Homework 5: Hadoop, Sqoop & Hive

E-63 Big Data Analytics
Harvard University, Autumn 2017

Tim Hagmann

Oktober 07, 2017

Contents

WSL Setup

A lot of company notebook don't allow running virtual machines (VT-X is disabled). This is also the case in my case. An alternative of using VM is to use the docker image. After installing docker nativily, it is possible to run docker against the WSL engine. This is because Docker can expose a TCP endpoint which the CLI (i.e., WSL) can attach to.

Note: The TCP endpoint on Windows is turned off by default. To activate it, right-click the Docker icon in your taskbar and choose Settings, and tick the box next to "Expose daemon on tcp://localhost:2375 without TLS".

Add docker to WSL

With that done, all we need to do is instruct the CLI under Bash to connect to the engine running under Windows instead of to the non-existing engine running under Bash, like this:

```
echo "export DOCKER_HOST='tcp://0.0.0.0:2375'" >> ~/.bashrc
source ~/.bashrc
```

Problem 1 (10%)

Download Quick Start for CDH 5.12. Start the image. Examine whether *hadoop-hdfs-** , *hadoop-mapreduce-** and *hadoop-yarn-** daemons are running. If those daemons are not running start all of them. If any of daemons fails to run, try to fix it.

Install CDH 5.12

```
docker pull cloudera/quickstart:latest
```

Start docker

```
docker run --hostname=quickstart.cloudera --privileged=true -it --rm -i -t -p 8888:8888 \
cloudera/quickstart /usr/bin/docker-quickstart
```

Change to cloudera user

The docker image starts by default into the root user. We're changing to the installed cloudera user (password=cloudera).

```
su cloudera
cd ~
```

Check status

```
for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x status ; done
```

```
[root@quickstart /]# for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x status ; done
Hadoop datanode is running                                 [  OK  ]
Hadoop journalnode is running                              [  OK  ]
Hadoop namenode is running                                 [  OK  ]
Hadoop secondarynamenode is running                        [  OK  ]
Hadoop httpfs is running                                   [  OK  ]
Hadoop historyserver is running                            [  OK  ]
Hadoop nodemanager is running                              [  OK  ]
Hadoop proxyserver is not running                          [FAILED]
Hadoop resourcemanager is running                          [  OK  ]
```

The proxyserver appears to have failed at load time. In order to fix this issue, we can edit the yarn-site.xml file

Stop services

First we're stoping the services

```
for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x stop ; done
```

Fix proxyserver

Next we're configuring the proxyserver file

```
sudo vi /etc/hadoop/conf.pseudo/yarn-site.xml
```

We have to add the following property:

```
  <property>
    <description>web proxy</description>
    <name>yarn.web-proxy.address</name>
    <value>localhost:3122</value>
  </property>
```

Add environment variables

In order for spark to run, we have to set the environment variables for *SPARK_HOME* and *JAVA_HOME*. We're adding those lines to the .bashrc file.

```
vi /home/cloudera/.bashrc
```

```
# Environment variables
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
```

Next we have to source the *.bashrc* startup file.

```
source .bashrc
```

Start services

Next we're starting all the services

```
for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x start ; done
sudo service hadoop-yarn-resourcemanager restart
```

Modify log4j.properties

In order to eliminate the error messages in the **spark-shell**, we have to create the log4j.properties file from the template file and setting the log level to ERROR and log4j.rootCategory=ERROR.

```
# Create log4j.properties
sudo mv /etc/spark/conf/log4j.properties.template /etc/spark/conf/log4j.properties

# Edit file
sudo vi /etc/spark/conf/log4j.properties

# Replace INFO, WARN with ERROR.
:g /WARN/s//ERROR/g
:g /INFO/s//ERROR/g
:wq
```

