Hadoop Developer Training – Pig Programming Lab Book

Table of Contents

Lab 1 : Getting Started with Pig

Location of Files:	/<*userpath*>/labs/pig_basic

Note: Please use the below paths to avoid error while practicing
1. <*userpath*> should be your working directory(user home directory)
2. for Cloudera HDFS path is : `user/training/`
3. for Hartonworks HDFS path is : `user/root/`

Step 1: View the Raw Data

1.1. Change directories to the Lab5.1 folder:

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

1.2. Unzip the archive in the **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/labs/Lab5.1/whitehouse_visits.txt
whitehouse/visits.txt
```

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

```
grunt> ls whitehouse
hdfs://sandbox:8020/user/root/whitehouse/visits.txt<r 1>
 175153242
```

Step 3: Define a Relation

3.1. 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();
```

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

```
grunt> DESCRIBE A;
```

```
Schema for A unknown.
```

3.3. 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**.

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**:

```
 (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: chararray,state:
chararray,arrival: chararray}
```

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 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 verify they contain the B relation in a tab-delimited format:

```
grunt> cat whouse_tab/part-m-00000
```

5.4. Each record should contain 7 fields. What happened to the rest of the fields from the raw data that was loaded from **whitehouse/visits.txt**?

_____

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 the **whouse_json folder** was created:

```
grunt> ls whouse_json
```

6.3. View one of the output files:

```
grunt> cat 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":"U81
961","status":"73574","state":"VA","arrival":"2/10/2011
11:14"}
{"lname":"MCALPINEDILEM","fname":"JENNIFER","mname":"J","id
":"U81961","status":"78586","state":"VA","arrival":"2/10/20
11 10:49"}
```

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 2: Exploring Data with Pig

Location of Files:	**whitehouse/vistis.txt** in HDFS

Step 1: Load the White House Visitor Data

1.1. You will use the **TextLoader** to load the **visits.txt** file. Define the following **LOAD** relation:

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

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**. What is the datatype of the **group** field? _____ Where did this datatype come from?

_____

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

2.4. How many groups are in the relation **B**? _____

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);
```

NOTE: The **'rowcount'** string in the **FOREACH** statement is simply to demonstrate that you can have constant values in a **GENERATE** clause. It is

certainly not necessary - just makes the output nicer to read.

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

```
(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.

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**:

```
(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,SCHUYLER,N,U51772,,VA,,,,)
(ADAMS,CHRISTINE,M,U51772,,VA,,,,)
(BERRY,STACEY,,U49494,79029,VA,10/15/2010
12:24,D0101,10/15/2010 14:06,D1S)
```

NOTE: Because **LIMIT** uses an arbitrary sample of the data, your output will be different names, but the format should look similar.

Notice from the output that the first three fields are the person's name. The next 7 fields are a unique ID, badge number, access type, time of arrival, post of arrival, time of departure and post of departure.

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 7 fields (**$19** to **$25**) of **visits**. Use **LIMIT** to only output 500 records. The output should look like:

```
(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,KA
RL,)
```

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:

```
(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

19:11,SL,POTUS,,WH,STATE FLOO,GALLAGHER,CLARE,,RECEPTION

,5/27/2011,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,
,,)
```

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 AL**L, and then a **FOREACH** projection that uses the **COUNT** function.

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

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. Start by loading the data using **PigStorage(',')**, which you may already have defined:

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

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

```
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
AS visitee:chararray;
```

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/_SUC
CES S<r 3> 0
hdfs://sandbox.hortonworks.com:8020/user/root/potus/part
-r-00000<r 3> 501378
```

6.7. Notice 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 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 executed on your cluster. You can view the log files of these jobs in the JobHistory UI. Point your browser to **http://*ipaddress*:19888/**:

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

RESULT: You have written several Pig scripts to analyze and query the data in the White House visitors' log. You should now be comfortable with writing Pig scripts with the Grunt shell and using common Pig commands like **LOAD**, **GROUP**, **FOREACH**, **FILTER**, **LIMIT**, **DUMP** and **STORE**.

ANSWERS:
Step 2: The **group** field is a **chararray** because it is just the string **"all"** and is a result of performing a **GROUP ALL**. The **A** field of **B** contains no schema because the **A** relation has no schema. The **B** relation can only contain 1 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.

SOLUTIONS:

Step 3:

