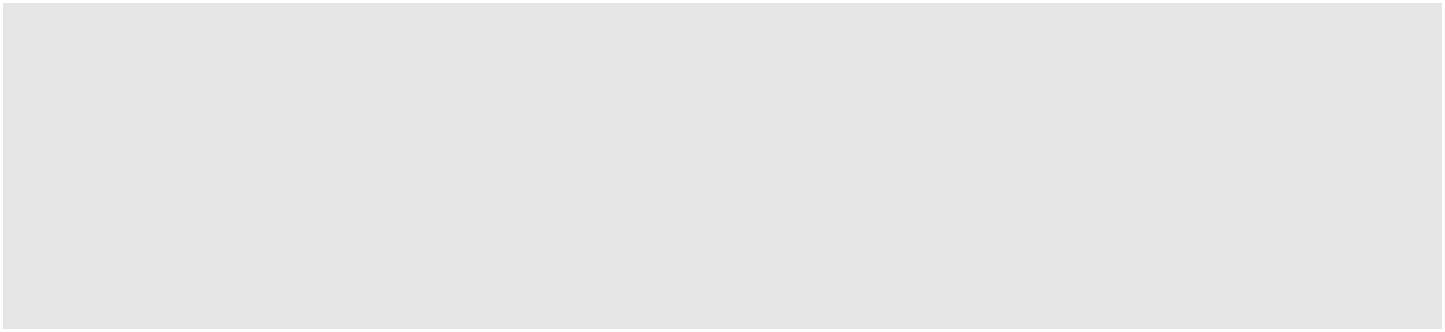
hive> load data local inpath '/root/hdp/pigandhive/labs/demos/hivedata\_co.txt' into table names partition (state = 'CO');

load data local inpath '/root/hdp/pigandhive/labs/demos/hivedata\_sd.txt' into table names partition (state = 'SD');



c) Verify that all of the data made it into the names table:

hive> select \* from names;



|  |  |  |
| --- | --- | --- |
| OK | Ulf | CA |
| 1 |

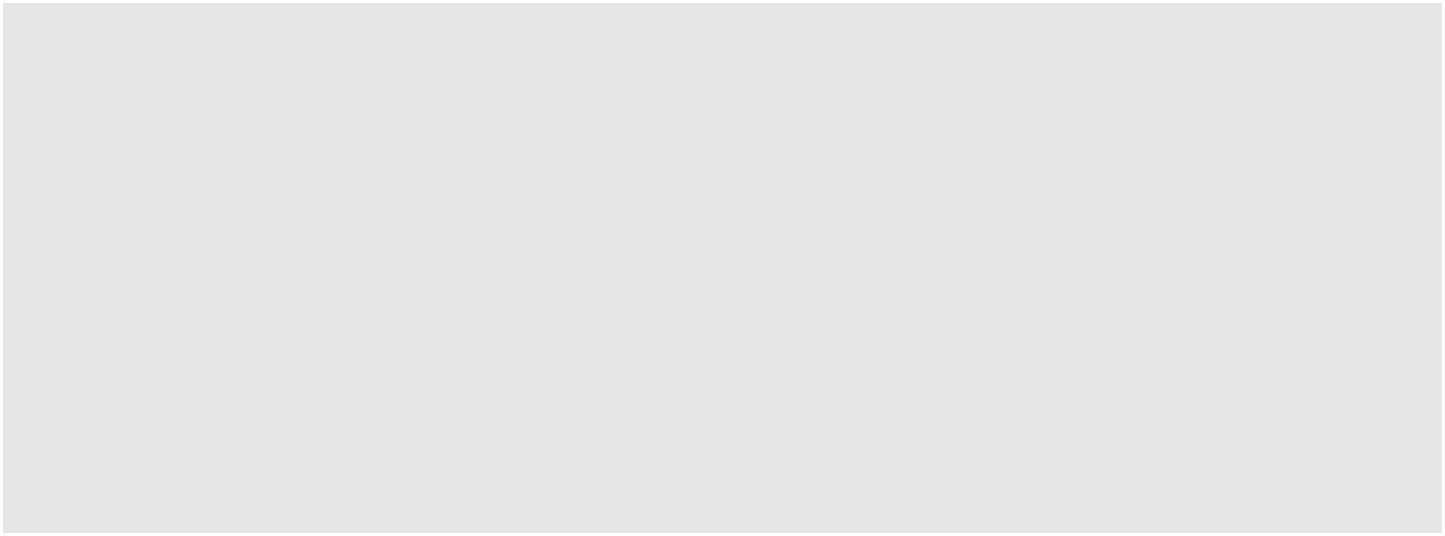
1. Manish CA
2. Brian CA
3. George CO

|  |  |  |
| --- | --- | --- |
| 5 | Mark | CO |
| 6 | Rich | SD |

4) View the Directory Structure

a) List the partitions and view the contents of /apps/hive/warehouse/names:

hive> show partitions names;



OK

state=CA

state=CO

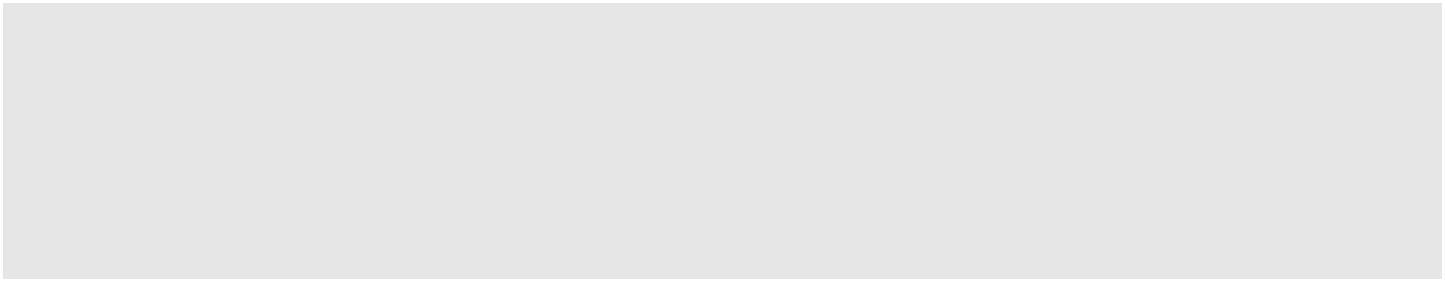
state=SD

hive> dfs -ls -R /apps/hive/warehouse/names/; 0 /apps/hive/warehouse/names/state=CA

1. /apps/hive/warehouse/names/state=CA/hivedata\_ca.txt
2. /apps/hive/warehouse/names/state=CO
3. /apps/hive/warehouse/names/state=CO/hivedata\_co.txt

|  |  |
| --- | --- |
| 0 | /apps/hive/warehouse/names/state=SD |
| 6 | /apps/hive/warehouse/names/state=SD/hivedata\_sd.txt |
|  | Notice that each partition has its own subfolder for storing its contents. |
| 5) | Perform a Query |
|  | a) When you specify a where clause that includes a partition, Hive is smart |
|  | enough to only scan the files in that partition. For example: |

hive> select \* from names where state = 'CA'; OK



1 Ulf CA

1. Manish CA
2. Brian CA
   1. Notice that a MapReduce job was not executed. Why? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Answer:** The result of the query is exactly the contents of the underlying files,so there is no need to run a MapReduce job. The files can simply be read and displayed.

* 1. You can select the partition field, even though it is not actually in the data file. Hive uses the directory name to retrieve the value:



hive> select name, state from names where state = 'CA';



1. You can still run queries across the entire dataset. For example, the following query spans multiple partitions. When you are done, use quit to exit the Hive shell.:

hive> select name, state from names where state = 'CA' or state = 'SD';



hive> quit;

6) Create a Skewed Table

* 1. Verify the existence of the salaries.txt folder in ~/hdp/pigandhive/labs/demos/ and then put it into the /user/root/salarydata/ folder in HDFS.

1. ls salaries.txt
2. hdfs dfs -put salaries.txt /user/root/salarydata/salaries.txt
   1. View the contents of demos/skewdemo.hive, which defines a skewed table named skew\_demo using the salaries.txt data:
3. more skewdemo.hive
   1. Which values are skewing this table? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



***Answer:*** The skewed values are the 95102 and 94040 zip codes.

1. Run the skewdemo.hive script:

# hive -f skewdemo.hive

* 1. View the contents of the underlying Hive warehouse folder:

# hdfs dfs -ls -R /apps/hive/warehouse/skew\_demo

* 1. Select a few records to make sure the table has data behind it:

# hive -f show\_skewdemo.hive



**Lab: Analyzing Big Data with Hive**

**About this Lab**

**Objective:**

Analyze the White House visitor data.

**File locations:**

/root/hdp/pigandhive/devph/labs/Lab7.2

**Successful**

**outcome:**

You will have discovered several useful pieces of information about the White House visitor data.

**Before you begin:**

Complete the Understanding Hive Tables Lab.

**Related lesson:**

***Hive Programming***

**Lab Steps**

1 ) Find the First Visit

1. If not already done, open a git-bash or putty Terminal to your VM.
2. Using an editor, create a new text file named whitehouse.hive and save it in

your ~/hdp/pigandhive/labs/Lab7.2 folder.

1. In this step, you will instruct the hive script to find the first visitor to the White House (based on our dataset). This will involve some clever handling of timestamps. This will be a long query, so enter it on multiple lines (note the lack of a ";" at the end of this first step). Start by selecting all columns where the time\_of\_arrival is not empty:



select \* from wh\_visits where time\_of\_arrival != ""

1. To find the first visit, we need to sort the result. This requires converting the time\_of\_arrival string into a timestamp. We will use the unix\_timestamp function to accomplish this. Add the following order by clause (again, no ";" at the end of the line):



order by unix\_timestamp(time\_of\_arrival, 'MM/dd/yyyy hh:mm')

1. Since we are only looking for one result, we certainly don’t need to return every row. Let’s limit the result to 10 rows, so we can view the first

10 visitors (this finishes the query, so will end with the ";" character):



limit 10;

f. Save your changes to whitehouse.hive.



|  |  |
| --- | --- |
|  |  |

Lab: Analyzing Big Data with Hive

* 1. Execute the script whitehouse.hive and wait for the results to be displayed:

1. cd ~/hdp/pigandhive/labs/Lab7.2
2. hive -f whitehouse.hive
   1. The results should be 10 visitors, and the first visit should be in 2009, since that is when the dataset begins. The first visitors are Charles Kahn and Carol Keehan on 3/5/2009.



2 ) Find the Last Visit

1. This one is easy: just take the previous query and reverse the order by adding desc to the order by clause:

order by unix\_timestamp(time\_of\_arrival, 'MM/dd/yyyy hh:mm') desc



* 1. Run the query again, and you should see that the most recent visit was Jackie Walker on 3/18/2011.

1. hive -f whitehouse.hive



3 ) Find the Most Common Comment

1. In this step, you will explore the info\_comment field and try to determine the most common comment. You will use some of Hive’s aggregate functions to accomplish this. Start by using an editor to create a new text file

named comments.hive and save it in ~/hdp/pigandhive/labs/Lab7.2 folder.

1. You will now create a query that displays the 10 most frequently occurring comments. Start with the following select clause:

from wh\_visits



select count(\*) as comment\_count, info\_comment

This runs the aggregate count function on each group (which you will define later in the query) and names the result comment\_ count. For example, if “OPEN HOUSE” occurs five times then comment\_count will be five for that group.

Notice we are also selecting the info\_comment column so we can see what the comment is.

1. Group the results of the query by the info\_comment column: group by info\_comment



1. Order the results by comment\_count, because we are only interested in comments that appear most frequently:

order by comment\_count DESC



* 1. We only want the top results, so limit the result set to 10:

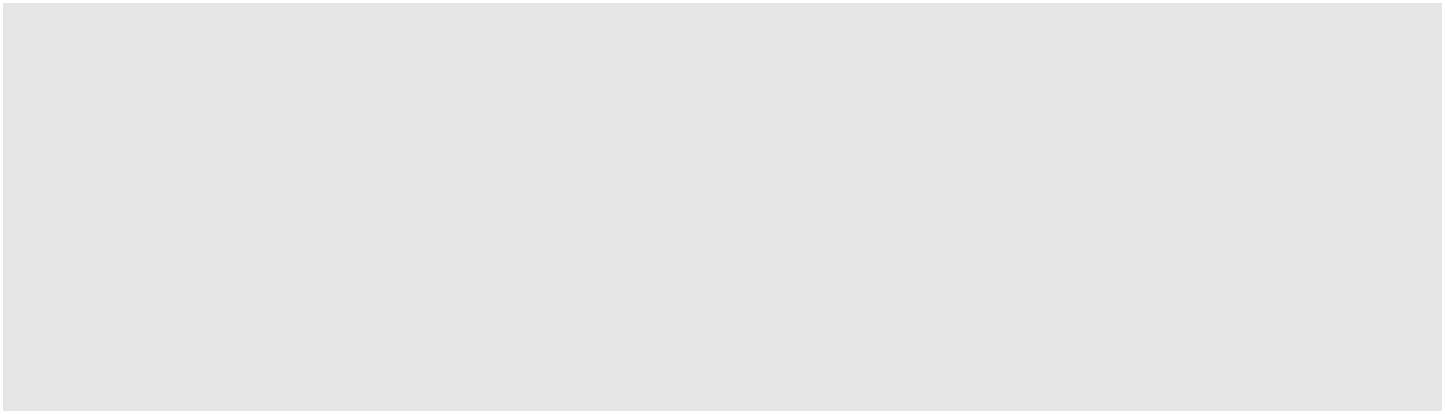
limit 10;

* 1. Save your changes to comments.hive and execute the script. Wait for the MapReduce job to execute.

1. hive -f comments.hive
   1. The output will be 10 comments and should look like:



9036



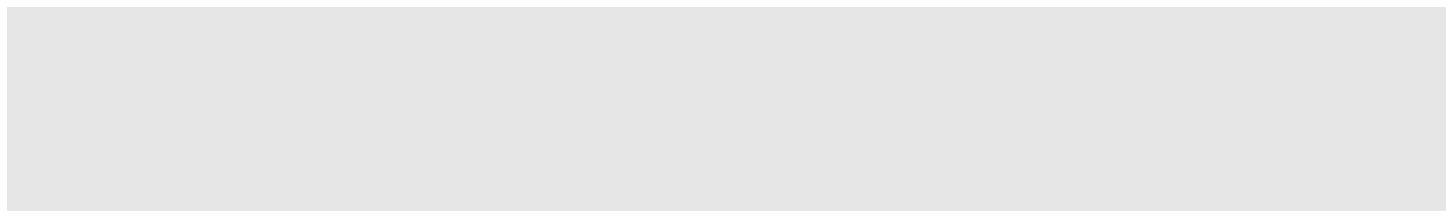
1253 HOLIDAY BALL ATTENDEES/

894 WHO EOP RECEP 2

700 WHO EOP 1 RECEPTION/

1. RESIDENCE STAFF HOLIDAY RECEPTION/
2. PRESS RECEPTION ONE (1)/
3. GENERAL RECEPTION 1
4. HANUKKAH RECEPTION./
5. GEN RECEP 5/
6. GENERAL RECEPTION 3
   1. It appears that a blank comment is the most frequent comment, followed by the HOLIDAY BALL, then a variation of other receptions.
   2. Modify the query so that it ignores empty comments. If it works, the comment “GEN RECEP 6/” will show up in your output.

**Solution:**



--In comments.hive, insert the following line between your select and group statements:

where info\_comment != ""

Save the changes, then back at the command line, re-run the query:



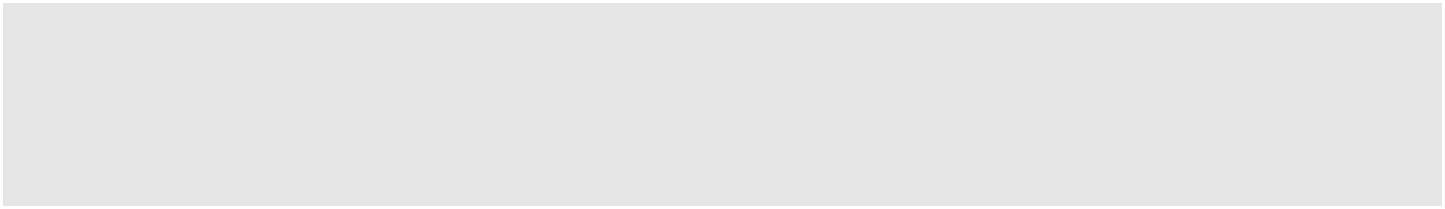
# hive -f comments.hive



4 ) Least Frequent Comment

1. Run the previous query again, but this time, find the 10 least occurring comments.

--Remove DESC from your order statement so that it looks like this:



order by comment\_count

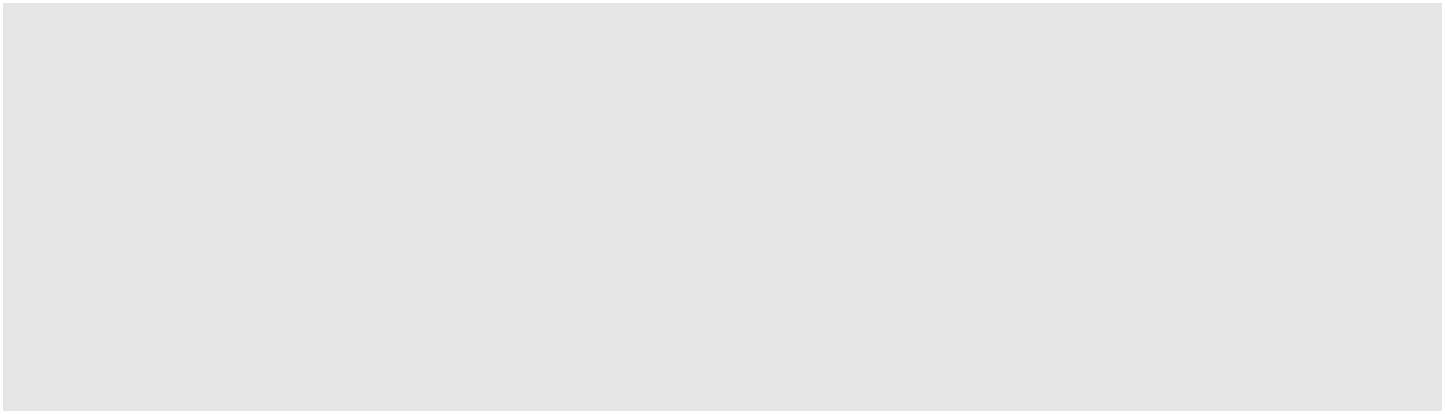
Save the changes, then back at the command line, re-run the query:



# hive -f comments.hive

The output should look something like:

1. CONGRESSIONAL BALL/
2. CONG BALL/
3. merged to u59031
4. CONGRESSIONAL BALL
5. CONG BALL
6. COMMUNITY COLLEGE SUMMIT
7. 48 HOUR WAVE EXCEPTION GRANTED
8. DROP BY VISIT
9. WHO EOP/
10. "POTUS LUNCH WITH WASHINGTON



This seems accurate since 1 is the least number of times a comment can appear.

5 ) Analyze the Data Inconsistencies

1. Analyzing the results of the most- and least-frequent comments, it appears that several variations of GENERAL RECEPTION occur. In this step, you will try to determine the number of visits to the POTUS involving a general reception by trying to clean up some of these inconsistencies in the data.

**Note**

Inconsistencies like these are very common in big data, especially when human input is involved. In this dataset, we likely have different people entering similar comments but using their own abbreviations.

1. Modify the query in comments.hive. Instead of searching for empty comments. Search for comments that contain variations of the string

“GEN RECEP.”



where info\_comment rlike '.\*GEN.\*\\s+RECEP.\*'



* 1. Change the limit clause from 10 to 30:

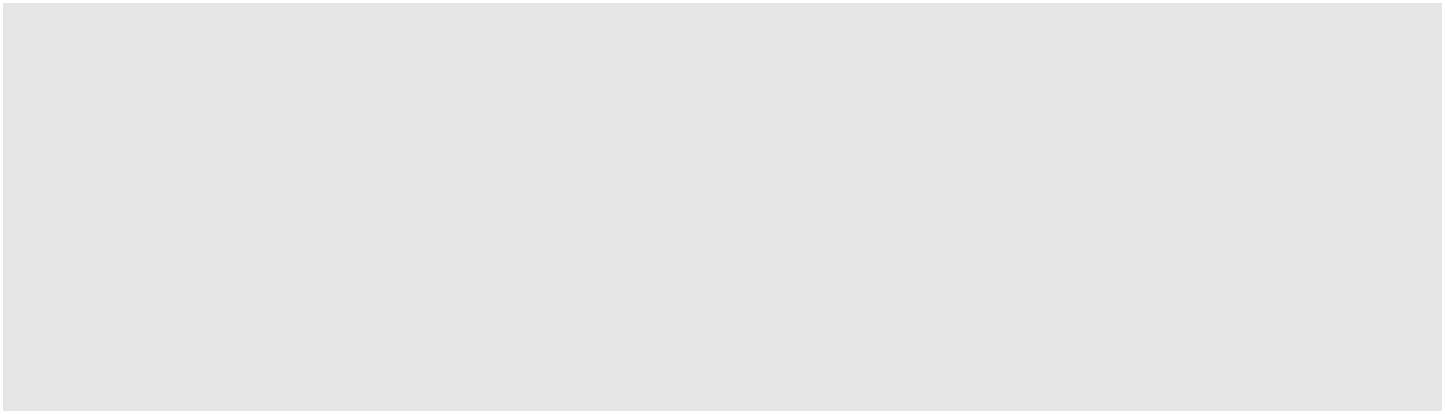
limit 30;

* 1. Run the query again.

