Lecture 05

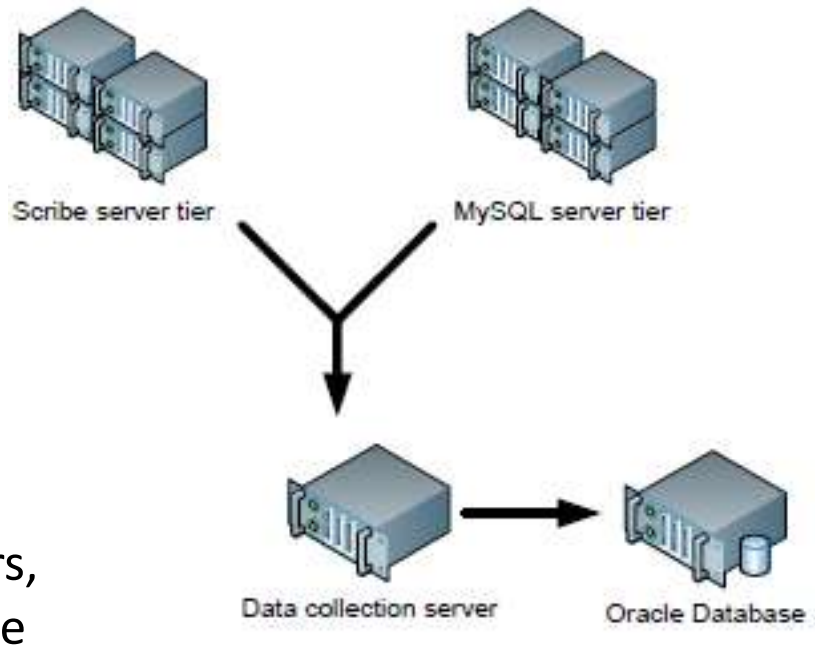
Apache Hive

Zoran B. Djordjević

csci e63 Big Data Analytics

Facebook 2006

- Started at Facebook.
- At that time, 2006, everything at Facebook was ran on MySQL.
- Facebook has a lot of Web Servers that produce a lot of logs with user related information that they wanted to extract.
- User data were held in farms of MySQL servers.
- All logs went through a locally developed log aggregation framework called Scribe.
- To run large scale reports: how many users, how many emails, who the messages were sent to, etc., a nightly cron job pushed data through an ETL process (Python scripts) into an Oracle database. Oracle spilled the reports.



Facebook Scales, 2007 > 2009 >2011> 2014

- In 2006 Facebook produced several tens of GBs of data and schema worked well.
- In mid 2007, Facebook logs accumulated 1TB of data per day.
- In 2009 the volume grew to 10 TB per day.
- January 2011, added 625 TB of compressed data
- In July 2011 Facebook Hadoop cluster had 30 PB of data.
- In 2014 daily volume reached 600 TB
- In 2014, the total size of Hive Warehouse was 300 PB
- Since Facebook was aware of Hadoop, they decided to push log data into Hadoop and try processing logs by MapReduce techniques.

Hadoop's MapReduce Jobe

- They started loading data from Scribe and MySQL data into Hadoop HDFS
- Facebook people started writing Hadoop MapReduce jobs to process data.
- To write MapReduce jobs you have to be a relatively sophisticated Java programmer. Hadoop was developed by geeks for geeks.
- It soon became apparent that some things were missing. Most notably:
  - Command-line interface for "end users" was not there.
    - End users are typically "marketing" people and they had no facilities to write queries on the fly. They had to bag real engineers to develop MapReduce programs and even run those programs for them.
    - The above broke the proper social hierarchy of Facebook Corp. Marketing people should never speak with geeks. In military they call that fraternization and can shoot you for it.
  - Information on data structures (schema) was not readily available.
    - Log data files have an implicit schema.
    - That schema is embedded in the code that knows how to read log files.
    - Schema of data is not readily visible.

Hive was Conceived

- In response to those challenges, Facebook developed Hive so that the "users" could perform:
  - Ad-hoc queries without writing full MapReduce jobs
  - Extract or create Schema information

  (Even Hive queries are still written and run by true engineers.)
- Hive could be used for
  - Log processing
  - Text mining
  - Document indexing
  - Customer-facing business intelligence (e.g., Google Analytics)
  - General Statistical Analysis, Predictive modeling, Hypothesis testing, etc.
- Hive has support for various aggregations and joins.
- Hive is considerably closer to standard SQL than PIG.
- Hive is more a data warehouse query language. PIG is more process oriented.

Hive Components

Hive has the following five major components:

- Shell, Driver, Compiler, Execution Engine and Meta Store

Shell

- A tool for user interaction. Allows interactive queries, just like DB shell connected to database. Supports web and JDBC clients.

Driver

- Management core of Hive engine. Driver manages query lifecycle, submits queries to the compiler, handles session, fetches the results and returns them to shell.

Compiler:

- Processing core that takes HQL language statements, parses them, creates query plan considering the schema of the database, selects optimal set of HDFS fetches, and other operations.

Hive Components

Execution engine:

- Directed Acyclic Graph of stages implemented as a set (tree) of
- Map/Reduce jobs, direct unfiltered fetches from HDFS and perhaps communications with Meta Store.

Meta Store:

- An actual relational database: Java Derby, MySQL, or any other.
- Meta Store contains schema of your tables,
- where in HDFS the table data are located, and
- a system called: SerDe (Serializer-Deserializer) which describes how to load data from HDFS or outside files and represent data as tables.

Hive vs. OLAP Warehouses

- Traditional OLAP warehouses create cubes. i.e. materialized views.
- OLAP cubes can not scale to 500 machines. Work on small clusters at best.
- Hive has no automatically generated materialized views.
- If you know what your marketing users are looking for nothing in Hive prevents you from prefabricating data sets that would allow the “users” to quickly find what they are looking for.
- Hive could operate on clusters of 500 or 10,000 machines
- There is no solution that could work on 1TB and return results in minutes.
- If you have 100 GB of data you are better off loading data into an Oracle database. Let Oracle index everything and then run your queries.
- Once you are in TB range Hadoop or Hive are faster than Oracle or any other RDBMS.
- Hive is approximately as fast as Hadoop itself. Hive simplifies your work.
- Hadoop and Hive are not targeting small incremental changes of data sets.
- Hadoop and Hive are meant for global enterprises.

