Practices taken from: HDP Developer: Apache Pig and Hive.

Lab 1: Importing RDBMS Data into HDFS

This lab explores importing data from a database into HDFS.

Table 4.

Objective:	Import data from a database into HDFS.
File locations:	/root/devph/labs/Lab3.1/
Successful outcome:	You will have imported data from MySQL into folders in HDFS.
Before you begin:	Your HDP 2.1 cluster should be up and running within your VM.
Related lesson:	Inputting Data into HDFS

Perform the following steps:

Step 1: Create a Table in MySQL

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

1.2. View the contents of salaries.txt:

```
# tail salaries.txt
```

```
# cp salaries.txt  /tmp
```

1.4. Run the salaries.sql script using the following command:

```
#  mysql test < salaries.sql
```

Step 2: View the Table

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

```
# mysql
```

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

```
mysql> use test;
```

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

```
mysql>        show tables;
+................               +
| Tables_in_test  |
+................               +
|  salaries                    |
+---------------               +

1  row in set           (0.00 sec)
```

2.4. Select 10 items from the table to verify that it is populated:

```
mysql> select * from salaries  limit 10;----
+--------      +    ------ +    -------- +    ---------  +         +
| gender    | age        | salary   | zipcode     | id |
+--------      +           +            +   ---------95103  +         +
| F           |   ------66 |  --------41000 |                   | ----1  |
| 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|
```

2.5. Exit the mysql

```
mysql> exit
```

Step 3: Import the Table into HDFS

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

```
# sqoop import --connect  jdbc:mysql://sandbox/test?user=root                    --table
```

3.2. A MapReduce job should start executing, and it may take a couple of minutes for the job to complete.

Step 4: Verify the Import

4.1. View the contents of your HDFS folder:

```
# hadoop fs -ls
```

4.2. You should see a new folder named salaries. View its contents:

```
#  hadoop fs -ls salaries Found
4 items
-rw-r--r-- 1 root hdfs 272 salaries/part-m-00000 -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
```

4.3. 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 used four mappers, so there are four output files (one from each mapper).

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

```
# hadoop fs -cat salaries/part-m-00000
```

Step 5: Specify Columns to Import

```
#  sqoop import --connect jdbc:mysql://sandbox/test?user=root --table salaries
```

```
#  sqoop import --connect jdbc:mysql://sandbox/test?user=root \ --table
salaries \
--columns salary,age \ -m 1
\
--target-dir  salaries2
```

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

```
#  hadoop fs -ls salaries2 Found
1 items
-rw-r--r--    1 root hdfs   482  salaries2/part-m-00000
```

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

```
# hadoop fs -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
```

Step 6: Importing from a Query

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 --connect jdbc:mysql://sandbox/test?user=root \ --query "select *
from salaries s where s.salary > 90000.00 and \ \$CONDITIONS" \
-- split-by gender \ -m 2
\
--target-dir  salaries3
```

--This is how it should appear in the command line:

```
# sqoop import --connect jdbc:mysql://sandbox/test?user=root --query "select
* from salaries s where s.salary > 90000.00 and \$CONDITIONS" -- split-by gender -
m 2 --target-dir salaries3
```

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

```
# hadoop fs -ls salaries3
```

6.3. View the contents of part-m-00000 and part-m-00001.

```
# hadoop fs -cat salaries3/part-m-00000

# hadoop fs -cat salaries3/part-m-00001
```

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

Lab 2: Exporting HDFS Data to an RDBMS

This lab explores exporting data from HDFS into a MySQL table using

Table 5.

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 in MySQL named salary2.
Before you begin:	Your HDP 2.1 cluster should be up and running within your VM.
Related lesson:	Inputting Data into HDFS

Perform the following steps:

Step 1: Put the Data into HDFS

1.1. Change directories to /root/devph/labs/Lab3.2:

```
# cd ~/devph/labs/Lab3.2
```

1.2. View the contents of salarydata.txt:

```
#  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.

