**Owen Galvin**

**HU Extension Assignment 05 E63 Big Data Analytics**

Issued on: February 27, 2016 Due by 11:30PM EST, March 04, 2016

Please work in Hue’s Hive Editor.

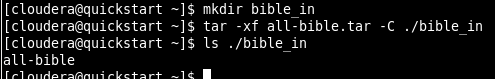
**Problem 1.** Create your own tables KINGJAMES with columns for words and frequencies and insert into the table the result of Hadoop MapReduce GREP program which produce word counts on file all-bible. File is provided with this assignment. Tell us all words in table KINGJAMES which start with letter “w” and are 4 or more characters long and appear more than 250. There are not that many of those words so you can count them by hand. However, you want to be more automated so please change your query so that it gives you the number of such words as its output. When comparing a word with a string your use LIKE operator, like

word like ‘a%’ or word like ‘%th%’

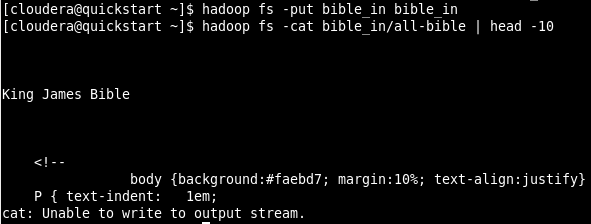
Symbol ‘%’ means any number of characters. You measure the length of a string using function length() and you change the case of a word to all lower characters using function lower().

**NOTE: Command text for all problems located at end of word document. Most text results/output is inline but sample output for Problem 4, re apache log, is uploaded as separate file.**

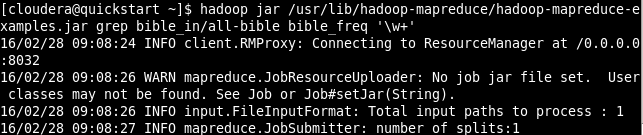
**Solution: With all-bible.tar present in cloudera’s home directory, create new /bible\_in directory and extract contents into it, confirm file exists.**



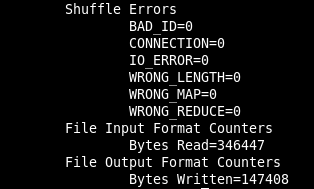
**Copy entire directory into hdfs with the put command, confirm success by doing a quick head on the file.**



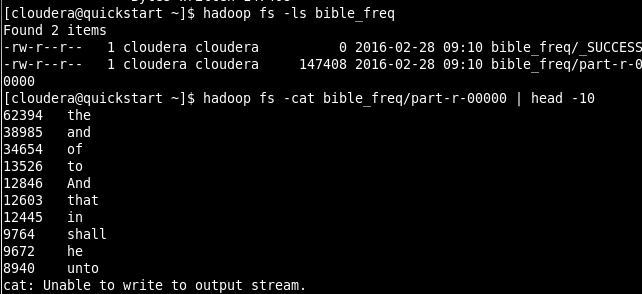
**Begin the grep/word-frequency-count job as described in lecture notes, to count all multi-characters words, telling it to put output in bible\_freq directory.**



**…**



**Check contents out output directory and then open the part-\* file and review first 10 lines using head command, everything looks good, with mostly (case-sensitive) noise words counted as most frequent.**



**TEXT:**

|  |
| --- |
| **62394 the**  **38985 and**  **34654 of**  **13526 to**  **12846 And**  **12603 that**  **12445 in**  **9764 shall**  **9672 he**  **8940 unto** |