1. hive -f comments.hive
   1. Notice there are several GENERAL RECEPTION entries that only differ by a number at the end or use the GEN RECEP abbreviation:



1. GENERAL RECEPTION 1
2. GEN RECEP 5/
3. GENERAL RECEPTION 3
4. GEN RECEP 6/
5. GEN RECEP 4
6. GENERAL RECEPTION 2
7. GENERAL RECEPTION 3
8. GENERAL RECEPTION 6
9. GENERAL RECEPTION 5
10. GENERAL RECEPTION 1



1. Let’s try one more query to try and narrow GENERAL RECEPTION visit. Modify the WHERE clause in comments.hive to include “%GEN%”:

where info\_comment like "%RECEP%"



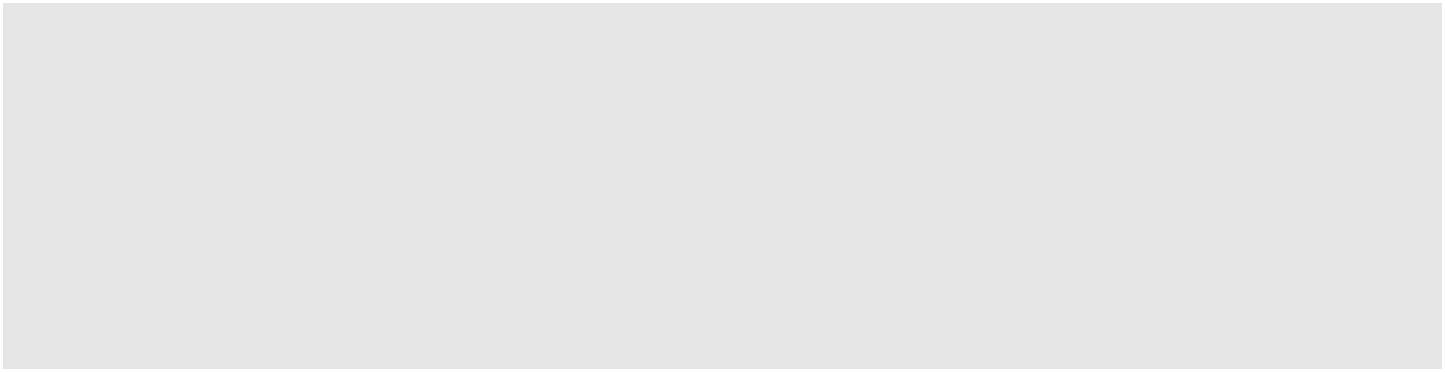
and info\_comment like "%GEN%"

* 1. Leave the limit at 30, save your changes, and run the query again.

1. hive -f comments.hive
   1. The output this time reveals all the variations of GEN and RECEP. Next, let’s add up the total number of them by running the following query:



from wh\_visits



select count(\*)

where info\_comment like "%RECEP%"

and info\_comment like "%GEN%";

--Then save your changes and run the query again from the command line:

1. hive -f comments.hive
   1. **.** Notice there are 2,697 visits to the POTUS withGEN RECEPin thecomment field, which is about 12% of the 21,819 total visits to the POTUS in our dataset.

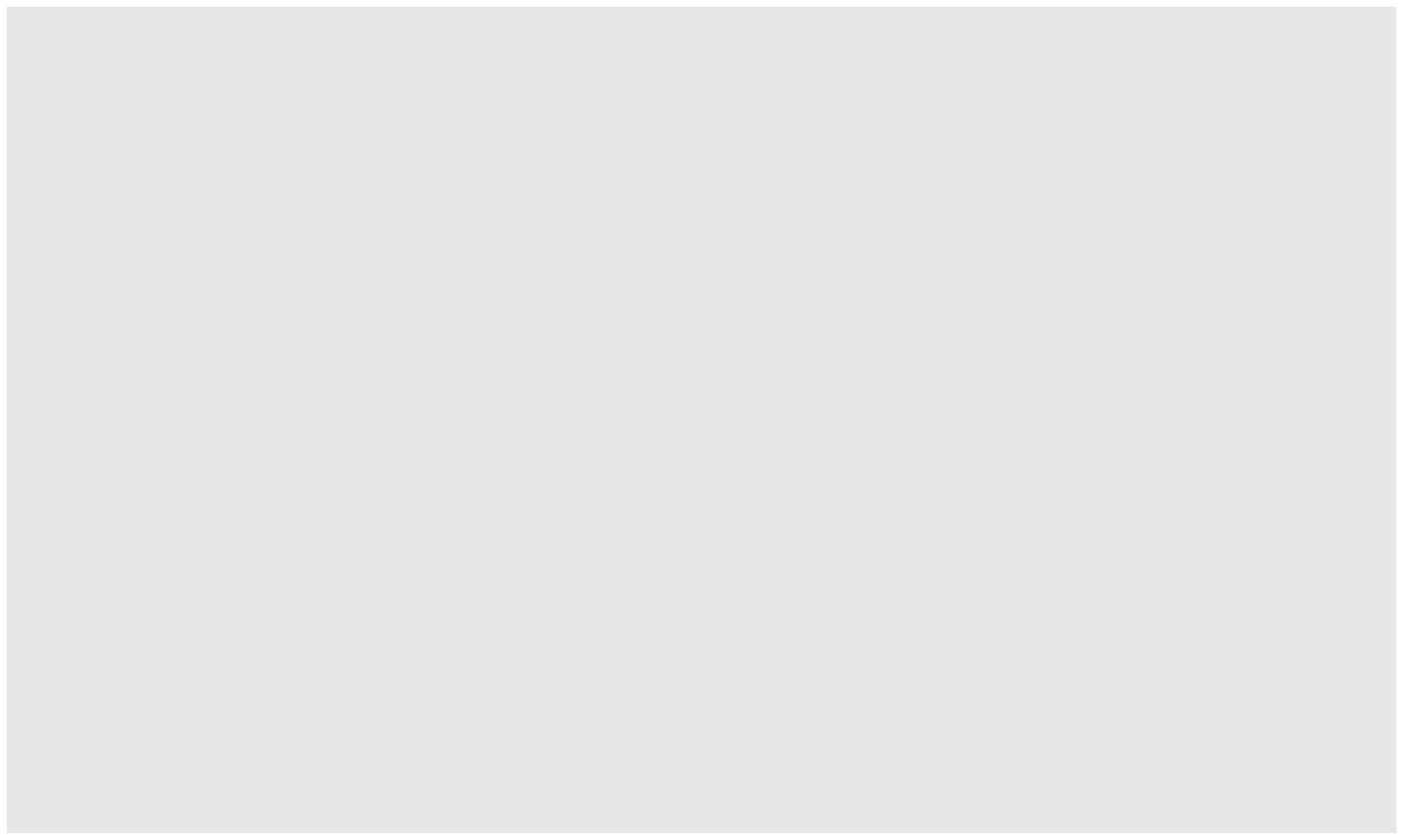


**Note**

More importantly, these results show that the conclusion from our first query, where we found that the most likely reason to visit the President was the HOLIDAY BALL with 1,253 attendees, is incorrect. This type of analysis is common in big data, and it shows how data analysts need to be creative and thorough when researching their data.

6 ) Verify the Result

1. We have 12% of visitors to the POTUS going for a general reception, but there were a lot of statements in the comments that contained WHO and EOP. Modify the query from the last step and display the top 30 comments that contain “WHO” and “EOP.”



--You should be able to undo changes to comments.hive and restore it to the state before the last lab. Then make the following two additional edits:

--Change the where clause to match WHO and EOP

where info\_comment like "%WHO%"

and info\_comment like "%EOP%";

--Add the DESC command back to the end of the order statement order by comment\_count DESC

--Finally, double-check select count(\*) as comment\_count, info\_count

--Make sure the "as..." portion is there

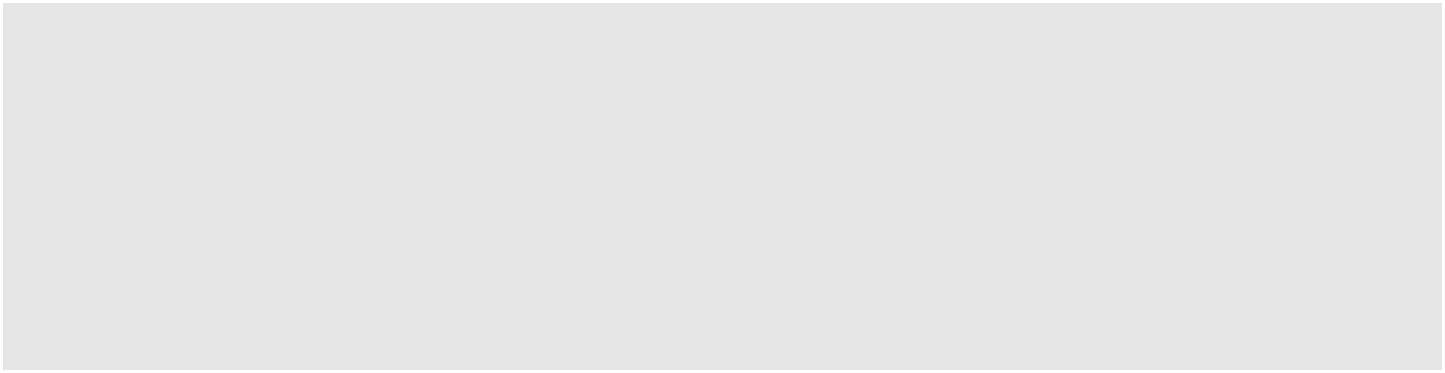
--Then save your changes and run the query again from the command line:

# hive -f comments.hive



The result should look like:

894 WHO EOP RECEP 2

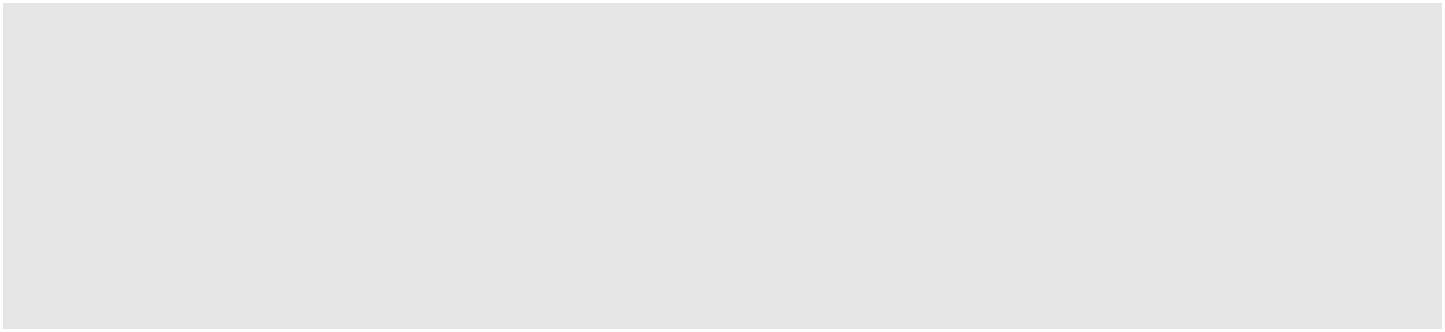


1. WHO EOP 1 RECEPTION/
2. WHO EOP RECEP/
3. WHO EOP HOLIDAY RECEP/
4. WHO/EOP #2/
5. WHO EOP RECEPTION
6. WHO EOP RECEP

|  |  |  |
| --- | --- | --- |
| 1 | WHO | EOP/ |
| 1 | WHO | EOP RECLEAR |

1. Modify the script again, this time to run a query that counts the number of records with WHO and EOP in the comments, and run the query:

from wh\_visits



select count(\*)

where info\_comment like "%WHO%"

and info\_comment like "%EOP%";

--Run the query from the command line:

# hive -f comments.hive

You should get 1,687 visits, or 7.7% of the visitors to the POTUS. So GENERAL RECEPTION still appears to be the most frequent comment.

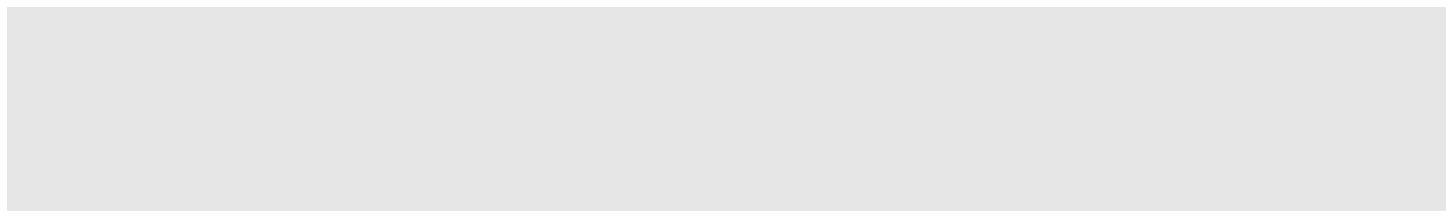
7 ) Find the Most Visits

1. See if you can write a Hive script that finds the top 20 individuals who visited the POTUS most. Use the Hive command from Step 3 earlier in this lab as a guide.

**Tip**

Use a grouping by both fname and lname.

The following script will accomplish the intention of the previous step:



from wh\_visits

select count(\*) as most\_visit, fname, lname group by fname, lname

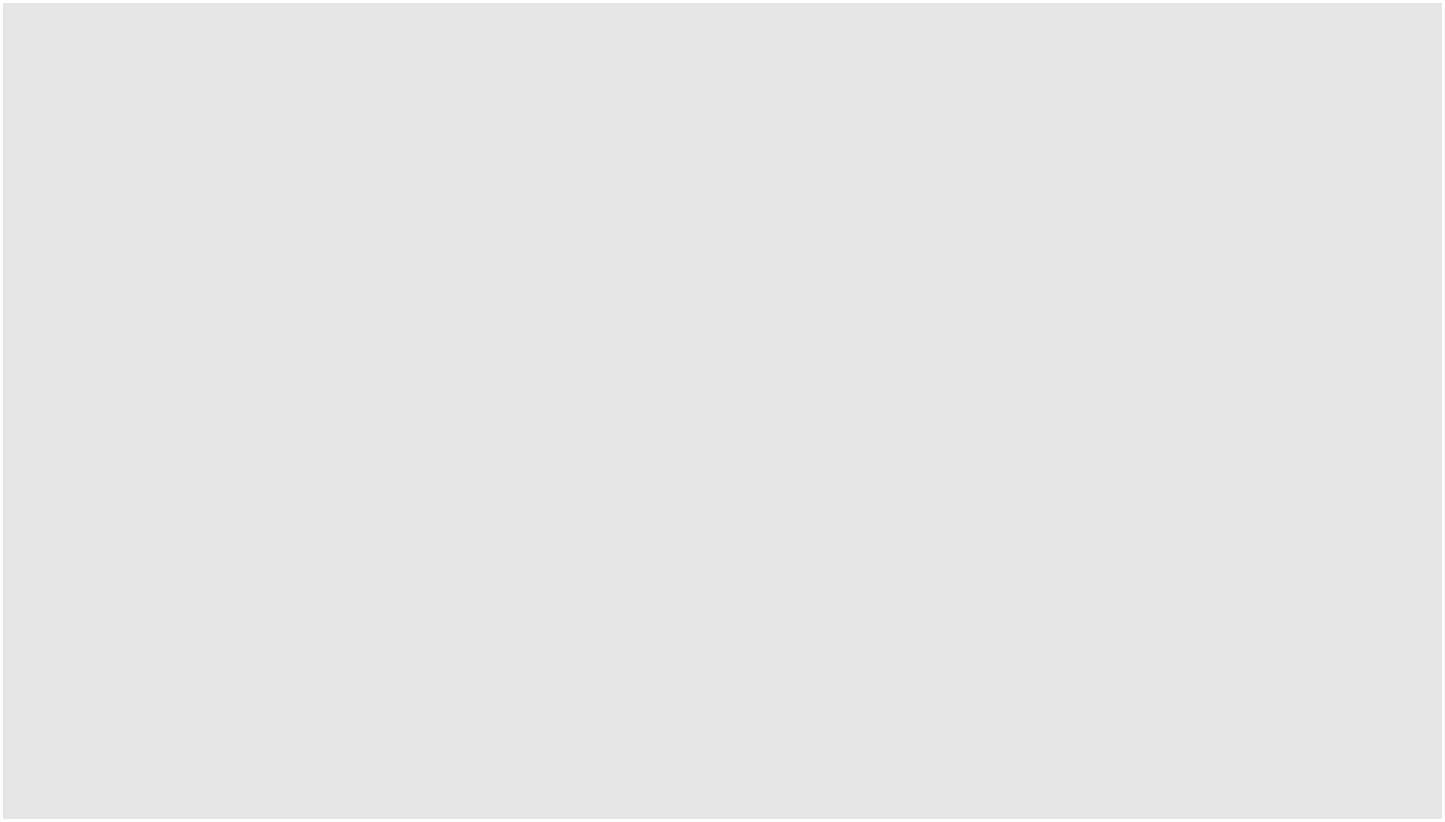
order by most\_visit DESC

limit 20;



To verify that your script worked, here are the top 20 individuals who visited the POTUS along with the number of visits (your output may vary slightly due to randomization of names):

16 ALAN PRATHER



15 CHRISTOPHER FRANKE

15 ANNAMARIA MOTTOLA

14 ROBERT BOGUSLAW

14 CHARLES POWERS

12 SARAH HART

12 JACKIE WALKER

12 JASON FETTIG

12 SHENGTSUNG WANG

12 FERN SATO

12 DIANA FISH

11 JANET BAILEY

11 PETER WILSON

11 GLENN DEWEY

11 MARCIO BOTELHO

11 DONNA WILLINGHAM

10 DAVID AXELROD

10 CLAUDIA CHUDACOFF

10 VALERIE JARRETT

10 MICHAEL COLBURN

**Result**

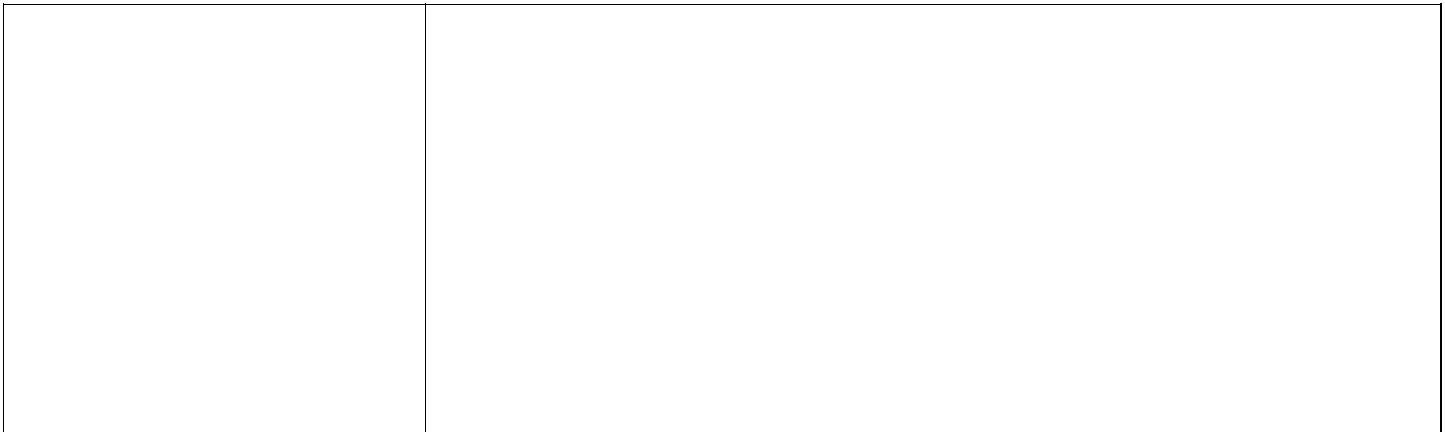
You have written several Hive queries to analyze the White House visitor data. The goal is for you to become comfortable with working with Hive, so hopefully you now feel like you can tackle a Hive problem and be able to answer questions about your big data stored in Hive.



