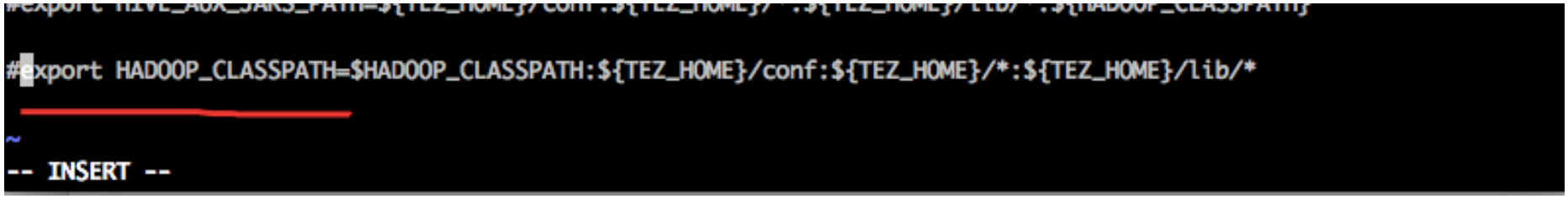
Table of Contents

***Note* in this particular version, once migrated to tez, it cant use mapreduce further.**

Hive Commands – Load and Select - 90 Minutes.

Goals: You will learn how to create database, tables and load data in it. Finally, you will be able to execute select statement to fetch data from the hive table.

If you have configured Tez then comment the following, which will be the last line else skip this instruction.
#vi $HIVE_HOME/conf/hive-env.sh



Ensure that Hadoop cluster and Hive services are up before running the Hive examples.

Start Hive CLI using the following commands.

#beeline -u jdbc:hive2://hadoop0:10000

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

hive> CREATE TABLE invites (foo INT, bar STRING) STORED AS TEXTFILE;

```
0: jdbc:hive2://hadoop0:10000> CREATE TABLE invites (foo INT, bar STRING) STORED AS TEXTFILE;
INFO  : Compiling command(queryId=root_20210416075701_0127e95b-ce96-484c-a303-df914155af07): CREATE TABLE invites (f
oo INT, bar STRING) STORED AS TEXTFILE
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=root_20210416075701_0127e95b-ce96-484c-a303-df914155af07); Time taken: 0
.048 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20210416075701_0127e95b-ce96-484c-a303-df914155af07): CREATE TABLE invites (f
oo INT, bar STRING) STORED AS TEXTFILE
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=root_20210416075701_0127e95b-ce96-484c-a303-df914155af07); Time taken: 0
.19 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (0.265 seconds)
```

Display the tables created in your databases.

hive> SHOW TABLES;