Data Model

- The Major benefit of Hive is that it could make unstructured data look like tables.
- Simply organized data like comma separated values naturally look like tables.
- In `CREATE EXTERNAL TABLE` command you just specify "delimited by ', ' and data will be loaded properly.
- You could also write elaborate serializers/deserializers that could read complex files and populate tables with data contained in those files.
- Hive is a database (warehouse) with strongly typed tables.
- Columns could have atomic types: `int, float, string, date, boolean`
- Composite types: `list, map` (associative array) or `struct` (convenient for JSON-like data).
- Elements of composite types could be any types, including composite types, meaning that types could be arbitrarily complex.

Hive Extends SQL

- Hive has various extensions of SQL. For example:
  - EXPLODE operator would take lists of data and create several columns with atomic data.
  - COLAPSE operator takes lists of data and pushes them into a single column of comma separated data.

Partitions

- You can break large tables by ranges of values in a column, for example by date.
- Date partitioned tables are stored in directories which have subdirectories with names stamped with date.
- You can make queries against individual partitions. Such queries are naturally much faster than queries over entire domain of partition column.
- Within partitions you have sub-partitions called Buckets
- Buckets are Hash partitions within ranges.
- Buckets are useful for sampling. For example: you can perform 5% of query on a valid sample.
- Buckets are also used by optimizer .

Meta Store

- Meta store does not reside in HDFS. Usually it is a Java Derby or MySQL database. Could use almost any other relational databases with a JDBC connector.
- Meta store uses derby by default;
- Meta store is a Database and:
  - Contains a namespace containing a set of tables
  - Holds table definitions (column types, physical layout (where in HDFS tables live as files, etc.))
  - Partitioning data (what are partition boundaries, etc.)
- Database storage location of Meta Store is determined by the hive configuration variable named `javax.jdo.option.ConnectionURL`.
- By default (see `conf/hive-site.xml`), this location is `./metastore_db`
- Right now, in the default configuration, this metadata can only be seen by one user at a time.

Physical Layout

- Warehouse directory is stored in HDFS e.g. `/user/hive/warehouse` or a similarly named directory
- Every table is stored in a subdirectory of `/user/hive/warehouse`
- Partitions, buckets form subdirectories of tables.
- Those files are under Hive control.
- Hive documentation suggests that you could backup those files by just making a copy to another directory or machine.
- Table data stored in
  - Flat Control char-delimited text files (`ctrl A` is the default delimiter)
  - SequenceFiles which are native to Hadoop.
  - With custom serializer-deserilizers, called `SerDe`, files could use arbitrary data organization format

Installing Hive

- If you want to install Hive on a bare machine, you could go to www.apache.org and navigate to `Project > Hive`.
- You need to have Hadoop installed and `HADOOP_HOME` environmental variable in your path. Hive will work with Hadoop 2 and Hadoop 1.
- Hive could be downloaded from http://hive.apache.org/downloads.html or from Subversion.

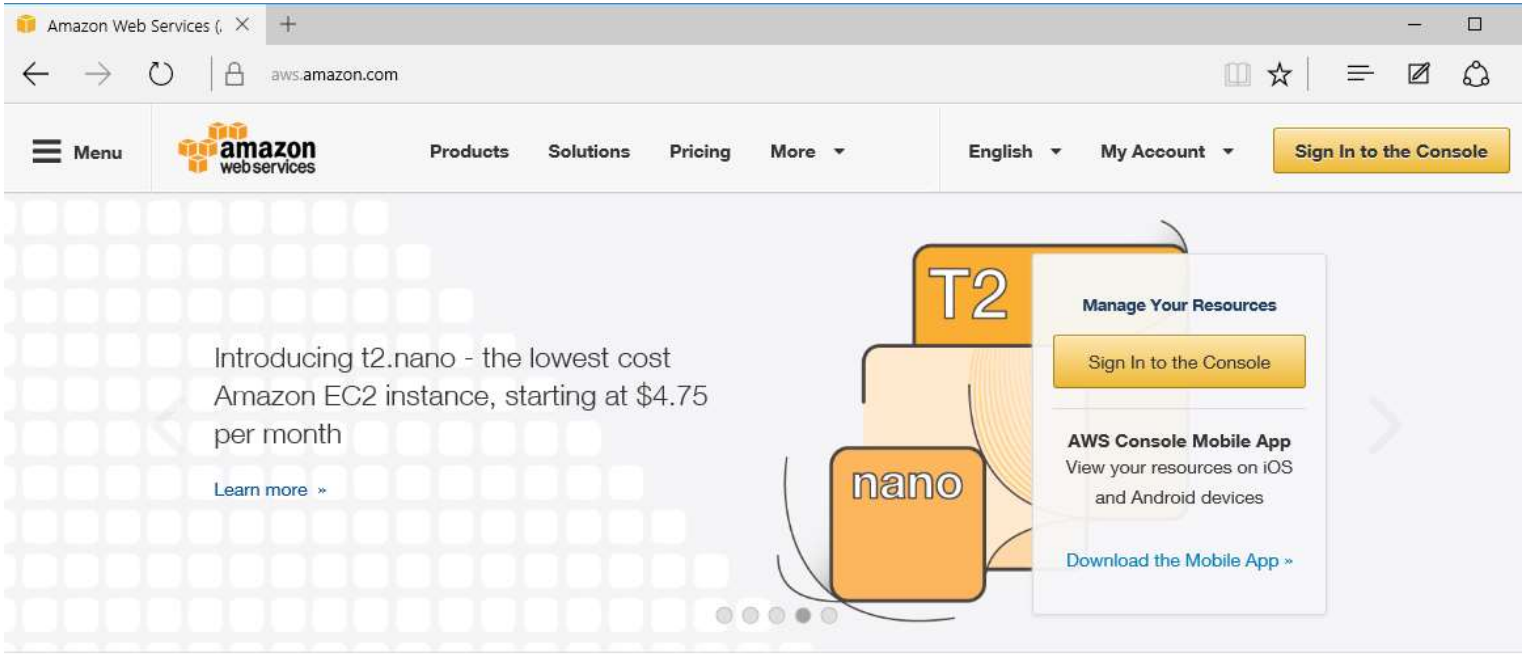
```
$ svn co http://svn.apache.org/repos/asf/hive/branches/branch-#.# hive
$ cd hive    -- Run Ant
$ ant package
$ cd build/dist
$ ls
README.txt
bin/ (all the shell scripts)
lib/ (required jar files)
conf/ (configuration files)
examples/ (sample input and query files)
```