**Lab: Writing a Hive UDF**

**Write a Hive User Defined Function (UDF)**

*Table 28.* *About this Lab*



|  |  |  |
| --- | --- | --- |
| **Objective:** |  | Write a Hive User Defined Function (UDF). |
|  |  |  |
| **File locations:** |  | /root/hdp/Java/labs/workspace/Lab11.1 |
|  |  |  |
| **Successful outcome:** |  | A Hive UDF that computes the money flow of a collection of stock |
|  |  | prices. |
|  |  |  |
| **Before you begin** |  | Start Eclipse. |
|  |  |  |
| **Related lesson:** |  | Hive Programming |
|  |  |  |

**Perform the following steps:**

**Step 1:** Define a Hive Table

**1.1.** Createa folder Lab11.1 in /root/hdp/Java/labs/workspace From a putty or git-bash terminal window. SCP the file importstock.hive from Techsoft\Courses\Java\sandbox\workspace\Hive to /root/hdp/Java/labs/workspace/Lab11.1 , change directories to~/hdp/Java/labs/workspace/Lab11.1/.View hive file’s contents and see if you can determine what it does:

**# more importstocks.hive**

**1.2.** Run theimportstocks.hivescript using the following command:

**# hive -f importstocks.hive**

**1.3.** Verify data is in thestockstable using the Hive CLI:

**# hive -e “SELECT \* FROM stocks LIMIT 100”**

**Step 2:** View the Project

**2.1.** copy a projectHivefromTechsoft\Courses\Java\sandbox\workspace\ toTechsoft\Courses\Java\eclipse-workspaceIn Eclipse, import and expand the folder of the Hive project.

**2.2.** Define a new package namedhiveudfs.

**Step 3:** Write a UDF Class

**3.1.** Add a class namedMoneyFlowto thehiveudfspackage that extends theUDFclass.

**3.2.** Add anevaluatemethod that has the following signature:

**public DoubleWritable evaluate(double high, double low, double close, int volume)**



**Note:** Useorg.apache.hadoop.hive.serde2.io.DoubleWritablefor thereturn type of the evaluate method above instead of the version in org.apache.hadoop.io.



**3.3.** Compute and return the money flow fromevaluate, which is computed with thefollowing formulas:

**typical price = (high + low + close) / 3 money flow = typical price x volume**

**3.4.** Save your changes toMoneyFlow.java.

**Step 4:** Build the JAR

**4.1.** Build the project to createhiveudfs.jar. SCP the jar to Lab11.1 on sandbox

**Step 5:** Write a Hive Script

**5.1.** Create a new text file in/root/hdp/Java/workspace/Lab11.1namedmoneyflow.hive.

**5.2.** Use theADD JARcommand to registerhiveudfs.jar.

**5.3.** Define yourhiveudfs.MoneyFlowclass as a temporary function named‘moneyflow’.

**5.4.** Write aSELECTquery that retrieves thesymbol,priceDateandmoneyflow(byinvoking the moneyflow function) from the stocks table:

**SELECT symbol, pricedate, moneyflow(high, low, close, volume) FROM stocks LIMIT 10000;**

**5.5.** Save your changes tomoneyflow.hive.

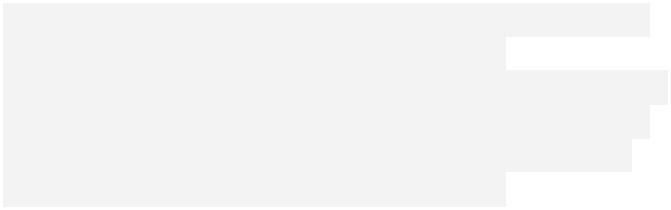
**Step 6:** Run the Hive Script

**6.1.** Run the script using the following command:

**# hive -f moneyflow.hive**

**6.2.** To verify, the end of your output should look like the following:

**AIT** **1993-09-10 856916.0099029541**



**AIT** **1993-09-09 1.72985E7**

**AIT** **1993-09-08 1258016.6666666667**

**AIT** **1993-09-07 6563453.232383728**

**AIT** **1993-09-03 94480.0033569336**

**AIT** **1993-09-02 1306400.0**

**Result:** In this lab, you defined a Hive table namedstocksand populated it with data from aCSV file. Then you wrote a Hive UDF to compute the money flow of the data in stocks, and tested the UDF with a Hive script.

**Demonstration: Computing ngrams**

**(Note - This demo works iff set hive.execution.engine=mr)**

**About this Demonstration**

**Objective:** To understand how to compute ngrams using Hive.

**During this** Watch as your instructor performs the following steps.

**demonstration:**

**Related** ***Hive Programming***

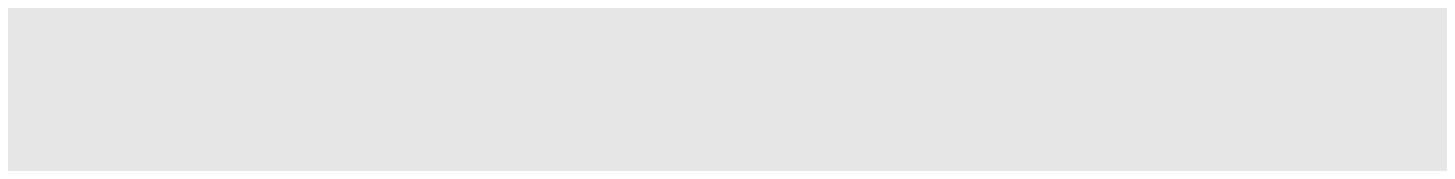
**lesson:**

**Demonstration Steps**

1 ) Create a Hive Table for the Data

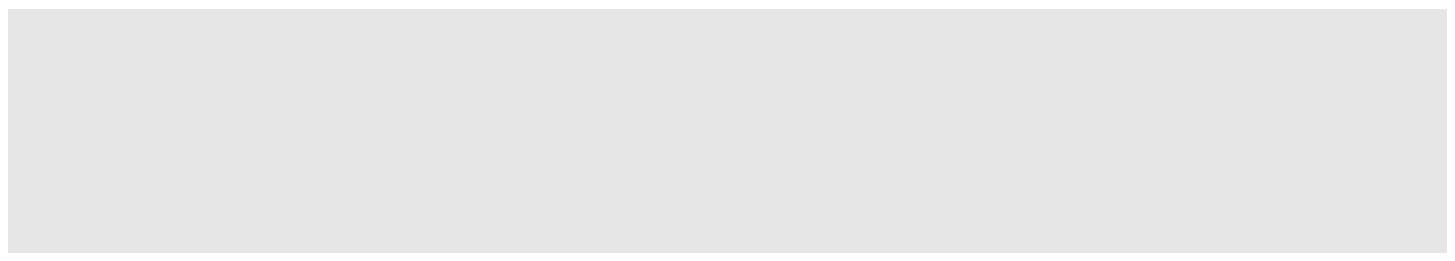
* 1. If not already done, open a putty or git-bash Terminal to sandbox.
  2. This demonstration computes ngrams on the U.S. Constitution, which is in a text file in the /root/devph/labs/demos folder:

1. cd ~/hdp/pigandhive/labs/demos/
2. more constitution.txt



--press q to exit more

c. Start the Hive shell and define the following table:



# hive

hive> create table constitution (

line string

)

ROW FORMAT DELIMITED;

Each line of text in the text file is going to be a record in our Hive table.

2 ) Load the Hive Table

a. Load constitution.txt into the constitution table:



hive> load data local inpath

'/root/hdp/pigandhive/labs/demos/constitution.txt' into table constitution;

1. Verify that the data is loaded:

hive> select \* from constitution;

You should see the contents of constitution.txt again.



|  |  |
| --- | --- |
|  |  |

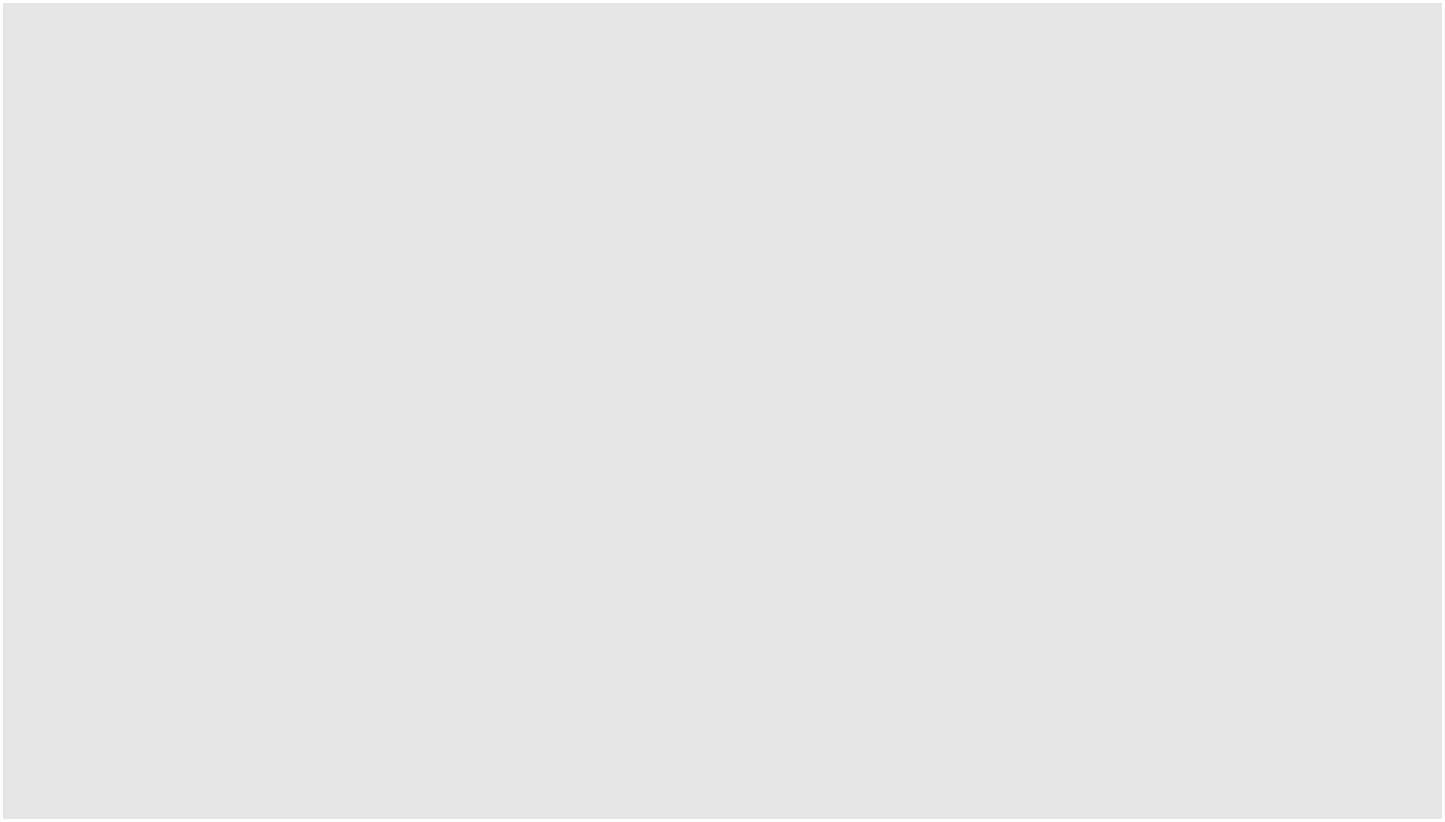
3 ) Compute a Bigram

hive> select explode(ngrams(sentences(line),2,15)) as x from constitution;

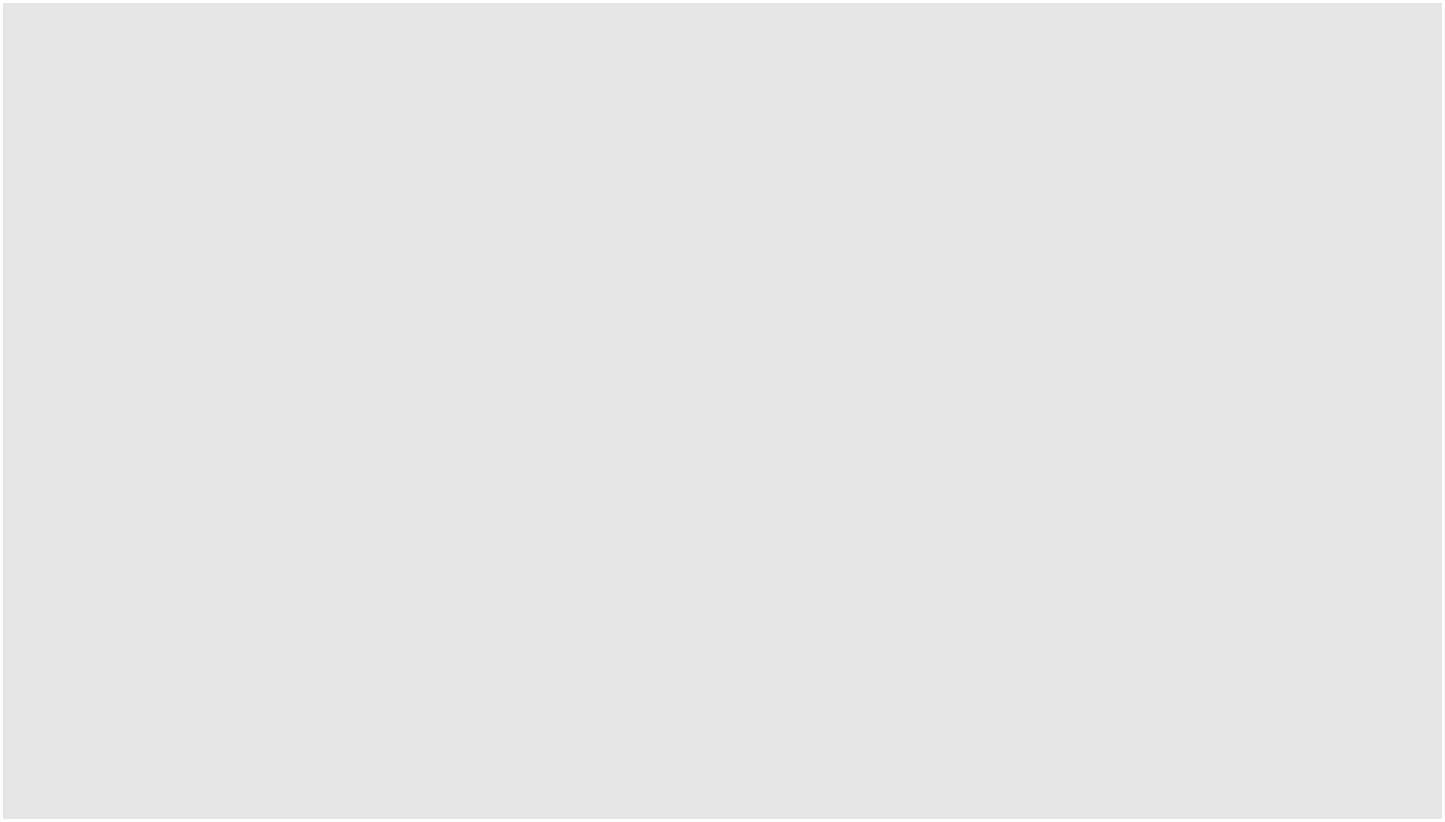


The result should look similar to:

{"ngram":["of","the"],"estfrequency":194.0} {"ngram":["shall","be"],"estfrequency":100.0} {"ngram":["the","United"],"estfrequency":76.0} {"ngram":["United","States"],"estfrequency":76.0} {"ngram":["to","the"],"estfrequency":57.0} {"ngram":["shall","have"],"estfrequency":44.0} {"ngram":["the","President"],"estfrequency":30.0} {"ngram":["shall","not"],"estfrequency":29.0} {"ngram":["in","the"],"estfrequency":28.0} {"ngram":["by","the"],"estfrequency":25.0} {"ngram":["the","Congress"],"estfrequency":22.0} {"ngram":["and","the"],"estfrequency":21.0} {"ngram":["for","the"],"estfrequency":21.0} {"ngram":["Vice","President"],"estfrequency":21.0} {"ngram":["the","Senate"],"estfrequency":21.0} {"ngram":["States","and"],"estfrequency":20.0} {"ngram":["States","shall"],"estfrequency":19.0} {"ngram":["any","State"],"estfrequency":18.0} {"ngram":["Congress","shall"],"estfrequency":18.0} {"ngram":["on","the"],"estfrequency":17.0}



4 ) Compute a Trigram



{"ngram":["the","United","States"],"estfrequency":68.0}

{"ngram":["of","the","United"],"estfrequency":51.0} {"ngram":["shall","not","be"],"estfrequency":16.0} {"ngram":["of","the","Senate"],"estfrequency":14.0} {"ngram":["States","shall","be"],"estfrequency":13.0} {"ngram":["House","of","Representatives"],"estfrequency":13.0} {"ngram":["United","States","shall"],"estfrequency":13.0} {"ngram":["shall","have","been"],"estfrequency":12.0} {"ngram":["the","several","States"],"estfrequency":12.0} {"ngram":["President","of","the"],"estfrequency":11.0} {"ngram":["United","States","and"],"estfrequency":11.0} {"ngram":["The","Congress","shall"],"estfrequency":10.0} {"ngram":["the","House","of"],"estfrequency":10.0} {"ngram":["United","States","or"],"estfrequency":10.0} {"ngram":["Congress","shall","have"],"estfrequency":10.0} {"ngram":["the","Vice","President"],"estfrequency":9.0} {"ngram":["of","the","President"],"estfrequency":8.0} {"ngram":["Consent","of","the"],"estfrequency":8.0} {"ngram":["shall","be","the"],"estfrequency":7.0} {"ngram":["by","the","Congress"],"estfrequency":7.0}

5 ) Compute a Contextual ngram

a. Let’s find the 20 most frequent words that follow “the”:

hive> select explode(context\_ngrams(sentences(line), array("the",null),20)) as result

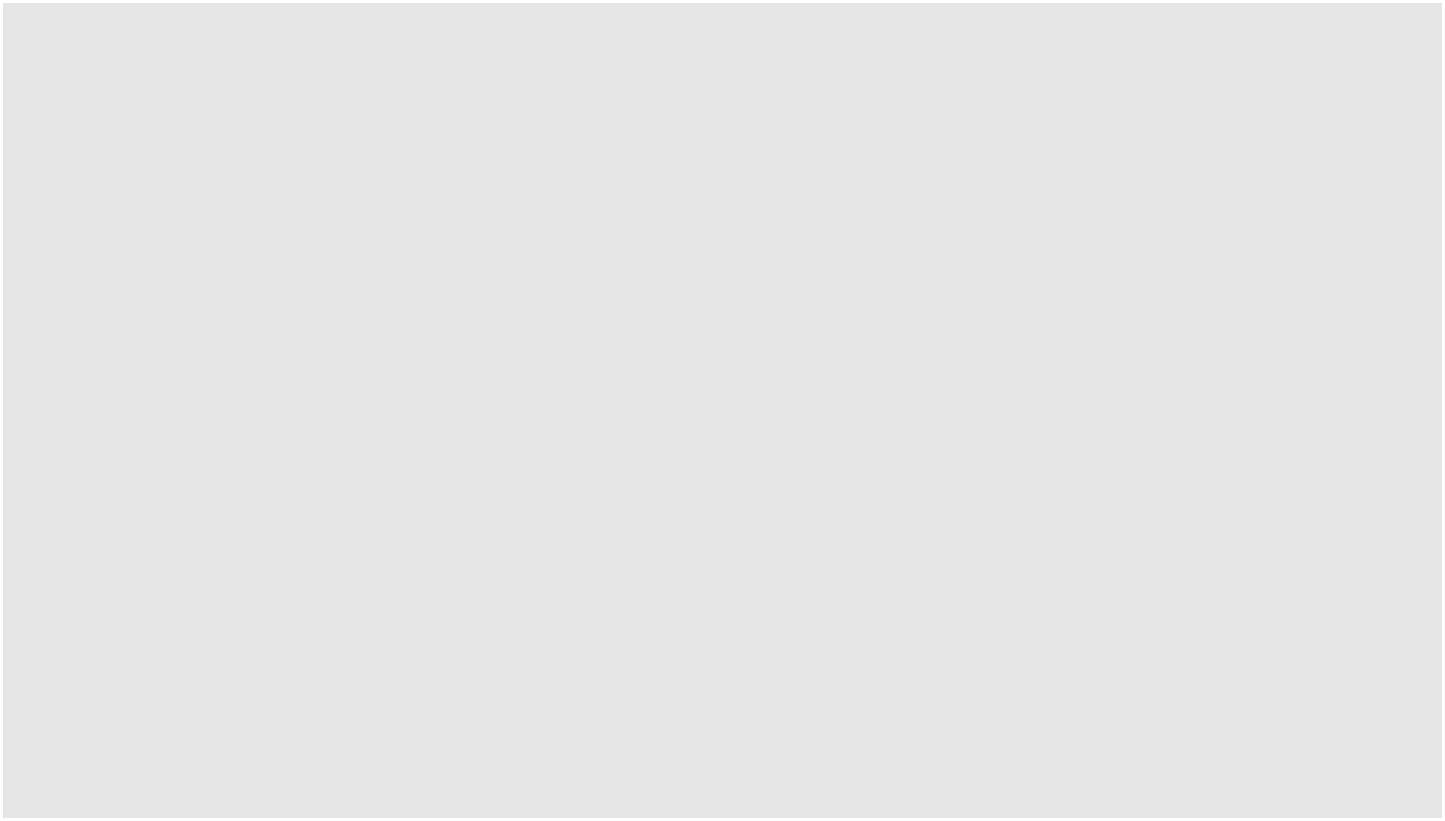


from constitution;



b. The result should look similar to:

{"ngram":["United"],"estfrequency":76.0} {"ngram":["President"],"estfrequency":30.0} {"ngram":["Congress"],"estfrequency":22.0} {"ngram":["Senate"],"estfrequency":21.0} {"ngram":["several"],"estfrequency":15.0} {"ngram":["Vice"],"estfrequency":12.0} {"ngram":["State"],"estfrequency":11.0} {"ngram":["same"],"estfrequency":10.0} {"ngram":["Constitution"],"estfrequency":10.0} {"ngram":["States"],"estfrequency":10.0} {"ngram":["House"],"estfrequency":10.0} {"ngram":["whole"],"estfrequency":10.0} {"ngram":["office"],"estfrequency":9.0} {"ngram":["right"],"estfrequency":8.0} {"ngram":["Legislature"],"estfrequency":8.0} {"ngram":["Consent"],"estfrequency":6.0} {"ngram":["powers"],"estfrequency":6.0} {"ngram":["supreme"],"estfrequency":6.0} {"ngram":["people"],"estfrequency":6.0} {"ngram":["first"],"estfrequency":6.0}



**Lab: Joining Datasets in Hive**

**About this Lab**

**Objective:** Perform a join of two datasets in Hive.

**File locations:** /root/hdp/pigandhive/devph/labs/Lab7.4

**Successful outcome:** A table named stock\_aggregates that contains a join of NYSE stock prices with the stock’s dividend prices.