**Log in to Hue, All OK.**

|  |
| --- |
|  |

**Go to Query Editors drop-down & select Hive, getting default query.**

|  |
| --- |
|  |

**Create table named KingJames (all upper-case in a table name goes against my SQL instincts), using syntax from lecture pdf. Output message at least indicates no errors.**

|  |
| --- |
|  |

**Run SHOW TABLES to confirm table was created, note that casing didn’t matter at all.**

**(I’ll generally follow the format of the lecture and highlight the phrases that are being executed in any given step, trying to place commands toward bottom of query window so that screenshot can be shorter but still capture output.)**

|  |
| --- |
|  |

**Confirm structure of table**

|  |
| --- |
|  |

**Since my data was put into HDFS, load frequency data from that bible\_freq directory that was created as a result of running the grep job.**

|  |
| --- |
|  |

**Run a select to see the 10 most common words and they match the similar Head results on the original file. Notable that the SORT BY is not strictly necessary in this example, get same results without. Pesumably due the pre-sortedness of the words when they were loaded in?**

|  |
| --- |
|  |

**TEXT:**

|  |
| --- |
| **kingjames.freq kingjames.word**  **0 62394 the**  **1 38985 and**  **2 34654 of**  **3 13526 to**  **4 12846 And**  **5 12603 that**  **6 12445 in**  **7 9764 shall**  **8 9672 he**  **9 8940 unto** |

**Write a query to assignment specs, using LOWER and LENGTH functions as necessary. Truncated output in screenshot, followed by full text results.**

|  |
| --- |
|  |

**TEXT:**

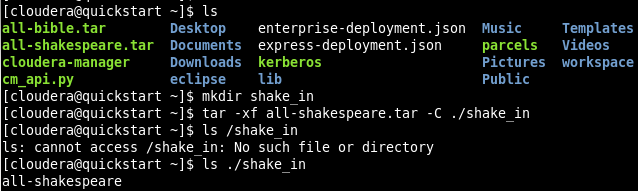
|  |
| --- |
| **kingjames.freq kingjames.word**  **0 6057 with**  **1 4297 which**  **2 3819 will**  **3 2767 were**  **4 2487 when**  **5 1399 went**  **6 732 whom**  **7 694 word**  **8 652 what**  **9 546 words**  **10 512 work**  **11 443 would**  **12 436 without**  **13 407 wife**  **14 396 water**  **15 355 woman**  **16 349 When**  **17 343 wicked**  **18 335 What**  **19 335 where**  **20 304 wilderness**  **21 301 works**  **22 288 world**  **23 286 waters**  **24 284 whose**  **25 283 written**  **26 261 Wherefore**  **27 253 well** |

**Now I’ll update the query to simply output the number of words that match the filter and see that the count matches the number of rows returned by original query.**

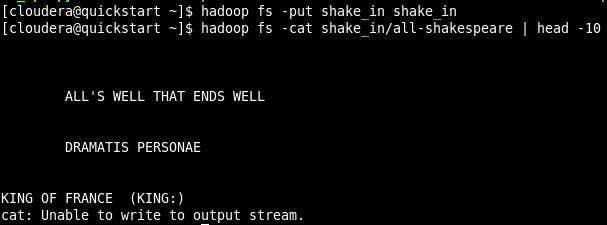
|  |
| --- |
|  |

**Problem 2**. Create your own table SHAKE similar to the one we used in class and populate it with results of MapReduce GREP program applied to the file all-shakespeare which is provided with this assignment. Create your own MERGED table similar to the one we used in class. The table will list all the word and the frequencies with which they appear in either table SHAKE or KINGJAMES. Your table will be “better” than the one we used in class. In class we only inserted into that table words that appear in both texts. Please use **outer joins** to populate the table with words that also appear in one but not the other text. Tell us how many words appear in table SHAKE but not in KINGJAMES and how many appear in KINGJAMES and not in SHAKE. Select 10 words from each group for us. To solve this problem you will have to consult Hive Tutorial at <https://cwiki.apache.org/confluence/display/Hive/Tutorial> or simply Google around the Web.

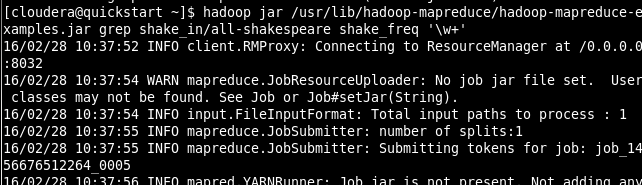
**Solution: With the all-shakespeare.tar file present in cloudera’s home directory, extract into new /shake\_in directory and confirm file exists.**