Check status

```
for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x status ; done
```

```
[root@quickstart /]# for x in `cd /etc/init.d ; ls hadoop-*`; do sudo service $x status ; done
Hadoop datanode is running                                 [  OK  ]
Hadoop journalnode is running                              [  OK  ]
Hadoop namenode is running                                 [  OK  ]
Hadoop secondarynamenode is running                        [  OK  ]
Hadoop httpfs is running                                   [  OK  ]
Hadoop historyserver is running                            [  OK  ]
Hadoop nodemanager is running                              [  OK  ]
```

```
Hadoop proxyserver is not running                          [  OK  ]
Hadoop resourcemanager is running                          [  OK  ]
```

Problem 2 (10%)

Examine whether there are HDFS home directories for users: *spark*, *hive*, *oozie*, and *cloudera*. If the directories are present, find the content of those directories. If the directories are not present, create them. Please do not format the namenode.

Check HDFS directory

```
hadoop fs -ls /user
```

```
drwxr-xr-x   - cloudera cloudera            0 2016-04-06 02:25 /user/cloudera
drwxr-xr-x   - mapred   hadoop              0 2016-04-06 02:26 /user/history
drwxrwxrwx   - hive     supergroup          0 2016-04-06 02:27 /user/hive
drwxrwxrwx   - hue      supergroup          0 2016-04-06 02:26 /user/hue
drwxrwxrwx   - jenkins  supergroup          0 2016-04-06 02:26 /user/jenkins
drwxrwxrwx   - oozie    supergroup          0 2016-04-06 02:27 /user/oozie
drwxrwxrwx   - root     supergroup          0 2016-04-06 02:26 /user/root
drwxr-xr-x   - hdfs     supergroup          0 2016-04-06 02:27 /user/spark
```

The above output shows, that all users appear to be added correctly.

Check health

```
hdfs fsck /
```

```
.............Status: HEALTHY
 Total size:    837790910 B (Total open files size: 166 B)
 Total dirs:    76
 Total files:   913
 Total symlinks:                0 (Files currently being written: 3)
 Total blocks (validated):      911 (avg. block size 919638 B) (Total open file blocks (not validated):
 Minimally replicated blocks:   911 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          1
 Number of racks:               1
FSCK ended at Sat Oct 07 09:37:21 UTC 2017 in 337 milliseconds


The filesystem under path '/' is HEALTHY
```

The filesystem appears to be healthy. We can move on to the next exercice.

Problem 3 (15%)

Create new Linux user *smith*. Make that user a member of the *mapred* Linux group. Make that user a *sudo* user. Create the home directory of user *smith* in HDFS. Download provided files *bible.tar* and *shakespeare.tar*. Unzip both tar files and copy the resulting files into HDFS directory input of user smith. As user *smith* run Hadoop *grep* on both *bible* and *shakespeare* texts. Every Hadoop run requires separate output directory. Examine content of first 20 lines of files generated by Hadoop *grep*.

Create User

```
# add new user
sudo useradd smith

# add password for the new user
sudo passwd smith
```

```
Changing password for user smith.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

```
# Verify presence of new user
grep smith /etc/passwd
```

```
smith:x:502:504::/home/smith:/bin/bash
```

```
# make new user member of group mapred
sudo usermod -g smith mapred
sudo usermod -g cloudera mapred

# login to the new user account using su command.
su smith
cd ~
```

Create home directory

```
#create HDFS directory for the new user
sudo -u hdfs hadoop fs -mkdir -p /user/smith
sudo -u hdfs hadoop fs -mkdir -p /user/cloudera

#change permissions 777 (read/write/exec for owner/group/other)
sudo -u hdfs hadoop fs -chmod -R 777 /user/smith
sudo -u hdfs hadoop fs -chmod -R 777 /user/cloudera

#change ownership to the new user
sudo -u hdfs hadoop fs -chown smith:mapred /user/smith
sudo -u hdfs hadoop fs -chown cloudera:mapred /user/cloudera

#verify HDFS dir created
sudo -u hdfs hadoop fs -ls -R /user/smith
sudo -u hdfs hadoop fs -ls -R /user/cloudera
```

Copy data

