Introduction to Hive

Mahran Farhat (farhatmahran@gmail.com)

Presentation:

This section teaches you how to try out Hive on the VM. The Hive data warehouse facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

In this section, we'll see how to use Hive to query a dataset of Shopping Mall purchases. The dataset is located in the udacity_training/data/purchases.txt. Hive is already installed and setup on your VM, so getting started is just a command away. Type the following in the terminal :

```
$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties

Hive history file=/tmp/training/hive_job_log_training_201602010336_645327107.txt

hive>
```

Let's use the SHOW TABLES command to see if any tables exist. Interacting with Hive is exactly like any relational database - so if you've worked with MySQL or PostgreSQL before you'll feel right at home.

```
hive> SHOW TABLES;
OK

Time taken: 2.028 seconds
```

Since no tables exist, we see a blank list. Now let's go ahead and create a table.

```
CREATE TABLE test_tables;
FAILED: SemanticException [Error 10043]: Either list of columns or a custom serializer should be
specified


hive> CREATE TABLE test_tables (some_text STRING);

OK

Time taken: 0.713 seconds
```

```
hive> SHOW TABLES;
OK
test_tables
Time taken: 0.053 seconds

hive> select * from test_tables;
OK
Time taken: 0.161 seconds
```

Press Ctrl-D to exit from the Hive console. So we created the test_tables above, but the first command failed because we didn't provide any schema. Every table that you wish to create must have a schema. In the subsequent command, we provide a dummy column name - some_text of the STRING type to make the command succeed. Once the table is created, SHOW TABLES correctly lists our table. However, since our table has no data, the select query returns a blank list.

Now we know how to create a table in Hive, we'll create a table and add real data. But before we do that, let us inspect the data that we're going to put in. This will be required for coming up with a schema for our database.

```
head udacity_training/data/purchases.txt

2012-07-20 09:59:00,Corpus Christi,CDs,327.91,Cash
2012-03-11 17:29:00,Durham,Books,115.09,Discover
2012-07-31 11:43:00,Rochester,Toys,332.07,MasterCard
2012-06-18 14:47:00,Garland,Computers,31.99,Visa
2012-03-27 11:40:00,Tulsa,CDs,452.18,Discover
2012-05-31 10:57:00,Pittsburgh,Garden,492.25,Amex
2012-08-22 14:35:00,Richmond,Consumer Electronics,346,Amex
2012-09-23 16:45:00,Scottsdale,CDs,21.58,Cash
2012-10-17 11:29:00,Baton Rouge,Computers,226.26,Cash
2012-07-03 11:05:00,Virginia Beach,Women's Clothing,23.47,Cash
```

We can see above that our data has 5 columns, each separated by a comma. The first column is a timestamp indicating the time and date of the sale. The second column indicates the store location where the sale occured, followed by the category of the product. The last two columns indicate the price and the mode of payment associated with the sale respectively.

Let's log into hive and load this data into HDFS. We'll start by first creating a new table.

```
hive> CREATE TABLE purchases (
  `sales_date` TIMESTAMP,
  `store_location` STRING,
  `category` STRING,
  `price` FLOAT,
  `card` STRING
) row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.177 seconds
```

In the above command, we create a new table called purchases with the appropriate schema shown above. The row format ... command tells Hive that we'll be loading in data from a csv file eventually. The next step is to move our data into HDFS so that we can import it in Hive. Exit from hive and run the following command from the csds-material directory.

```
$ hadoop fs -copyFromLocal udacity_training/data/purchases.txt /user/csds/input
$ hadoop fs -ls /user/csds/input
Found 1 items
-rw-r--r--   1 training supergroup      53755 2016-02-01 04:29 /user/csds/input/purchases.txt
```

Now we have everything in place to load data in Hive. Log back into the hive console and run the following :

```
hive> LOAD DATA INPATH '/user/csds/input/purchases.txt' INTO TABLE purchases;
Loading data to table default.purchases
OK
Time taken: 2.436 seconds
```

Great our data is now loaded. Now let's explore the data

```
hive> show tables;
OK
purchases
test_tables
Time taken: 0.133 seconds


hive> select * from purchases limit 10;
OK
2012-07-20 09:59:00     Corpus Christi  CDs     327.91  Cash
```

```
2012-03-11 17:29:00	Durham	Books	115.09	Discover
2012-07-31 11:43:00	Rochester	Toys	332.07	MasterCard
2012-06-18 14:47:00	Garland	Computers	31.99	Visa
2012-03-27 11:40:00	Tulsa	CDs	452.18	Discover
2012-05-31 10:57:00	Pittsburgh	Garden	492.25	Amex
2012-08-22 14:35:00	Richmond	Consumer Electronics	346.0	Amex
2012-09-23 16:45:00	Scottsdale	CDs	21.58	Cash
2012-10-17 11:29:00	Baton Rouge	Computers	226.26	Cash
2012-07-03 11:05:00	Virginia Beach	Women's Clothing	23.47	Cash
Time taken: 0.155 seconds

hive> select sum(price) from purchases where card = "Cash";
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201602010237_0001, Tracking URL = http://0.0.0.0:50030/jobdetails.jsp?jobid=job_201602010237_0001
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=0.0.0.0:8021 -kill job_201602010237_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-02-01 04:36:15,505 Stage-1 map = 0%,  reduce = 0%
2016-02-01 04:36:17,537 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.57 sec
2016-02-01 04:36:18,558 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.57 sec
2016-02-01 04:36:19,565 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.23 sec
2016-02-01 04:36:20,582 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.23 sec
MapReduce Total cumulative CPU time: 1 seconds 230 msec
Ended Job = job_201602010237_0001
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 1.23 sec   HDFS Read: 0 HDFS Write: 0 SUCCESS
```

```
Total MapReduce CPU Time Spent: 1 seconds 230 msec
OK
55199.32992255688
Time taken: 9.002 seconds
```

In this last query, we ran a simple query to calculate the total price of all the products that were paid in cash. We can see that our query got mapped to MapReduce tasks and got run by Hadoop. Finally at the end, we see the answer to our query - $55199 and also the total time taken.

So this is how you can use Hive and the power of SQL to analyse your big data that is already stored on an HDFS cluster. For more practice, try running your own queries and see what you can uncover. If you're feeling adventurous, try to formuate the queries for answering the questions below -

1. What is the average price of the products that were purchased via Mastercard?
2. Which date recorded the highest total sales?
3. What is the minimum value of a product under the Computers category?
4. How many distinct categories of products are there?
5. Which store location had the lowest total sales?

Great Day