Cloudera VM

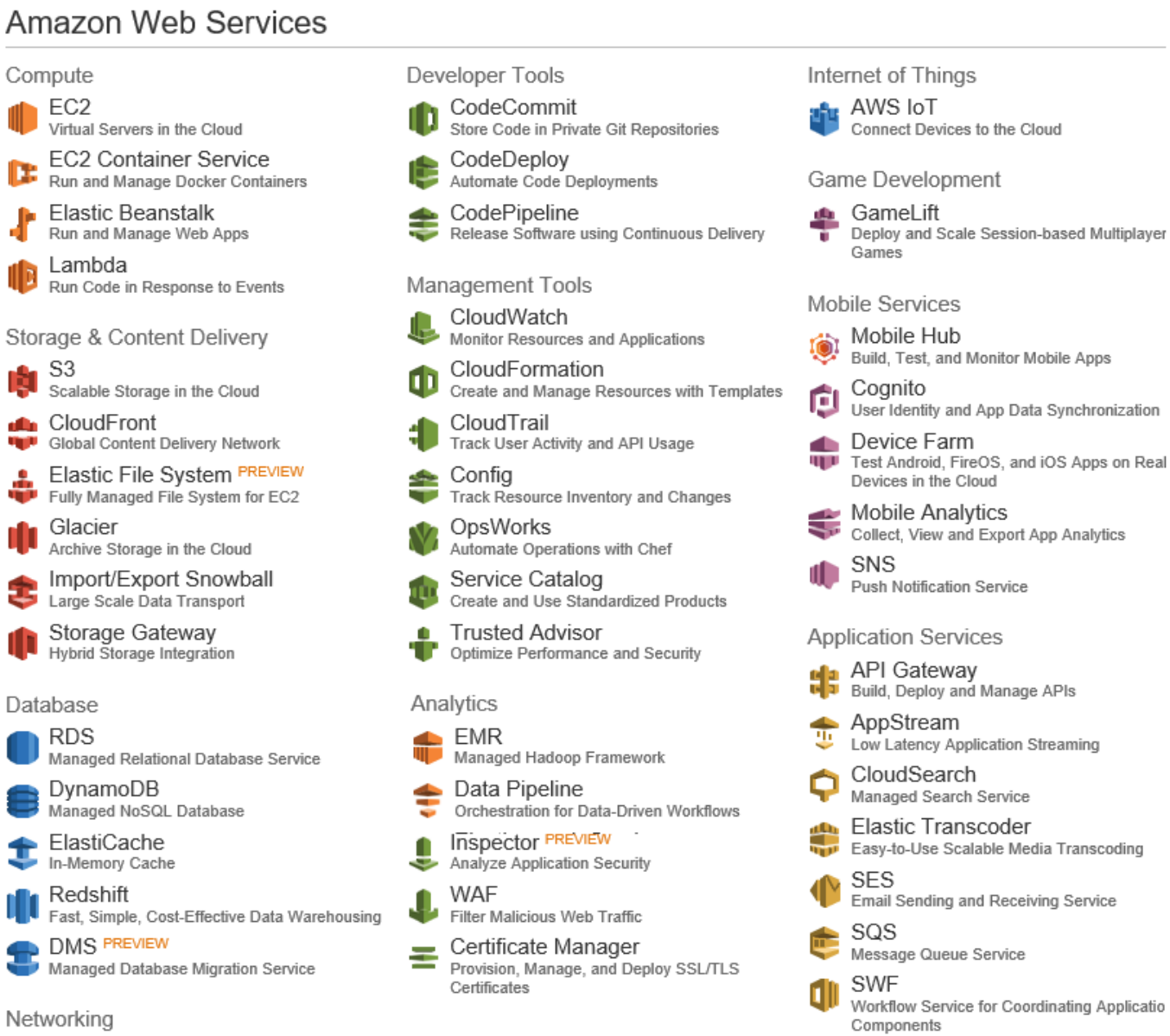
- You used to be able to go to www.cloudera.com and download a 64 bit VM with preconfigured versions of Hadoop, Hive and many other related applications.
- `cloudera-quickstart-vm-5.5.1-0-vmware.7z` or whatever was the name of the downloaded file has approximately 7GB and takes some time to download. You need utility 7-zip to open the archive.
- Once you open the archive, that is it. You run the VM using VMWare VMPlayer or, if you have VMWare Workstation or Fusion installed, simply open the VM.
- Check the option that VMWare Tools are updated at Startup time.
- Once you launch the VM, log in with the following account details:
  - **username:** `cloudera`
  - **password:** `cloudera`
- Some of us will have a problem. Cloudera offer only 64 bit VMs and those require more than 4GB of RAM. If you have a 32bit OS or have less than 4GB of RAM on your machine, you are almost out of luck. Actually, if Hive is what you really need, you can run older versions of CDH some of which require less space and memory.

Where to run Hive, AWS EMR Service

- If you have issues with your Cloudera VM or want to run Hive on a realistic cluster one option is to go to http://aws.amazon.com and open EMR (Elastic Map Reduce) service.



Under Analytics, Select EMR Service



EMR

- Select Create Cluster

- Resulting cluster will have a master node and zero or more slave modes.
- You will be able to ssh into the master node and run Hive shell and Hive programs from there.

How to run Hive

- On the master node of your EMR master or your Cloudera VM on the command line type `hive` and Hive shell opens:

```
[cloudera@quickstart ~]$ hive
Logging initialized using configuration in
file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline
is recommended.
hive>
```

