Lab 05
HDFS, Pyspark, & Apache Hive

Blagoje Z. Djordjević

csci e63 Big Data Analytics

HDFS: Copying a file to new HDFS directory

- Adding files to HDFS.
- We can fetch the .txt version of James Joyce's Ulysses by issuing the following command on the command prompt:

```
$ wget http://www.gutenberg.org/files/4300/4300-0.txt
```

- We can place file `4300-0.txt` into `ulysses` directory of user `cloudera`

```
$ hadoop fs -mkdir ulysses
$ hadoop fs -put 4300-0.txt ulysses
$ hadoop fs -ls ulysses
Found 1 items
-rw-r--r-- 1 cloudera cloudera 1580890 2017-09-27 19:54 ulysses/4300-0.txt
```

- The number 1 in the above listing tells us how many times a particular file is replicated. Since we have a single machine, 1 is appropriate.
- The replication factor is 3 by default, but could be set to any number.

Fetching and examining Files from HDFS

- The Hadoop command `get` does the exact reverse of `put`. It copies files from HDFS to the local file system.

```
$ hadoop fs -put 4300-0.txt .
```

- To retrieve file `4300-0.txt` from HDFS and copy it into the current local working directory, we run the command

```
$ hadoop fs -get 4300-0.txt .
```

- Generally we would like to examine the data. For small files, Hadoop `cat` command is convenient.

```
$ hadoop fs -cat 4300-0.txt
```

- We can use any Hadoop file command with Unix pipes to forward its output for further processing by other Unix commands. For example, if the file is huge (as Hadoop files typically are) and you're interested in a quick check of its content, you can pipe the output of Hadoop's `cat` into a Unix `head`.

```
$ hadoop fs -cat 4300-0.txt  | head
```

- Hadoop natively supports `tail` command for looking at the last kilobyte of a file.

```
$ hadoop fs -tail 4300-0.txt
```

Pyspark: Setup your Environment

- For pyspark we need to make sure our environment is properly setup

```
[cloudera]$ echo $JAVA_HOME
/usr/java/jdk1.7.0_67-cloudera
[cloudera@quickstart ~]$ ls $JAVA_HOME
bin  lib  src.zip LICENSE  db  man include jre  release . . .
$ echo $SPARK_HOME
$ SPARK_HOME is not define. Change user to root:
[cloudera]$ su -
Password: cloudera
[root]#  cd /
[root@quickstart /]# find . -name start-master.sh -print
./usr/lib/spark/sbin/start-master.sh
```

- This means that `SPARK_HOME` is `/usr/lib/spark`
- As user `cloudera` change `.bash_profile` file, add:

```
[cloudera]$ vi .bash_profile
JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
export JAVA_HOME
SPARK_HOME=/usr/lib/spark
export SPARK_HOME
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin:$SPARK_HOME/bin
[cloudera]$ source .bash_profile
```

Modify `log4j.properties`, `start-master.sh`

- If we do not know where `log4j.properties` file reside, we can find it:

```
[root@quickstart /]# find . -name log4j.properties.template -print
./etc/spark/conf.dist/log4j.properties.template
[cloudera]$ cd /etc/spark/conf.dist
$ sudo cp log4j.properties.template log4j.properties
```

- Replace `INFO, WARN` with `ERROR`.

```
$ vi log4j.properties
  :g/WARN/s//ERROR/g
```

- Start spark master

```
$ cd /usr/lib/spark/sbin
$ sudo ./start-master.sh
[cloudera]$ cd ~
$ pyspark
```

Spark can read HDFS

- On Cloudera VM we have Spark 1.6. This means that we have to rely on old fashioned SparkContext object called sc.

```
$ pyspark
17/09/27 20:10:06 WARN util.Utils: Set SPARK_LOCAL_IP if you need
to bind to another address
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as
sqlContext.
>>>
```

Spark could read from HDFS

- If we type:

```
>>> ul = sc.textFile("4300-0.txt")
>>> ul.count()
```

- We will initially be presented with errors of the form:

```
org.apache.hadoop.mapred.InvalidInputException: Input path does
not exist: hdfs://quickstart.cloudera:8020/user/cloudera/4300-
0.txt
```

- InSpark 1.6, the default file location is in the HDFS home directory of the current user, e.g. `/user/cloudera`.
- We could can file `4300-0.txt` to `/user/cloudera` by typing:

```
[cloudera@quickstart ~]$ hadoop fs -put 4300-0.txt
[cloudera@quickstart ~]$ hadoop fs -ls
Found 5 items
-rw-r--r--   1 cloudera cloudera    1580890 2017-09-28 15:09 4300-0.txt
drwxr-xr-x   - cloudera cloudera          0 2017-09-27 20:18 input
drwxr-xr-x   - cloudera cloudera          0 2017-09-27 20:21 output
```