**Before you begin:** Your HDP 2.6 cluster should be up and running within your VM.

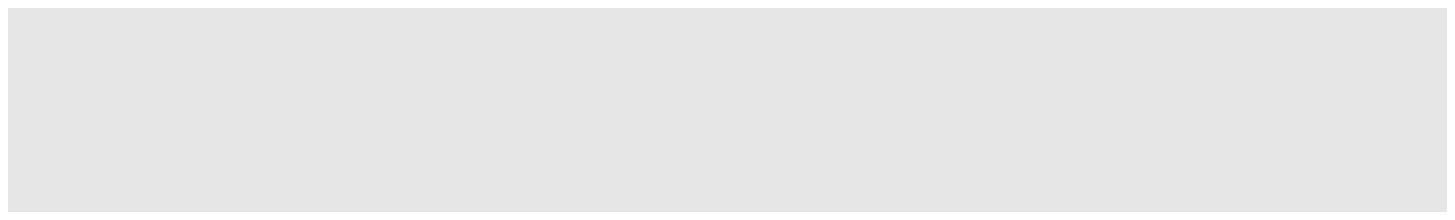
**Related lesson:** ***Hive Programming***

**Lab Steps**

1 ) Load the Data into Hive

* 1. If not already done, open a putty or git-bashTerminal to your sandbox.
  2. View the contents of the file setup.hive in /root/hdp/pigandhive/labs/Lab7.4:

1. cd ~/hdp/pigandhive/labs/Lab7.4/
2. more setup.hive
   1. Notice that this script creates three tables in Hive. The nyse\_data table is filled with the daily stock prices of stocks that start with the letter K and the dividends table that contains the quarterly dividends of those stocks. The stock\_aggregates table is going to be used for a join of these two datasets and contain the stock price and dividend amount on the date the dividend was paid.
   2. Run the setup.hive script from the Joining Datasets in Hive lab folder:
3. hive -f setup.hive
   1. To verify that the script worked, enter the Hive Shell and run the following following queries:
4. hive



hive> select \* from nyse\_data limit 20;

hive> select \* from dividends limit 20;

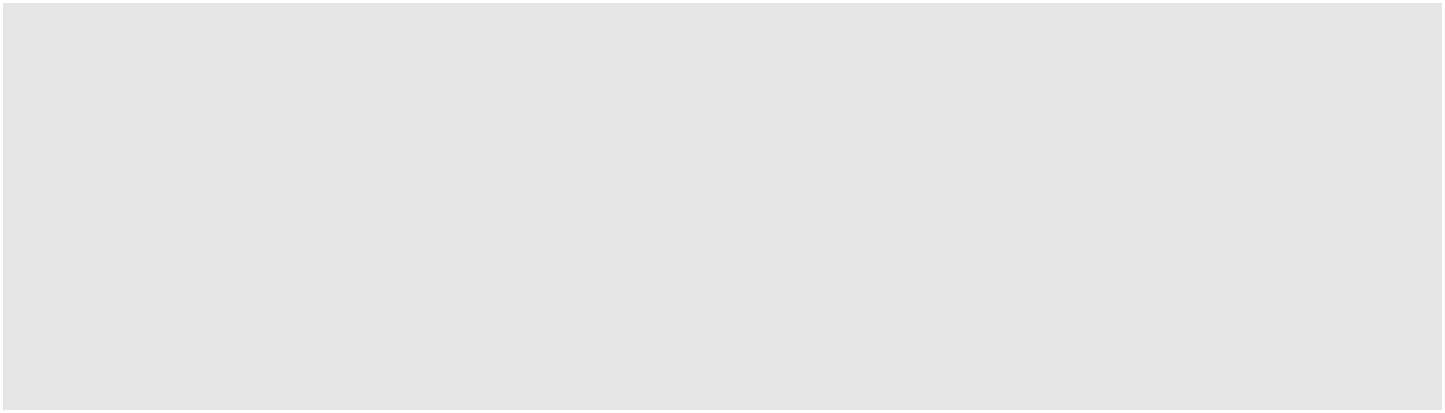
You should see daily stock prices and dividends from stocks that start with the letter K.



|  |  |
| --- | --- |
|  |  |

1. The stock\_aggregates table should be empty, but view its schema to verify that it was created successfully, then type quit to exit the Hive Shell:

hive> describe stock\_aggregates;



|  |  |
| --- | --- |
| OK | string |
| symbol |
| year | string |
| high | float |
| low | float |
| average\_close | float |
| total\_dividends | float |
| hive> quit; |  |
| 2 ) Join the Datasets |  |

1. The join statement is going to be fairly long, so let’s create it in a text file. Use an editor to create a new text in the /root/hdp/pigandhive/devph/labs/Lab7.4/ folder

named join.hive.

1. We will break the join statement down into sections. First, the result of the join is going to be put into the stock\_aggregates table, which requires an insert:

insert overwrite table stock\_aggregates



The overwrite causes any existing data in stock\_aggregates to be deleted.

1. The data being inserted is going to be the result of a select query that contains various insightful indicators about each stock. The result is going to contain the stock symbol, date traded, maximum high for the stock, minimum low, average close, and the sum of dividends, as shown here:

select a.symbol, year(a.trade\_date), max(a.high), min(a.low), avg(a.close), sum(b.dividend)



1. The from clause is the nyse\_data table: from nyse\_data a
2. The join is going to be a left outer join of the dividends table: left outer join dividends b
3. The join is by stock symbol and trade date:



on (a.symbol = b.symbol and a.trade\_date = b.trade\_date)



1. Let’s group the result by symbol and trade date: group by a.symbol, year(a.trade\_date);
2. Save your changes to join.hive.



3 ) Run the Query

* 1. Run the query and wait for Tez to execute:

1. hive -f join.hive
   1. How many total mappers and reducers does it take to perform this query?

***Answer*:**Two mappers and one reducer.



4 ) Verify the Results

a. Enter the Hive Shell and run a select query to view the contents of

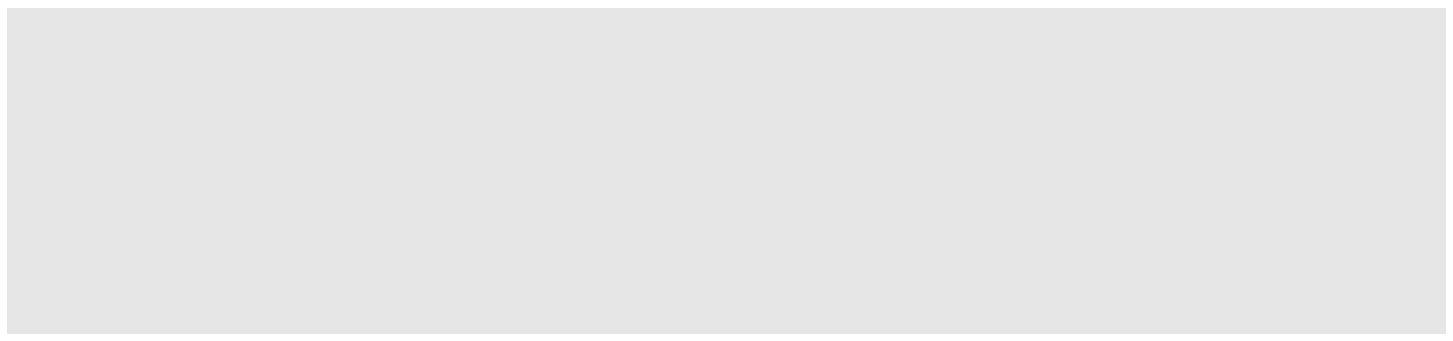
stock\_aggregates:



# hive

hive> select \* from stock\_aggregates;

The output should look like:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| KYO | 2004 | 90.9 | 66.25 | 75.79952 | 0.544 |
| KYO | 2005 | 78.45 | 62.58 | 72.042656 | 0.91999996 |
| KYO | 2006 | 98.01 | 71.73 | 85.80327 | 0.851 |
| KYO | 2007 | 110.01 81.09 | | 3.737686 | NULL |
| KYO | 2008 | 100.78 | 45.41 | 79.6098 | NULL |
| KYO | 2009 | 93.2 | 52.98 | 77.04389 | NULL |
| KYO 2010 93.83 85.94 90.71 | | | | | NULL |
| stock\_symbolNULLNULLNULLNULLNULL | | | | |  |

1. ) List the contents of the stock\_aggregates directory in HDFS. The 000000\_0 file was created as a result of the join query:



hive> dfs -ls -R /apps/hive/warehouse/stock\_aggregates/;

-rw-r--r-- 3 root hdfs 41109 /apps/hive/warehouse/stock\_aggregates/000000\_0

1. View the contents of the stock\_aggregates table using the cat command: hive> dfs -cat /apps/hive/warehouse/stock\_aggregates/000000\_0;



**Result**

The stock\_aggregates table is a joining of the daily stock prices and the quarterly dividend amounts on the date the dividend was announced, and the data in the table is an aggregate of various statistics like max high, min low, etc.



**Lab: Computing ngrams of Emails in Avro Format**

**About this Lab**

**Objective:**

Use Hive to compute ngrams.

**File locations:**

/root/hdp/pigandhive/labs/Lab7.5

**Successful outcome:**

A bigram of words found in a collection of Avro-formatted emails.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:**

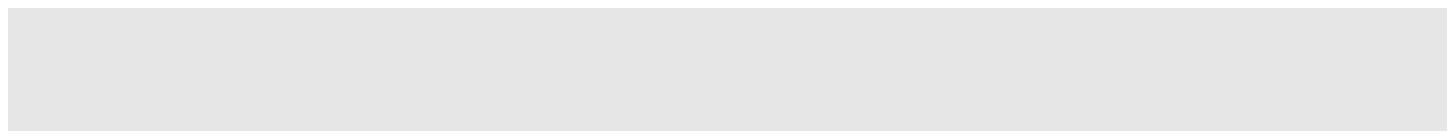
***Hive Programming***

**Lab Steps**

1 ) View an Avro Schema

* 1. If not already done, open a putty or git-bashTerminal to your sandbox.
  2. Change directories to the /root/hdp/pigandhive/labs/Lab7.5 folder. Notice this folder contains an Avro file named sample.avro.

1. cd ~/hdp/pigandhive/devph/labs/Lab7.5



1. ls sample.avro
2. Enter the following command to print the schema of the contents of

sample.avro:

1. 
2. Enter the following command to create the schema of the contents of

sample.avro:



1. 
   1. How many fields do records in sample.avro have? ***Answer*:**Four fields
   2. Put the schema file in HDFS:
2. hdfs dfs -put sample.avsc



2 ) Create a Hive Table from an Avro Schema

* 1. View the contents of the CREATE TABLE query defined in the

create\_sample\_table.hive file in your Computing ngrams of Emails in Avro Format lab folder:

1. more create\_sample\_table.hive
   1. Make sure the avro.schema.url property points to the schema file you created in the previous step:



WITH SERDEPROPERTIES (



'avro.schema.url'='hdfs:///user/root/sample.avsc')

* 1. Run the CREATE TABLE query:

1. hive -f create\_sample\_table.hive

3 ) Verify the Table

* 1. Start the Hive shell.

1. hive
   1. Run the show tables command and verify that you have a table named



sample\_table.

hive> show tables;



1. Run the describe command on sample\_table. Notice the schema for sample\_table matches the Avro schema from sample.avsc.

hive> describe sample\_table;

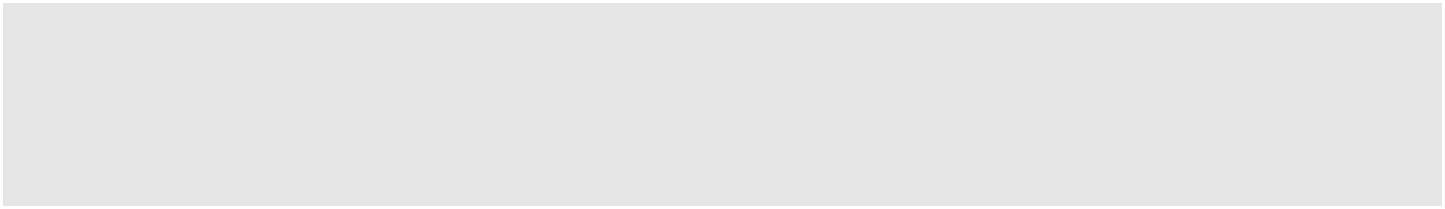


1. Let’s associate some data with sample\_table. Copy sample.avro into the Hive warehouse folder by running the following command (all on a single line):

hive> dfs -put /root/hdp/pigandhive/labs/Lab7.5/sample.avro /apps/hive/warehouse/sample\_table;

1. View the contents of sample\_table, then quit the Hive Shell:

hive> select \* from sample\_table;



|  |  |
| --- | --- |
| OK | 10, Bar Eggs Spam 800 |
| Foo 19 |

hive> quit;

Note that there is only one record in sample.avro. You have now seen how to create a Hive table using an Avro schema file. This was a simple example; next you will complete these steps using a large data file that contains emails in an Avro format.



4 ) Create an Email-User Table

* 1. There is an Avro file in your /root/devph/labs/Lab7.5 folder named mbox7.avro, which represents emails in an Avro format from a Hive mailing list for the month of July. Use the getschema option of java jar to view the schema of this file.
  2. 
  3. How many fields do records in mbox7.avro have? ***Answer:*** Four fields
  4. Run the getschema command again, but this time output the schema to a file named mbox.avsc:



* 1. Put the Avro schema file into /user/root in HDFS:

1. hdfs dfs -put mbox.avsc
   1. Use more to view the contents of the create\_email\_table.hive ;:script in

your /root/devph/labs/Lab7.5 folder. Verify the avro.schema.url property

is correct.

1. more create\_email\_table.hive
   1. Run the script to create the hive\_user\_email table:
2. hive -f create\_email\_table.hive
   1. Copy mbox7.avro into the warehouse directory:
3. hdfs dfs -put mbox7.avro /apps/hive/warehouse/hive\_user\_email
   1. Start the Hive shell and verify the table has data in it:
4. hive



hive> select \* from hive\_user\_email limit 20;

5 ) Compute a Bigram

a. Use the Hive ngrams function to create a bigram of the words in

mbox7.avro:



hive> select

ngrams(sentences(content),2 ,10)

from hive\_user\_email;



The output will be kind of a jumbled mess:

[{"ngram":["2013","at"],"estfrequency":802.0},{"ngram":["of","the"

],"estfrequency":391.0},{"ngram":["I","am"],"estfrequency":368.0},

{"ngram":["I","have"],"estfrequency":340.0},{"ngram":["J","E9r"],"

estfrequency":306.0},{"ngram":["for","the"],"estfrequency":291.0},

{"ngram":["you","are"],"estfrequency":289.0},{"ngram":["user","hiv



e.apache.org"],"estfrequency":289.0},{"ngram":["to","the"],"estfre

quency":276.0},{"ngram":["E9r","F4me"],"estfrequency":270.0}]

1. To clean this up, use the Hive explode function to display the output in a more readable format:

hive> select

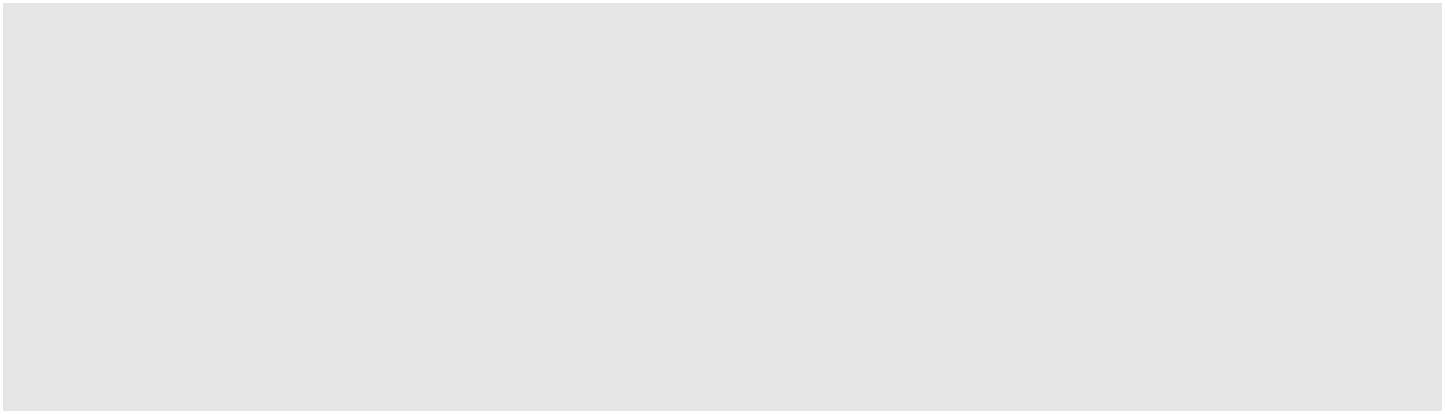


explode(ngrams(sentences(content),2 ,10))

from hive\_user\_email;

You should see a nice, readable list of 10 bigrams:

{"ngram":["2013","at"],"estfrequency":802.0} {"ngram":["of","the"],"estfrequency":391.0} {"ngram":["I","am"],"estfrequency":368.0} {"ngram":["I","have"],"estfrequency":340.0} {"ngram":["J","E9r"],"estfrequency":306.0} {"ngram":["for","the"],"estfrequency":291.0} {"ngram":["you","are"],"estfrequency":289.0} {"ngram":["user","hive.apache.org"],"estfrequency":289.0} {"ngram":["to","the"],"estfrequency":276.0} {"ngram":["E9r","F4me"],"estfrequency":270.0}