```
git clone https://github.com/greenore/e63
```

Unzip

```
tar xf e63/bible.tar
tar xf e63/shakespeare.tar
```

Copy file

```
# put the files into HDFS
hadoop fs -put /home/smith/all-bible
hadoop fs -put /home/smith/all-shakespeare

# List files in hadoop directory
hadoop fs -ls -R /user/smith
```

```
-rw-r--r--   1 smith mapred    5258688 2017-10-07 09:44 /user/smith/all-bible
-rw-r--r--   1 smith mapred    5284231 2017-10-07 09:44 /user/smith/all-shakes
```

Bible grep

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep \
/user/smith/all-bible /user/smith/bible-freq '\w+'

hadoop fs -cat /user/smith/bible/bible-freq/part-r-00000 | head -20
```

```
[smith@quickstart ~]$ hadoop fs -cat /user/smith/bible-freq/part-r-00000 |
head -20
62394 the
38985 and
34654 of
13526 to
12846 And
12603 that
12445 in
9764 shall
9672 he
8940 unto
8854 I
8385 his
8057 a
7270 for
6974 they
6913 be
6884 is
6649 him
6647 LORD
6591 not
```

Shakespear grep

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar grep \
/user/smith/shakespeare/all-shakespeare /user/smith/shakespeare/shake-freq '\w+'

hadoop fs -cat /user/smith/shakespeare/shake-freq/ part-r-00000 | head -20
```

```
[smith@quickstart ~]$ hadoop fs -cat /user/smith/shake-freq/part-r-00000 |
head -20
25578 the
23027 I
19654 and
17462 to
16444 of
13524 a
12697 you
11296 my
10699 in
8857 is
8851 that
8402 not
8033 me
8020 s
7800 And
7231 with
7165 it
6812 his
6753 be
6246 your
```

Problem 4 (20%)

Create your own version of **Hadoop grep** program using Spark. Compare your results with the results of Hadoop grep when applied to the texts of King James Bible, and all of **Shakespears** works, contained in files *bible.tar* and *shakespear.tar* respectively. Notice small differences between results obtained by your Spark program and Hadoop grep. Try to explain what causes those differences. Save results of your Spark grep operations both in HDFS and on your local file system. You can implement your solution using one of interactive shells or a standalone program.

Bible

Import libraries

```
# Import libraries
import findspark
findspark.init("/usr/local/spark")
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext, Row
from pyspark.sql.types import *
import re
from operator import add

# Start session
```

```
conf = SparkConf().setMaster("local").setAppName("rdd")
sc = SparkContext(conf = conf)
```

Import and split data

```
rdd_bible = sc.textFile("/home/smith/all-bible")
rdd_counts = rdd_bible.flatMap(lambda x:x.split(" ")).map(lambda x: (x,1)).reduceByKey(add)
print(rdd_counts.take(5))
```

```
[('', 605968), ('King', 60), ('James', 33), ('Bible', 4), ('<!--', 68)]
```

Exchange

```
rdd_counts2 = rdd_counts.map(lambda x: (x[1],x[0]))
print(rdd_counts2.take(5))
```

```
[(605968, ''), (60, 'King'), (33, 'James'), (4, 'Bible'), (68, '<!--')]
```

Sort

```
rdd_sorted = rdd_counts2.sortBy(lambda x: x[0], ascending=False)
print(rdd_sorted.take(20))
```

```
[(605968, ''), (62384, 'the'), (38711, 'and'), (34618, 'of'), (13505, 'to'),
(12735, 'And'), (12478, 'that'), (12279, 'in'), (9764, 'shall'), (9513, 'he'),
(8930, 'unto'), (8708, 'I'), (8362, 'his'), (8054, 'a'), (7183, 'for'), (6897,
'they'), (6754, 'be'), (6747, 'is'), (6047, 'with'), (5878, 'not')]
```

Shakespear

Import and split data