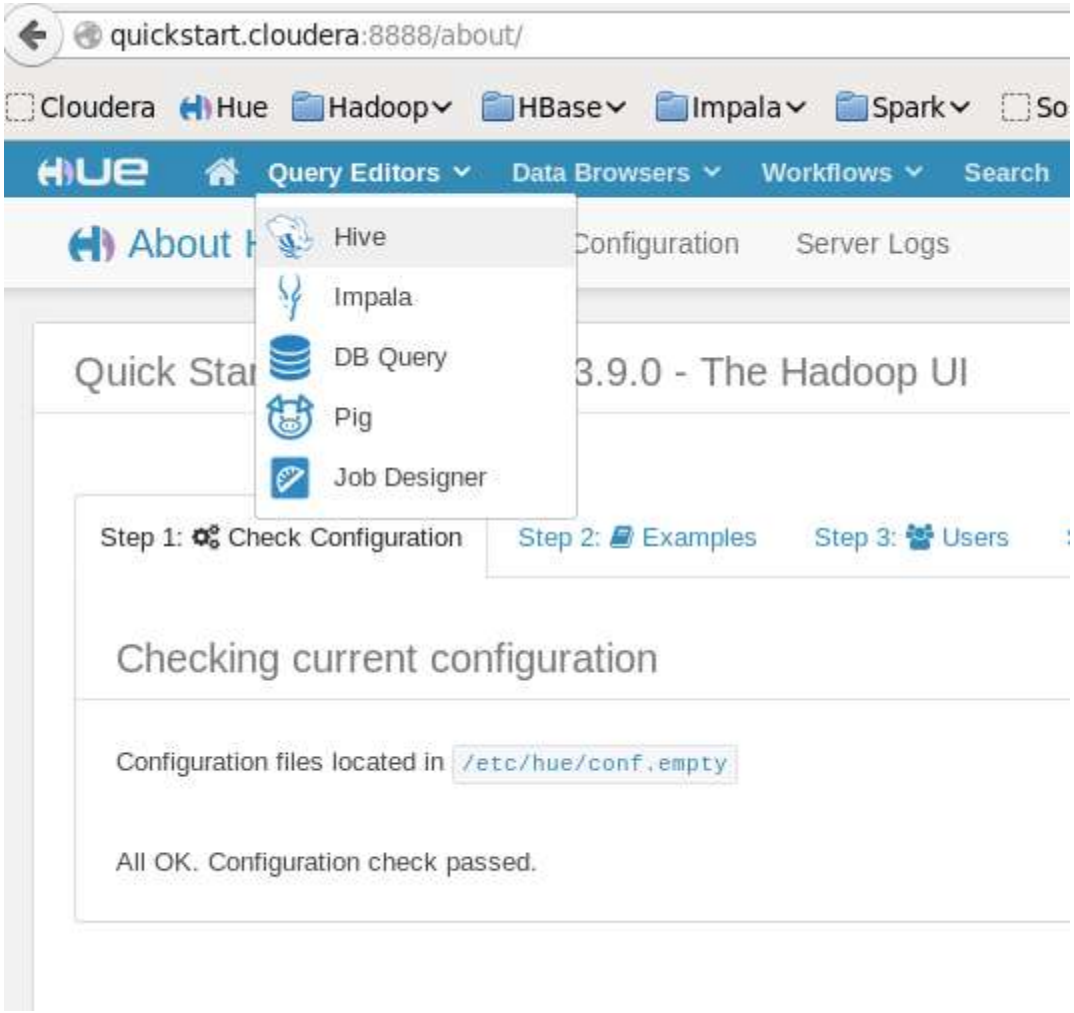
- To get out you used to type “quit”. Quitting is passé, so you do either Ctrl-d or old fashioned Ctrl-c. Next you type bee-line, and get:

```
[cloudera@quickstart ~]$ beeline
Beeline version 1.1.0-cdh5.5.0 by Apache Hive
beeline>
```

- This CLI should work

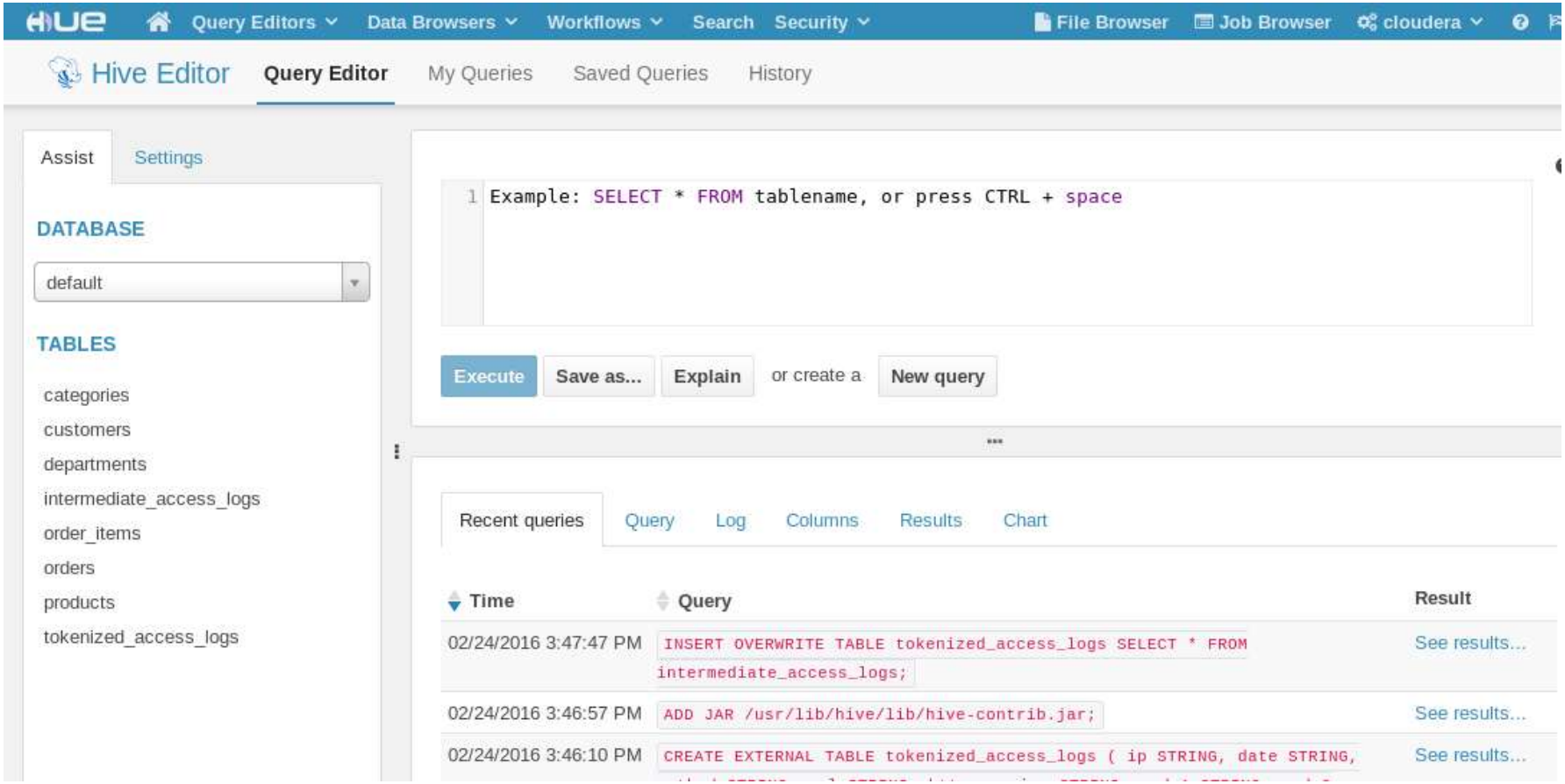
Could go to Hue as well

- When you open your Fire Fox on the tool's bar you will see Hue. Click it.
- On Cloudera's VM, username/password are `cloudera/cloudera`.
- On the next screen, select Query Editors and Hive



Hue Hive Editor

- Hive Query Editor opens with regions to enter queries, results, saved queries and more.