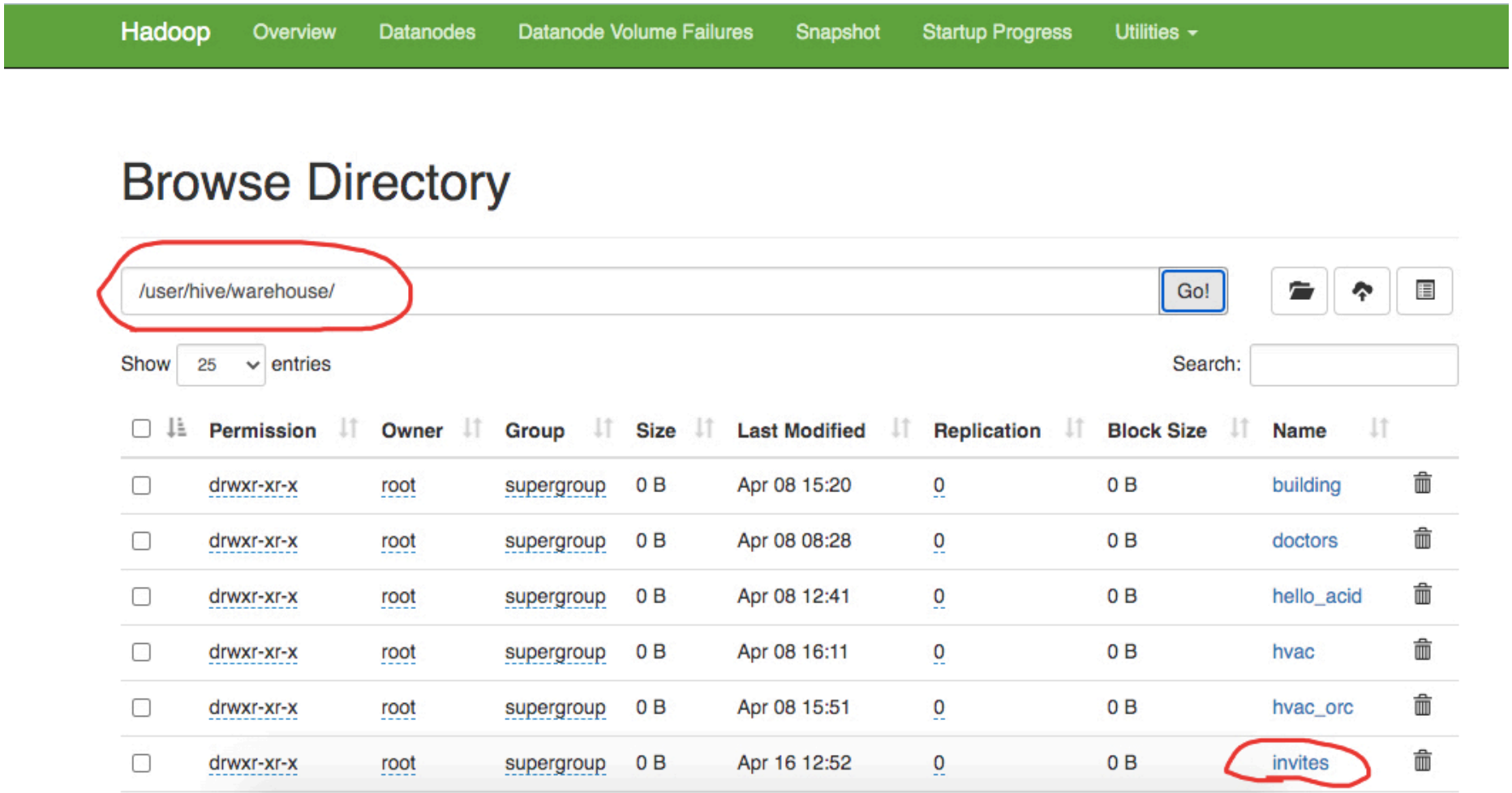
```
hive>
    >
    >
    > show tables;
OK
invites
Time taken: 0.41 seconds
hive>
```

Get the definition of the table.