- If we now run it:

```
>>> ul2 = sc.textFile("4300-0.txt")
>>> ul2.count()
32710
```

Spark found the file in HDFS, read it and counted properly

Spark could read from the Local File

- To be more specific, we can write:

```
>>>  ul = sc.textFile("file:///home/cloudera/4300-0.txt")
>>>  ul.count()
32170
```

- This time Spark reads the file from the local file system.
- Let us create a more complex RDD

```
from operator import add
>>> counts = ul.flatMap(lambda x:x.split(" ")).map(lambda x:
(x,1)).reduceByKey(add)
>>> counts.take(5)
[(u'', 10549), (u'fawn', 3), (u'noctambules', 1), (u'considered?', 1),
(u'clotted', 4)]
```

- Perhaps we would like to sort word counts in descending order. One way to do it:

```
>>> exchanged = counts.map(lambda x: (x[1],x[0]))
>>> exchanged.take(5)
[(10549, u''), (3, u'fawn'), (1, u'noctambules'), (1, u'considered?'), (4,
u'clotted’)]
```

- Numbers are now keys so we could sort by keys `(RDD.sortByKey())`

```
sorted = exchanged.sortByKey(False)   # False (a Boolean) refers to ascending argument
```

Saving Spark Objects to File System

- Perhaps we want to save this new RDD, `sorted`, to the file system.

```
>>> sorted.take(5)
[(13609, u'the'), (10549, u''), (8134, u'of'), (6551, u'and'), (5841, u'a')]
>>> sorted.saveAsTextFile("file:///home/cloudera/sorted")
```

- If we go to the directory `/home/cloudera` and do

```
$ ls -la sorted
drwxrwxr-x   2 cloudera cloudera    4096 Sep 28 18:54 sorted
```

- We see that `sorted` is not a file but rather a directory.

```
$ ls -la sorted.txt/
drwxrwxr-x  2 cloudera cloudera   4096 Sep 28 18:58 .
drwxrwxr-x 32 cloudera cloudera   4096 Sep 28 18:58 ..
-rw-r--r--  1 cloudera cloudera 276753 Sep 28 18:52 part-00000
-rw-r--r--  1 cloudera cloudera   2172 Sep 28 18:52 .part-00000.crc
-rw-r--r--  1 cloudera cloudera 548833 Sep 28 18:52 part-00001
-rw-r--r--  1 cloudera cloudera   4296 Sep 28 18:52 .part-00001.crc
-rw-r--r--  1 cloudera cloudera      0 Sep 28 18:52 _SUCCESS
-rw-r--r--  1 cloudera cloudera      8 Sep 28 18:52 ._SUCCESS.crc
```

- You can examine the content of files `part-00000` and `part-00001` and you will see sorted words and corresponding counts.

Saving Spark Objects to HDFS

- We want to save this new RDD, `sorted`, to the HDFS system. Let us try:

```
>>> sorted.saveAsTextFile("hdfs:///user/cloudera/sorted")
```

- If we go to the directory `/home/cloudera` and do

```
[cloudera@quickstart ~]$ hadoop fs -ls sorted
Found 3 items
-rw-r--r--   1 cloudera cloudera      0 2017-09-28 19:05 sorted/_SUCCESS
-rw-r--r--   1 cloudera cloudera  276753 2017-09-28 19:05 sorted/part-00000
-rw-r--r--   1 cloudera cloudera  548833 2017-09-28 19:05 sorted/part-00001
```

- We can fetch part of the `part-000x` file and examine it with vi or cat.

```
[cloudera@quickstart ~]$ hadoop fs -cat sorted/part-00000 | head
(13609, u'the')
(10549, u'')
(8134, u'of')
(6551, u'and')
(5841, u'a')
(4788, u'to')
(4619, u'in')
(3034, u'his')
(2712, u'he')
(2430, u'I')
```

Hive: Create Table to accept `grep` Data

- HiveServer2, (or Beeline), is another approach to analysis data sets.
- Connect to beeline:

```
beeline> !connect jdbc:hive2://127.0.0.1:10000/default
hive cloudera org.apache.hive.jdbc.HiveDriver
```

- In preparation for import of Shakespeare frequency data on hive prompt we create a table `shakespeare`.
- Note, whenever you enter `hive` shell, type the following:

```
beeline> add jar /usr/lib/hive/lib/hive-contrib.jar;
```

- That file contains various tools Hive editor needs, Hue might not.
- Then, let us create the table

```
beeline> create table shakespeare (freq INT, word STRING) ROW
   FORMAT DELIMITED FIELDS TERMINATED BY '\t' stored as textfile;
beeline> show tables;
```

This created table `shakespeare` with out any data

- If we already have a table we wish to remove, use the following:

```
beeline> drop table shakespeare;
```