To Get Examples, Get Hive Release

- If you need hive code and examples, you could get the entire hive release. Go to: http://hive.apache.org/releases.html
- Select **Download a release now!**
- Select a Download Mirror
- Select `hive-2.0.0-bin.tar.gz` (Note, any version will work for our examples)
- Un-tar it in your Cygwin window or on your VM.

```
$ tar -zxf apache-hive-2.0.0-bin.tar.gz    # Note: z tells tar to run gzip filter first
[cloudera@quickstart]$ cd apache-hive-2.0.0-bin
[cloudera@quickstart apache-hive-2.0.0-bin]$ ls
bin  conf  docs  examples  lib  LICENSE  NOTICE  README.txt
[cloudera@quickstart apache-hive-2.0.0-bin]$ cd examples
[cloudera@quickstart examples]$ ls
files  queries
[cloudera@quickstart examples]$ cd files
[cloudera@quickstart files]$ ls
apache.access.2.log     in7.txt                 lt100.txt
srcbucket22.txt apache.access.log  in8.txt   lt100.txt.deflate
```

Fetch Example Files and Queries

- Hive installation comes with a fair number of examples. On cloudera command prompt, type:

```
[cloudera@quickstart ~] cd hive/example/
[cloudera@quickstart ~]/hive/examples$ ls
T1.txt                in2.txt    kv6.txt                srcbucket1.txt
T2.txt                in3.txt    lineitem.txt           srcbucket20.txt
T3.txt                in4.txt    lt100.sorted.txt       srcbucket21.txt
TestSerDe.jar         in5.txt    lt100.txt              srcbucket22.txt
apache.access.2.log   in6.txt    lt100.txt.deflate      srcbucket23.txt
[cloudera@quickstart ~]/hive/examples/$ cd ../queries
[cloudera@quickstart ~]/hive/examples/queries$ ls
case_sensitivity.q  input1.q   input8.q   join3.q    sample3.q  udf6.q
cast1.q      input2.q   input9.q   join4.q    sample4.q  udf_case.q
groupby1.q   input20.q  input_part1.q  join5.q    sample5.q  udf_when.q
groupby2.q  input3.q  input_testsequencefile.q  join6.q  sample6.q  union.q
groupby3.q  input4.q   input_testxpath.q         join7.q    sample7.q
```

- Study them. They are excellent educational tools
- You can tar the examples directory and copy it to you computer using scp command

```
hadoop@..~/hive$ tar cvf ex.tar example     # On EC2 Linux prompt
$ scp -i cloudera@192.168.205.107:/home/cloudera/hive/ex.tar .
$ tar xvf ex.tar
```

- The last 2 commands are run on your Cygwin prompt, on your local machine

Sample Data, Unpack Shakespeare

- On my machine in directory `~date` there are files:

  `shakespeare.tar.gz and bible.tar.gz files.`
- One contains complete works of Shakespeare and the other King James Bible. Both works use somewhat archaic form of English.
- We can unzip (un tar) both files. Command

  `$ tar zxf shakespeare.tar.gz`
- will un-tar Shakespeare's works and create directory `~/input`
- which contains a file `all-shakespeare` with all of Shakespeare works.
- You could examine the file by perhaps doing:

```
$ cat all-shakespeare | wc  or
$ cat all-shakespeare | tail -n 100
```

Copy Shakespeare into HDFS

- We will copy local directory "`input`" into the HDFS directory input:

```
$hadoop fs -put input input
```

- We could convince ourselves that the data inside HDFS is still the same Shakespeare by typing something like:
  - `$ hadoop fs -cat input/all-shakespeare | head -n 20`
  - `  1 KING HENRY IV`
  - `  DRAMATIS PERSONAE`
  - `KING HENRY     the Fourth. (KING HENRY IV:)`

Unpack King James Bible

- We un-tar `bible.tar.gz`, what creates new local directory `bible`:

```
$ tar zxf bible.tar.gz
```

- We copy that directory to HDFS, as well

```
$hadoop fs -put bible bible
```

- If we now list files/directories in HDFS we will see both `input` and `bible`.

Running a MapReduce `grep` Job

- On our VM example MapReduce scrips (jobs) are contained in `$HADOOP_HOME/hadoop-mapreduce-examples.jar` file.
- One of the scripts is a `grep` job which counts how many times every word appears in the analyzed corpus.
- In our case, Hadoop `grep` would scan the file (with all Shakespeare's works) placed in the specified (HDFS) directory "`input`" and create a tab delimited report named `shakespeare_freq`.
- Hadoop `grep` uses regular exp `'\w+'` to select all multi-character words.
- This `grep` is different from Unix (Linux) `grep`. Unix `grep` returns lines where a pattern appears. Hadoop `grep` counts word frequencies.
- The command to run Hadoop grep reads:

```
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-
  examples.jar grep input/all-shakespeare shakespeare_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 `shakespeare_freq`

Examine Result of `grep`

- Examin the output in HDFS directory `shakespeare_freq` by typing:

```
$ cloudera@quickstart:]$ hadoop fs -ls
Found 2 items
drwxr-xr-x   - cloudera      input
drwxr-xr-x   - cloudera      shakespeare_freq
$ hadoop fs -ls shakespeare_freq
Found 2 items
drwxr-xr-x   - cloudera        shakespeare_freq/_SUCCESS
-rw-r--r--   1 cloudera        shakespeare_freq/part-r-00000
cloudera@quickstart:~/data/input$
```

Examine Content of the `output` file

- We could also see partial content of the output file:

```
$ hadoop fs -cat shakespeare_freq/part-00000 | head -n 20
25848   the
23031   I
19671   and
18038   to
16700   of
14170   a
12702   you
11297   my
10797   in
6817    his
6773    be
6309    for
cat: Unable to write to output stream.
```

- These are frequency - word pairs, as expected.

Create Table to accept `grep` Data

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

```
hive> 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

```
hive> create table shakespeare (freq INT, word STRING) ROW FORMAT
     DELIMITED FIELDS TERMINATED BY '\t' stored as textfile;
hive> show tables; # This created table shakespeare with out any data
OK
shakespeare
Time taken: 8.268 seconds
hive> describe shakespeare;
OK
freq    int
word    string
Time taken: 1.253 seconds
hive>
```

Load grep Data into `shakespeare` Table

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

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

- On the load command, Hive moved HDFS directory `shakespeare_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.