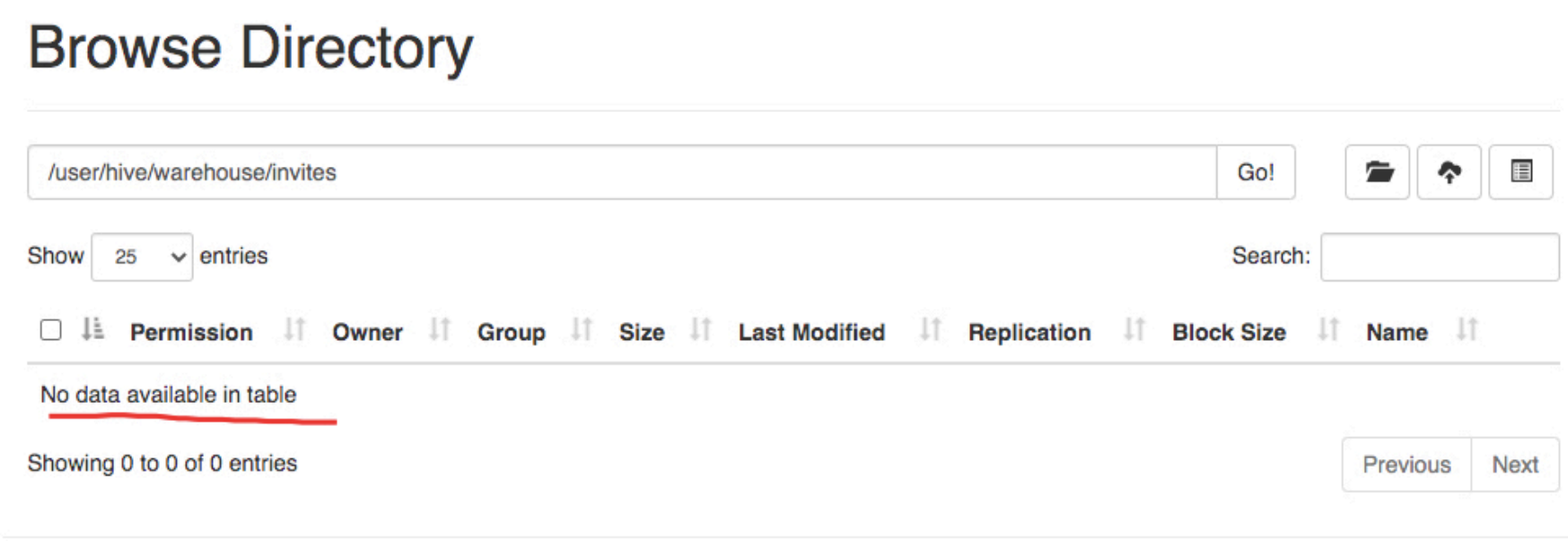
hive> DESCRIBE invites;

```
INFO  : Concurrency mode is disabled, not creating a lock manager
+-----------+------------+----------+
| col_name  | data_type  | comment  |
+-----------+------------+----------+
| foo       | int        |          |
| bar       | string     |          |
+-----------+------------+----------+
2 rows selected (0.325 seconds)
0: jdbc:hive2://hadoop0:10000>
```

It will create a folder by the Table name i.e ***invites*** inside the Hive ware house folder. You can view from the HDFS utilities as shown below.



Browse the folder.