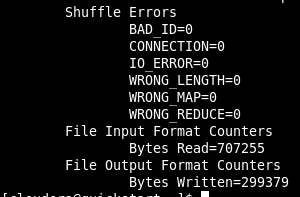
**Copy file into hdfs, into similar /shake\_in directory, confirm success by doing a quick head on the file.**



**Run the grep job again as was done with bible\_in directory in Problem 1.**



**…**



**Create the target Shake table in Hive, using similar structure as for KingJames table.**

|  |
| --- |
|  |

**And confirm it worked by running DESCRIBE command.**

|  |
| --- |
|  |

**Load the Shakespeare grep/frequency data into the new table, pulling from the HDFS directory created by the grep job.**

|  |
| --- |
|  |

**And confirm results by SELECTng the 10 most common words. Again, skipping the SORT BY clause happens returns the same results.**

|  |
| --- |
|  |

**TEXT:**

|  |
| --- |
| **shake.freq shake.word**  **0 25578 the**  **1 23027 I**  **2 19654 and**  **3 17462 to**  **4 16444 of**  **5 13524 a**  **6 12697 you**  **7 11296 my**  **8 10699 in**  **9 8857 is** |

**Create the target merged table that will hold word/frequency data from both the Shake table and KingJames table. In this case we will need separate columns for word data coming from the two different sources vs. the single column that was used in lecture notes where there was only an INNER JOIN responsible for the data collection.**

|  |
| --- |
|  |

**Now to populate the merged table, use a query similar to the one from lecture but updated to do a FULL OUTER JOIN and populated word columns for both data sources. As opposed to the lecture example, here we are inserting all words, not just those with frequency > 1.**

|  |
| --- |
|  |

**A quick snapshot of some results just to confirm that some data made its way into merged table, Assignment queries will confirm data from both tables is in there.**

|  |
| --- |
|  |

**Since the next requirement comes up directly after populating merged, I’ll assume that the query in question should be built using that table as opposed to the separate tables.**

**How many words appear in Shake but not King James = 21,428.**

|  |
| --- |
|  |

**How many words appear in KingJames but not Shake = 6,575.**

|  |
| --- |
|  |

**10 word sample for words that are in Shake but not in KingJames, I figure including the frequency count won’t hurt. The results from a simple SELECT were kind of boring since they were alphabetized and included words that weren’t ‘real’. Full text results follow.**

|  |
| --- |
|  |

**TEXT:**

|  |
| --- |
| **shakeword shakefreq**  **0 baring 1**  **1 Hercules 36**  **2 Lovell 9**  **3 Domitius 2**  **4 Cleitus 2**  **5 prompter 2**  **6 splendor 4**  **7 Prefer 2**  **8 favourer 2**  **9 intrinse 1** |

**And similar 10-sample for words in KingJames but not in Shake.**

|  |
| --- |
|  |

**Text:**

|  |
| --- |
| **bibleword biblefreq**  **0 Beeri 2**  **1 crucify 12**  **2 antichrists 1**  **3 Restore 4**  **4 research 4**  **5 wonderously 1**  **6 backbitings 1**  **7 LICENSE 4**  **8 holden 12**  **9 Ozni 1** |

**Problem 3**. When you have your three queries for counting common words, words that are present in Bible but not in Shakespeare and the words present in Shakespeare but not in Bible refined and working, collect the execution times of those queries. This is not straightforward, since Hive does not give you a simple tool to time your queries. You can look in query logs (a tab next to the Results tab) and sum execution times of map and reduce jobs. That is close enough. Then change your Hue Query Editor and switch to Impala Editor. Run your queries in that editor. This time you have no way of read the time. You just make a subjective estimate. Compare the execution time of queries with Impala and Hive. Impala is usually much faster. One thing to notice here is that you can use Impala on some of Hive tables. Unfortunately not all. Hive is more versatile than Impala.