```
rdd_shake = sc.textFile("/home/smith/all-shakespeare")
rdd_counts = rdd_shake.flatMap(lambda x:x.split(" ")).map(lambda x:(x,1)).reduceByKey(add)
print(rdd_counts.take(5))
```

```
[('', 64531), ("\tALL'S", 25), ('WELL', 52), ('THAT', 27), ('ENDS', 26)]
```

Exchange

```
rdd_counts2 = rdd_counts.map(lambda x: (x[1],x[0]))
print(rdd_counts2.take(5))
```

```
[(64531, ''), (25, "\tALL'S"), (52, 'WELL'), (27, 'THAT'), (26, 'ENDS')]
```

Sort

```
rdd_sorted = rdd_counts2.sortBy(lambda x: x[0], ascending=False)
print(rdd_sorted.take(20))
```

```
[(64531, ''), (25069, 'the'), (18793, 'and'), (16436, 'to'), (16069, 'of'),
(15223, 'I'), (12982, 'a'), (11180, 'my'), (10134, 'in'), (9109, 'you'), (8109,
'is'), (7773, 'that'), (7123, 'not'), (7001, 'with'), (6594, 'his'), (6202, 'be'),
(6119, 'your'), (5955, '\tAnd'), (5781, 'for'), (5311, 'have')]
```

The two grep results, i.e., *hadoop grep* and my own appear to differ in there results. The reason for this is, that the *hadoop grep* appears to be removing blanks and is taking the lower() command.

Problem 5 (15%)

Create your own tables *KINGJAMES* with columns for words and frequencies and insert into the table the result of your Spark grep program which produces word counts in file bible. Find all words in table *KINGJAMES* which start with letter *w* and are 4 or more characters long and appear more than 250 times. Write a query that will tell us the number of such words. Before counting turn all words in lower case. 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()*.

Import libraries and set context

```
from pyspark import SparkConf, SparkContext, SQLContext
from pyspark.sql import SQLContext, Row
from pyspark.sql.types import *
import re
from operator import add

sc.setLogLevel("ERROR")
sqlContext = SQLContext(sc)
```

Cleanup

```
rdd_bible = sc.textFile("all-bible")
rdd_bible = rdd_bible.flatMap(clean_up)
bible_lines = sc.textFile("all-bible").flatMap(lambda l: l.split())\
                .map(lambda x: re.sub("[^a-zA-Z]+", "", x.lower()))\
                .filter(lambda x: x != "")
print(bible_lines.take(10))
```

```
['king', 'james', 'bible', 'body', 'backgroundfaebd', 'margin',
'textalignjustify', 'p', 'textindent', 'em']
```

Define rows

```
rdd_words = rdd_bible.map(lambda p: Row(words=str(p)))
print(rdd_words.take(10))
```

```
[Row(words='king'), Row(words='james'), Row(words='bible'), Row(words='\t\tbody'),
Row(words='backgroundfaebd7'), Row(words='margin10'), Row(words='textalignjustify'),
Row(words='p'), Row(words='textindent'), Row(words='1em')]
```

Create SQL context

```
df_bible = sqlContext.createDataFrame(rdd_words)
print(df_bible.show(10))
```

```
+----------------+
|           words|
+----------------+
|            king|
|           james|
|           bible|
|            body|
|backgroundfaebd7|
|        margin10|
|textalignjustify|
|               p|
|      textindent|
|             1em|
+----------------+
only showing top 10 rows

None
```

Get counts

```
df_bible.registerTempTable("KINGJAMES")
tbl_bible = sqlContext.sql("SELECT words, COUNT(*) freq FROM KINGJAMES WHERE lower(words) \
like 'w%' AND length(words) > 4  GROUP BY words HAVING COUNT(*) > 250")
tbl_bible = tbl_bible.orderBy(tbl_bible['freq'].desc())
print(tbl_bible.show())
```