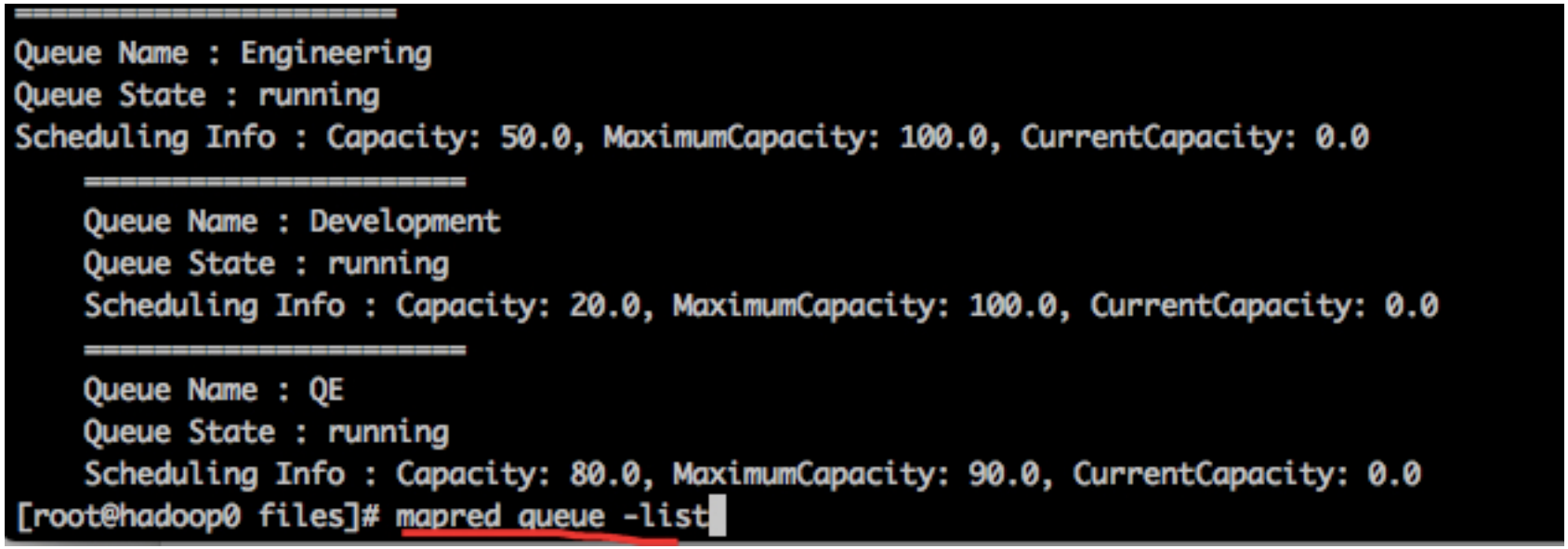


There is nothing inside the folder.

Set a specific Queue for the Hive if you have configured Queue else skip.
#set mapred.job.queue.name=QE;

Note:

You can determine the queue configured using the following command.

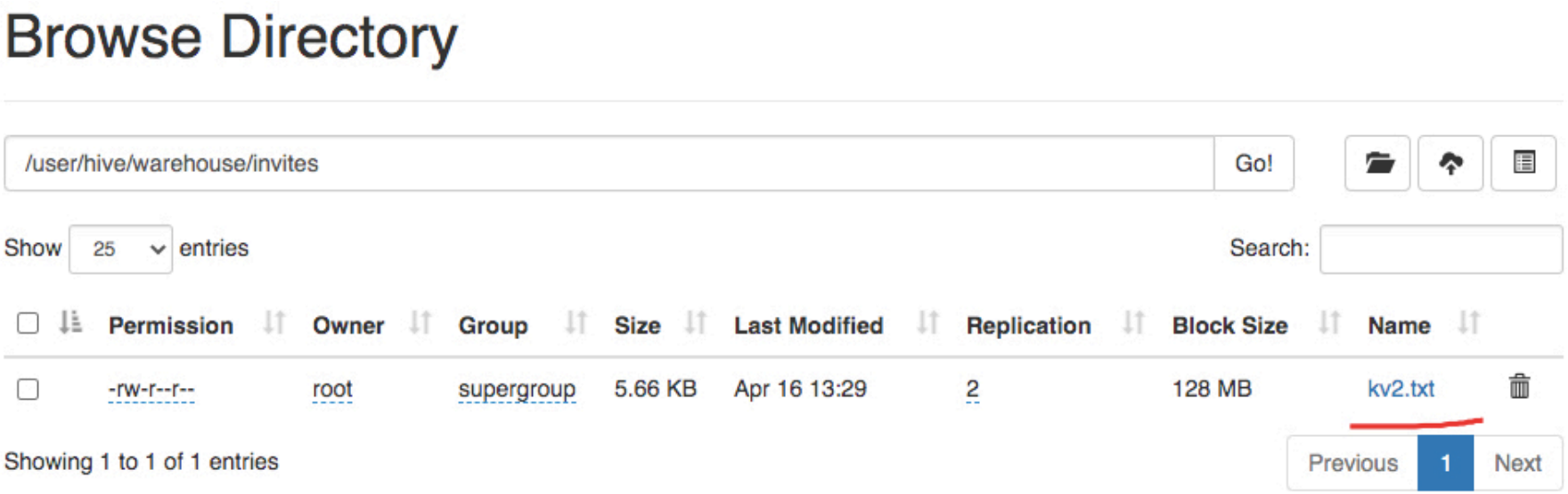


Load the data from the Local system to the Hive cluster table. Execute the following.