Verify that `shakespeare` has `grep` Data

```
hive> 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
hive>
```

- This statement read from the table (actually as part of optimization, it read directly from the HDFS file) and presented us with the first 10 lines.
- This is the same data we saw previously.

More Advanced Query

- Slightly more advanced query would perhaps be this one:

```
hive> 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.

```
hive> 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:

```
hive> 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
. . . .
```

- Those words we call stop words. In Google-like analysis of relevance for text finding, we simply ignore stop word. 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.

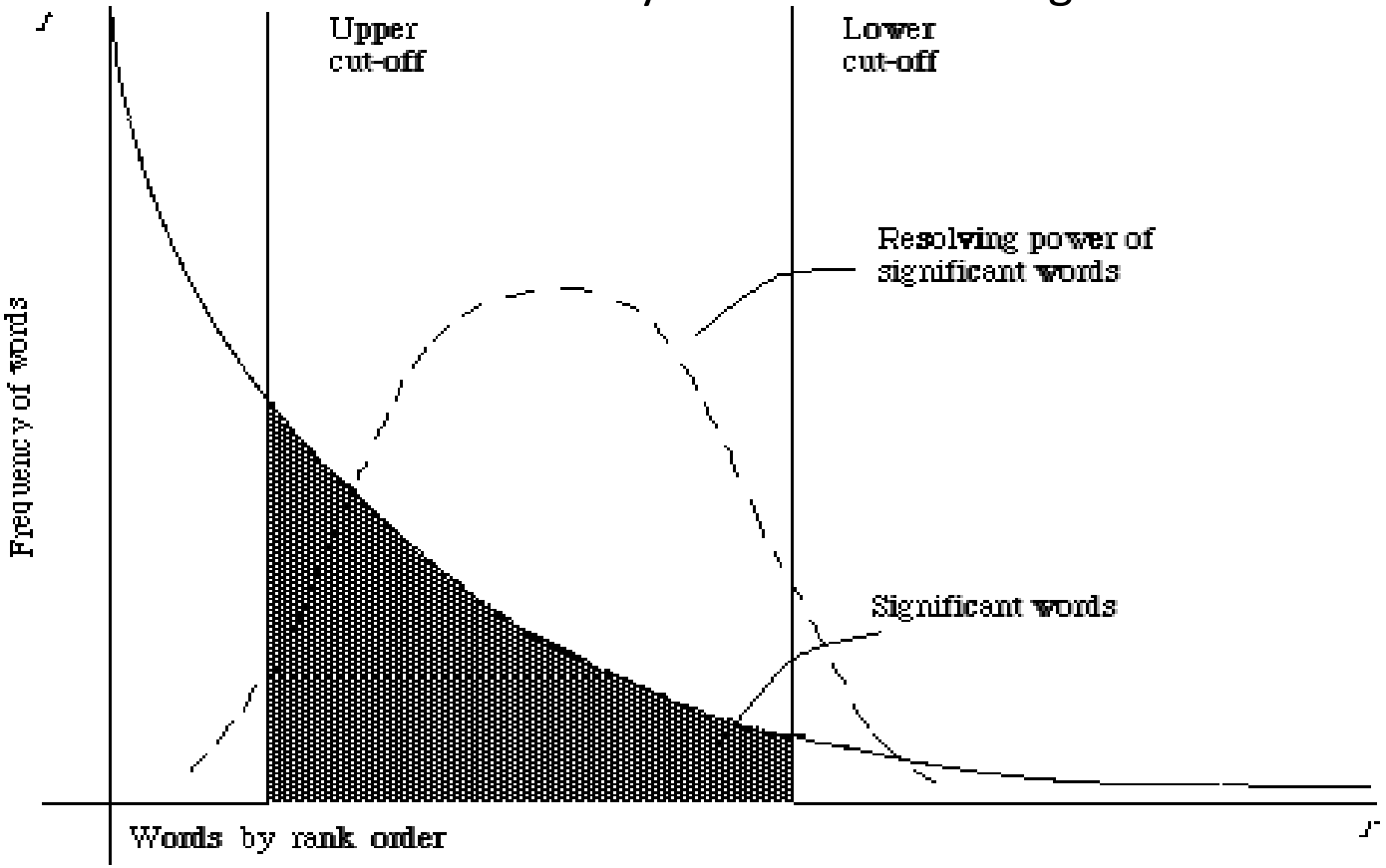


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adapted from Schultz[44] page 120)

How is Query Executed, Explain

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

```
hive> 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

```
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

```
            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 most powerful feature of Hive is the ability to create queries that joins tables together using regular SQL syntax.
- We have (freq, word) data for Shakespeare
- We could 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 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`
- Again we remove `_logs and –SUCCESS` directories if present.
- `$ hadoop fs –rm -r bible_freq/_logs`

Create KingJamesBible Table

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

Import data into KingJamesBible

```
hive> LOAD DATA INPATH "/user/cloudera/bible_freq" INTO TABLE
   Bible;
OK
Time taken: 0.781 seconds
hive> select * from kingjamesbible 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
hive>
```

Examine `bible_freq` directory in HDFS

- Once you imported data into `KingJamesBible` 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.
- You do recall 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 most common words in both volumes with corresponding frequencies

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

- For this table we do not need to specify how will date be stored.
- Hive will determine that by itself.

- Next, we will run a query that will select data from tables: `shakespeare` and `kingjamesbible`, create a 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

```
hive> INSERT OVERWRITE TABLE merged
SELECT s.word, s.freq, k.freq FROM
shakespeare s JOIN kingjamesbible k ON
(s.word = k.word)
WHERE s.freq >= 1 AND k.freq >= 1;
....
Ended Job = job_201404021324_0013
Loading data to table merged
7826 Rows loaded to merged
hive> . . . .
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
```

Most common common words

- What words appeared most frequently in both corpuses?

```
hive> 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
```

Hive's DDL Operations, `Create Table`

- We already know how to create Hive tables and browse through them

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
```

- Creates a table called `pokes` with two columns, the first being an integer and the other a string

```
hive> CREATE TABLE invites (foo INT, bar STRING)
  PARTITIONED BY (ds STRING);
```

- Creates a table called `invites` with two columns and a partition column called `ds`.
- The partition column is a virtual column. It is not a part of the data itself but is derived from the partition that a particular dataset is loaded into.
- By default, tables are assumed to be of text input format and the delimiters are assumed to be `^A(ctrl-a)`.