**Solution: There wasn’t anything in Problem 2 regarding a query for counting common words so I’ll first create one and execute.**

|  |
| --- |
|  |

**And the log output for the in-common query.**

|  |
| --- |
| **job\_1456676512264\_0022**  **INFO : Number of reduce tasks determined at compile time: 1**  **INFO : In order to change the average load for a reducer (in bytes):**  **INFO : set hive.exec.reducers.bytes.per.reducer=<number>**  **INFO : In order to limit the maximum number of reducers:**  **INFO : set hive.exec.reducers.max=<number>**  **INFO : In order to set a constant number of reducers:**  **INFO : set mapreduce.job.reduces=<number>**  **INFO : Starting Job = job\_1456676512264\_0022, Tracking URL = http://quickstart.cloudera:8088/proxy/application\_1456676512264\_0022/**  **INFO : Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job\_1456676512264\_0022**  **INFO : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1**  **INFO : 2016-02-28 14:44:04,347 Stage-1 map = 0%, reduce = 0%**  **INFO : 2016-02-28 14:44:25,398 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.14 sec**  **INFO : 2016-02-28 14:44:45,701 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 7.01 sec**  **INFO : MapReduce Total cumulative CPU time: 7 seconds 10 msec**  **INFO : Ended Job = job\_1456676512264\_0022** |

**My reading of the output is that the value displayed for ‘MapReduce Total cumulative CPU time ‘ is a good enough measure since the values being displayed are in fact cumulative and we aren’t going for exacting measurement. So I’ll go with total execution time of 7 seconds, 10 milliseconds.**

**My query again for number of words in Shakespeare but not in King James Bible**

|  |
| --- |
| ***SELECT COUNT(\*) AS NumWordsInShakeButNotBible***  ***FROM merged***  ***WHERE shakeFreq IS NOT NULL AND bibleFreq IS NULL*** |

**Leads to log output of**

|  |
| --- |
| **job\_1456676512264\_0023**  **INFO : Number of reduce tasks determined at compile time: 1**  **INFO : In order to change the average load for a reducer (in bytes):**  **INFO : set hive.exec.reducers.bytes.per.reducer=<number>**  **INFO : In order to limit the maximum number of reducers:**  **INFO : set hive.exec.reducers.max=<number>**  **INFO : In order to set a constant number of reducers:**  **INFO : set mapreduce.job.reduces=<number>**  **INFO : Starting Job = job\_1456676512264\_0023, Tracking URL = http://quickstart.cloudera:8088/proxy/application\_1456676512264\_0023/**  **INFO : Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job\_1456676512264\_0023**  **INFO : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1**  **INFO : 2016-02-28 15:04:06,452 Stage-1 map = 0%, reduce = 0%**  **INFO : 2016-02-28 15:04:25,899 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.02 sec**  **INFO : 2016-02-28 15:04:45,307 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 6.97 sec**  **INFO : MapReduce Total cumulative CPU time: 6 seconds 970 msec**  **INFO : Ended Job = job\_1456676512264\_0023** |

**Record that one as 6 seconds, 970 msec.**

**The final query, for words in King James Bible but not Shakespeare.**

|  |
| --- |
| ***SELECT COUNT(\*) AS NumWordsInBibleButNotShake***  ***FROM merged***  ***WHERE bibleFreq IS NOT NULL AND shakeFreq IS NULL*** |

**Log output**

|  |
| --- |
| **job\_1456676512264\_0024**  **INFO : Number of reduce tasks determined at compile time: 1**  **INFO : In order to change the average load for a reducer (in bytes):**  **INFO : set hive.exec.reducers.bytes.per.reducer=<number>**  **INFO : In order to limit the maximum number of reducers:**  **INFO : set hive.exec.reducers.max=<number>**  **INFO : In order to set a constant number of reducers:**  **INFO : set mapreduce.job.reduces=<number>**  **INFO : Starting Job = job\_1456676512264\_0024, Tracking URL = http://quickstart.cloudera:8088/proxy/application\_1456676512264\_0024/**  **INFO : Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job\_1456676512264\_0024**  **INFO : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1**  **INFO : 2016-02-28 17:31:54,156 Stage-1 map = 0%, reduce = 0%**  **INFO : 2016-02-28 17:32:14,769 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.24 sec**  **INFO : 2016-02-28 17:32:34,299 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 7.26 sec**  **INFO : MapReduce Total cumulative CPU time: 7 seconds 260 msec**  **INFO : Ended Job = job\_1456676512264\_0024** |