```
visits = LOAD '/user/root/whitehouse/'
USING PigStorage(',');
firstten = FOREACH visits GENERATE $
0..$9; firstten_limit = LIMIT firstten
50;
DUMP firstten_limit;
```

Step 4:

```
lastfields = FOREACH visits GENERATE $19..$25;
lastfields_limit = LIMIT lastfields 500;
DUMP lastfields_limit;

--find the POTUS
potus = FILTER visits BY $19 MATCHES
'POTUS'; potus_limit = LIMIT potus 500;
DUMP potus_limit;
```

Step 5:

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

Lab 3 : Splitting a Dataset

Location of Files	/user/root/whitehouse/visits.txt.

Perform the following steps:

Step 1: Explore the Comments Field

1.1. In this step, you will explore the comments field of the White House visitor data. Start by loading **visits.txt**:

```
cd whitehouse
visits = LOAD 'visits.txt' USING PigStorage(',');
```

1.2. Field **$25** is the comments. Filter out all records where field **$25** is null:

```
not_null_25 = FILTER visits BY ($25 IS NOT NULL);
```

1.3. Now define a new relation that is a projection of only column **$25**:

```
comments = FOREACH not_null_25 GENERATE $25 AS comment;
```

1.4. View the schema of **comments** and make sure you understand how this relation ended up as a tuple with one field:

```
grunt> describe comments;
comments: {comment: bytearray}
```

Step 2: Test the Relation

2.1. A common Pig task is to test a relation to make sure it is consistent with what you are intending it to be. But using **DUMP** on a big data relation might take too long or not be practical, so define a **SAMPLE** of **comments**:

```
comments_sample = SAMPLE comments 0.001;
```

2.2. Now **DUMP** the **comments_sample** relation. The output should be non-null comments about visitors to the White House, similar to:

```
(ATTENDEES VISITING FOR A
MEETING) (FORUM ON IT MANAGEMENT
REFORM/) (FORUM ON IT MANAGEMENT
REFORM/) (HEALTH REFORM MEETING)
(DRIVER TO REMAIN WITH VEHICLE)
```

Step 3: Count the Number of Comments

3.1. The **comments** relation represents all non-null comments from **visits.txt**. Write Pig statements that output the number of records in the **comments** relation. The correct result is 222,839 records.

Step 4: Split the Dataset

> **NOTE**: Our end goal is find visitors to the White House who are also members of Congress. We could run our MapReduce job on the entire **visits.txt** dataset, but it is common in Hadoop to split data into smaller input files for specific tasks, which can greatly improve the performance of your MapReduce applications. In this step, you will split **visits.txt** into two separate datasets.

4.1. In this step, you will split visits.txt into two datasets: those that contain "CONGRESS" in the comments field, and those that do not. Start by loading the data:

```
visits = LOAD 'visits.txt' USING PigStorage(',')
```

4.2. Use the **SPLIT** command to split the **visits** relation into two new relations named **congress** and **not_congress**:

```
SPLIT visits INTO congress IF($25 MATCHES
'.* CONGRESS .*'), not_congress IF (NOT($25 MATCHES
'.* CONGRESS .*'));
```

4.3. Store the **congress** relation into a folder named **'congress'** using a JSON format:

```
STORE congress INTO 'congress';
```

4.4. Similarly, **STORE** the **not_congress** relation in a folder named **'not_congress'**.

4.5. View the output folders using **ls**. The file sizes should be equivalent to the following:

```
grunt> ls congress
whitehouse/congress/part-m-00000<r 1>   45618
whitehouse/congress/part-m-00001<r 1>   0
grunt> ls not_congress
whitehouse/not_congress/part-m-00000<r 1>    90741587
whitehouse/not_congress/part-m-00001<r 1>    272381
```

4.6. View one of the output files in **congress**: and make sure the string "CONGRESS" appears in the comment field:

```
cat congress/part-m-00000
```

Step 5: Count the Results

5.1. Write Pig statements that output the number of records in the **congress** relation. This will tell us how many visitors to the White House have "CONGRESS" in the comments of their visit log. The correct result is 102.

> **NOTE**: You now have two datasets: one in **'congress'** with 102 records, and the remaining records in the **'not_congress'** folder. These records are still in their original, raw format.

RESULT: You have just split '**visits.txt**' into two datasets, and you have also discovered that 102 visitors to the White House had the word "CONGRESS" in their comments field. We will further explore these visitors in the next lab as we perform a join with a dataset containing the names of members of Congress.

SOLUTIONS: Here is a solution to Step 3:

```
comments_all = GROUP comments ALL; comments_count
= FOREACH comments_all GENERATE
                COUNT(comments);
DUMP comments_count;
```

Here is a solution to Step 5:

```
congress_grp = GROUP congress ALL; congress_count
= FOREACH congress_grp GENERATE
                COUNT(congress);
DUMP congress_count;
```

Lab 4 : Joining Datasets

Location of Files	/root/labs/Lab6.2

Perform the following steps:

Step 1: Upload the Congress Data

1.1. Put the file **/root/labs/Lab6.2/congress.txt** into the **whitehouse** directory in HDFS.

1.2. Use the **hadoop fs -ls** command to verify the **congress.txt** file is in **whitehouse**, and use **hadoop fs -cat** to view its contents. The file contains the names and other information about the members of the U.S. Congress.

Step 2: Create a Pig Script File

2.1. In this lab, you will not use the Grunt shell to enter commands. Instead, you will enter your script in a text file. Using a text editor, create a new file named **join.pig** in the **Lab6.2** folder.

2.2. At the top of the file, add a comment:

```
--join.pig: joins congress.txt and visits.txt
```

Step 3: Load the White House Visitors

3.1. Define the following **visitors** relations, which will contain the first and last names of all White House visitors:

```
visitors = LOAD 'whitehouse/visits.txt' USING
PigStorage(',') AS (lname:chararray, fname:chararray);
```

That is the only data we are going to use from **visits.txt**.

Step 4: Define a Projection of the Congress Data

4.1. Add the following load command that loads the **'congress.txt'** file into a relation named **congress**. The data is tab-delimited, so no special Pig loader is needed:

```
congress = LOAD 'whitehouse/congress.txt' AS
   ( full_title:chararray,
   district:chararray,
   title:chararray,
   fname:chararray,
   lname:chararray,
   party:chararray
);
```

4.2. The names in **visits.txt** are all uppercase, but the names **in congress.txt** are not. Define a projection of the **congress** relation that consists of the following fields:

```
congress_data = FOREACH congress
   GENERATE district,
   UPPER(lname)        AS
   lname,    UPPER(fname)
   AS fname, party;
```

Step 5: Join the Two Datasets

5.1. Define a new relation named **join_contact_congress** that is a **JOIN** of **visitors** and **congress_data**. Perform the join on both the first and last names.

5.2. Use the **STORE** command to store the result of **join_contact_congress** into a directory named **'joinresult'**.

Step 6: Run the Pig Script

6.1. Save your changes to **join.pig**.

6.2. Run the script using the following command:

```
# pig join.pig
```

6.3. Wait for the MapReduce job to execute. When it is finished, write down the number of seconds it took for the job to complete (by subtracting the **StartedAt** time from the **FinishedAt** time) and write down the result: _____

6.4. The type of join used is also output in the job statistics. Notice the statistics output has "**HASH_JOIN**" underneath the "**Features**" column, which means a hash join was used to join the two datasets.

Step 7: View the Results

7.1. The output will be in the **joinresult** folder in HDFS. Verify the folder was created:

```
# hadoop fs -ls -R joinresult
```

```
-rw-r--r--   1 root hdfs      40892 joinresult/part-r-00000
```

7.2. View the resulting file:

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

The output should look like the following:

```
DUFFY  SEAN  WI07  DUFFY SEAN Republican
JONES  WALTER     NC03  JONES WALTER    Republican
SMITH  ADAM  WA09  SMITH ADAM Democrat
CAMPBELL    JOHN  CA45  CAMPBELL   JOHN  Republican
CAMPBELL    JOHN  CA45  CAMPBELL   JOHN  Republican
SMITH  ADAM  WA09  SMITH ADAM Democrat
```

Step 8: Try Using Replicated on the Join

8.1. Delete the **joinresult** directory in HDFS:

```
# hadoop fs -rm -R joinresult
```

8.2. Modify your **JOIN** statement in **join.pig** so that is uses replication.

8.3. Save your changes **to join.pig** and run the script again.

8.4. Notice this time that the statistics output shows Pig used a "**REPLICATED_JOIN**" instead of a "**HASH_JOIN**".

8.5. Compare the execution time of the REPLICATED_JOIN vs. the HASH_JOIN. Did you have any improvement or decrease in performance?

> **NOTE**: Using replicated does not necessarily increase the join time. There are way too many factors involved, and this example is using small datasets. The point is that you should try both techniques (if one dataset is small enough to fit in memory) and determine which join algorithm is faster for your particular dataset and use case.

Step 9: Count the Results

9.1. In **join.pig**, comment out the **STORE** command:

```
--STORE join_contact_congress INTO 'joinresult';
```

You have already saved the output of the **JOIN**, so there is no need to perform the **STORE** command again.

9.2. Notice in the output of your **join.pig** script that we know which party the visitor belongs to: Democrat, Republican or Independent. Using the **join_contact_congress** relation as a starting point, see if you can figure out how to output the number of Democrat, Republican and Independent

members of Congress that visited the White House. Name the relation **counters** and use the **DUMP** command to output the results:

```
DUMP counters;
```

HINT: When you group the **join_contact_congress** relation, group it by the **party** field of **congress_data**. You will need to use the **::** operator in the **BY** clause. It will look like:

```
congress_data::party
```

9.3. The correct results are shown here:

```
(Democrat,637)
(Republican,351)
(Independent,2)
```

Step 10: Use the EXPLAIN Command

10.1. At the end of **join.pig**, add the following statement:

```
EXPLAIN counters;
```

If you do not have a **counters** relation, then use **join_contact_congress** instead.

10.2. Run the script again. The Logical, Physical and MapReduce plans should dispay at the end of the output.

10.3. How many MapReduce jobs did it take to run this job? _____

RESULT: You should have a folder in HDFS named **joinresult** that contains a list of members of Congress who have visited the White House (within the timeframe of the historical data in **visits.txt**).

SOLUTIONS: The **JOIN** and **STORE** commands in Step 5 look like:

```
join_contact_congress = JOIN visitors BY (lname,fname),
                        congress_data BY (lname,fname);
STORE join_contact_congress INTO 'joinresult';
```

A solution for Step 9 is:

```
join_group = GROUP join_contact_congress
             BY congress_data::party;
counters = FOREACH join_group GENERATE group,
             COUNT(join_contact_congress);
DUMP counters;
```

Lab 5: Preparing Data for Hive

Location of Files	/root/labs/Lab6.3

Perform the following steps:

Step 1: Review the Pig Script

1.1. From a command prompt, change directories to **Lab6.3**:

```
# cd ~/labs/Lab6.3
```

1.2. View the contents of **wh_visits.pig**:

```
# more wh_visits.pig
```

1.3. Notice the **potus** relation is all White House visitors who met with the President.

1.4. Notice the **project_potus** relation is a projection of the last name, first name, time of arrival, location and comments from the visit.

Step 2: Store the Projection in the Hive Warehouse

2.1. Open **wh_visits.pig** with a text editor.

2.2. 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/';
```