`Show Tables` Command

- `hive> SHOW TABLES '.*s';`
- `OK`
- `invites`
- `ip_locations`
- `pokes`

- `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:

```
  hive> ALTER TABLE pokes ADD COLUMNS (new_col INT);
  hive> ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT
   'this is a comment');
  hive> ALTER TABLE pokes RENAME TO happenings;
OK
Time taken: 0.17 seconds
  hive> ALTER TABLE happenings RENAME TO pokes;
```

Running commands on OS

- From within a `hive` shell you could run outside (OS) commands by placing an exclamation mark in front of the command. Like:

```
hive> !pwd;
/home/cloudera/hive/examples/files
Hive> !ls;
union_input.txt
x.txt
y.txt
z.Txt
. . .
```

- OS commands similar to the above appear not to work in Hue.

DML Operations, Loading Data from Flat Files

- To use examples provided with Elastic Map Reduce AMI distribution go to: `apache-hive-2.0.0-bin/examples/` directory of your hive untared download. I moved that directory to `/home/cloudera/hive/examples`
- Open hive shell in `~cloudera/hive` directory, or use Hue

```
hive> LOAD DATA LOCAL INPATH
   '/home/cloudera/examples/files/kv1.txt' OVERWRITE INTO TABLE
   pokes;
Copying data from file:/home/cloudera/hive/examples/files/kv1.txt
Loading data to table pokes.
```

- Use of key word `LOCAL` forces Hive to load data from a local flat file.
- If the keyword is not present, Hive presumes that you are loading data from HDFS.
- The keyword `'OVERWRITE'` signifies that existing data in the table is deleted.
- If the `'OVERWRITE'` keyword is omitted, data files are appended to existing data sets.

Every Table corresponds to a Directory

```
[cloudera@quickstart files]$ hadoop fs -ls /user/hive/warehouse
Found 4 items
drwxr-xr-x   - cloudera cloudera          0 2016-02-25 06:51
/user/hive/warehouse/kingjamesbible
drwxr-xr-x   - cloudera cloudera          0 2016-02-25 07:48
/user/hive/warehouse/merged
drwxr-xr-x   - cloudera cloudera          0 2016-02-25 08:42
/user/hive/warehouse/pokes
drwxr-xr-x   - cloudera cloudera          0 2016-02-25 06:50
/user/hive/warehouse/shakespeare
[cloudera@quickstart files]$
[cloudera@quickstart files]$ hadoop fs -ls
/user/hive/warehouse/kingjamesbible
Found 2 items
-rw-r--r--   1 cloudera cloudera          0 2016-02-25 06:42
/user/hive/warehouse/kingjamesbible/_SUCCESS
-rw-r--r--   1 cloudera cloudera     147408 2016-02-25 06:42
/user/hive/warehouse/kingjamesbible/part-00000
```

- Data of a non-partitioned table are contained in a single file

Every Partition corresponds to a Directory

```
[..@quickstart files]$ hadoop fs -ls /user/hive/warehouse/invites
Found 2 items
drwxr-xr-x   - cloudera cloudera          0 2016-02-25 09:17
/user/hive/warehouse/invites/ds=2008-08-08
drwxr-xr-x   - cloudera cloudera          0 2016-02-25 09:16
/user/hive/warehouse/invites/ds=2008-08-15
```

- Note that partitioned indexes are used as new directory names.
- Insider every "partition" directory there is a file containing data for that partition:

```
[..@..files]$ hadoop fs -ls /user/hive/warehouse/invites/ds=2008-08-
08
Found 1 items
-rw-r--r--   1 cloudera cloudera        216 2016-02-25 09:17
/user/hive/warehouse/invites/ds=2008-08-08/kv3.txt
[..@..]hadoop fs -cat /user/hive/warehouse/invites/ds=2008-08-
08/kv3.txt
213 val_213
146 val_146
406 val_406
```

Verify Presence of Data in `pokes`

```
hive> select * from pokes limit 20;
OK
238     val_238 NULL
86      val_86  NULL
311     val_311 NULL
27      val_27  NULL
165     val_165 NULL
409     val_409 NULL
255     val_255 NULL
278     val_278 NULL
98      val_98  NULL
484     val_484 NULL
265     val_265 NULL
193     val_193 NULL
401     val_401 NULL
150     val_150 NULL
273     val_273 NULL
224     val_224 NULL
369     val_369 NULL . . . .
Time taken: 0.272 seconds
hive>
```

NOTE:

- NO verification of data against the schema is performed by the load command.
- If the file is in HDFS, it is moved into the Hive-controlled file system namespace.
- The root of the Hive directory is specified by the option `'hive.metastore.warehouse.dir'` in `conf/hive-site.xml`.
- If that directory is not there, the users should create this directory before trying to create tables via Hive.

Loading data into Partitioned Table

```
hive> LOAD DATA LOCAL INPATH
  '/home/cloudera/hive/examples/files/kv2.txt' OVERWRITE INTO
  TABLE invites PARTITION (ds='2008-08-15');
hive> LOAD DATA LOCAL INPATH
  '/home/cloudera/hive/examples/files/kv3.txt' OVERWRITE INTO
  TABLE invites PARTITION (ds='2008-08-08');
```

- The two LOAD statements above load data into two different partitions of the table invites.
- Table invites must be created as partitioned by the key `ds` for this to succeed.
- To verify that data is loaded run:

```
SELECT a.foo FROM invites a WHERE a.ds='2008-08-08';
```

- The statement selects column `'foo'` from all rows of partition `<2008-08-08>` of `invites` table. The results are not stored anywhere, but are displayed on the console.
- For fast access to data, partitioned tables should have a partition index selected in the WHERE clause of the statement.

Without Partition the Query makes a Full Table Scan

```
hive> select count(*) from invites;
Starting Job = job_201211300612_0015, Tracking URL =
http://quickstart:50030/jobdetails.jsp?jobid=job_201602250612_001
5
Kill Command = /usr/lib/hadoop/bin/hadoop job  -
Dmapred.job.tracker=quickstart:8021 -kill job_201602250612_0015
Ended Job = job_201602250612_0015
OK
525
Time taken: 54.988 seconds
hive> select count(*) from invites where ds='2008-08-08';
25
Time taken: 15.792 seconds
hive> select count(*) from invites where ds='2008-08-15';
500
Time taken: 31.454 seconds
hive>
```

Exporting Data from a Table into an HDFS directory

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

```
INSERT OVERWRITE DIRECTORY './hdfs_out' SELECT a.* FROM invites a
   WHERE a.ds='2008-08-08';
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
hive>
```

- Verify presence of the directory

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

Exporting Data into a Local Directory

- You could as well send data into a local directory:

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM
pokes a;
cloudera@quickstart files]$ cd /tmp/local_out
$ ls
000000_0
$ cat 000000_0
87 val_87 \N
364 val_364 \N
179 val_179 \N
118 val_118 \N
134 val_134 \N
395 val_395 \N
282 val_282 \N
138 val_138 \N
238 val_238 \N . . . .
```

GROUP BY Statements

```
hive> create table events (bar string, foo int);
```

- Note that the following statements are equivalent.