**Total time : 7 seconds 260 msec.**

**Run the same queries in Impala, though at first the tables are not visible. Thanks to a discussion post I run the ‘INVALIDATE METADATA;’ so that Impala knows about the work that has been done in Hive.**

|  |
| --- |
|  |

**Now to run the first query. As noted in the description for this problem, no internal data is provided to indicate processing time. Via stopwatch, time it at about 9 seconds from clicking Execute until the result comes back. That’s higher than the roughly 7 seconds reported in the log for the Hive query but overall it felt quicker than Hive’s did… I usually found myself switching to doing something else while waiting for those queries to finish. Using a stopwatch on that first Hive (words in common) query from this Problem comes out to 66 seconds from initial click until actual rendered result.**

|  |
| --- |
|  |

**Back to Impala and NumWordsInShakeButNotBible query, record stopwatch results below.**

|  |
| --- |
|  |

**And NumWordsInBibleButNotShake, record stopwatch results below.**

|  |
| --- |
|  |

**My general conclusion is that the Cumulative time values displayed in the Hive log results are not necessarily relevant to actual processing time, or at least not to the time it takes for results to appear in the web page for an end user. Maybe Beeline would perform differently.**

**So I went back and ran the Hive queries and used a stopwatch on those. Below are the combined observed click-wait-Results for each type of run. (The 2nd & 3rd Impala runs below were done on a different day vs. the 1st and when I re-ran the 1st it was in the 4 second range, of course maybe some type of caching was going on.)**

|  |  |  |
| --- | --- | --- |
| **COUNT(\*) query** | **Stopwatch time (seconds)** | |
| **Word in both tables** | **9** | **66** |
| **In Shake but not KingJames** | **4** | **65** |
| **In KingJames but not Shake** | **4** | **68** |

**Problem 4.** Please create Hive table APACHELOG for extraction of the content of Apache server logs:

CREATE TABLE apachelog (

host STRING,

identity STRING,

user STRING,

time STRING,

request STRING,

status STRING,

size STRING,

referer STRING,

agent STRING)

ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe' WITH SERDEPROPERTIES ( "input.regex" = "([^ ]\*) ([^ ]\*) ([^ ]\*) (-|\\[[^\\]]\*\\]) ([^ \"]\*|\"[^\"]\*\") (-|[0-9]\*) (-|[0-9]\*)(?: ([^ \"]\*|\"[^\"]\*\") ([^ \"]\*|\"[^\"]\*\"))?", "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s" )

STORED AS TEXTFILE;

Please expand the above regular expression to single line before copying the entire statement to Hue Hive editor.

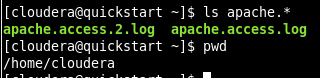
Test success of creation of that table using two single line samples of Apache logs contained in files apache.access.2.log and apache.access.log (note files do not have .txt suffix) contained in the attached file examples\_older.zip. Once you are convinced that you can safely insert those two samples into your table apachelog, insert a bigger log contained in file apache\_log\_1.txt. Tell us how many lines of apache logs you have in table apachelog.

We are also attaching two groups of example data files for Hive: examples\_older.zip and examples.zip. You might find those files useful if you want to keep on learning about the technology. You could get those files by downloading Hive distributions, as described in notes.

**Solution: The first task is to create the table given the provided command/schema. Note that I needed to uncheck the ‘Enable parameterization’ in Setting, apparently cause it thought the $ character pre-pended a parameter named ‘s’.**

|  |
| --- |
|  |

**The single-line apache log files are present in cloudera’s home directory.**



**There really isn’t any reason to copy into HDFS so I’ll load the two files with the LOCAL keyword.**

|  |
| --- |
|  |

**Do a simple SELECT \* on the table and it looks like both made it in there.**

|  |
| --- |
|  |

**FULL RESULTS (columns broken into two sections for better readability):**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | apachelog.host | apachelog.identity | apachelog.user | apachelog.time | apachelog.request | | 0 | 127.0.0.1 | - | - | [26/May/2009:00:00:00 +0000] | GET /someurl/?track=Blabla(Main) HTTP/1.1 | | 1 | 127.0.0.1 | - | frank | [10/Oct/2000:13:55:36 -0700] | GET /apache\_pb.gif HTTP/1.0 | |
| |  |  |  |  | | --- | --- | --- | --- | | apachelog.status | apachelog.size | apachelog.referer | apachelog.agent | | 200 | 5864 | - | Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/525.19 (KHTML, like Gecko) Chrome/1.0.154.65 Safari/525.19 | | 200 | 2326 | NULL | NULL | |