1.3. Create a new directory in HDFS named salarydata.

```
# hadoop fs -mkdir salarydata
```

1.4. Put salarydata.txt into the salarydata directory in HDFS.

```
# hadoop fs -mkdir salarydata
```

Step 2: Create a Table in the Database

2.1. 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:

```
# more salaries2.sql
```

2.2. Run this script using the following command:

```
# mysql test < salaries2.sql
```

2.3. Verify that the table was created successfully in MySQL:

```
# mysql
mysql>   use test;
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          |             |
+---------      +-----------      +------     +-----  +---------      +-------      +
```

2.4. Exit the mysql

```
mysql> exit
```

Step 3: Export the Data

3.1. 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://sandbox/test?user=root \ --table
salaries2 \
--export-dir  salarydata  \
--input-fields-terminated-by              ","
```

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

```
# mysql
mysql>   use     test;
mysql>   select * from salaries2  limit 10;
+--------        +    ------ +  --------    +     ---------    +
| gender | age        | salary         | zipcode   |
+--------        +          +              +                  +
| M             |  ------57 | --------39000  |   ---------95050 |
| 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  |
+--------     +    ------ +    -------- +     --------- +
```

3.3. Exit the mysql prompt.

HDP Developer: Apache Pig and Hive

Lab 3: Understanding MapReduce

This lab explores how MapReduce works.

Table 6.

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

Perform the following steps:

Step 1: Put the File into HDFS

1.1. Change directories to the demos folder:

```
# cd ~/devph/labs/demos/
```

1.2. Use more to look at a file named constitution.txt. Press q to exit when finished.

```
# more constitution.txt
```

1.3. Put the file into HDFS:

```
#  hadoop fs -put constitution.txt
```

Step 2: Run the WordCount Job

2.1. The following command runs a wordcount job on the constitution.txt and writes the output to wordcount_output:

```
#  yarn jar /usr/hdp/current/hadoop-mapreduce-historyserver/hadoop-mapreduce-
examples.jar wordcount constitution.txt wordcount_output
```

2.2. Notice that a MapReduce job gets submitted to the cluster. Wait for the job to complete.

Step 3: View the Results

3.1. View the contents of the wordcount_output folder:

3.2. Why is there one part-r file in this directory?
Answer: The job only used one reducer.

3.3. What does the “r” in the filename stand for?
Answer: The “r” stands for “reducer.”

3.4. View the contents of part-r-

```
# hadoop fs -cat wordcount_output/part-r-00000
```

3.5. Why are the words sorted alphabetically?________________________

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

3.6. What was the key output by the WordCount reducer?_______

Answer: The reducer's output key was each word.

3.7. What was the value output by the WordCount reducer?________

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

3.8. 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 as each value.

Lab 4: Running a MapReduce Job

This lab explores how to run a Java MapReduce job.

Table 7.

Objective:	Run a Java MapReduce job.
File locations:	/root/devph/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 2.2 cluster should be up and running within your VM.
Related lesson:	The MapReduce Framework

Perform the following steps:

Step 1: Put the Data into HDFS

1.1. 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/labs/Lab4.1:

```
# cd ~/devph/labs/Lab4.1
```

1.2. Use more to view the contents of the file hortonworks.txt.

```
# more hortonworks.txt
```

Each line looks like:

```
http://hortonworks.com/,hadoop,webinars,articles,download,enterprise,team ,rel iability
```

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

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

```
# hadoop fs -mkdir -p inverted/input
```

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

```
#  hadoop fs -mkdir -p inverted/input
```

Step 2: Run the Inverted Index Job

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

```
# hadoop jar invertedindex.jar inverted.IndexInverterJob inverted/input
inverted/output
```

2.2. 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
```

Step 3: View the Results

3.1. List the contents of the inverted/output folder.

```
# hadoop fs -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 of only one part-r-n file in the output directory.

3.2. Use the cat command to view the contents of inverted/output/part-r-00000. The file should look like:

```
# hadoop fs -cat inverted/output/part-r-00000 about
http://hortonworks.com/about-us/,apache
          http://hortonworks.com/products/hortonworksdataplatform/,http://hort onw
orks.com/about-us/,
articleshttp://hortonworks.com/community/,http://hortonworks.com/,
...
```

Step 4: Specify the Number of Reducers

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

```
# hadoop jar invertedindex.jar inverted.IndexInverterJob -D
mapreduce.job.reduces=3 inverted/input inverted/output
```

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

```
# hadoop fs -ls inverted/output Found
3 items
 1 root    hdfs              1221  inverted/output/part-r-00000
 1 root    hdfs               977 inverted/output/part-r-00001

 1 root    hdfs               799 inverted/output/part-r-00002
```

4.3. 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 hashing of the key and using the remainder of dividing by the number of reducers.

Lab· 5: Understanding and Getting started with Pig

This lab explores Pig scripts and relations.

Table 8.

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

Perform the following steps:

Step 1: Start the Grunt Shell

1.1. Review the contents of the file pigdemo.txt located in