```
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar,
   count(*) WHERE a.foo > 0 GROUP BY a.bar;
hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(1) FROM
   invites a WHERE a.foo > 0 GROUP BY a.bar;
```

- 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 events SELECT a.bar, sum(a.foo) FROM
   invites a WHERE a.foo > 0 GROUP BY a.bar;
```

- The following syntax works:

```
hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT
   OVERWRITE TABLE events SELECT t1.bar, t2.foo;
```

Multi-Table Insert

- Modified syntax, where query starts with a FROM clause has its benefits.
- Could 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` need 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:

```
$hadoop fs –copyFromLocal /usr/lib/hive/contrib/hive-contrib.jar \
       /user/hive/hive-contrib.jar
hive> add jar /user/hive/hive_contrib.jar;
```

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

```
hive> 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: `apache.access.2.log` and `apache.access.log`. We will use them to test `CREATE TABLE` command with the regular expression.

```
hive> load data local inpath
'/home/cloudera/hive/examples/files/apache.access.2.log'
into table apachelog;
hive> load data local inpath
'/home/cloudera/hive/examples/files/apache.access.txt' into
table apachelog;
hive> 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
```

Test data in pig-apache samples s3 folder

- Large apache.access.log data file(s) could be found in an S3 bucket:
  `s3n://elasticmapreduce/samples/pig-apache/input`
- If on an AWS EMR cluster machine it is easy to fetch that data:

```
$ hadoop fs -ls s3n://elasticmapreduce/samples/pig-apache/input
Found 6 items
-rwxrwxrwx   1    8754118 2009-08-04 20:33 /samples/pig-
apache/input/access_log_1
-rwxrwxrwx   1    8902171 2009-08-04 20:33 /samples/pig-
apache/input/access_log_2
. . .
-rwxrwxrwx   1    8892828 2009-08-04 20:34 /samples/pig-
apache/input/access_log_6
```

- Wee will copy apache log data from the S3 bucket into local directory.

```
$ hadoop fs -copyToLocal \
   s3n://elasticmapreduce/samples/pig-apache/input .
$ ls input
access_log_1
. . .
access_log_6
```

apache_log_1.txt

- We could transfer file `apache_log_1` from our examples directory to Cloudera VM

```
$ scp access_log_1.txt  \
      cloudera@192.168.209.129:/home/cloudera
cloudera@192.168.209.129's password:
access_log_1.txt      100% 8549KB   8.4MB/s   00:01

zdjor@DESKTOP-0NUU7AF /cygdrive/e/CLASSES/code
```

- On VM side add readability to the file:

```
$ chmod + r access_log_1
```

Load Data

```
LOAD DATA LOCAL INPATH '/home/cloudera/access_log_1.txt' into table
apachelog;
Copying data from file:/home/cloudera/access_log_1.txt
Loading data to table apachelog
OK
Time taken: 3.794 seconds
```

- Next examine the data

```
SELECT * from apachelog;
46      "http://example.org/"   "Mozilla/5.0 (Macintosh; U; Intel Mac OS X
10_5_7; en-us) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0
Safari/530.17"
66.249.67.3     -       -       [20/Jul/2009:20:13:21 -0700]    "GET
/gallery/main.php?g2_controller=exif.SwitchDetailMode&g2_mode=detailed&g2_retur
n=%2Fgallery%2Fmain.php%3Fg2_itemId%3D30893&g2_returnName=photo HTTP/1.1"
302     5       "-"     "Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)"
66.249.67.3     -       -       [20/Jul/2009:20:13:24 -0700]    "GET
/gallery/main.php?g2_itemId=30893&g2_fromNavId=xfc647d65 HTTP/1.1"      200
8196    "-"     "Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)"
```

Session History

- Hive writes session history to a file `hive.log` it stores in a local directory:

```
$ls /tmp/cloudera/hive.log
$ vi hive.log
2016-02-26 09:32:47,832 INFO  [main]: session.SessionState
(SessionState.java:createPath(626)) - Created HDFS directory:
/tmp/hive/cloudera/6f8ea094-76dd-476d-b14c-70b353078dc7
2016-02-26 09:32:47,834 INFO  [main]: session.SessionState
(SessionState.java:createPath(626)) - Created local directory:
/tmp/cloudera/6f8ea094-76dd-476d-b14c-70b353078dc7
```

hive_job_log_hadoop_201303012359_200507540.txt

```
SessionStart SESSION_ID="hadoop_201303012359"
TIME="1362182359726"

QueryStart QUERY_STRING="create table shakespeare (freq INT, word
STRING) ROW FORMAT       DELIMITED FIELDS TERMINATED BY '\t'
stored as textfile" QUERY_ID="hadoop_20130301235959_3456b91d-
e074-4f05-a39e-b3c51e183164" TIME="1362182369190"

Counters plan="{"queryId":"hadoop_20130301235959_3456b91d-e074-
4f05-a39e-
b3c51e183164","queryType":null,"queryAttributes":{"queryString":"
create table shakespeare (freq INT, word STRING) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t' stored as
textfile"},"queryCounters":"null","stageGraph":{"nodeType":"STAGE
","roots":"null","adjacencyList":"]"},"stageList":[{"stageId":"St
age-
0","stageType":"DDL","stageAttributes":"null","stageCounters":"}"
,"taskList":[{"taskId":"Stage-
0_OTHER","taskType":"OTHER","taskAttributes":"null","taskCounters
":"null","operatorGraph":"null","operatorList":"]","done":"false"
,"started":"false"}],"done":"false","started":"false"}],"done":"f
alse","started":"true"}" TIME="1362182369220"

TaskStart TASK_NAME="org.apache.hadoop.hive.ql.exec.DDLTask"
TASK_ID="Stage-0" QUERY_ID="hadoop_20130301235959_3456b91d-e074-
4f05-a39e-b3c51e183164" TIME="1362182369223"
```

References

- http://wiki.apache.org/hadoop/Hive/GettingStarted
- http://www.cloudera.com/videos/introduction_to_hive
- http://www.cloudera.com/videos/hive_tutorial
- http://issues.apache.org/jira/browse/HIVE-662