hive> LOAD DATA LOCAL INPATH
'/opt/hive/examples/files/kv2.txt' OVERWRITE INTO TABLE invites ;

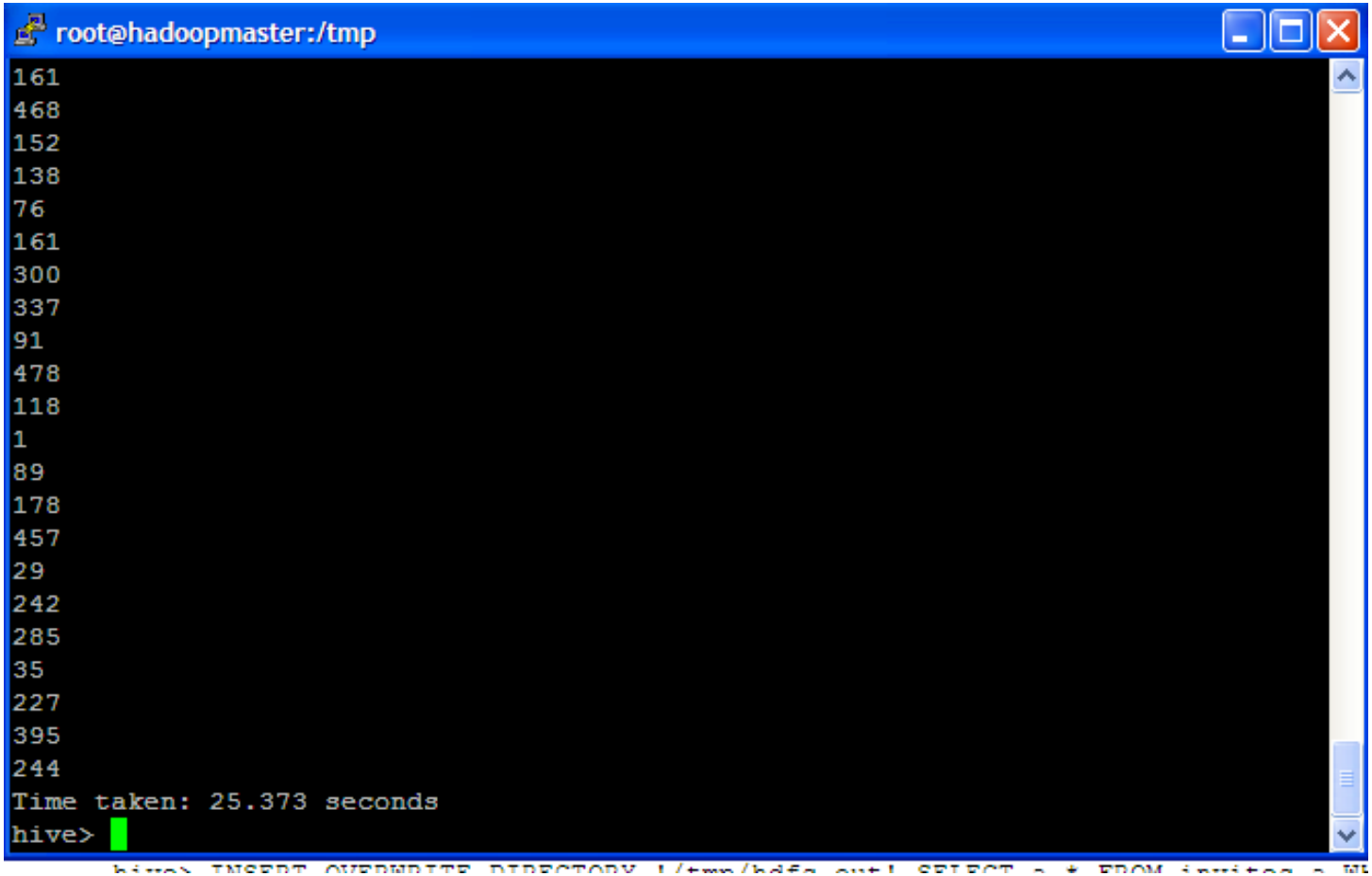
```
INFO  : Executing command(queryId=root_2[redacted]5938bd8e-2de1-416b-a938-e8134d17c200): LOAD DATA LOCAL INPATH
'/opt/hive/examples/files/kv2.txt' OVERWRITE INTO TABLE invites
INFO  : Starting task [Stage-0:MOVE] in serial mode
INFO  : Loading data to table default.invites from file:/opt/hive/examples/files/kv2.txt
INFO  : Starting task [Stage-1:STATS] in serial mode
INFO  : Completed executing command(queryId=root_20210416075901_5938bd8e-2de1-416b-a938-e8134d17c200); Time taken: 0
.49 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (0.645 seconds)
0: jdbc:hive2://hadoop0:10000>
```