Running a MapReduce `grep` Job

- The files for all-bible and all-shakespeare are provided and can be extract from the CLI.
  `$ tar xf all-shakespeare.tar`
- We shall run Hadoop grep to send the data sets to HDFS:
  ```
  $ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep input/all-shakespeare shake_freq '\w+'
  ```
- Job takes a few minutes. You could monitor progress of all map jobs and
- reduce jobs. The output is placed in HDFS directory `shake_freq`
- We repeat the same process for all-bible

Load grep Data into `shakespeare` Table

- To load data we go back to the `Hue editor` and type:

```
beeline> LOAD DATA INPATH "/user/cloudera/shake_freq/part-r-00000"
   INTO TABLE shakespeare;  # From HDFS file system  or
beeline> LOAD DATA LOCAL INPATH '/home/cloudera/part-r-00000' INTO
   TABLE shakespeare2;     # From the local file system
    Loading data to table shakespeare
    OK
    Time taken: 0.213 seconds
```

- On the load command, Hive moved HDFS directory `shake_freq` into its own HDFS directory. That directory is specified in `hive-site.xml` file

```
cloudera@quickstart:~/etc/hive/conf$ vi hive-site.xml
. . . .
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/user/hive/warehouse</value>
  <description>location of default database for the warehouse</description>
</property>
```

- Note again, the directory `/user/hive/warehouse` is in HDFS, not on Linux OS.

```
cloudera@quickstart$ hadoop fs -ls /user/hive/warehouse/shakespeare2
```

Verify that `shakespeare` has `grep` Data

```
beeline> select * from shakespeare limit 10;
OK
25848   the
23031   I
19671   and
18038   to
16700   of
14170   a
12702   you
11297   my
10797   in
8882    is
Time taken: 0.095 seconds
beeline>
```

- This statement is read from the table (actually as part of optimization, it read directly from the HDFS file) and presents us with the first 10 lines.

More Advanced Query

- Slightly more advanced query:

```
beeline> SELECT * FROM shakespeare WHERE freq > 100 SORT BY
   freq ASC LIMIT 10;
```

- Notice that for a large data set this is not an entirely trivial job.
- Data has to be sorted before we could see 10 rows of words that have frequency just above 100.
- Notice how hive reports on map-reduce job it is starting.
- If the job takes too long you are given the job id and the command that you could execute to tell Hadoop to kill the job:

```
Starting Job = job_201404021324_0005, Tracking URL =
   http://quickstart:50030/jobdetails.jsp?jobid=job_201404021324_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job  -
   Dmapred.job.tracker=quickstart:8021 -kill job_201404021324_0005
```

Even More Complex Query

- The “users”, linguists perhaps, would like to know the number of words which appear with the most common frequencies.

```
beeline> SELECT freq, COUNT(1) AS f2
  FROM shakespeare GROUP BY freq SORT BY f2 DESC LIMIT 10;
```

- OK
- 1 13426
- 2 4274
- 3 2342
- 4 1502
- 5 1111
- 6 873
- 7 656
- 8 598
- 9 474
- 10 381

- This tells us that there are 13426 words that appears only once.
- 4274 words appear twice. 2342 words appear three times, etc.
- SQL command with minor deviation: ORDER BY is replaced by SORT BY.

Zipf's Law

- Rank (r): The numerical position of a word in a list sorted by decreasing frequency (f).
- Zipf (1949) "discovered" that:

- If probability of word of rank r is p_r and N is the total number of word occurrences:

$$f \cdot r = k \quad (\text{for constant } k)$$

$$p_r = \frac{f}{N} = \frac{A}{r} \quad \text{for corpus indp. const. } A \approx 0.1$$

Stop Word or the most Frequent Words

- Most frequent words are simple to find:

```
beeline> select freq, word from shakespeare sort by freq  desc
   limit 20;
OK
25848   the
23031   I
19671   and
18038   to
16700   of
14170   a
12702   you
11297   my
10797  in
. . . .
```

- We call these stop words. In Google-like analysis of relevance for text finding, we simply ignore stop words. When we create Tf-Idf weighted vectors we by rule do not include “stop words”.

Zipf and Term Weighting

- Luhn (1958) suggested that both extremely common and extremely uncommon words were not very useful for indexing.