1. Typically when working with word comparison we ignore case. Run the query again, but this time add the Hive lower function and compute 20 bigrams:

hive> select



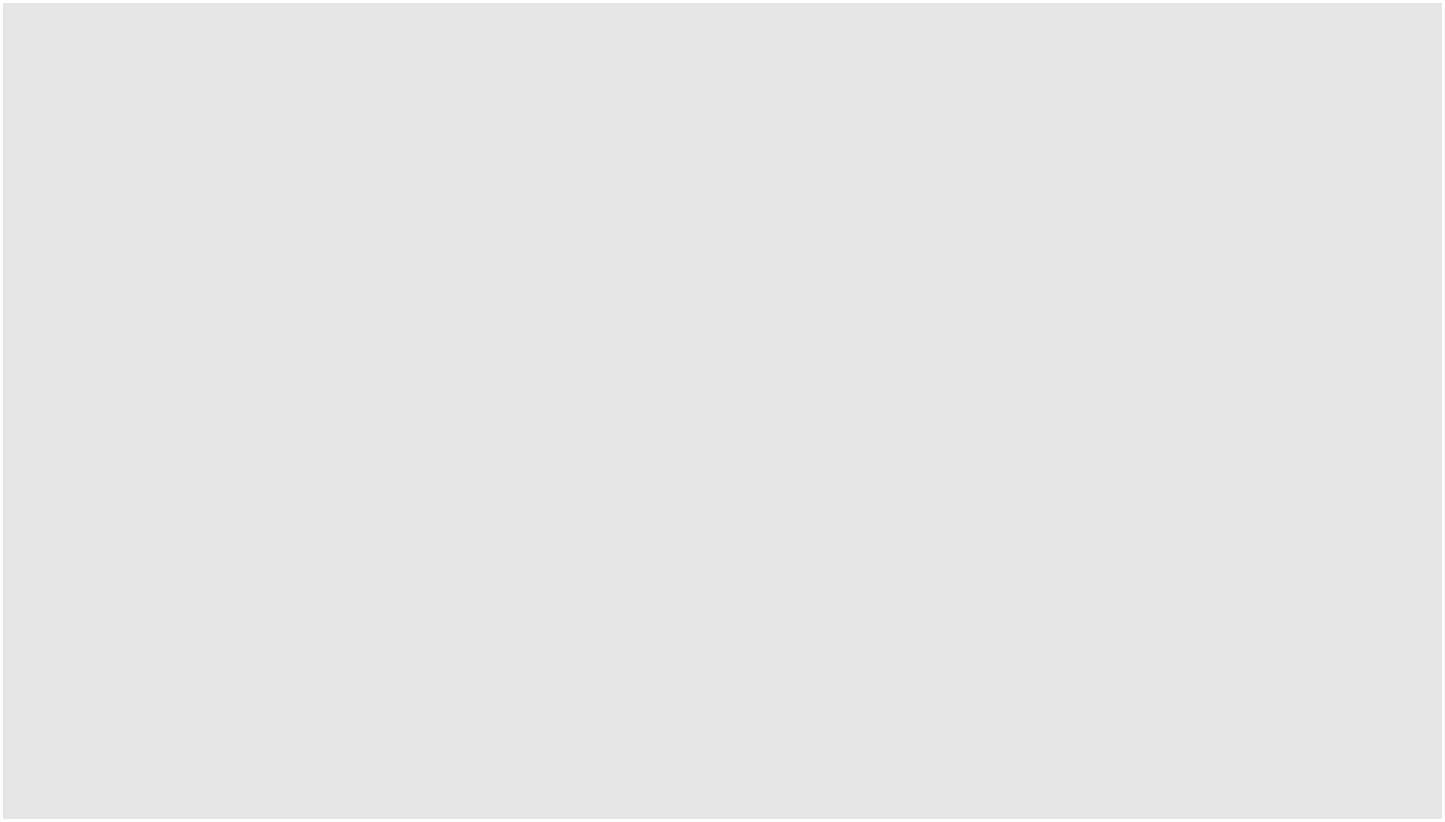
explode(ngrams(sentences(lower(content)),2 ,20))

from hive\_user\_email;



The output should look like the following:

{"ngram":["2013","at"],"estfrequency":802.0} {"ngram":["i","have"],"estfrequency":409.0} {"ngram":["of","the"],"estfrequency":391.0} {"ngram":["i","am"],"estfrequency":372.0} {"ngram":["if","you"],"estfrequency":347.0} {"ngram":["in","hive"],"estfrequency":337.0} {"ngram":["for","the"],"estfrequency":309.0} {"ngram":["j","e9r"],"estfrequency":306.0} {"ngram":["you","are"],"estfrequency":289.0} {"ngram":["user","hive.apache.org"],"estfrequency":289.0} {"ngram":["to","the"],"estfrequency":276.0} {"ngram":["outer","join"],"estfrequency":271.0} {"ngram":["2013","06"],"estfrequency":270.0} {"ngram":["e9r","f4me"],"estfrequency":270.0} {"ngram":["left","outer"],"estfrequency":270.0} {"ngram":["in","the"],"estfrequency":252.0} {"ngram":["gmail.com","wrote"],"estfrequency":248.0} {"ngram":["17","16"],"estfrequency":248.0} {"ngram":["06","17"],"estfrequency":246.0} {"ngram":["wrote","hi"],"estfrequency":234.0}



6 ) Compute a Context ngram

1. From the Hive shell, run the following query, which uses the

context\_ngrams function to find the top 20 terms that follow the word “error”:

hive> select



explode(context\_ngrams(sentences(lower(content)), array("error", null) ,20))

from hive\_user\_email;



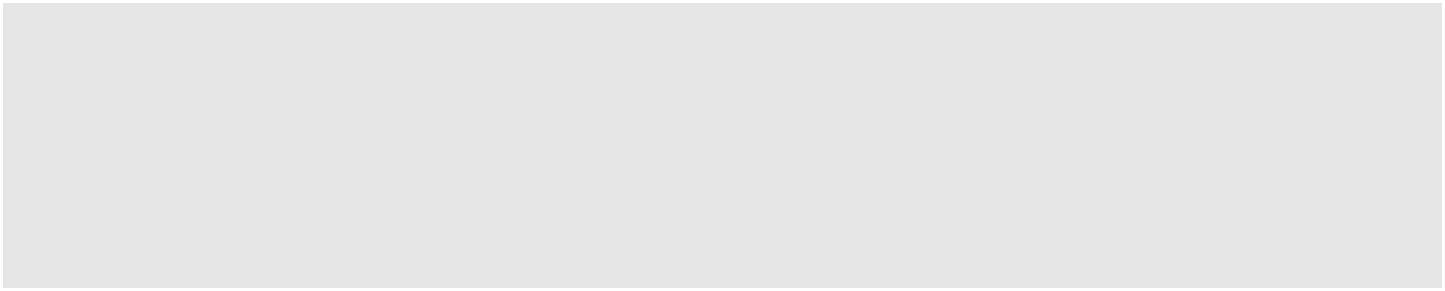
The output should look like the following:

{"ngram":["in"],"estfrequency":102.0}



{"ngram":["return"],"estfrequency":97.0}

{"ngram":["org.apache.hadoop.hive.ql.exec.udfargumenttypeexception



"],"estfrequency":49.0}

{"ngram":["failed"],"estfrequency":49.0}

{"ngram":["is"],"estfrequency":41.0}

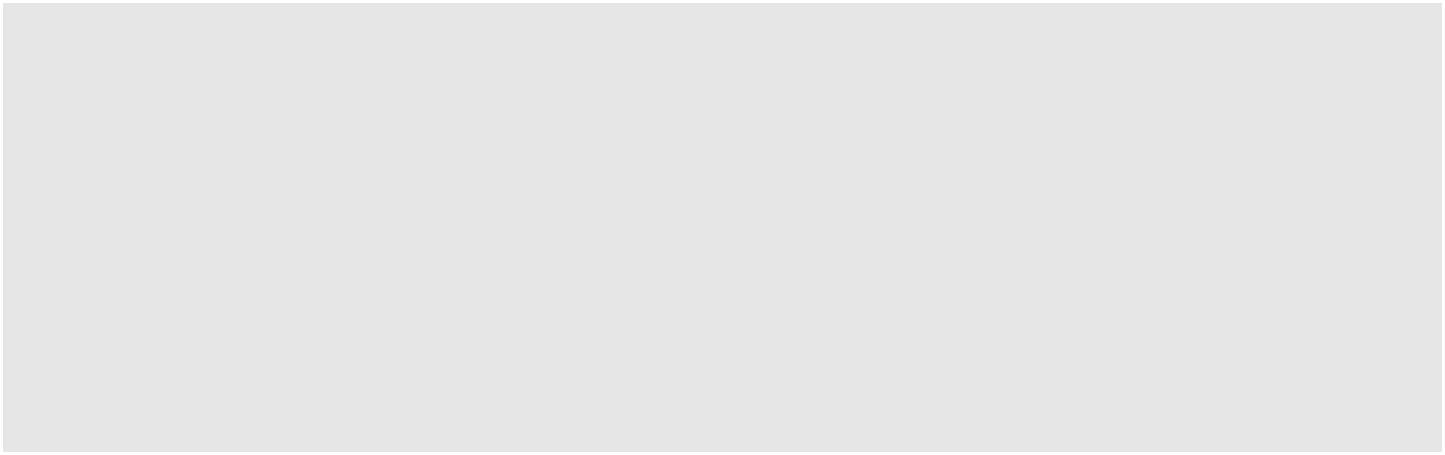
{"ngram":["message"],"estfrequency":40.0}

{"ngram":["when"],"estfrequency":39.0}

{"ngram":["please"],"estfrequency":36.0}

{"ngram":["while"],"estfrequency":28.0}

{"ngram":["org.apache.thrift.transport.ttransportexception"],"estf



requency":28.0}

{"ngram":["datanucleus.plugin"],"estfrequency":26.0}

{"ngram":["during"],"estfrequency":18.0}

{"ngram":["query"],"estfrequency":16.0}

{"ngram":["hive"],"estfrequency":16.0}

{"ngram":["could"],"estfrequency":16.0}

{"ngram":["java.lang.runtimeexception"],"estfrequency":13.0}

{"ngram":["13"],"estfrequency":12.0}

{"ngram":["error"],"estfrequency":12.0}

{"ngram":["exec.execdriver"],"estfrequency":10.0}

{"ngram":["exec.task"],"estfrequency":10.0}

1. What is the most likely word to follow “error” in these emails? ***Answer*:**“in”
2. Run a Hive query that finds the top 20 results for words in mbox7.avro that follow the phrase “error in.”

***Solution*:**



select

explode(context\_ngrams(sentences(lower(content)),

array("error", null, null) ,20))

from hive\_user\_email;

**Result**

You have written several Hive queries that computed bigrams based on the data in the mbox7.avro file. You should also be familiar with working with Avro files, a popular file format in Hadoop.





**Lab: Advanced Hive Programming**

**About this Lab**

**Objective:**

To understand how some of the more advanced features of Hive work, including multi-table inserts, views, and windowing.

**File locations:**

/root/hdp/pigandhive/devph/labs/Lab9.1

**Successful**

**outcome:**

You will have executed numerous Hive queries on customer order data.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:**

***Advanced Hive Programming***

**Lab Steps**

1 ) Create and Populate a Hive Table

* 1. If not already done, open a Terminal in your VM and type "ssh sandbox".
  2. From the command line, change directories to /root/devph/labs/Lab9.1 folder:

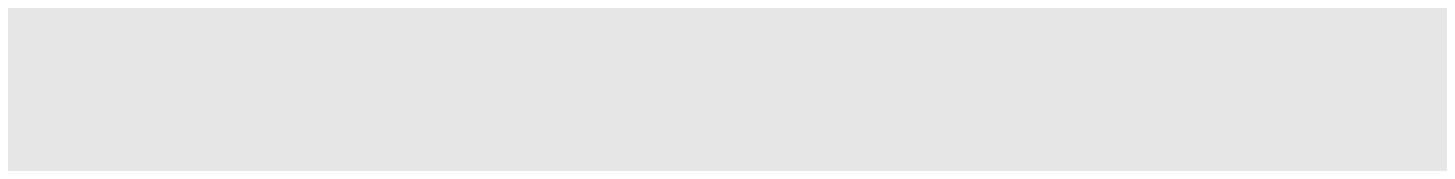
1. cd ~/hdp/pigandhive/devph/labs/Lab9.1
   1. View the contents of the orders.hive file in that folder:
2. more orders.hive



Notice it defines a Hive table named orders that has seven columns.

* 1. Execute the contents of orders.hive:

1. hive -f orders.hive
   1. From the Hive shell, verify that the script worked by running the following commands:
2. hive



hive> describe orders;

hive> select count(\*) from orders;

Your orders table should contain 99,999 records.



|  |  |
| --- | --- |
|  |  |

2 ) Analyze the Customer Data

1. Let’s run a few queries to see what this data looks like. Start by verifying that the username column actually looks like names:

hive> SELECT username FROM orders LIMIT 10;



You should see 10 first names.

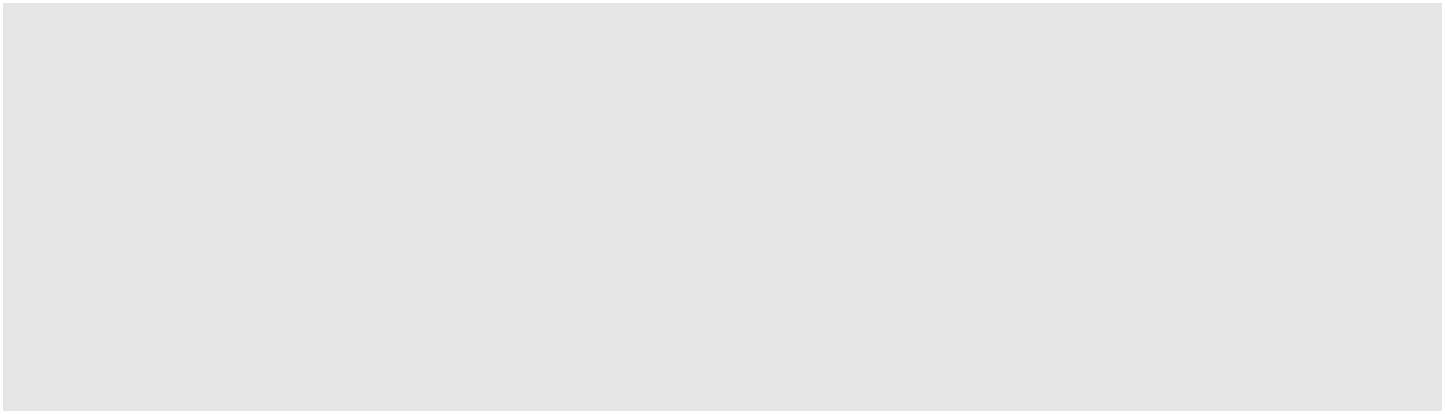
1. The orders table contains orders placed by customers. Run the following query, that shows the 10 lowest-price orders:

hive> SELECT username, ordertotal FROM orders ORDER BY ordertotal LIMIT 10;



The smallest orders are each $10, as you can see from the output:

Chelsea 10



Samantha 10

Danielle 10

Kimberly 10

Tiffany 10

Megan 10

Maria 10

Megan 10

Melissa 10

Christina 10

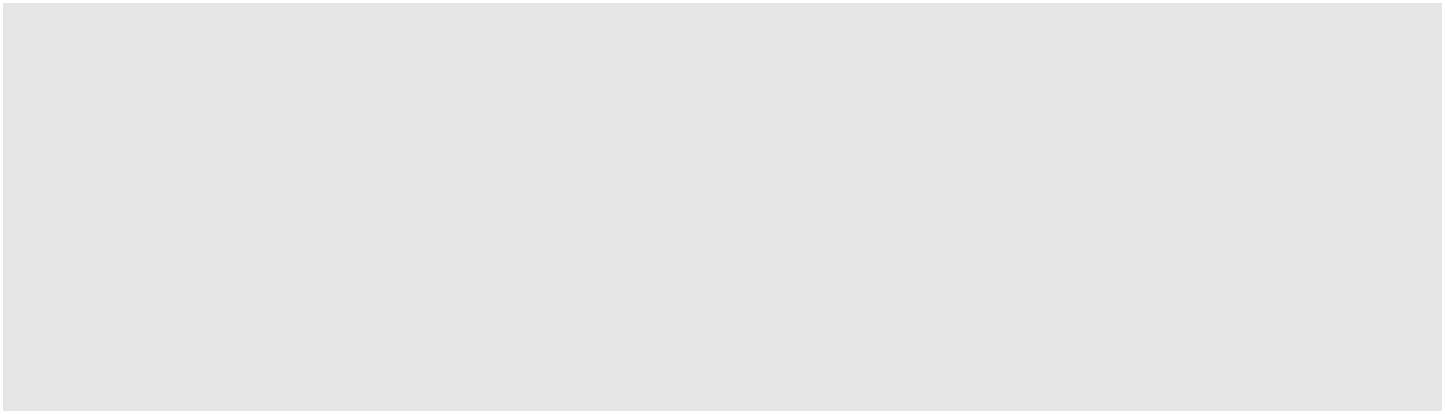
c. Run the same query, but this time use descending order:

hive> SELECT username, ordertotal FROM orders ORDER BY ordertotal DESC LIMIT 10;



The output this time is the 10 highest-priced orders:

Brandon 612



Mark 612

Sean 612

Jordan 612

Anthony 612

Paul 611

Jonathan 611

Eric 611

Nathan 611

Jordan 610

d. Let’s find out if men or women spent more money:

hive> SELECT sum(ordertotal), gender FROM orders GROUP BY gender;



Based on the output, which gender has spent more money on purchases?

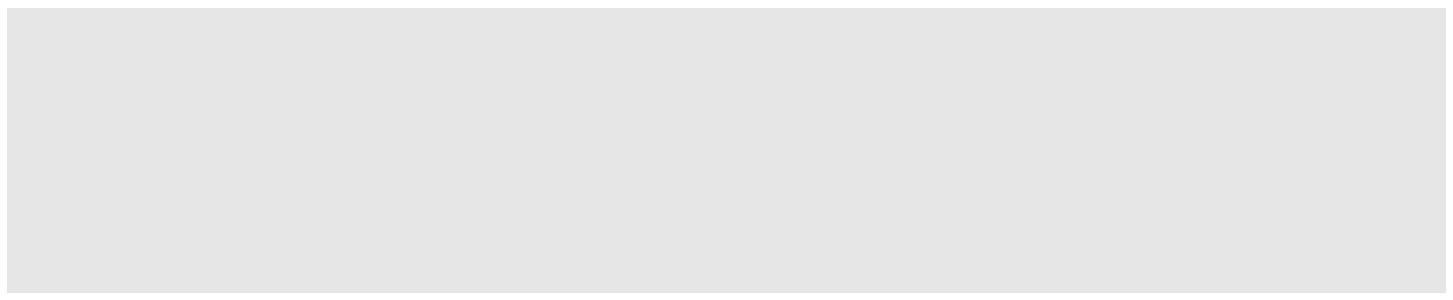
**Answer:** Men spent $9,919,847, and women spent $9,787,324.

1. The orderdate column is a string with the format yyyy-mm-dd. Use the year function to extract the various parts of the date. For example, run the following query, which computes the sum of all orders for each year:



hive> SELECT sum(ordertotal), year(order\_date) FROM orders GROUP BY year(order\_date);

The output should look like this. **Verify, then quit the Hive shell**:



4082780 2009

4404806 2010

4399886 2011

4248950 2012

2570749 2013

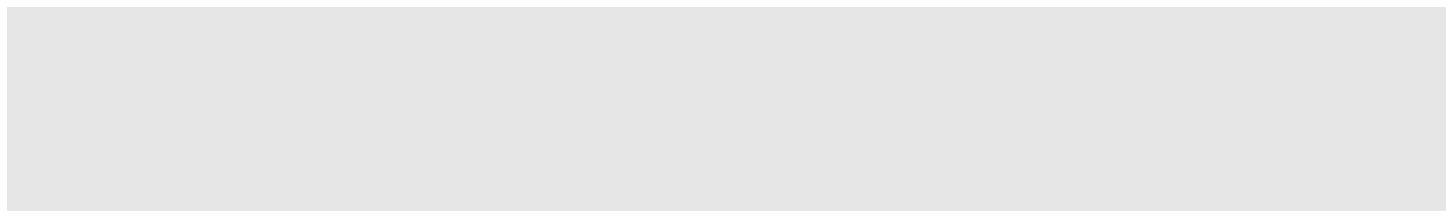
hive> quit;

3 ) Multi-File Insert

1. In this step, you will run two completely different queries, but in a single job. The output of the queries will be in two separate directories in HDFS. Start by using gedit to create a new text file in the

/root/devph/labs/Lab9.1 folder named multifile.hive.

1. Within the text file, enter the following query. Notice there is no semicolon between the two INSERT statements:



FROM ORDERS o

INSERT OVERWRITE DIRECTORY '2010\_orders'

SELECT o.\* WHERE year(order\_date) = 2010

INSERT OVERWRITE DIRECTORY 'software'

SELECT o.\* WHERE itemlist LIKE '%Software%';

* 1. Save your changes to multifile.hive.
  2. Run the query from the command line:

1. hive -f multifile.hive
   1. The above query executes in a single job. Even more interesting, it only requires a map phase.