**Instructions say to load in apache\_log\_1.txt, I’m going to assume this refers to access\_log\_1.txt since the lecture pdf seems to show the former file being renamed as the latter. I copy that over via my VM’s shared folder and confirm file is in Cloudera’s home directory.**



**Load the access\_log\_1.txt file, again from its local file path.**

|  |
| --- |
|  |

**Run a COUNT(\*) to get the full count of apache log lines in the ApacheLog table, which is 39,346 and include the contents of the two single-line apache.access files that were initially loaded.**

|  |
| --- |
|  |

**(I’ve also uploaded a 10 row sample from the ApacheLog table, as file named ApacheLog\_query\_result.csv.txt, in csv format.)**

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of your command. We cannot retype text that is in JPG images. Please, always submit a copy of original, working scripts and class files you used as separate files. Sometimes we need to run your code and retyping is too costly. Please, submit to the class drop box. For issues and comments visit the class Discussion Board .

**Problem 1 cmd/HUE text**

**[cloudera@quickstart ~]$ mkdir bible\_in**

**[cloudera@quickstart ~]$ tar -xf all-bible.tar -C ./bible\_in**

**[cloudera@quickstart ~]$ ls ./bible\_in**

**[cloudera@quickstart ~]$ hadoop fs -put bible\_in bible\_in**

**[cloudera@quickstart ~]$ hadoop fs -ls**

**[cloudera@quickstart ~]$ hadoop fs -put bible\_in bible\_in**

**[cloudera@quickstart ~]$ hadoop fs -cat bible\_in/all-bible | head -10**

**[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep bible\_in/all-bible bible\_freq '\w+'**

**[cloudera@quickstart ~]$ hadoop fs -ls bible\_freq**

**[cloudera@quickstart ~]$ hadoop fs -cat bible\_freq/part-r-00000 | head -10**

**CREATE TABLE KingJames (freq INT, word STRING)**

**ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'**

**STORED AS TEXTFILE;**

**SHOW TABLES**

**DESCRIBE KingJames**

**LOAD DATA INPATH '/user/cloudera/bible\_freq' INTO TABLE KingJames;**

**SELECT \* FROM KingJames SORT BY freq LIMIT 10;**

-- 2) Tell us all words in table KINGJAMES which start with letter “w” and are 4 or more characters long and appear more than 250

**SELECT \* FROM KingJames**

**WHERE LOWER(word) LIKE 'w%'**

**AND LENGTH(word) >= 4**

**AND freq > 250**

**Problem 2 cmd/HUE text**

**[cloudera@quickstart ~]$ rmdir shake**

**[cloudera@quickstart ~]$ ls**

**[cloudera@quickstart ~]$ mkdir shake\_in**

**[cloudera@quickstart ~]$ tar -xf all-shakespeare.tar -C ./shake\_in**

**[cloudera@quickstart ~]$ ls ./shake\_in**

**[cloudera@quickstart ~]$ hadoop fs -put shake\_in shake\_in**

**[cloudera@quickstart ~]$ hadoop fs -cat shake\_in/all-shakespeare | head -10**

**[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep shake\_in/all-shakespeare shake\_freq '\w+'**

**[cloudera@quickstart ~]$ hadoop fs -ls shake\_freq**

**CREATE TABLE Shake (freq INT, word STRING)**

**ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'**

**STORED AS TEXTFILE;**

**DESCRIBE Shake**

**LOAD DATA INPATH '/user/cloudera/shake\_freq' INTO TABLE Shake;**

**SELECT \* FROM Shake SORT BY freq LIMIT 10;**

**CREATE TABLE merged (shakeWord STRING, shakeFreq INT, bibleWord STRING, bibleFreq INT);**

**INSERT OVERWRITE TABLE merged**

**SELECT s.word, s.freq, k.word, k.freq**

**FROM Shake s**

**FULL OUTER JOIN KingJames k ON (k.word = s.word)**

-- 3) Tell us how many words appear in table SHAKE but not in KINGJAMES and how many appear in KINGJAMES and not in SHAKE.