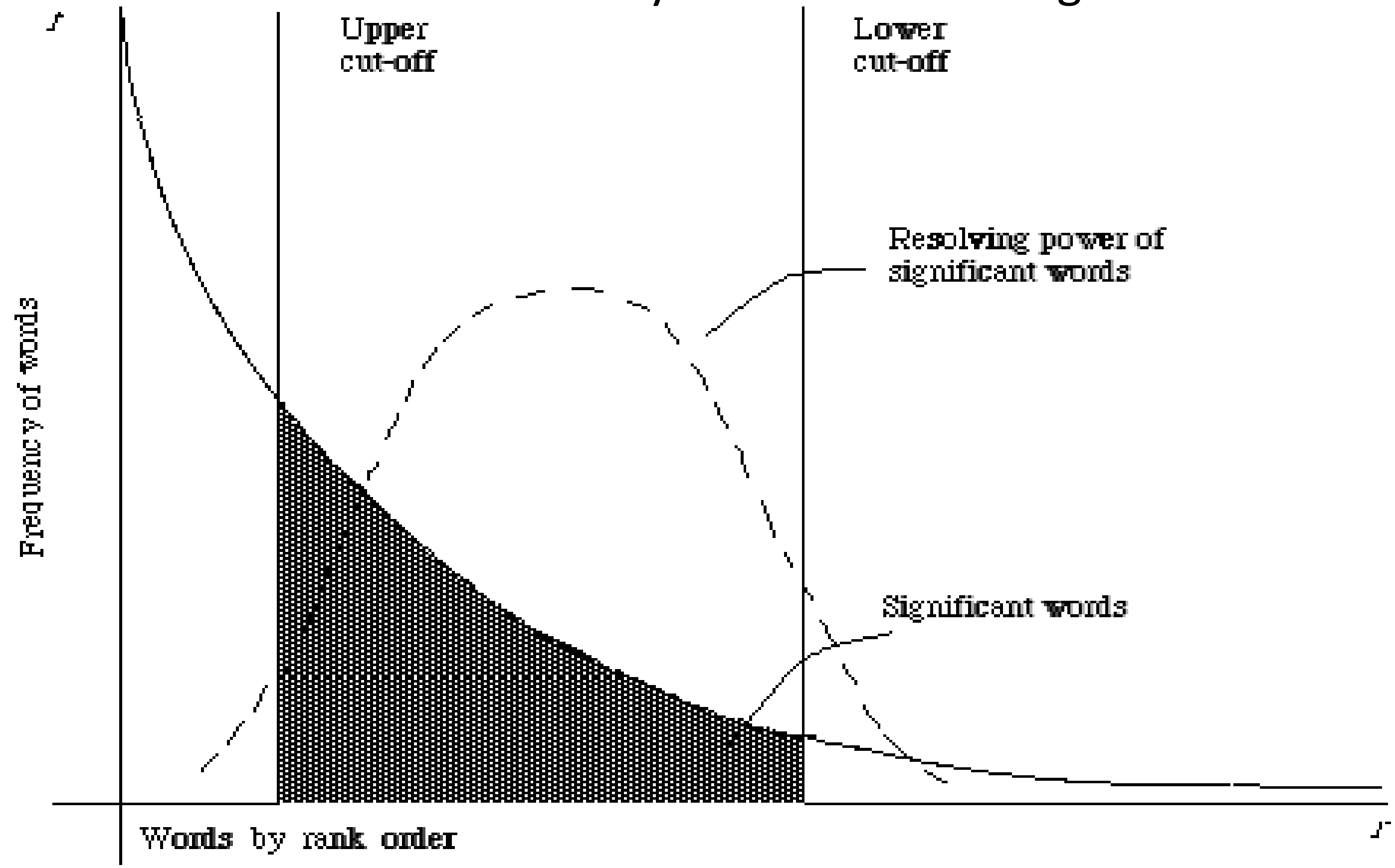


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

How is Query Executed, Explain - Output

- If we are curious how a query is executed we could use Explain command:

```
beeline> EXPLAIN SELECT freq, COUNT(1) AS f2 FROM shakespeare
   GROUP BY freq SORT BY f2 DESC LIMIT 10;

ABSTRACT SYNTAX TREE:
  (TOK_QUERY (TOK_FROM (TOK_TABREF shakespeare)) (TOK_INSERT
   (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
   (TOK_SELEXPR (TOK_TABLE_OR_COL freq)) (TOK_SELEXPR
   (TOK_FUNCTION COUNT 1) f2)) (TOK_GROUPBY (TOK_TABLE_OR_COL
   freq)) (TOK_SORTBY (TOK_TABSORTCOLNAMEDESC (TOK_TABLE_OR_COL
   f2))) (TOK_LIMIT 10)))

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-3 depends on stages: Stage-2
  Stage-0 is a root stage
```

How is Query Executed, Explain - Output

```
STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        shakespeare
          TableScan
            alias: shakespeare
            Select Operator
              expressions:
                    expr: freq
                    type: int
              outputColumnNames: freq
              Reduce Output Operator
                key expressions:
                      expr: freq
                      type: int
                sort order: +
```

How is Query Executed, Explain - Output

```
        Map-reduce partition columns:
                expr: freq
                type: int
          tag: -1
          value expressions:
                expr: 1
                type: int
Reduce Operator Tree:
  Group By Operator
    aggregations:
          expr: count(VALUE._col0)
    keys:
          expr: KEY._col0
          type: int
    mode: complete
```

…… .