Why did this job not require a reduce phase?

***Answer:*** Because the query only does aSELECT \*, noreducephase wasneeded.



* 1. Verify that the two queries executed successfully by viewing the folders in HDFS:

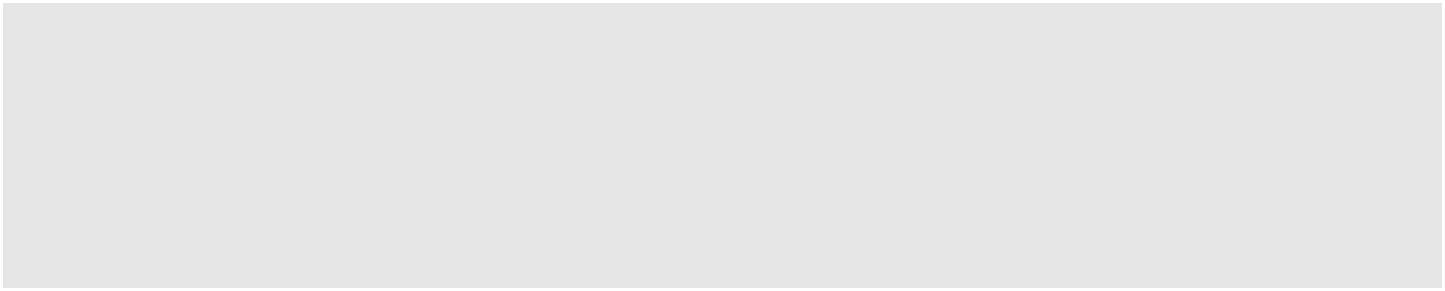
1. hdfs dfs -ls



You should see two new folders: 2010\_orders and software.

* 1. View the output files in these two folders. Verify that the 2010\_orders directory contains orders from only the year 2010, and verify that the software directory contains only orders that included ‘Software.’

1. hdfs dfs -ls 2010\_orders
2. hdfs dfs -cat 2010\_orders/000000\_0
3. hdfs dfs -ls software
4. hdfs dfs -cat software/000000\_0



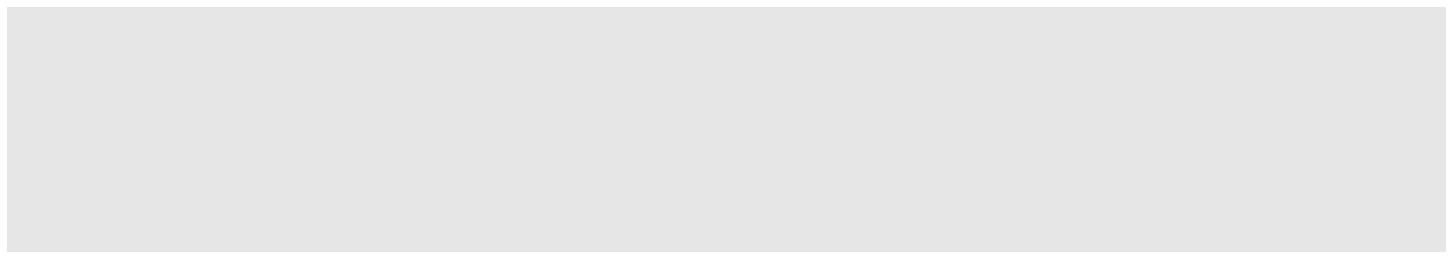
4 ) Define a View

a. Start the Hive shell. Define a view named 2013\_orders that contains the

orderid, order\_date, username, and itemlist columns of the orders table

where the order\_date was in the year 2013.

***Solution*:**The2013\_ordersview:



# hive

hive> CREATE VIEW 2013\_orders AS

SELECT orderid, order\_date, username, itemlist FROM orders

WHERE year(order\_date) = '2013';

1. Run the show tables command: hive> show tables;



You should see 2013\_orders in the list of tables.

1. To verify your view is defined correctly, run the following query:

hive> SELECT COUNT(\*) FROM 2013\_orders;

The 2013\_orders view should contain 13,104 records.



5 ) Find the Maximum Order of Each Customer

1. Suppose you want to find the maximum order of each customer. This can be done easily enough with the following Hive query. Run this query now:

hive> SELECT max(ordertotal), userid FROM orders GROUP BY userid;



1. How many different customers are in the orders table? \_\_\_\_\_\_\_\_\_\_\_\_\_

***Answer:*** There are 100 unique customers in the orders table.

1. Suppose you want to add the itemlist column to the previous query. Try adding it to the SELECT clause by the following method and see what happens:



hive> SELECT max(ordertotal), userid, itemlist FROM orders GROUP BY userid;

Notice this query is not valid because itemlist is not in the GROUP BY key.

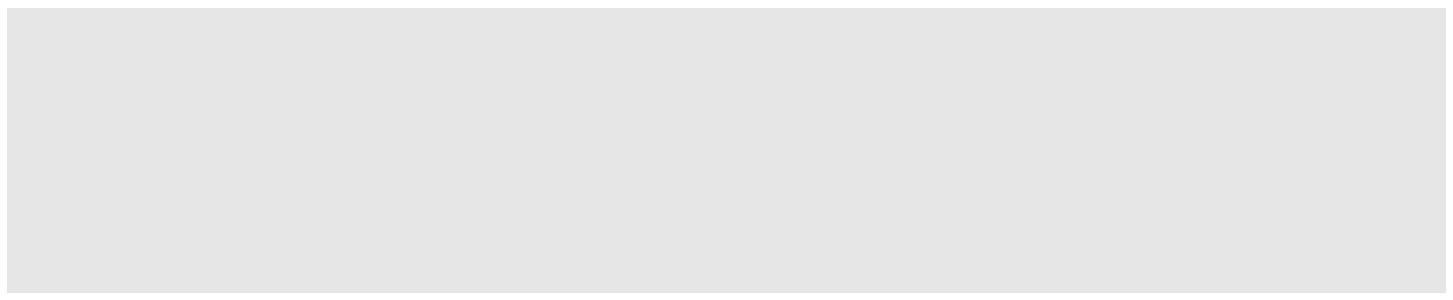
1. We can join the result set of the max-total query with the orders table to add the itemlist to our result. Start by defining a view named max\_ordertotal for the maximum order of each customer:



hive> CREATE VIEW max\_ordertotal AS

SELECT max(ordertotal) AS maxtotal, userid FROM orders GROUP BY userid;

e. Now join the orders table with your max\_ordertotal view:



hive> SELECT ordertotal, orders.userid, itemlist FROM orders

JOIN max\_ordertotal ON

max\_ordertotal.userid = orders.userid

AND

max\_ordertotal.maxtotal = orders.ordertotal ORDER BY orders.userid;

1. What did the Tez job look like for this query? \_\_\_\_\_\_\_\_\_\_\_

***Answer:*** The query resulted in a map-reduce-reduce Tez job.



The end of your output should look like:

1. 98

Grill,Freezer,Bedding,Headphones,DVD,Table,Grill,Software,Dishwash

er,DVD,Microwave,Adapter

1. 99Washer,Cookware,Vacuum,Freezer,2-Way Radio,Bicycle,Washer & Dryer,Coffee Maker,Refrigerator,DVD,Boots,DVD
2. 100Bicycle,Washer,DVD,Wrench Set,Sweater,2-Way Radio,Pants,Freezer,Blankets,Grill,Adapter,pillows



6 ) Fixing the GROUP BY Key Error

1. Let’s compute the sum of all of the orders of all of the customers. Start by entering the following query:

hive> SELECT sum(ordertotal), userid FROM orders GROUP BY userid;



Notice that the output is the sum of all orders, but displaying just the userid is not very exciting.

1. Try to add the username column to the SELECT clause in the following manner and see what happens:

hive> SELECT sum(ordertotal), userid, username



FROM orders

GROUP BY userid;

This generates an “Expression not in GROUP BY key” error, because the username column is not being aggregated but the ordertotal is.

1. An easy fix is to aggregate the username values using the collect\_set function, but output only one of them:

hive> SELECT sum(ordertotal), userid, collect\_set(username)[0] FROM orders



GROUP BY userid;

You should get the same output as before, but this time the username is included.

7 ) Using the OVER Clause

1. Now let’s compute the sum of all orders for each customer, but this time use the OVER clause to not group the output and to also display the

itemlist column:

hive> SELECT userid, itemlist, sum(ordertotal)



OVER (PARTITION BY userid)

FROM orders;

Notice the output contains every order, along with the items they purchased and the sum of all of the orders ever placed from that particular customer.



8 ) Using the Window Functions

a. It is not difficult to compute the sum of all orders for each day using the

GROUP BY clause:

hive> select order\_date, sum(ordertotal)



FROM orders

GROUP BY order\_date;

Run the query above and the tail of the output should look like:

2013-07-28 18362



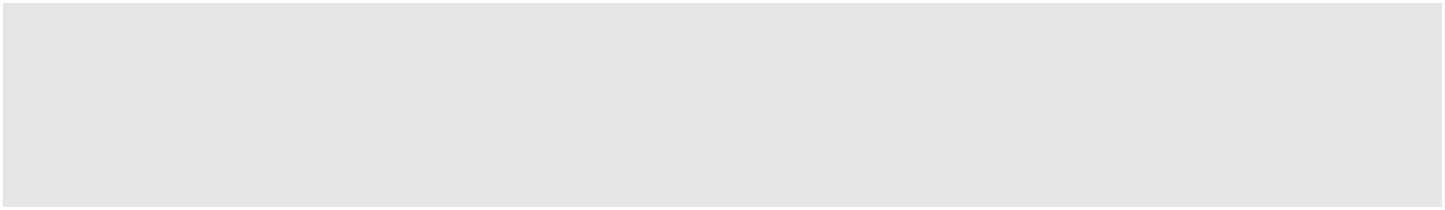
2013-07-29 3233

2013-07-30 4468

2013-07-31 4714

1. Suppose you want to compute the sum for each day that includes each order. This can be done using a window that sums all previous orders along with the current row:

hive> SELECT order\_date, sum(ordertotal)



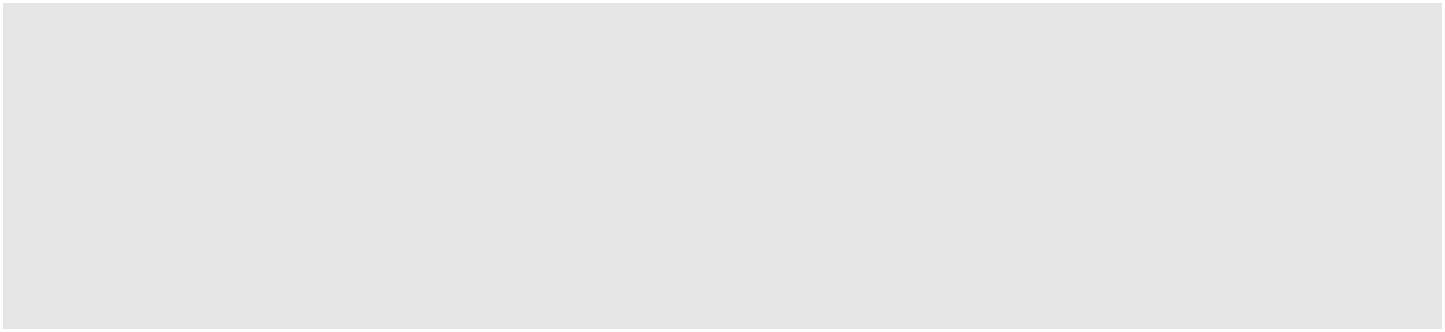
OVER

(PARTITION BY order\_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)

FROM orders;

To verify that it worked, your tail of your output should look like:

2013-07-31 3163



2013-07-31 3415

2013-07-31 3607

2013-07-31 4146

2013-07-31 4470

2013-07-31 4610

2013-07-31 4714

9 ) Using the Hive Analytics Functions

1. Run the following query, which displays the rank of the ordertotal by day:

hive> SELECT order\_date, ordertotal, rank()



OVER

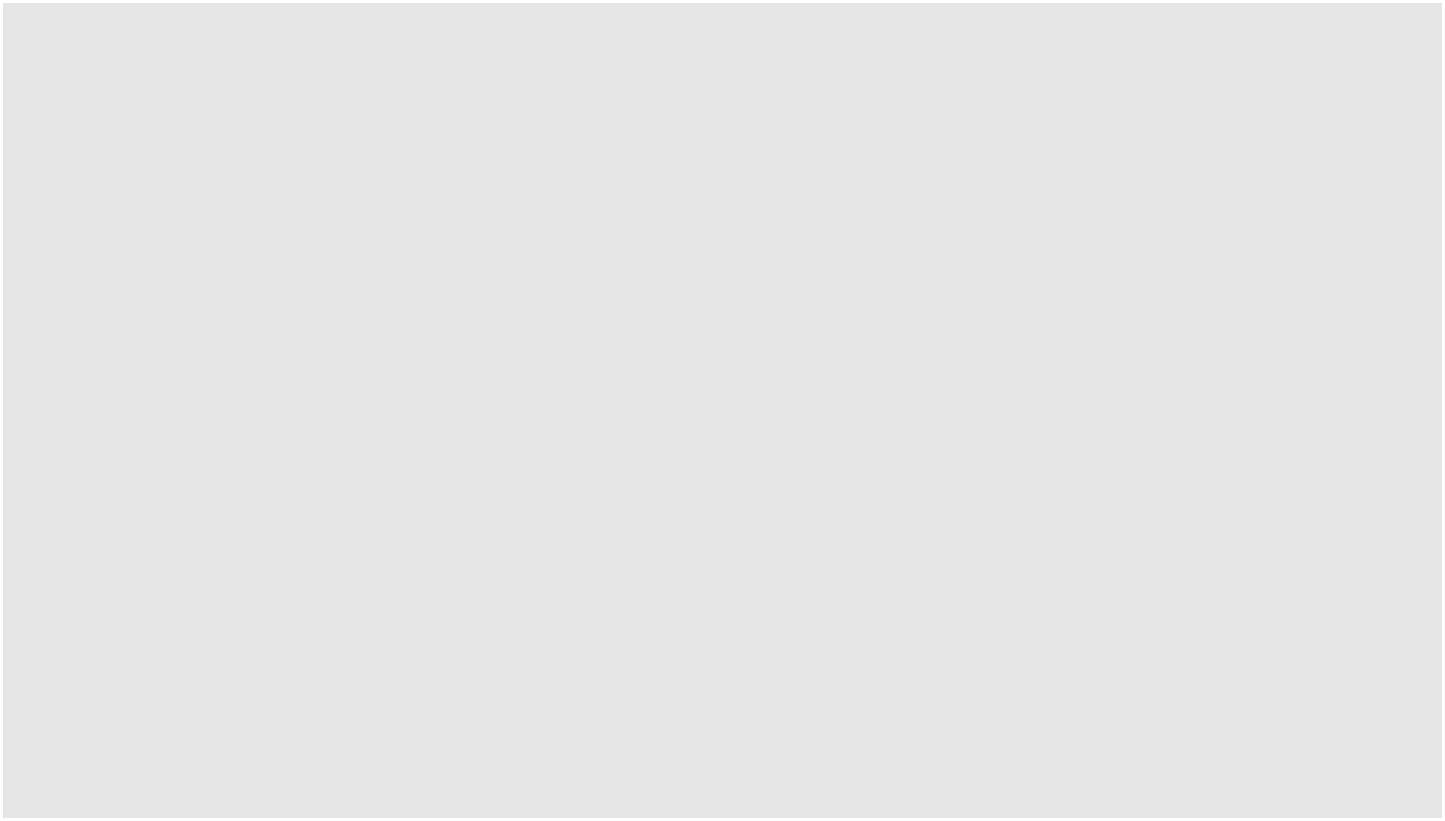
(PARTITION BY order\_date ORDER BY ordertotal)

FROM orders;



b. To verify it worked, the output of July 31, 2013, should look like:

2013-07-31 48 1



2013-07-31 104 2

2013-07-31 119 3

2013-07-31 130 4

2013-07-31 133 5

2013-07-31 135 6

2013-07-31 140 7

2013-07-31 147 8

2013-07-31 156 9

2013-07-31 192 10

2013-07-31 192 10

2013-07-31 196 12

2013-07-31 240 13

2013-07-31 252 14

2013-07-31 296 15

2013-07-31 324 16

2013-07-31 343 17

2013-07-31 500 18

2013-07-31 528 19

2013-07-31 539 20

1. As a challenge, see if you can run a query similar to the previous one except compute the rank over months instead of each day. ***Solution:*** The rank query by month:



SELECT substr(order\_date,0,7), ordertotal, rank()

OVER

(PARTITION BY substr(order\_date,0,7) ORDER BY ordertotal) FROM orders;

10 ) Histograms

1. Run the following Hive query, which uses the histogram\_numeric function to compute 20 (x,y) pairs of the frequency distribution of the total order amount from customers who purchased a microwave (using the orders table):



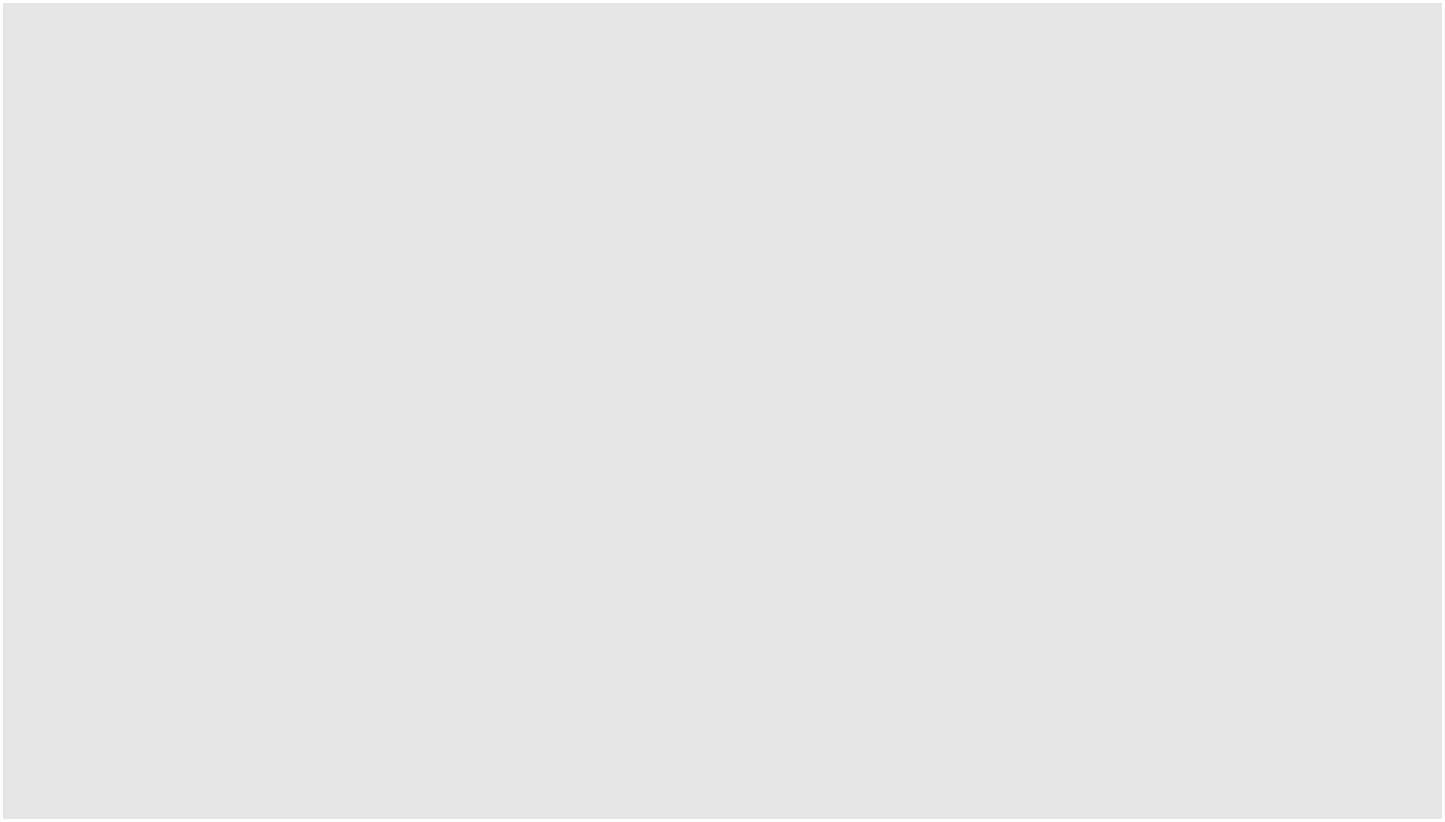
hive> SELECT explode(histogram\_numeric(ordertotal,20)) AS x FROM orders