```
+----------+----+
|     words|freq|
+----------+----+
|     which|4427|
|     words| 548|
|     would| 451|
|   without| 442|
|     where| 407|
|     water| 396|
|     woman| 357|
| wherefore| 348|
|    wicked| 344|
|     whose| 314|
|wilderness| 304|
|     works| 302|
|    waters| 287|
|     world| 287|
|   written| 283|
+----------+----+

None
```

Problem 6 (15%)

Transfer content of your Hive *KINGJAMES* table to a Spark DataFrame. Perform the analysis from problem 5 using any available API in Spark. Please note that you are working with Spark 1.6.

Hive query

The necessary python code can be found in the *problem_6.py* script. However, the code can also be directly run from the RMarkdown script.

```
hivecontext = HiveContext(sc)
df_bible_freq = hivecontext.sql("SELECT freq, lower(word) word FROM kingjames \
WHERE lower(word) like 'w%' AND length(word) > 4 AND freq > 250 ORDER BY freq \
DESC")
```

Frequency plot

```
print(df_bible_freq.show(20))
```

```
+----+----------+
|freq| word|
+----+----------+
|4297|     which|
| 546|     words|
| 443|     would|
| 436|   without|
| 396|     water|
| 355|     woman|
| 343|    wicked|
| 335|     where|
| 304|wilderness|
| 301|     works|
| 288|     world|
| 286|    waters|
| 284|     whose|
| 283|   written|
| 261| wherefore|
+----+----------+
```

Cumsum

```
print(df_bible_freq.show.agg({"freq": "sum"}).show())
```

```
+---------+
|sum(freq)|
+---------+
|     9158|
+---------+
```

Number of words

```
print("Number of words:")
print(df_bible_freq.count())
```

```
# Number of words in Query
15
```

Problem 7 (15%)

Use Sqoop to transfer the content of MySQL database *retail_db* which is present on the Cloudera VM into Hive. Demonstrate that new Hive tables are created and correspond to the original MySQL tables. Find the number of rows in each table. Compare those row counts with row counts in MySQL database.

MySQL tables

```
sudo mysql --user=retail_dba -p
```

Show tables

```
show databases;
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| retail_db          |
+--------------------+
2 rows in set (0.00 sec)
```

```
show databases;
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| retail_db          |
+--------------------+
2 rows in set (0.00 sec)
```

```
use retail_db;
```

```
mysql> use retail_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

```
show tables;
```

```
mysql> show tables;
+----------------------+
| Tables_in_retail_db |
+----------------------+
| categories           |
| customers            |
| departments          |
| order_items          |
| orders               |
| products             |
+----------------------+
6 rows in set (0.00 sec)
```

Sqoop transfer

Next we're importing the data into hive.

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

```
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
17/10/07 11:02:13 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.5.0
.....................
File Input Format Counters
Bytes Read=0
File Output Format Counters
Bytes Written=0
...
        Map output records=1345
        Input split bytes=87
        Spilled Records=0
        Failed Shuffles=0
        Merged Map outputs=0
        GC time elapsed (ms)=252
        CPU time spent (ms)=2980
        Physical memory (bytes) snapshot=181051392
        Virtual memory (bytes) snapshot=1522274304
        Total committed heap usage (bytes)=60620800
    File Input Format Counters
        Bytes Read=0
    File Output Format Counters
        Bytes Written=0
17/10/07 11:06:00 INFO mapreduce.ImportJobBase: Transferred 46.1318 KB in 41.3821 seconds (1.2983 KB/se
17/10/07 11:06:00 INFO mapreduce.ImportJobBase: Retrieved 1345 records.
```

The above output shows, that the data was successfuly expordet. In order to check the results we can have a look at the tables in hive.