Joining Tables

- One of the more powerful features of Hive is the ability to create queries that join tables together using regular SQL syntax.
- We have the (freq, word) data for Shakespeare
- We can generate similar data for King James Bible and then examine which words show up in both volumes of text.
- To generate `grep` data for King James Bible we run Hadoop grep command:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar
   grep all-bible bible_freq '\w+'
```

- This will generate HDFS directory `bible_freq`
- `$ hadoop fs -ls`
- `Found 4 items`
- `drwxr-xr-x   - cloudera   /user/cloudera/bible`
- `drwxr-xr-x   - cloudera   /user/cloudera/bible_freq`
- If present, remove `_logs` directories.
- `$ hadoop fs -rm -r bible_freq/_logs`

Create `bible` Table

```
beeline> CREATE TABLE bible (freq INT, word STRING)
      ROW FORMAT DELIMITED
       FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;
beeline> show tables;
OK
bible
shakespeare
Time taken: 0.165 seconds
beeline> desc bible;
OK
freq   int
word   string
Time taken: 0.228 seconds
```

Import data into `bible`

```
beeline> LOAD DATA INPATH "/user/cloudera/bible_freq/part-r-
   00000" INTO TABLE bible;
OK
Time taken: 0.781 seconds
beeline> select * from bible limit 20;
OK
62394   the
38985   and
34654   of
13526   to
12846   And
12603   that
12445   in
6913    be
6884    is
6649    him
6647    LORD
. . .
Time taken: 0.111 seconds
beeline>
```

Examine `bible_freq` directory in HDFS

- Once you imported data into `bible` table examine `bible_freq` directory in HDFS.

```
$ hadoop fs -ls bible_freq
```

- **There is nothing there???**
- Hive took `part-r-00000` out and moved it somewhere else. Where?
- For every table you create, Hive creates a directory in HDFS
- If your table is partitioned, there will be as many directories as partitions
- Those directories live (usually) in HDFS directory `/user/hive/warehouse` we spoke about already
- On some VMs you have to create that directory as user HDFS.
- Recall the commands:

```
$ sudo –u hdfs hadoop fs –mkdir –p /user/hive/warehouse
$ sudo –u hdfs hadoop fs –chown hive /user/hive
$ sudo –u hdfs hadoop fs –chmod 1777 /user/hive
```

Create an Intermediate Table

- We need a table that will list the most common words in both volumes with the corresponding frequencies

```
beeline> CREATE TABLE merged(word STRING, shake_f INT,
   kjb_f INT);
```

- For this table we do not need to specify how will data be stored.
- Hive will determine that by itself.
- Next, we will run a query that will select data from tables: `shakespeare` and `bible`, create, join and insert, i.e. overwrite the content of new table.
- In our case the table happens to be empty. If it were not empty and we insist on overwriting, table data would be lost. If we only perform an insert, new data would be appended to the old.

Populate `merged` table

```
beeline> INSERT OVERWRITE TABLE merged SELECT s.word, s.freq,
   k.freq FROM shakespeare s JOIN bible k ON (s.word = k.word);

(WHERE s.freq >= 1 AND k.freq >= 1; unnecessary)
....
Ended Job = job_201404021324_0013
Loading data to table merged
7826 Rows loaded to merged
beeline> . . . .
A       2027    236
AND      102    5
AS        25    2
Aaron    26     350
Abel     2      16
Abhor    2      1
Abide    1      5
About    41     6
Above    25     3
Abraham 4       250
Time taken: 0.107 seconds
```

- The above procedure creates a merged table via an **inner join** (only selects shared words). One can also perform an **outer join**, which will give us words only from one table**.**

Populate `mergedfull` table

```
beeline> CREATE TABLE mergedfull (word STRING, shake_f INT, kjb_f
   INT);
beeline> INSERT OVERWRITE TABLE mergedfull SELECT s.word, s.freq,
   k.freq FROM shakespeare s FULL OUTER JOIN bible k ON (s.word =
   k.word);
```

- We can compare the number of lines of `merged` vs `mergedfull`

```
beeline> select count(*) from merged;
+-------+--+
| 7755  |
+-------+--+
beeline> select count(*) from mergedfull;
+--------+--+
| 35758  |
+--------+--+
beeline> select count(*) from mergedfull where shake_f is NULL;
+-------+--+
| 6575  |
+-------+--+
beeline> select count(*) from mergedfull where kjb_f is NULL;
+--------+--+
| 21428  |
+--------+--+
```

Most common common words

- What words appeared most frequently in both corpuses?