```
# more root/devph/labs/demos/pigdemo.txt
```

1.2. Start the Grunt shell:

```
# pig
```

1.3. Notice that the output includes where the logging for your Pig session will go as well

```
[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
```

Step 2: Make a New Directory

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

```
grunt> ls
```

2.2. Make a new directory named demos:

```
grunt> mkdir demos
```

2.3. Use copyFromLocal to copy the pigdemo.txt file into the demos

```
grunt> copyFromLocal  /root/devph/labs/demos/pigdemo.txt           demos/
```

2.4. Verify the file was uploaded successfully:

```
grunt> ls demos
hdfs://sandbox.hortonworks.com:8020/user/root/demos/pigdemo.txt<r              3>
```

2.5. Change the present working directory to demos:

```
grunt> cd demos
grunt> pwd
hdfs://sandbox.hortonworks.com:8020/user/root/dem
```

2.6. View the contents using the cat command:

```
grunt>    cat pigdemo.txt
SD          Rich
NV          Barry
CO
George CA
Ulf
IL          Danielle
OH          Tom

manish CA
Brian  CO
Mark
```

Step 3: Define a Relation

3.1. Define the employees relation, using a schema:

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

3.2. Demonstrate the describe command, which describes what a relation looks

```
grunt> describe  employees;
employees:  {state:  bytearray,name:            bytearray}
```

3.3. 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)
```

Step 4: Filter the Relation by a Field

4.1. Let's filter the employees whose state field equals CA:

```
grunt> ca_only  = FILTER employees  BY (state=='CA');
grunt> DUMP ca_only;
```

4.2. The output is still tuples, but only the records that match the filter

```
(CA,Ulf)
(CA,manish
)
```

Step 5: Create a Group

5.1. Define a relation that groups the employees by the state **grunt> emp_group = GROUP employees BY state;**

5.2. Bags represent groups in Pig. A bag is an unordered collection of

```
grunt> describe  emp_group;
emp_group:  {group:  bytearray,employees:           {(state:  bytearray,name:
```

5.3. All records with the same state will be grouped together, as shown by the output of

```
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)})
```

Step 6: The STORE Command

6.1. The DUMP command dumps the contents of a relation to the console. The STORE

```
grunt> STORE emp_group  INTO 'emp_group';
```

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

```
grunt> ls
hdfs://sandbox.hortonworks.com:8020/user/root/demos/emp_group                  <dir>
hdfs://sandbox.hortonworks.com:8020/user/root/demos/pigdemo.txt<r              3>      89
```

6.3. View the contents of the output

```
grunt> cat emp_group/part-r-00000
```

```
CA
{(CA,Ulf),(CA,manish),(CA,Brian)} CO
{(CO,George),(CO,Mark)}
IL
{(IL,Danielle)} NV
```

```
{(NV,Barry)}
grunt> STORE emp_group  INTO 'emp_group_csv'            USING PigStorage(',');
```

To view the results:

```
grunt > ls

grunt > cat emp_group_csv/part-r-00000
```

Step 7: Show All Aliases

7.1. The aliases command shows a list of currently defined

```
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.

Step 8: Monitor the Pig Jobs

8.1. Point your browser to the JobHistory UI at http://sandbox:19888/.

8.2. View the list of jobs, which should contain the MapReduce jobs that were executed from your Pig Latin code in the Grunt shell.

8.3. Notice you can view the log files of the ApplicationMaster and also each map and reduce task.

HDP Developer: Apache Pig and Hive

Lab 5: Understanding and Getting Started with Pig

This lab explores using Pig to navigate through HDFS and explore a dataset.

Table 9.

Objective:	**Successful outcome:**
File locations:	

Before you begin:

Related lesson:

Use Pig to navigate through HDFS and explore a dataset.

/root/devph/labs/Lab5.1

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.

Your HDP 2.2 cluster should be up and running within your VM.

Introduction to Pig

Perform the following steps:

Step 1: View the Raw Data

1.1. Change directories to the /root/devph/labs/Lab5.1 folder:

```
# cd ~/devph/labs/Lab5.1
```

1.2. Unzip the archive in the /root/devph/labs/Lab5.1 folder, which contains a file named whitehouse_visits.txt that is quite large:

```
# unzip whitehouse_visits.zip
```

1.3. View the contents of this file:

```
# tail whitehouse_visits.txt
```

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

Step 2: Load the Data into HDFS

2.1. Start the Grunt shell:

```
# pig
```

2.2. From the Grunt shell, make a new directory in HDFS named whitehouse:

```
grunt> mkdir whitehouse
```

2.3. 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/devph/labs/Lab5.1/whitehouse_visits.txt
```

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

```
grunt> ls whitehouse
hdfs://sandbox.hortonworks.com:8020/user/root/whitehouse/visits.txt<r
3>
```

Step 3: Define a Relation

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

Define the following LOAD relation:

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

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

```
grunt> DESCRIBE  A;
Schema for A
```

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

```
grunt> A_limit  = LIMIT A 10
```

3.4. 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,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,
```

Step 4: Define a Schema

4.1. 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
);
```

4.2. Use the DESCRIBE command to view the schema:

```
grunt> describe  B;
B:   {lname: chararray,fname: chararray,mname: chararray,id:
chararray,status:
```

Step 5: The STORE Command

5.1. 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');
```

5.2. Verify that the whouse_tab folder was created:

```
grunt> ls whouse_tab;
```

You should see two map output files.

5.3. View one of the output files to v~~erify they contain the B relation in a tab-~~ delimited format:

```
grunt> fs -tail whouse_tab/part-m-00000;
```

5.4. 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 from HDFS.

Step 6: Use a Different Storer

6.1. 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();
```

6.2. Verify that the whouse_json folder was created:

```
grunt> ls whouse_json;
```

6.3. View one of the output files:

```
grunt> fs -tail whouse_json/part-m-00000;
```

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

```
{"lname":"MATTHEWMAN","fname":"ROBIN","mname":"H","id":"U81961","status":"735
74","state":"VA","arrival":"2/10/2011 11:14"}
{"lname":"MCALPINEDILEM","fname":"JENNIFER","mname":"J","id":"U81961","status
":"78586","state":"VA","arrival":"2/10/2011 10:49"}
```

HDP Developer: Apache Pig and Hive

Lab 6: Exploring Data with Pig

This lab explores using Pig to navigate through HDFS and explore a dataset.

Table 10. About&this&Lab

Objective:	Use Pig to navigate through HDFS and explore a dataset.
File locations:	whitehouse/visits.txt in HDFS
Successful outcome:	You will have written several Pig scripts that analyze and query the White House visitors' data, including a list of people who visited the President.
Before you begin:	At a minimum, complete steps 1 and 2 of the Getting Started with Pig lab.
Related lesson:	Introduction to Pig

Perform the following steps:

Step 1: Load the White House Visitor Data

1.1. You will use the TextLoader to load the visits.txt file. From the Pig Grunt shell, define the following LOAD relation:

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

Step 2: Count the Number of Lines

2.1. Define a new relation named B that is a group of all the records in A:

```
grunt> B = GROUP A ALL;
```

2.2. Use DESCRIBE to view the schema of B.

```
grunt> DESCRIBE  B;
```

What is the datatype of the group field?____________________

Where did this datatype come from?

____

Answer: The group field is a chararray because it is just the string "all" and is a result of performing a GROUP ALL.

2.3. Why does the A field of B contain no schema?____________________

Answer: The A field of B contains no schema because the A relation has no schema.

2.4. How many groups are in the relation B? ______________

Answer: The B relation can only contain one group because it a grouping of every single record. Note that the A field is a bag, and A will contain any number of tuples.

2.5. The A field of the B tuple is a bag of all of the records in visits.txt. Use the COUNT function on this bag to determine how many lines of text are in visits.txt:

```
grunt> A_count  = FOREACH  B GENERATE  'rowcount',  COUNT(A);
```

2.6. Use DUMP on A_count to view the result. The output should look like:

```
grunt> DUMP A_count;
(rowcount,447598)
```

We can now conclude that there are 447,598 rows of text in visits.txt. **Step 3:** Analyze the Data's Contents

3.1. We now know how many records are in the data, but we still do not have a clear picture of what the records look like. Let's start by looking at the fields of each record. Load the data using PigStorage(',') instead of TextLoader():

```
grunt> visits = LOAD '/user/root/whitehouse/'                    USING PigStorage(',');
```

This will split up the fields by comma.