WHERE itemlist LIKE "%Microwave%";



The output should look like the following:

{"x":14.333333333333332,"y":3.0}



{"x":33.87755102040816,"y":441.0}

{"x":62.52577319587637,"y":679.0}

{"x":89.37823834196874,"y":965.0}

{"x":115.1242236024843,"y":1127.0}

{"x":142.6468885672939,"y":1382.0}

{"x":174.07664233576656,"y":1370.0}

{"x":208.06909090909105,"y":1375.0}

{"x":242.55486381322928,"y":1285.0}

{"x":275.8625954198475,"y":1048.0}

{"x":304.71100917431284,"y":872.0}

{"x":333.1514423076924,"y":832.0}

{"x":363.7630208333335,"y":768.0}

{"x":397.51587301587364,"y":756.0}

{"x":430.9072847682117,"y":604.0}

{"x":461.68715083798895,"y":537.0}

{"x":494.1598360655734,"y":488.0}

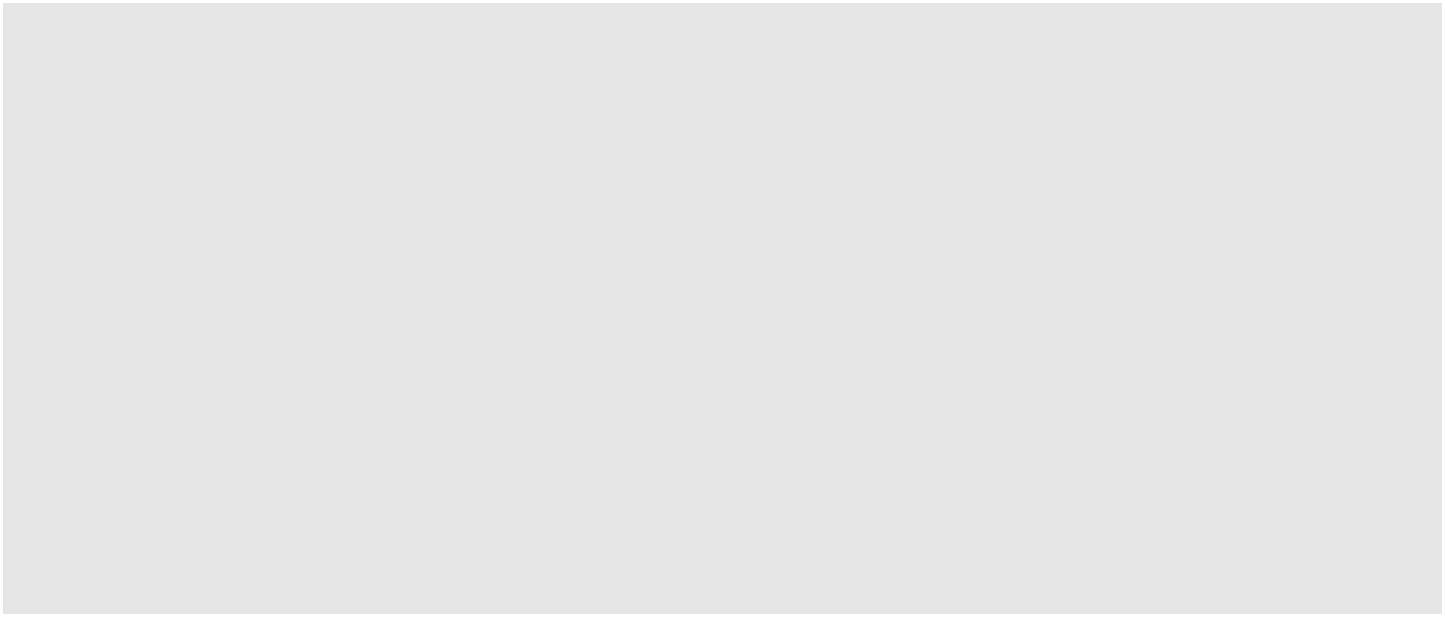
{"x":528.5816326530613,"y":294.0}

{"x":555.5166666666672,"y":180.0}

{"x":588.7979797979801,"y":198.0}

1. Write a similar Hive query to compute 10 frequency-distribution pairs for the ordertotal from orders table where ordertotal is greater than $200.

SELECT explode(histogram\_numeric(ordertotal,10)) AS x FROM orders



WHERE ordertotal > 200;

{"x":218.8195174551819,"y":7419.0}

{"x":254.10237580993478,"y":6945.0}

{"x":293.4231618807192,"y":6338.0}

{"x":334.57302573203015,"y":5635.0}

{"x":379.79714934930786,"y":4841.0}

{"x":428.1165628891644,"y":4015.0}

{"x":473.1484734420741,"y":2391.0}

{"x":511.2576946288467,"y":1657.0}

{"x":549.0106899902812,"y":1029.0}

{"x":589.0761194029857,"y":670.0}

**Result**

You should now be comfortable running Hive queries and using some of the more advanced features of Hive, like views and the window functions.



**Lab: Running a YARN Application**

**About this Lab**

**Objective:**

To run a YARN application.

**File locations:**

n/a

**Successful**

**outcome:**

You will have executed the DistributedShell YARN application.

**Before you begin:**

Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:**

***Hadoop 2 and YARN***

**Lab Steps**

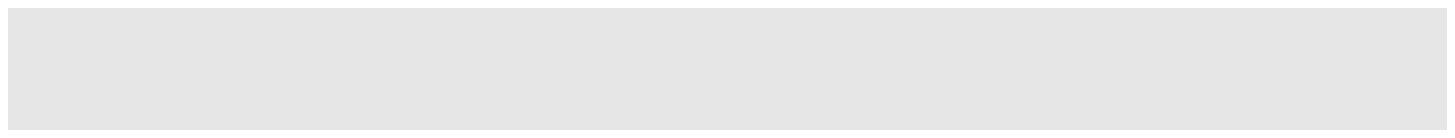
1 ) Run a DistributedShell Application

1. If not already done, open a putty or git-bash Terminal to your sandbox.
2. In a terminal window change directories to the /usr/hdp/current/hadoop-

yarn-client folder:



1. cd /usr/hdp/current/hadoop-yarn-client/
   1. Run the following command, which runs a sample YARN application that ships with HDP 2.x:
2. yarn jar hadoop-yarn-applications-distributedshell.jar org.apache.hadoop.yarn.applications.distributedshell.Client -jar hadoop-yarn-applications-distributedshell.jar -shell\_command cal



The DistributedShell command allows you to run a script or shell command on your cluster. The example above runs the Unix “cal” command, which displays a text calendar.

* 1. Wait for the YARN job to finish. 2 ) Verify the Result

a. Enter the following command (all on a single line):

1. yarn application -list -appStates FINISHED | grep Dist



|  |  |
| --- | --- |
|  |  |

You should see the application ID of the DistributedShell command that you just ran:

application\_1378331467073\_0004 DistributedShell YARN yarn



default FINISHED SUCCEEDED 100%

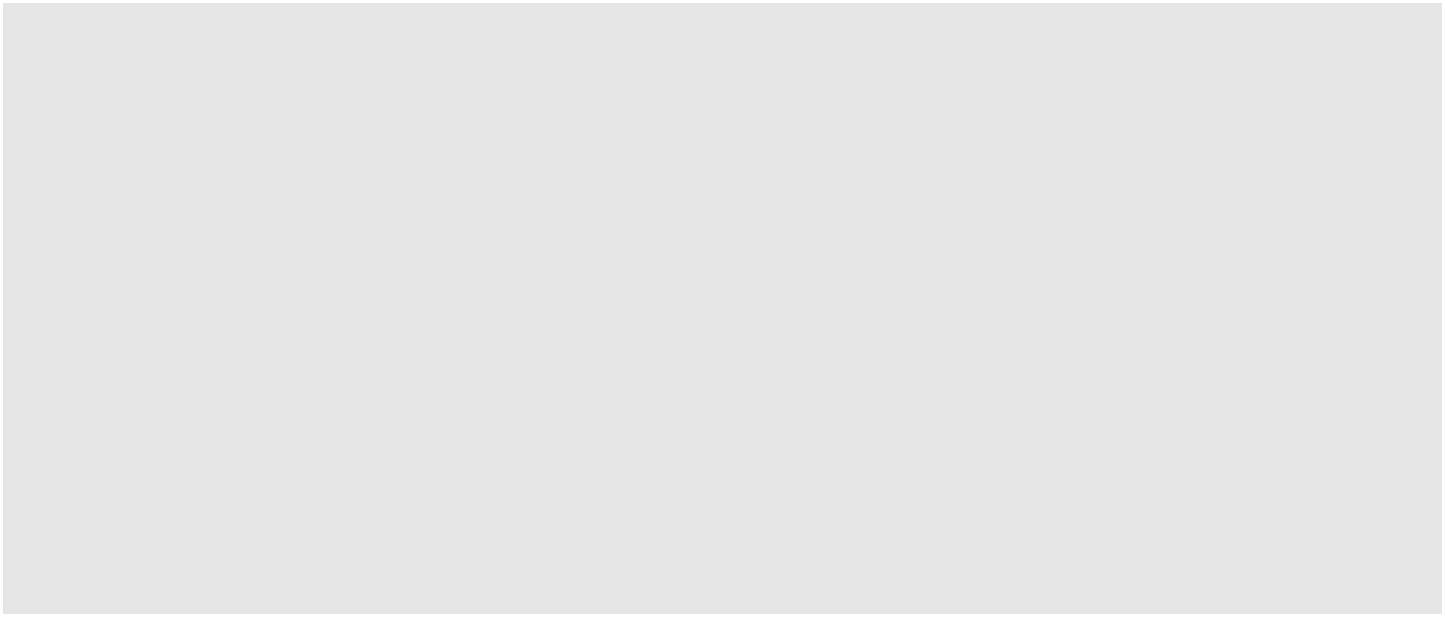
* 1. Copy and paste the application ID of your DistributedShell command and check its status using the following command (but replacing the ID shown here with your actual application ID):

1. yarn application -status application\_1378331467073\_0004



Notice that the details of the job are displayed. This was a simple application, so there is not a lot of information to analyze:

Application Report :



Application-Id : application\_1378331467073\_0004

Application-Name : DistributedShell

Application-Type : YARN

User : root

Queue : default

Start-Time : 1408060384688

Finish-Time : 1408060391340

Progress : 100%

State : FINISHED

Final-State : SUCCEEDED

Tracking-URL : N/A

RPC Port : -1

AM Host : sandbox.hortonworks.com/172.16.173.149

Diagnostics :

**Note**

The YARN application command also has a -kill option (followed by the application’s ID) that kills a running YARN job. This is a great tool when you have submitted a job and then need to stop it before it runs to completion.

3 ) View the Log File

* 1. Enter the following command to view the output for this YARN application that you just executed:

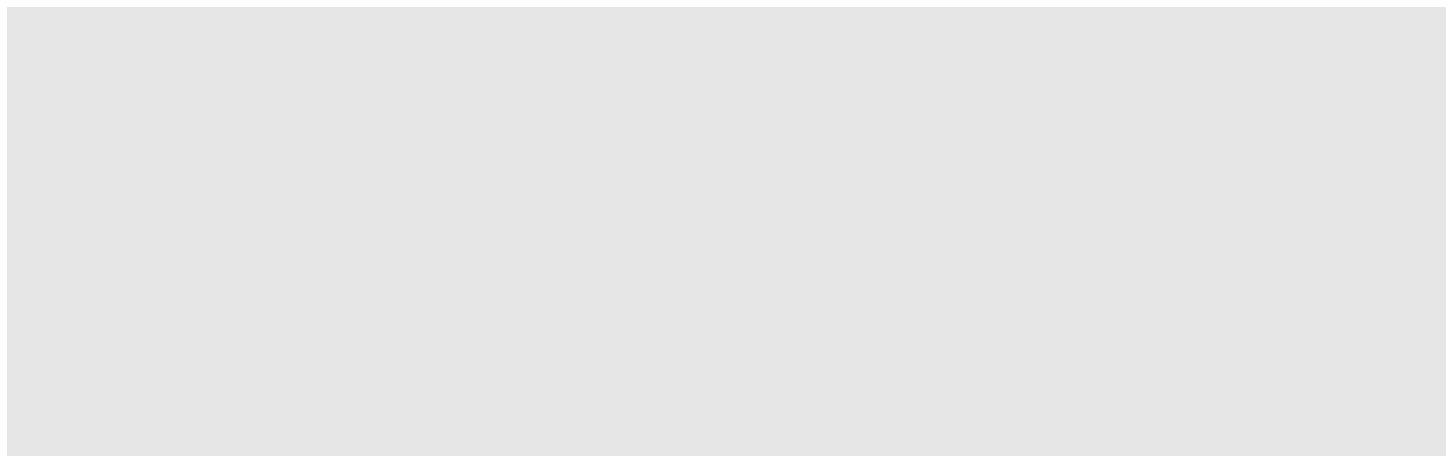
1. yarn logs -applicationId application\_1378331467073\_0004



**Important**

The -applicationId must have the correct case for every letter, or else you will see an error message stating that it is missing. The only capital letter in the option is the I. The d, and all other letters, are lower case.

Somewhere in the log file, you should see a text calendar of the current month. For example:



LogType: stdout

LogLength: 150

Log Contents:

August 2014

Su Mo Tu We Th Fr Sa

2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4 | 5 | 6 | 7 | 8 | 9 | 16 |
| 10 | 11 | 12 | 13 | 14 | 15 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 |  |  |  |  |  |  |

4 ) Optional: Run the Job in Six Containers

1. The DistributedShell application allows you to specify how many containers the ApplicationMaster uses. Add the following arguments to the end of the YARN command from Step 1.2:



-num\_containers 6 -container\_memory 20

* 1. Now find the applicationID and view the aggregated log file:

1. yarn application -list -appStates FINISHED | grep Dist
2. yarn logs -applicationId <applicationId>



You should see six calendars this time, one from each container.

1. Notice that this also demonstrates how the log output from multiple containers is aggregated into a single, convenient log file.

**Result**

In this lab you ran a simple YARN application called the DistributedShell (that ships with HDP 2.x). You also saw how to view the output of the aggregated log file using the YARN logs command.

**Lab: Getting Started with Apache Spark**

**About this Lab**

**Objective:** Read and manipulate HDFS files with Spark.

**File locations:** /root/hdp/pigandhive/devph/labs/Spark

**Successful outcome:** You will have processed several HDFS file via Spark Core.

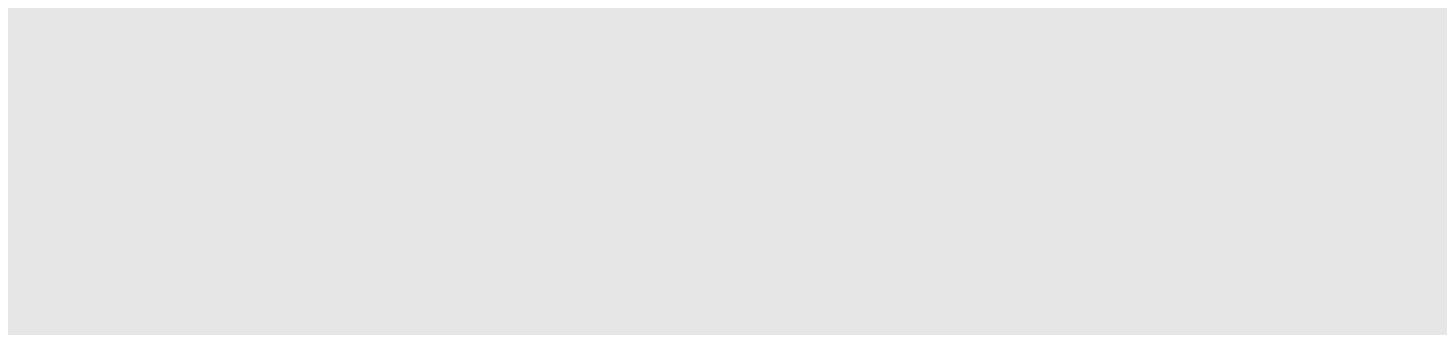
**Before you begin:** Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:** ***Programming with Apache Spark***

**Lab Steps**

1 ) Execute a WordCount with Spark.

1. If not already done, open a putty or git-bash Terminal to your sandbox.
2. Copy the constitution.txt file to HDFS.



[root@sandbox ~]# cd ~/hdp/pigandhive/labs/Spark

[root@sandbox Spark]# hdfs dfs -mkdir spark

[root@sandbox Spark]# hdfs dfs -put

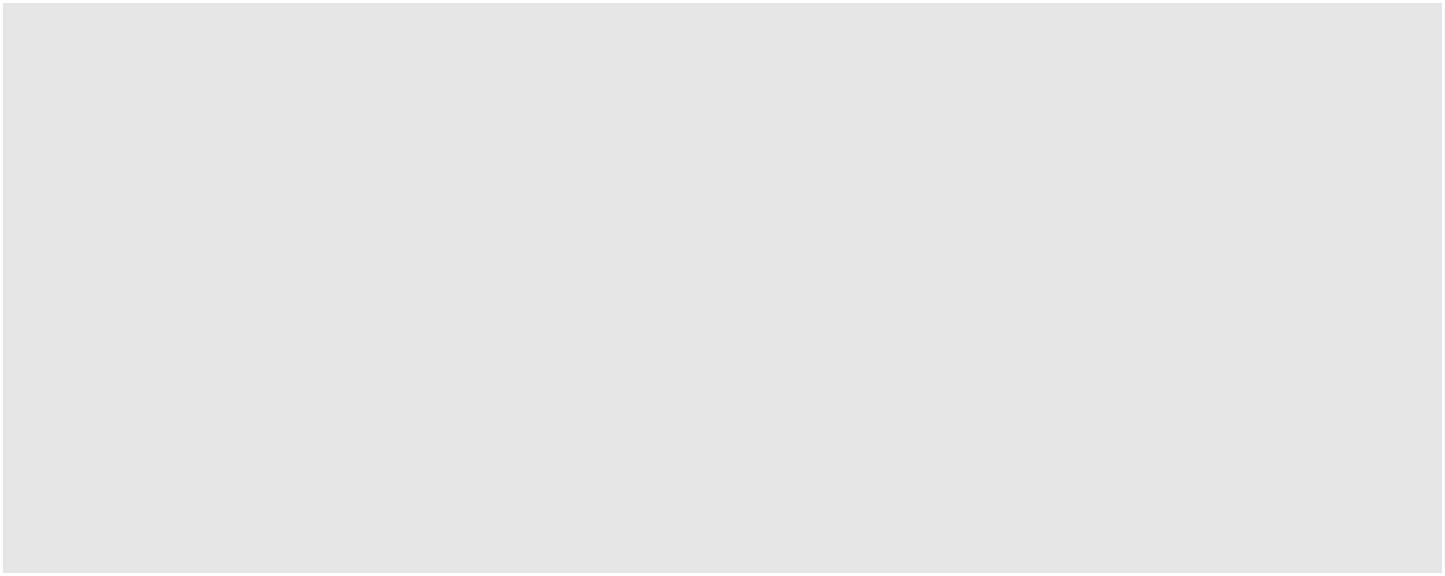
|  |  |  |  |
| --- | --- | --- | --- |
| ~/hdp/pigandhive/devph/labs/demos/constitution.txt | | | spark/ |
| [root@sandbox Spark]# hdfs dfs -ls spark | | | |
| Found 1 items | 1 root hdfs | 45489 | 2015-11-09 09:39 |
| -rw-r--r-- |

spark/constitution.txt



1. Launch the Python Spark Shell. NOTE: "INFO" comments will be removed from the output in this lab guide going forward.

[root@sandbox ~]# pyspark



Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56) [GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2

|  |  |  |
| --- | --- | --- |
| ... lines removed ... | |  |
| Welcome to | |  |
| \_\_\_\_ | \_\_ |  |
| / \_\_/\_\_ | \_\_\_ \_\_\_\_\_/ /\_\_ |  |
| \_\ \/ \_ \/ \_ `/ \_\_/ '\_/ | | version 1.4.1 |
| /\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ | |
| /\_/ |  |  |

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56) SparkContext available as sc, HiveContext available as sqlContext. >>>



* 1. Read in the file as a RDD.

1. baseFile = sc.textFile("hdfs://sandbox:8020/user/root/spark/constitution.txt"

)

1. baseFile.take(1)



[u'We the People of the United States, in Order to form a more perfect ']

* 1. Break the full lines down into a collection of words.

1. justWords = baseFile.flatMap(lambda line: line.split(' '))
2. justWords.take(5)



[u'We', u'the', u'People', u'of', u'the']

* 1. Map the words with a count of "1" for each.