```
beeline> SELECT word, shake_f, kjb_f,(shake_f + kjb_f) AS ss FROM merged
SORT BY ss DESC LIMIT 20;
the      25848    62394    88242
and      19671    38985    58656
of       16700    34654    51354
I        23031    8854     31885
to       18038    13526    31564
in       10797    12445    23242
a        14170    8057     22227
that     8869     12603    21472
And      7800     12846    20646
is       8882     6884     15766
my       11297    4135     15432
you      12702    2720     15422
he       5720     9672     15392
his      6817     8385     15202
not      8409     6591     15000
be       6773     6913     13686
for      6309     7270     13579
with     7284     6057     13341
it       7178     5917     13095
shall    3293     9764     13057
```

To examine common non-Stop Word, go deeper

```
SELECT word, shake_f, kjb_f, (shake_f + kjb_f) AS ss FROM merged
   SORT BY ss DESC LIMIT 200;
. . . . .
heaven  626     578     1204
When    847     349     1196
Of 1006 63      1191
most    1017    135     1152
where   813     335     1148
tell    960     188     1148
blood   699     447     1146
doth    961     63      1146
set     451     694     1145
It 890  241     1131
ever    634     475     1109
Which   977     130     1107
whom    375     732     1107
Time taken: 46.988 seconds
```

`Show Tables` Command

- `beeline> SHOW TABLES '.*t';`
- `OK`
- `test`

- `beeline> SHOW TABLES '.*e';`
- `OK`
- `bible`
- `shakespeare`

- `Show tables;` command lists all the table that end with an `'s'.`
- The pattern matching follows Java regular expressions.

Alter Table Command

- As for altering tables, table names can be changed and additional columns can be dropped:

```
beeline> ALTER TABLE test ADD COLUMNS (new_col INT);
beeline> ALTER TABLE test ADD COLUMNS (new_col2 INT COMMENT
 'this is a comment');
beeline> ALTER TABLE test RENAME TO test2;
OK
Time taken: 0.17 seconds
beeline> ALTER TABLE test2 RENAME TO test;
```

Exporting Data from a Table into an HDFS directory

- The following command will move data in table `shakespeare` to HDFS directory `hdfs_out`

```
INSERT OVERWRITE DIRECTORY '/user/cloudera/hdfs_out' SELECT a.*
   FROM shakespeare a WHERE a.freq='101';
Total MapReduce jobs = 2
Number of reduce tasks is set to 0 since there's no reduce operator
....
Ended Job = job_201602251324_0025
Moving data to: hdfs_out
25 Rows loaded to hdfs_out
OK
Time taken: 39.014 seconds
beeline>
```

- Verify presence of the directory

```
$ hadoop fs -ls
Found 5 items . . . .
0 2016-02-25 18:32 /user/cloudera/hdfs_out
```

GROUP BY Statements

```
beeline> create table counts (word string, freq int);
```

- Note that the following statements are equivalent.

```
beeline> FROM shakespeare a INSERT OVERWRITE TABLE counts SELECT
   a.word, count(*) WHERE a.freq > 0 GROUP BY a.word;
beeline> INSERT OVERWRITE TABLE counts SELECT a.word, count(1)
   FROM shakespeare a WHERE a.freq > 0 GROUP BY a.word;
```

- Note that COUNT(*) does not work on older hive installations. You have to use COUNT(1) instead.
- You can use SUM, AVG, MIN, MAX operators on any column as well

```
INSERT OVERWRITE TABLE counts SELECT a.word, sum(a.freq) FROM
   shakespeare a WHERE a.freq > 0 GROUP BY a.word;
```

- The following syntax works:

```
beeline> FROM bible t1 JOIN shakespeare t2 ON (t1.word = t2.word)
   INSERT OVERWRITE TABLE counts SELECT t1.word, t2.freq;
```

Multi-Table Insert

- Modified syntax, where query starts with a FROM clause has its benefits.
- Can you do this in your favorite RDBMS?

```
FROM src
INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100
INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE
   src.key >= 100 and src.key < 200
INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12')
   SELECT src.key WHERE src.key >= 200 and src.key < 300
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT
   src.value WHERE src.key >= 300;
```

- Apparently, this syntax allows you to perform inserts into several tables while visiting the original table only once. Since your table contains “big data”, Hive’s SQL engine has achieved a significant optimization.

Apache Weblog Analysis

- Regular expression serializer, deserializer `RegexSerDe` needs to be loaded into Hive from `hive-contrib.jar`. The file is introduced into Hive by copying it to HDFS and then adding it to hive:

```
beeline> add jar /usr/lib/hive/lib/hive-contrib.jar;
```

- For default Apache weblog, you can create a table with the following command

```
beeline> 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;
```

Insert data into `apachelog` Table

- Hive examples directory contains to single line samples of apache log files: `access_log_1.txt`. We will use it to test the `CREATE TABLE` command with the regular expression.