3.2. Use a FOREACH...GENERATE command to define a relation that is a projection of the first 10 fields of the visits relation.

```
grunt> firstten  = FOREACH  visits GENERATE  $0..$9;
```

3.3. Use LIMIT to display only 50 records then DUMP the result. The output should be 50 tuples that represent the first 10 fields of visits:

```
grunt> firstten_limit  = LIMIT firstten  50;
grunt> DUMP firstten_limit;
(PARK,ANNE,C,U51510,0,VA,10/24/2010           14:53,B0402,,)
(PARK,RYAN,C,U51510,0,VA,10/24/2010           14:53,B0402,,)
(PARK,MAGGIE,E,U51510,0,VA,10/24/2010
14:53,B0402,,)
(PARK,SIDNEY,R,U51510,0,VA,10/24/2010
14:53,B0402,,)
(RYAN,MARGUERITE,,U82926,0,VA,2/13/2011
17:14,B0402,,)         (WILE,DAVID,J,U44328,,VA,,,,)
(YANG,EILENE,D,U82921,,VA,,,,)
```

```
(ADAMS,CHRISTINE,M,U51772,,VA,,,,)
(BERRY,STACEY,,U49494,79029,VA,10/15/2010 12:24,D0101,10/15/2010
```

Step 4: Locate the POTUS (President of the United States of America)

4.1. There are 26 fields in each record, and one of them represents the visitee (the person being visited in the White House). Your goal now is to locate this column and determine who has visited the President of the United States. Define a relation that is a projection of the last seven fields ($19 to $25) of visits. Use LIMIT to only output 500 records. The output should look like:

```
grunt> lastfields = FOREACH visits GENERATE $19..$25; grunt>
lastfields_limit = LIMIT lastfields 500; grunt> DUMP lastfields_ limit;
(OFFICE,VISITORS,WH,RESIDENCE,OFFICE,VISITORS,HOLIDAY              OPEN HOUSE/)
(OFFICE,VISITORS,WH,RESIDENCE,OFFICE,VISITORS,HOLIDAY OPEN HOUSES/)
(OFFICE,VISITORS,WH,RESIDENCE,OFFICE,VISITORS,HOLIDAY              OPEN HOUSE/)
(CARNEY,FRANCIS,WH,WW,ALAM,SYED,WW     TOUR)
(CARNEY,FRANCIS,WH,WW,ALAM,SYED,WW     TOUR)
(CARNEY,FRANCIS,WH,WW,ALAM,SYED,WW     TOUR)
(CHANDLER,DANIEL,NEOB,6104,AGCAOILI,KARL,)
```

It is not necessarily obvious from the output, but field $19 in the visits relation represents the visitee. Even though you selected 500 records in the previous step, you may or may not see POTUS in the output above. (The White House has thousands of visitors each day, but only a few meet the President.)

4.2. Use FILTER to define a relation that only contains records of visits where field $19 matches POTUS. Limit the output to 500 records. The output should include only visitors who met with the President. For example:

```
grunt> potus = FILTER visits BY $19 MATCHES 'POTUS'; grunt>
potus_limit = LIMIT potus 500;
grunt> DUMP potus_limit;
```

```
(ARGOW,KEITH,A,U83268,,VA,,,,,2/14/2011 18:42,2/16/2011 16:00,2/16/2011
23:59,,154,LC,WIN,2/14/2011 18:42,LC,POTUS,,WH,EAST
ROOM,THOMPSON,MARGRETTE,,AMERICA'S     GREAT OUTDOORS  ROLLOUT
EVENT
,5/27/2011,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,
,,,,,,,)
(AYERS,JOHNATHAN,T,U84307,,VA,,,,,2/18/2011 19:11,2/25/2011
17:00,2/25/2011
23:59,,619,SL,WIN,2/18/2011
```

Step 5: Count the POTUS Visitors

5.1. Let's discover how many people have visited the President. To do this, we need to count the number of records in visits where field $19 matches POTUS. See if you can write a Pig script to accomplish this. Use the potus relation from the previous step as a starting point. You will need to use GROUP ALL and then a FOREACH projection that uses the COUNT function.

If successful, you should get 21,819 as the number of visitors to the White House who visited the President.

Solution:

```
grunt> potus = FILTER visits BY $19 MATCHES 'POTUS'; grunt>
potus_group = GROUP potus ALL;
grunt> potus_count = FOREACH potus_group GENERATE COUNT(potus);
grunt> DUMP potus_ count;
```

Step 6: Finding People Who Visited the President

6.1. So far you have used DUMP to view the results of your Pig scripts. In this step, you will save the output to a file using the STORE command.