Step 3: Run the Pig Script

3.1. Save your changes to **wh_visits.pig**.

3.2. Run the script from the command line:

```
# pig wh_visits.pig
```

Step 4: View the Results

4.1. The **wh_visits.pig** script creates a directory in the Hive warehouse named **wh_visits**. Use **ls** to view its contents:

```
# hadoop fs -ls /apps/hive/warehouse/wh_visits/

-rw-r--r-- 3 root hdfs 971339
/apps/hive/warehouse/wh_visits/part-m-00000
-rw-r--r-- 3 root hdfs 142850
/apps/hive/warehouse/wh_visits/part-m-00001
```

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

```
hadoop fs -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.

Demonstration: Computing PageRank

Objective:	To understand how to use the PageRank UDF in DataFu.
During this Demonstration:	Watch as your instructor performs the following steps.

Step 1: View the Data

1.1. Review the **edges.txt** file in the **/root/labs/demos** folder:

```
• cd ~/labs/demos/
• more edges.txt
0     2     3     1.0
0     3     2     1.0
0     4     1     1.0
0     4     2     1.0
0     5     4     1.0
0     5     2     1.0
0     5     6     1.0
0     6     5     1.0
0     6     2     1.0
0     100   2     1.0
0     100   5     1.0
0     101   2     1.0
0     101   5     1.0
0     102   2     1.0
0     102   5     1.0
0     103   5     1.0
0     104   5     1.0
```

- The first column is the topic, but since we only have a single graph, the topic is 0 for all the edges.
- The second and third columns are the source and destination of each

edge. For example, there is an edge from 2 to 3 based on the first row.

- The fourth column is the weight of the edge. Our graph is all evenly weighted.

1.5. Based on the data above, which pages should be ranked toward the top?

_____

Step 2: Put the Data in HDFS

2.1. Put **edges.txt** into HDFS:

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

Step 3: Define the PageRank UDF

3.1. View the contents of **demos/pagerank.pig**. The first two lines register the DataFu library and define the **PageRank** function:

```
register /root/labs/Lab10.1/datafu-0.0.10.jar; define
PageRank datafu.pig.linkanalysis.PageRank();
```

3.2. The edges are loaded and grouped by **topic** and **source**:

```
topic_edges = LOAD '/user/root/edges.txt' as
              (topic:INT,source:INT,dest:INT,weight:DOUBLE);
topic_edges_grouped = GROUP topic_edges by (topic, source);
```

3.3. The data is then prepared for the **PageRank** function, which is expecting a topic, a source, and its edges:

```
topic_edges_data = FOREACH topic_edges_grouped GENERATE
      group.topic as topic,
      group.source as source,
      topic_edges.(dest,weight) as edges;
```

3.4. We only have one topic, but the edges still need to be grouped by topic:

```
topic_edges_data_by_topic = GROUP topic_edges_data BY topic;
```

3.5. We can now invoke the **PageRank** function:

```
topic_ranks = FOREACH topic_edges_data_by_topic GENERATE
      group as topic,
      FLATTEN(PageRank(topic_edges_data.(source,edges)));
```

3.6. The results are stored in HDFS:

```
store topic_ranks into 'topicranks';
```

Step 4: Run the Script

4.1. From the command prompt, enter:

```
# pig pagerank.pig
```

The job will take a couple minutes to run.

Step 5: View the Results

5.1. View the contents of the **topicranks** folder in HDFS:

```
# hadoop fs -ls topicranks
Found 1 items
root hdfs          181   topicranks/part-r-00000
```

5.2. View the contents of the output file:

```
# hadoop fs -cat topicranks/part-r-00000
0      104   0.013636362
0      1     0.02764593
0      103   0.013636362
0      5     0.06821412
0      100   0.013636362
0      102   0.013636362
0      6     0.032963693
0      3     0.2891899
0      2     0.32418048
0      4     0.032963693
0      101   0.013636362
```

Step 6: Analyze the Results

- Which page should be ranked the highest? _____
- Which page should be ranked the lowest? _____
- Compare the actual results with your guess from Step 1.

Lab 6: Analyzing Clickstream Data

Location of Files	/root/labs/Lab6.4

Step 1: View the Clickstream Data

1.1. Open a command prompt and change directories to **Lab6.4**.

1.2. View the contents of **clicks.csv**:

```
more clicks.csv
```

The first column is the user's **id**, the second column is the **time** of the click stored as a long, and the third column is the **URL** visited.

1.3. Put the file in HDFS:

```
hadoop fs -put clicks.csv clicks.csv
```

Step 2: Define the Sessionize UDF

a. Using a text editor, open the file **sessions.pig** in the **Lab6.4** folder.

b. Notice two JAR files are registered: **datafu-0.0.10.jar** and **piggybank.jar**. The datafu JAR contains the **Sessionize** function that you are going to use, and the **piggybank.jar** contains a time utility function named **UnixToISO**, which is already defined for you in this Pig script.

2.3. Add the following **DEFINE** statement to define the **Sessionize** UDF:

```
DEFINE Sessionize datafu.pig.sessions.Sessionize('8m');
```

2.4. What does the '**8m**' mean in the constructor?

_____

Step 3: Sessionize the Clickstream

3.1. Notice the **clicks.csv** file is loaded for you in **sessions.pig**:

```
clicks = LOAD 'clicks.csv' USING
         PigStorage(',') AS (id:int, time:long,
         url:chararray);
```

3.2. Notice also that the **clicks** relation is projected onto **clicks_iso** with the long converted to an ISO time format, then grouped by **id** in the **clicks_group** relation:

```
clicks_iso = FOREACH clicks GENERATE
             UnixToISO(time) AS isotime, time, id;
clicks_group = GROUP clicks_iso BY id;
```

3.3. Sessionize the clickstream by adding the following nested **FOREACH** loop:

```
clicks_sessionized = FOREACH clicks_group
     {  sorted  =  ORDER  clicks_iso  BY
     isotime;                    GENERATE
     FLATTEN(Sessionize(sorted))
     AS (isotime, time, id, sessionid);
}
```

3.4. Dump the sessionized data:

```
dump clicks_sessionized;
```

3.5. Save your changes to **sessions.pig**.

Step 4: Run the Script

4.1. Let's verify the **Sessionized** function is working by running the script:

```
pig sessions.pig
```

4.2. Verify the tail of the output looks similar to the following:

```
(2013-01-10T07:15:20.520Z,1357802120520,2,51d89b38-b14a-
4158-8703-724525d9f787)
(2013-01-10T07:15:39.797Z,1357802139797,2,51d89b38-b14a-
4158-8703-724525d9f787)
(2013-01-10T07:26:30.602Z,1357802790602,2,711525c4-eff6-
```

```
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:26:53.357Z,1357802813357,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:26:58.800Z,1357802818800,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:27:05.253Z,1357802825253,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:27:57.844Z,1357802877844,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:28:20.610Z,1357802900610,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:29:01.128Z,1357802941128,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:29:02.190Z,1357802942190,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:29:23.190Z,1357802963190,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:30:04.181Z,1357803004181,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
(2013-01-10T07:30:32.455Z,1357803032455,2,711525c4-eff6-
4697-ade7-e2ad5ec555e5)
```

Step 5: Compute the Session Length

5.1 Comment out the **dump** statement:

```
--dump clicks_sessionized
```

5.2 Define a projection named **sessions** that is a projection of only the **time** and **sessionid** fields of the **clicks_sessionized** relation.

5.3 Define a relation named **sessions_group** that is the **sessions** relation grouped by **sessionid**.

5.4 Define a **session_times** relation using the following projection that computes the length of each session:

```
session_times = FOREACH
    sessions_group GENERATE group as
    sessionid,
   (MAX(sessions.time) - MIN(sessions.time)) / 1000.0 /
         60 as session_length;
```

5.5 Dump the **session_times** relation:

```
dump session_times;
```

5.6 Save your changes to sessions.pig and run the script. The output should look like the following:

```
(01e5259c-c5a6-45b0-8d04-1be86182d12e,0.16571666666666665)
(164be386-1df2-40dd-9331-563e1b8a7275,4.030883333333334)
(16ab9225-28d3-45f6-9d07-f065223046bb,38.809916666666666)
(18362695-d032-424a-a983-33ab45638700,0.0) (2699ef77-bd37-
4611-a239-ddbd80066043,10.398116666666665) (3077f9d1-a5d5-
4bf9-8212-87ae848b4ed8,3.44485) (3e732d19-e3ed-4cc4-810f-
f05c8534fb28,1.1402833333333333) (455183ea-c3bb-43fe-9f07-
63e0c0199008,14.648516666666666) (5a65d8dc-1a4e-4355-b86a-
f1efc519b084,63.620149999999995) (5ef45fc4-01df-40d8-805f-
a61c60fc421e,0.03173333333333333) (61e14bcf-1fb4-4f7e-
a3b4-2b67b8840756,1.0819833333333333) (63b53f03-31e9-4a01-
8029-6334020080e4,4.48765) (66f58bc2-7aeb-487d-a28e-
21090578cfe2,22.9298) (812a7fc4-9ea2-4c3b-a3da-
17bbd740a49a,0.006183333333333333) (84f8c113-d3c9-4590-
83a8-5a9edf44c5c5,86.69525) (85cd8b8c-644b-4fb9-a6c6-
3b5082d32f0c,2.509133333333333) (8e4cfed7-8500-47bb-a5e9-
3744de6b1595,0.0) (a35be8db-de7b-4b55-a230-
66389a4e4b5f,0.9713166666666667) (bcfef9fa-fd71-4962-8a0b-
ddcf77ea47a3,0.37246666666666667) (c092d0c4-3c7d-4cfc-
b7f9-078baaa7469f,1.6453333333333333) (d1d1b88e-b827-4005-
b088-233d56c4ea8f,0.6608333333333333) (e0f48349-1d2a-4cd7-
8258-e36b4b6118fc,31.88788333333333) (e1ccdf96-fc37-4b7e-
9a7c-95acb8f52fa7,0.0) (fd92f410-19fc-4927-
917f0f86b5d7edb2,17.197683333333334) (fdfcea38-ddf9-477a-
bb3e-401e8874e0ac,2.2512333333333334) (ff70c6b5-abb2-4606-
b12f-3054501947a4,0.05118333333333334)
```

5.7 How long was the longest session? _____

Step 6: Compute the Average Session Length

6.1. Comment out the **dump** statement:

```
--dump session_times;
```

- Define a relation named **sessiontimes_all** that is a grouping of all **session_times**.

- Define **sessiontimes_avg** using the following nested **FOREACH** statement:

```
sessiontimes_avg = FOREACH sessiontimes_all {
        ordered = ORDER session_times BY session_length;
        GENERATE AVG(ordered.session_length)
            AS avg_session;
}
```

6.4. Dump the **sessiontimes_avg** relation:

```
dump sessiontimes_avg;
```

- Save your changes to **sessions.pig** and run the script.

- Verify the output, which should be a single value representing the average session time:

```
(11.88608076923077)
```

> **NOTE**: This value is hard to find within the all the logging output. You may need to search carefully for the output!

Step 7: Compute the Median Session Length

" Using the **sessiontimes_avg** relation as an example, compute the median session time. You will need to define the **Median** function from the datafu library, which is named **datafu.pig.stats.Median()**.

" Verify you got the following value for the median session length:

```
(1.9482833333333334)
```

RESULT: You have taken clickstream data and sessionized it using Pig to determine statistical information about the sessions, like the length of each session and the average and median lengths of all sessions.

ANSWERS:

2.4: The '8m' stands for 8 minutes, which is the length of the session. You can pick any length of time you want to define your sessions.

5.7: The longest session was 86.69525 minutes.

SOLUTIONS:

Step 5.2:

```
sessions = FOREACH clicks_ sessionized
           GENERATE time, sessionid;
```

Step 5.3:

```
sessions_group = GROUP sessions BY sessionid;
```

Step 6.2:

```
sessiontimes_all = GROUP session_times ALL;
```

Step 7.1: A quick solution for computing the median is to simply add it to the existing nested **FOREACH** statement:

```
sessiontimes_avg = FOREACH sessiontimes_all {
 ordered = ORDER session_times BY session_length;
 GENERATE
    AVG(ordered.session_length) AS avg _session,
    Median(ordered.session_length) AS median_session;
}
```

Lab 7 : Analyzing Stock Market Data using Quantiles

Location of Files	/root/labs/Lab6.5

Step 1: Review the Stock Market Data

From the command prompt, change directories to the **Lab6.5** folder.

1.1 View the contents of the **stocks.csv** file, which contains the historical prices for New York Stock Exchange stocks that begin with the letter "Y":

```
# tail stocks.csv
```

1.2 The first column is always "NYSE". The second column is the stock's symbol. The third column is the date that the prices occurred. The next columns are the open, high, low, close and trading volume.

1.3 Put **stocks.csv** into your **/user/root** folder in HDFS:

```
# hadoop fs -put stocks.csv stocks.csv
```

Step 2: Define the Quantile Function

2.1. Create a new text file in the Lab6.5 folder named quantile.pig.

2.2. On the first line of the file, register the datafu JAR file.

2.3. Define the **datafu.pig.stats.Quantile** function as **Quantile**, and pass in the values for computing the *quartiles* of a set of numbers:

```
define Quantile datafu.pig.stats.Quantile(
                          '0.0','0.25','0.50','0.75','1.0');
```

Step 3: Load the Stocks

3.1. Enter the following **LOAD** command, which loads the first five values of each row:

```
stocks = LOAD 'stocks.csv' USING PigStorage(',') AS
           (nyse:chararray,

            symbol:chararray,
            closingdate:chararray,
            low:double,
            highprice:double);
```

Step 4: Filter Null Values

4.1. The **Quantile** function fails if any of the values passed to it are null. Define a relation named **stocks_filter** that filters the stocks relation where the **highprice** is not null.

Step 5: Group the Values

5.1. We want to compute the quantiles for each individual stock (as opposed to all the stocks prices that start with a "Y"), so define a relation named **stocks_group** that groups the **stock_filter** relation by **symbol**.

Step 6: Compute the Quantiles

6.1. Define the following relation that invokes the Quantile method on the highprice values:

```
quantiles = FOREACH stocks_group {

            sorted = ORDER stocks_filter BY highprice; GENERATE group AS
            symbol,

                  Quantile(sorted.highprice) AS quant;
```

```
}
```

6.2. How many times will the Quantile function be invoked in the nested FOREACH statement above? _____

6.3. Add a DUMP statement that outputs the quantiles relation:

```
DUMP quantiles;
```

Step 7: Run the Script

7.1. Save your changes to **quantile.pig**.

7.2. Run the script:

```
pig quantile.pig
```

7.3. There are only five stocks in the input data, so the output will be the quartiles of the high price of these five stocks:

```
(YGE,(3.22,10.97,14.79,19.6,41.5)) (YPF,(9.0,23.62,31.94,41.47,69.98))
(YSI,(1.56,8.04,16.435000000000002,19.93,23.61))
(YUM,(21.9,32.08,37.85,48.91,73.87))
(YZC,(4.41,14.4,20.795,47.13,116.73))
```

Step 8: Compute the Median

8.1. Now that you have a working Pig script for computing quantiles of the high prices of stocks, see if you can modify the script (you only have to make a few changes) to compute the median value of the high prices.

RESULT: You have used the DataFu library to compute quantiles of a collection of numbers using Pig.

ANSWERS:

6.2: The **FOREACH** statement iterates over the **stocks_group**, which is a grouping by symbol. So the **Quantile** function will be invoked once for each unique stock symbol in the **stocks.csv** file.

SOLUTIONS:

Step 4:

```
stocks_filter = FILTER stocks BY highprice is not null;
```

Step 5:

```
stocks_group = GROUP stocks_filter BY symbol;
```