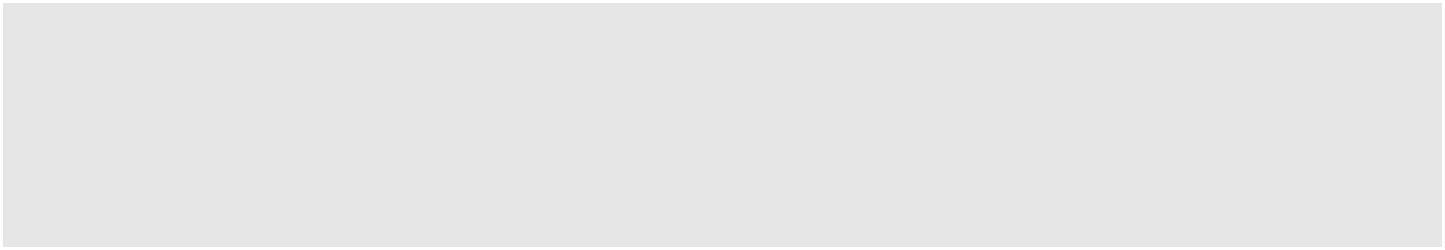
1. mappedWords = justWords.map(lambda word: (word, 1))
2. mappedWords.take(5)



[(u'We', 1), (u'the', 1), (u'People', 1), (u'of', 1), (u'the', 1)]

* 1. Count up the words; sorting them in reverse.

1. wordCounts = mappedWords.reduceByKey(lambda a,b: a+b).sortByKey(ascending=False)
2. wordCounts.take(10)



[(u'years;', 1), (u'years', 9), (u'year,', 1), (u'year', 1), (u'written', 6), (u'writs', 1), (u'writing,', 1), (u'would', 2), (u'work', 1), (u'witnesses', 2)]



* 1. Chain all the method invocations into a single operation, as is more the normal usage pattern. NOTE: Type as a single line without the new line or "\" characters.

asOneLine =

sc.textFile("hdfs://sandbox:8020/user/root/spark/constitution.txt" ) \

.flatMap(lambda line: line.split(' ')) \

.map(lambda word: (word, 1)) \

.reduceByKey(lambda a,b: a+b) \

.sortByKey(ascending=False)

1. asOneLine.take(10)



[(u'years;', 1), (u'years', 9), (u'year,', 1), (u'year', 1), (u'written', 6), (u'writs', 1), (u'writing,', 1), (u'would', 2), (u'work', 1), (u'witnesses', 2)]

* 1. Exit out of the pyspark REPL.

1. quit()

**Result:**

Successful use of Spark Core and RDD to read files and perform data analysis.



**Lab: Exploring Spark SQL**

**About this Lab**

**Objective:** Create DataFrame structure from Hive tables & HDFS files and utilize both the DataFrame API and SQL to refine returned data.

**File locations:** /root/hdp/pigandhive/labs/Spark

**Successful outcome:** You will have successfully queries data from multipleDataFrame objects as well as joined them together.

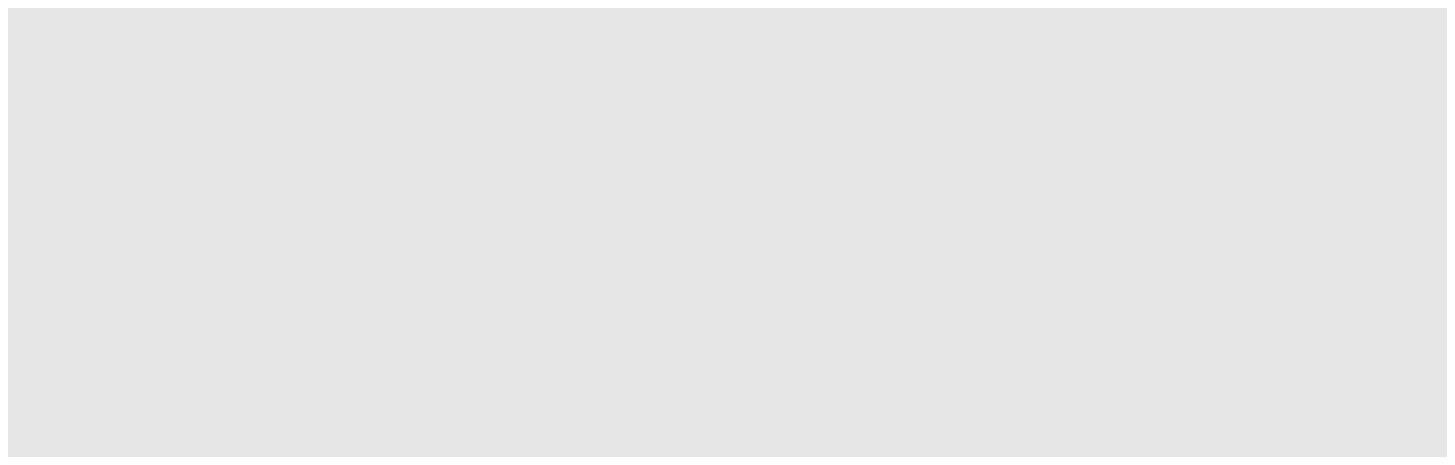
**Before you begin:** Your HDP 2.6 cluster should be up and running within your VM.

**Related lesson:** ***Spark SQL and DataFrames***

**Lab Steps**

1 ) Run a query on an existing Hive table.

a. Load file into HDFS and create a Hive table mapping to it.



[root@sandbox Spark]# hdfs dfs -mkdir spark/cust\_fav [root@sandbox Spark]# hdfs dfs -put cust\_fav.csv spark/cust\_fav/ [root@sandbox Spark]# hive -f cust\_fav.hive

[root@sandbox Spark]# hive -e 'select \* from cust\_fav limit 5;'

OK

Irvin Riesling

Owen Pinot Noir

August Sauvignon Blanc

Christian Merlot

Arlen Pinot Noir

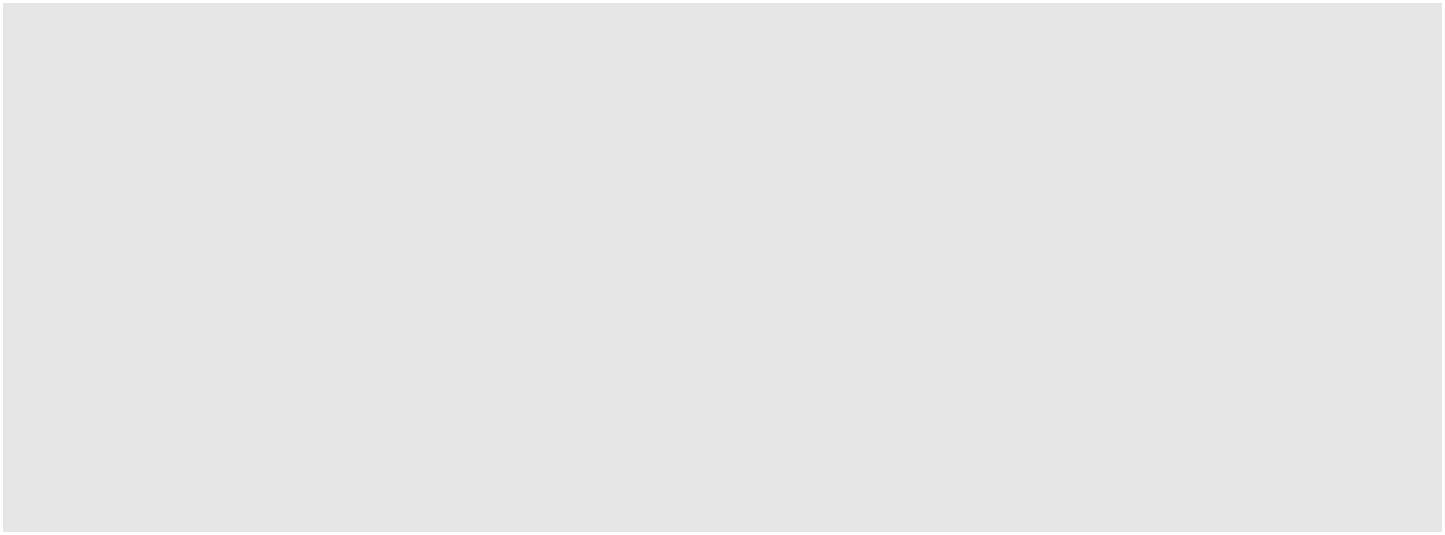
Time taken: 3.023 seconds, Fetched: 5 row(s)



|  |  |
| --- | --- |
|  |  |

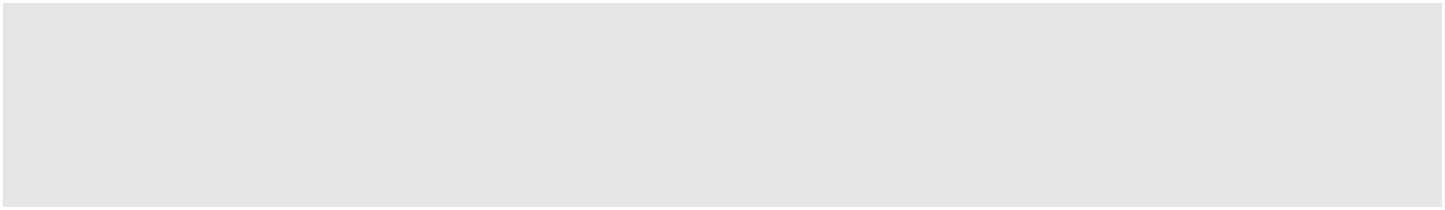
* 1. Create a DataFrame from querying the Hive table created in the prior step after starting up pyspark again.

1. from pyspark.sql import HiveContext
2. hc = HiveContext(sc)
3. custFavDF = hc.sql("SELECT \* FROM cust\_fav")
4. custFavDF.show(5)



|  |  |  |
| --- | --- | --- |
| + | ---------+ | ---------------+ |
| |cust\_name| | | wine\_type| |
| + | ---------+ | ---------------+ |
| | | Irvin| | Riesling| |
| | | Owen| | Pinot Noir| |
| | | August|Sauvignon Blanc| | |
| |Christian| | | Merlot| |
| | | Arlen| | Pinot Noir| |
| + | ---------+ | ---------------+ |

2 ) Use customer data from the prior lab to find average length of customers by gender and state.

* 1. Import the necessary Row definition and then create a RDD from the customer.csv file previously loaded into HDFS.

from pyspark.sql import Row

customerRaw = sc.textFile("hdfs://sandbox:8020/user/root/spark/customer.csv")

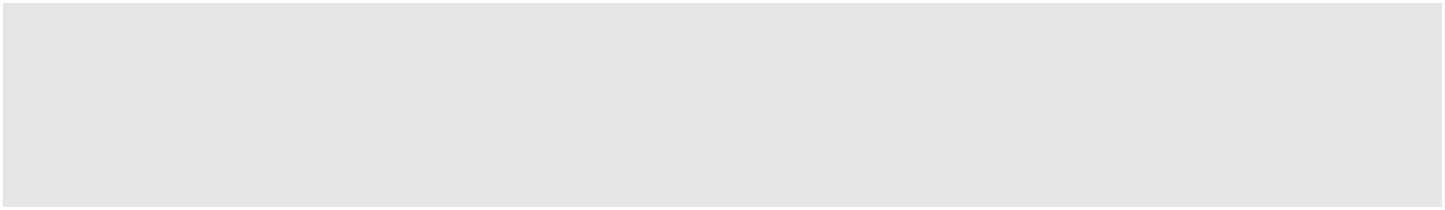
customerRaw.take(2)

[u'Irvin,M,Maryland,5.06', u'Owen,M,Illinois,2.01']

* 1. Break each long string representing a row from the input file into discrete customer records.

customerRecords = customerRaw.map(lambda line: line.split(','))

customerRecords.take(2)



[[u'Irvin', u'M', u'Maryland', u'5.06'], [u'Owen', u'M', u'Illinois', u'2.01']]

* 1. Convert that the RDD to a DataFrame.

customerDF = customerRecords.map(lambda c: Row(name=c[0], gender=c[1], state=c[2], length=float(c[3]))).toDF()

customerDF.show(2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + | ------+ | ------+ | -----+ | --------+ |
| |gender|length| | | | name| | state| |
| + | ------+ | ------+ | -----+ | --------+ |
| | | M| | 5.06|Irvin|Maryland| | | |
| | | M| | 2.01| Owen|Illinois| | | |
| + | ------+ | ------+ | -----+ | --------+ |

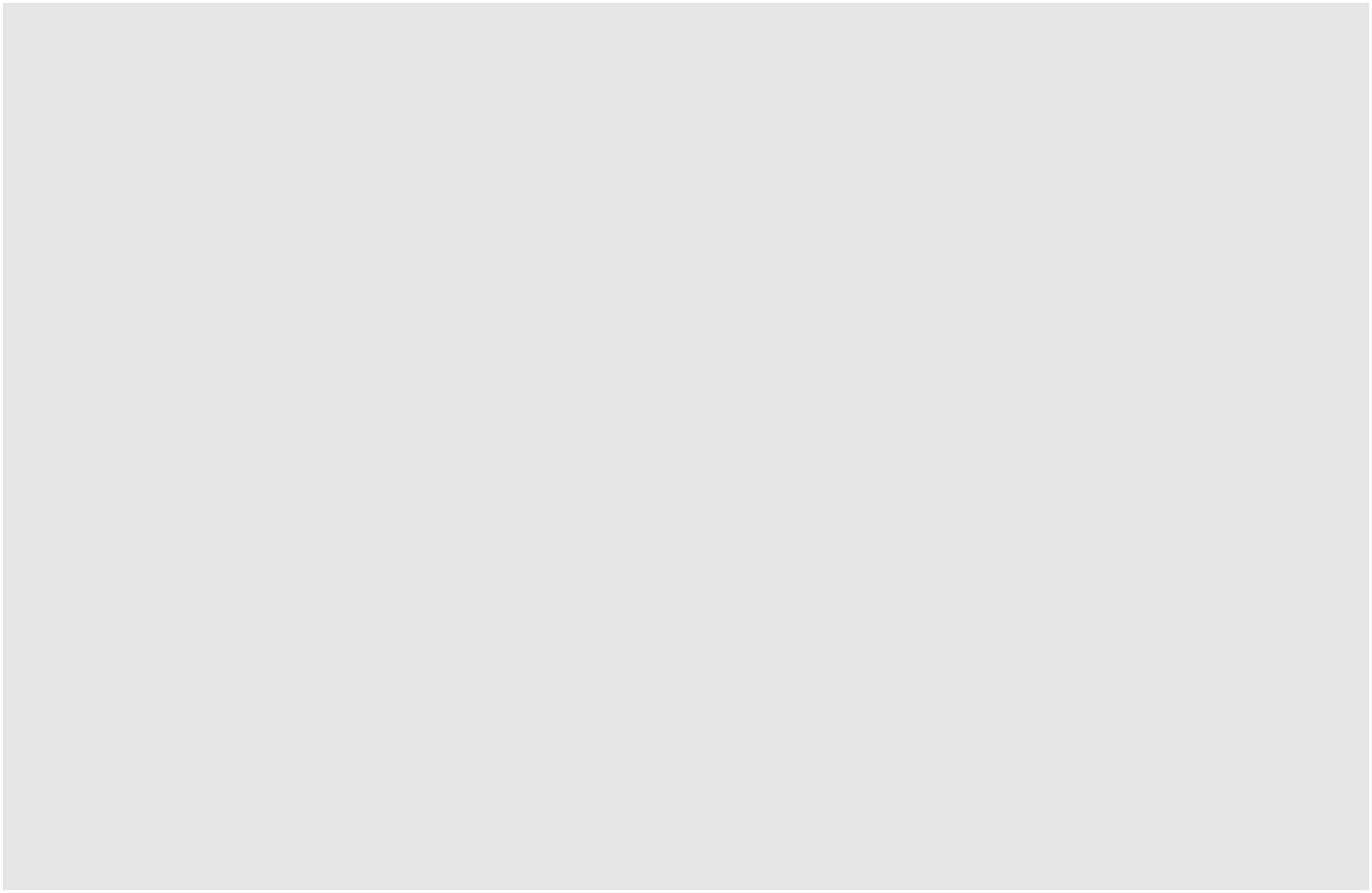


d. Search for the final results with the DataFrame API.

>>>



customerDF.select("gender","state","length").groupBy("gender","state").a

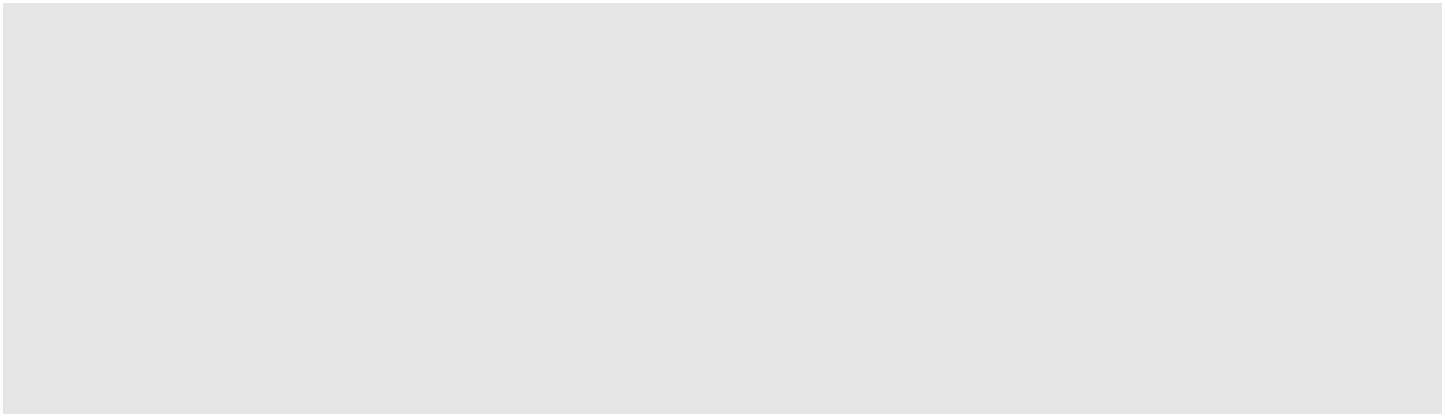


|  |  |  |  |
| --- | --- | --- | --- |
| vg("length").show() | | | ------------------+ |
| + | ------+ | ------------+ |
| |gender| | | state| | AVG(length)| |
| + | ------+ | ------------+ | ------------------+ |
| | | F| | Illinois| | 7.49| |
| | | F| | New Jersey| | 4.04| |
| | | F| | Minnesota| | 6.204| |
| | | F| | Michigan| 4.207142857142857| | |
| | | M| | Indiana| 4.369999999999999| | |
| | | M| | Maryland| 5.326250000000001| | |
| | | M| | Nebraska| | 6.516| |
| | | M| | Illinois| 4.858571428571428| | |
| | | M| | New Jersey|4.6499999999999995| | |
| | | M| | Minnesota| | 5.4375| |
| | | F| | Wisconsin| | 6.29125| |
| | | M| | Michigan|5.0337499999999995| | |
| | | F|Pennsylvania| | | 5.575| |
| | | F| | Ohio| | 5.93| |
| | | F| | Iowa| 7.430000000000001| | |
| | | M| | Wisconsin| | 0.33| |
| | | M|Pennsylvania| | | 3.665| |
| | | M| | Ohio| | 4.25| |
| | | M| | Iowa| 5.753333333333333| | |
| | | F| | Maryland| 5.003749999999999| | |
| + | ------+ | ------------+ | ------------------+ |

3 ) Join the prior two DataFrames.

* 1. Utilize the DataFrame API to perform the join.

customerDF.join(custFavDF, customerDF.name == custFavDF.cust\_name).show(5)



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| + | ------+ | ------+ | ---------+ | ----------+ | ---------+ | ---------------+ |
| |gender|length| | | | name| | state|cust\_name| | | wine\_type| |
| + | ------+ | ------+ | ---------+ | ----------+ | ---------+ | ---------------+ |
| | | M| | 5.06| | Irvin| | Maryland| | Irvin| | Riesling| |
| | | M| | 2.01| | Owen| | Illinois| | Owen| | Pinot Noir| |
| | | M| | 1.42| | August| | Illinois| | August|Sauvignon Blanc| | |
| | | M| | 8.17|Christian|New Jersey|Christian| | | | | Merlot| |
| | | M| | 2.24| | Arlen| | Indiana| | Arlen| | Pinot Noir| |
| + | ------+ | ------+ | ---------+ | ----------+ | ---------+ | ---------------+ |

**Result**

Successful creation of a DataFrame from a Hive tables and a HDFS file; as well as joining these two DataFrames.