```
beeline> load data local inpath
'/home/cloudera/access_log_1.txt' into table apachelog;
beeline> select * from apachelog;

127.0.0.1       -       -       [26/May/2009:00:00:00 +0000]
"GET /someurl/?t                    rack=Blabla(Main) HTTP/1.1"
200     5864    -       "Mozilla/5.0 (Windows; U
; Windows NT 6.0; en-US) AppleWebKit/525.19 (KHTML, like Gecko)
Chrome/1.0.154.6                    5 Safari/525.19"
127.0.0.1       -       frank   [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.                    gif HTTP/1.0"   200     2326
NULL    NULL
Time taken: 0.269 seconds
```

Examine data from `apachelog` Table

- For a more refined search:

```
beeline> select a.host , count(*) as host_count from
apachelog a group by a.host order by host_count desc
limit 10;
```

```
+----------------+-------------+--+
|     a.host     | host_count  |
+----------------+-------------+--+
| 66.249.67.3    | 6111        |
| 74.125.74.193  | 2508        |
| 64.233.172.17  | 2416        |
| 74.125.16.65   | 2341        |
| 72.14.192.65   | 2290        |
| 66.249.67.87   | 2245        |
| 72.14.193.68   | 1424        |
| 72.14.194.1    | 1316        |
| 72.30.142.87   | 1255        |
| 74.125.75.17   | 945         |
+----------------+-------------+--+
```

Sqoop

- RDBMS data are critical for operation of any enterprise and will remain so for long time to come. Enriching analysis of unstructured data on the Spark/Hadoop side with RDBMS data and vice versa is essential.
- One convenient tool for automating transfer of data from Relational Database Systems into the world of Hadoop and Spark is Sqoop.
- Sqoop transfers data directly into HDFS, making them available for further analysis.
- There are 2 versions of Sqoop
  - Sqoop 1 is a "thick client" and is what you use in this tutorial. The command you run will directly submit the MapReduce jobs to transfer the data.
  - Sqoop 2 consists of a central server that submits the MapReduce jobs on behalf of clients, and a much lighter weight client that you use to connect to the server

Sqoop Commands

- To get the list of supported commands, type:

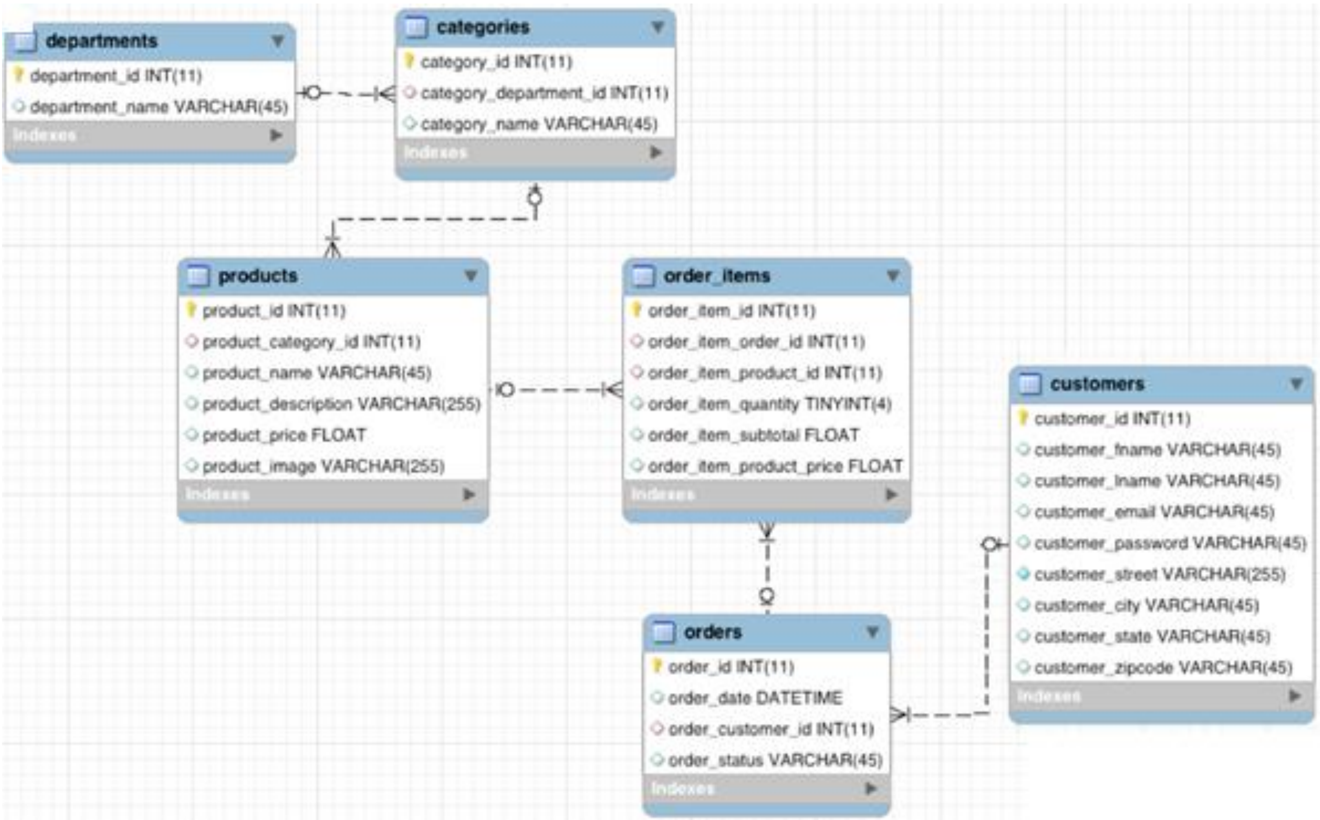
```
[cloudera@quickstart ~]$ sqoop help
Running Sqoop version: 1.4.6-cdh5.5.0
usage: sqoop COMMAND [ARGS]