6.2. Now FILTER the relation by visitors who met with the President:

```
grunt> potus = FILTER visits BY $19 MATCHES  'POTUS';
```

6.3. Define a projection of the potus relationship that contains the name and time of arrival of the visitor:

```
grunt> potus_details = FOREACH potus GENERATE
(chararray) $0 AS lname:chararray, (chararray)
$1 AS fname:chararray,
(chararray)  $6 AS
arrival_time:chararray,  (chararray)  $19
```

6.4. Order the potus_details projection by last name:

```
grunt> potus_details_ordered            = ORDER potus_details  BY lname ASC;
```

6.5. Store the records of potus_details_ordered into a folder named potus and using a comma delimiter:

```
grunt> STORE potus_details_ordered              INTO 'potus'  USING PigStorage(',');
```

6.6. View the contents of the potus folder:

```
grunt> ls potus
hdfs://sandbox.hortonworks.com:8020/user/root/potus/_SUCCESS<r                    3>
0  hdfs://sandbox.hortonworks.com:8020/user/root/potus/part-r-00000<r 3>
```

6.7. Notice that there is a single output file, so the Pig job was executed with one reducer. View the contents of the output file using cat:

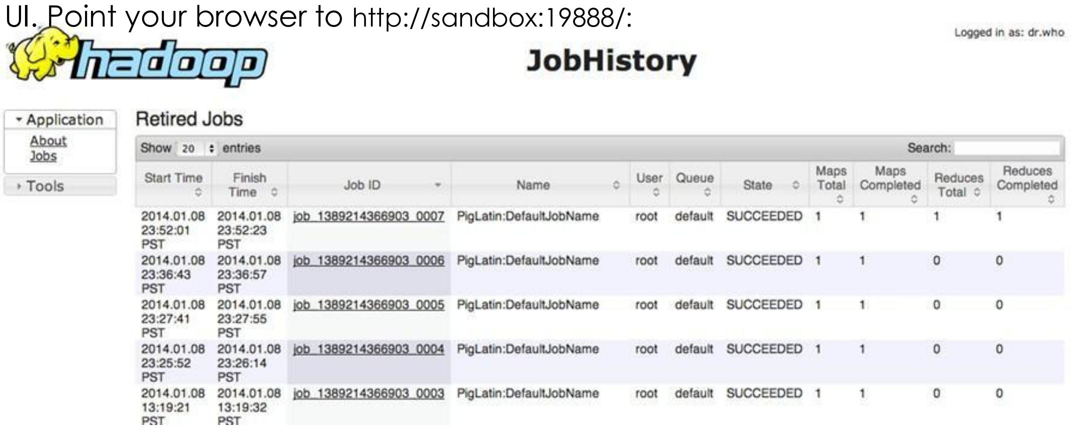
```
grunt> cat potus/part-r-00000
```

The output should be in a comma-delimited format and should contain the last name, first name, time of arrival (if available), and the string POTUS:

```
CLINTON,WILLIAM,,POTUS
CLINTON,HILLARY,,POTUS
CLINTON,HILLARY,,POTUS
CLINTON,HILLARY,,POTUS
CLONAN,JEANETTE,,POTUS
CLOOBECK,STEPHEN,,POTUS
CLOOBECK,CHANTAL,,POTUS
CLOOBECK,STEPHEN,,POTUS
CLOONEY,GEORGE,10/12/2010
14:47,POTUS
```

Step 7: View the Pig Log Files

7.1. Each time you executed a DUMP or STORE command, a MapReduce job is executed on your cluster. You can view the log files of these jobs in the JobHistory UI. Point your browser to http://sandbox:19888/:



Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2014.01.08 23:52:01 PST	2014.01.08 23:52:23 PST	job_1389214366903_0007	PigLatin:DefaultJobName	root	default	SUCCEEDED	1	1	1	1
2014.01.08 23:36:43 PST	2014.01.08 23:36:57 PST	job_1389214366903_0006	PigLatin:DefaultJobName	root	default	SUCCEEDED	1	1	0	0
2014.01.08 23:27:41 PST	2014.01.08 23:27:55 PST	job_1389214366903_0005	PigLatin:DefaultJobName	root	default	SUCCEEDED	1	1	0	0
2014.01.08 23:25:52 PST	2014.01.08 23:26:14 PST	job_1389214366903_0004	PigLatin:DefaultJobName	root	default	SUCCEEDED	1	1	0	0
2014.01.08 13:19:21 PST	2014.01.08 13:19:32 PST	job_1389214366903_0003	PigLatin:DefaultJobName	root	default	SUCCEEDED	1	1	0	0

7.2. Click on the job's ID to view the details of the job and its log files.