-- words in Shake but not KingJames

**SELECT COUNT(\*) AS NumWordsInShakeButNotBible**

**FROM merged**

**WHERE shakeFreq IS NOT NULL AND bibleFreq IS NULL**

-- words in KingJames but not Shake

**SELECT COUNT(\*) AS NumWordsInBibleButNotShake**

**FROM merged**

**WHERE bibleFreq IS NOT NULL AND shakeFreq IS NULL**

-- 4) Select 10 words from each group for us.

**SELECT shakeWord, shakeFreq**

**FROM merged**

**WHERE shakeFreq IS NOT NULL AND bibleFreq IS NULL**

**ORDER BY RAND()**

**LIMIT 10;**

**SELECT bibleWord, bibleFreq**

**FROM merged**

**WHERE bibleFreq IS NOT NULL AND shakeFreq IS NULL**

**ORDER BY RAND()**

**LIMIT 10;**

**Problem 3 HUE text**

--Hive

**SELECT COUNT(\*) AS NumWordsInBibleAndShake**

**FROM merged**

**WHERE bibleFreq IS NOT NULL AND shakeFreq IS NOT NULL**

**SELECT COUNT(\*) AS NumWordsInShakeButNotBible**

**FROM merged**

**WHERE shakeFreq IS NOT NULL AND bibleFreq IS NULL**

**SELECT COUNT(\*) AS NumWordsInBibleButNotShake**

**FROM merged**

**WHERE bibleFreq IS NOT NULL AND shakeFreq IS NULL**

-- Impala

**INVALIDATE METADATA;**

**SHOW TABLES;**

**SELECT COUNT(\*) AS NumWordsInBibleAndShake**

**FROM merged**

**WHERE bibleFreq IS NOT NULL AND shakeFreq IS NOT NULL**

**SELECT COUNT(\*) AS NumWordsInShakeButNotBible**

**FROM merged**

**WHERE shakeFreq IS NOT NULL AND bibleFreq IS NULL**

**SELECT COUNT(\*) AS NumWordsInBibleButNotShake**

**FROM merged**

**WHERE bibleFreq IS NOT NULL AND shakeFreq IS NULL**

**Problem 4 cmd/HUE text**

**TERMINAL:**

**[cloudera@quickstart ~]$ ls apache.\***

**[cloudera@quickstart ~]$ pwd**

**HUE:**

**CREATE TABLE apachelog (**

**host STRING,**

**identity STRING,**

**user STRING,**

**time STRING,**

**request STRING,**

**status STRING,**

**size STRING,**

**referer STRING,**

**agent STRING)**

**ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'**

**WITH SERDEPROPERTIES ( 'input.regex' = '([^ ]\*) ([^ ]\*) ([^ ]\*) (-|\\[[^\\]]\*\\]) ([^ \"]\*|\"[^\"]\*\") (-|[0-9]\*) (-|[0-9]\*)(?: ([^ \"]\*|\"[^\"]\*\") ([^ \"]\*|\"[^\"]\*\"))?', 'output.format.string' = '%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s' )**

**STORED AS TEXTFILE;**

**LOAD DATA LOCAL INPATH '/home/cloudera/apache.access.log' INTO TABLE ApacheLog;**

**LOAD DATA LOCAL INPATH '/home/cloudera/apache.access.2.log' INTO TABLE ApacheLog;**

**SELECT \* FROM ApacheLog;**

**LOAD DATA LOCAL INPATH '/home/cloudera/access\_log\_1.txt' INTO TABLE ApacheLog;**

**SELECT COUNT(\*) FROM ApacheLog;**

**SELECT \* FROM ApacheLog LIMIT 10;**