Available commands:
  codegen             Generate code to interact with database records
  create-hive-table   Import a table definition into Hive
  eval                Evaluate a SQL statement and display the results
  export              Export an HDFS directory to a database table
  help                List available commands
  import              Import a table from a database to HDFS
  import-all-tables   Import tables from a database to HDFS
  import-mainframe    Import datasets from a mainframe server to HDFS
  job                 Work with saved jobs
  list-databases      List available databases on a server
  list-tables         List available tables in a database
  merge               Merge results of incremental imports
  metastore           Run a standalone Sqoop metastore
  version             Display version information

See 'sqoop help COMMAND' for information on a specific command.
```

Ingesting Relational Data

- We will use tables provided in MySQL demo database `retail_db` provided with Cloudera's VM. Schema of that database is presented below:



- Various business relevant queries involving those tables, even when they contain hundreds of thousands of rows, can be efficiently done within modern relational databases systems.

Benefits of Merging Data with Sqoop

- A benefit of Spark, Hadoop, Impala, Hive, … platform is that you can do the same queries at greater scale at lower cost, and perform many other types of analysis not suitable for RDBMS.
- Seamless integration is important when evaluating any new infrastructure. Hence, it’s important to be able to do what you normally do, and not break any regular BI reports or workloads over the dataset you plan to migrate.
- To analyze the transaction data in the new platform, we need to bring it into the Hadoop Distributed File System (HDFS). We need a tool that easily transfers structured data from a RDBMS to HDFS, while preserving structure. One that enables us to query the data, but not interfere with or break any regular workload on it.
- Apache Sqoop, is such tool. With Sqoop we can automatically load our relational data from MySQL (Oracle, DB2, etc) into HDFS, while preserving the structure of that data.

Apache Parquet

- How is data organized within each file is the key matter for database design.
- Apache Parquet is a columnar storage format available to the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.
- Parquet is built from the ground up with complex nested data structures in mind, and uses the record shredding and assembly algorithm.
- Parquet is built to support very efficient compression and encoding schemes.
- Parquet allows compression schemes to be specified on a per-column level, and is future-proofed to allow adding more encodings as they are invented and implemented.
- Parquet is an efficient columnar storage substrate without the cost of extensive and difficult to set up dependencies.
- You should experiment and find out whether you should use: Avro, Parquet, RCFile (Record Columnar File) or ORC (Optimized Row Columnar) file system.

Import all tables

- To use sqoop to import all tables of an existing database schema, at the command prompt we type all at one line:

```
$mysql -uretail_dba -p
mysql> show databases;
mysql>  use retail_db;
mysql> show tables;
```

```
$ sqoop import-all-tables -m 1 --connect
jdbc:mysql://quickstart:3306/retail_db --username=retail_dba  --
password=cloudera --compression-codec=snappy --as-parquetfile --
warehouse-dir=/user/hive/warehouse --hive-import
```

- Import takes a while, more than several minutes on a fast laptop.
- You will notice that `Sqoop` is executing several map-reduce jobs in the process.
- Parquet is a format designed for analytical applications on Hadoop. Instead of grouping your data into rows like typical data formats, it groups your data into columns. This is ideal for many analytical queries where instead of retrieving data from specific records, you're analyzing relationships between specific variables across many records. Parquet is designed to optimize data storage and retrieval in these scenarios..

```
hadoop fs -ls /user/hive/warehouse/products
hadoop fs -get /user/hive/warehouse/products/8173c7ad-7d1f-4a27-a44c-
8fb4f5650a93.parquet
vi 8173c7ad-7d1f-4a27-a44c-8fb4f5650a93.parquet
```