Hive

```
hive
```

```
Logging initialized using configuration in file:/etc/hive/conf.dist/hivelog4j.
properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
```

```
SHOW TABLES;
```

```
hive> SHOW TABLES;
OK
apachelog
categories
customers
```

```
departments
kingjames
merged
order_items
orders
products
```

Next we're looking an the order

```
SELECT order_item_order_id, sum(order_item_quantity) AS total_order_item_quantity
FROM order_items
GROUP BY order_item_order_id
ORDER BY total_order_item_quantity DESC LIMIT 10;
```

```
hive> SELECT order_item_order_id, sum(order_item_quantity) AS total_order_item_quantity
FROM order_items GROUP BY order_item_order_id ORDER BY total_order_item_quantity
DESC LIMIT 10;
Query ID = cloudera_20171007181414_a8ca115b-c6a1-549f-bd26-e043b5f80ede
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
.....................
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 5.97 sec HDFS Read: 513123
HDFS Write: 1220606 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 4.33 sec HDFS Read:
1225407 HDFS Write: 90 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 300 msec
OK
49405 24
49429 24
21920 24
24967 24
33283 23
43848 23
29272 22
56414 22
17455 22
56559 22
Time taken: 59.2 seconds, Fetched: 10 row(s)
```

Comparisons

In order to run the comparisons the necessary sql and hive code can be found in the *problem_7.sql* and *problem_7.q* respectivly. However, The code can also be run from the RMarkdown file.

Categories

```
SELECT count(*) FROM categories;
```

```
mysql> SELECT count(*) FROM categories;
+----------+
| count(*) |
```

```
+----------+
| 58 |
+----------+
1 row in set (0.00 sec)
```

```
jdbc:hive2://127.0.0.1:10000/default> select count(*) from
categories;
```

```
0: jdbc:hive2://127.0.0.1:10000/default> select count(*) from
categories;
+------+--+
| _c0 |
+------+--+
| 58 |
+------+--+
1 row selected (28.531 seconds)
```

Departments

```
SELECT count(*) FROM departments;
```

```
mysql> SELECT count(*) FROM departments;
+----------+
| count(*) |
+----------+
| 6 |
+----------+
1 row in set (0.00 sec)
```

```
jdbc:hive2://127.0.0.1:10000/default> select count(*) from
departments;
```

```
0: jdbc:hive2://127.0.0.1:10000/default> select count(*) from
departments;
+------+--+
| _c0 |
+------+--+
| 6 |
+------+--+
1 row selected (21.434 seconds)
```

Order Items

```
SELECT count(*) FROM order_items;
```

```
mysql> SELECT count(*) FROM order_items;
+----------+
| count(*) |
+----------+
| 172198 |
+----------+
1 row in set (0.05 sec)
```

```
jdbc:hive2://127.0.0.1:10000/default> select count(*) from
order_items;
```

```
0: jdbc:hive2://127.0.0.1:10000/default> select count(*) from
order_items;
+---------+--+
| _c0 |
+---------+--+
| 172198 |
+---------+--+
1 row selected (27.422 seconds)
```

Orders

```
SELECT count(*) FROM orders;
```

```
mysql> SELECT count(*) FROM orders;
+----------+
| count(*) |
+----------+
| 68883 |
+----------+
1 row in set (0.04 sec)
```

```
jdbc:hive2://127.0.0.1:10000/default> select count(*) from
orders;
```

```
0: jdbc:hive2://127.0.0.1:10000/default> select count(*) from
orders;
+--------+--+
| _c0 |
+--------+--+
| 68883 |
+--------+--+
1 row selected (30.201 seconds)
```

Products

```
SELECT count(*) FROM products;
```

```
mysql> SELECT count(*) FROM products;
+----------+
| count(*) |
+----------+
| 1345 |
+----------+
1 row in set (0.00 sec)
```

```
jdbc:hive2://127.0.0.1:10000/default> select count(*) from
products;
```

```
0: jdbc:hive2://127.0.0.1:10000/default> select count(*) from
products;
+-------+--+
| _c0 |
+-------+--+
| 1345 |
+-------+--+
1 row selected (27.983 seconds)
```

As can be seen above, the table between hive and MySQL are the same.