You can see files under the invites folder.



Fetch record from the table with where condition.

hive> SELECT a.foo FROM invites a;



Extract the data from the Hive table and store in a HDFS directory.

hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a;

(Verify the file in the hdfs file system)

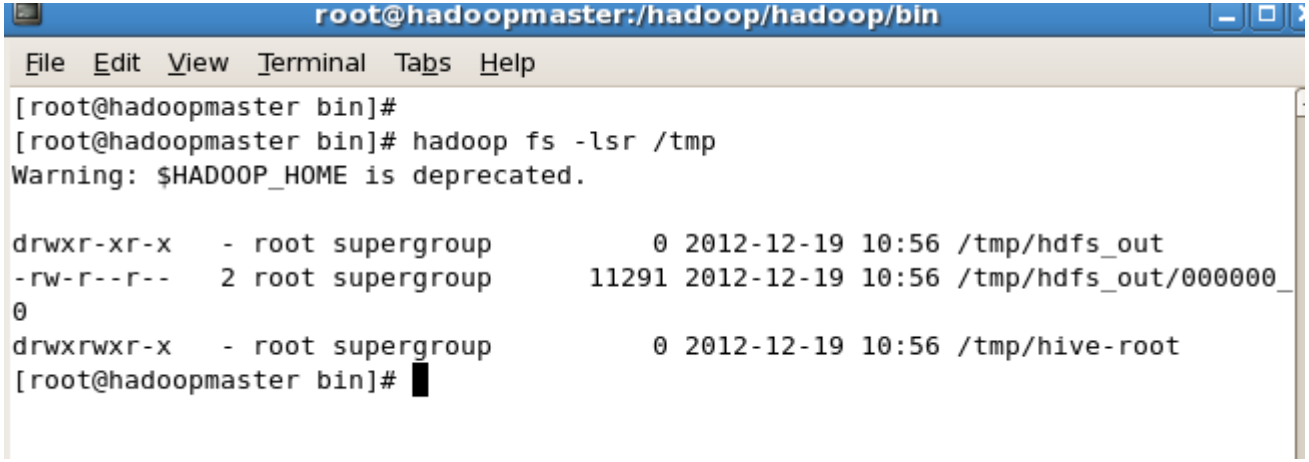
```
hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE
 a.ds='2008-08-15';
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201212191030_0002, Tracking URL = http://master:50030/jobdeta
ils.jsp?jobid=job_201212191030_0002
Kill Command = /hadoop/hadoop/libexec/../bin/hadoop job  -Dmapred.job.tracker=ma
ster:9001 -kill job_201212191030_0002
```

You can verify the job status using yarn command too:

```
#yarn application -list
```

```
[hdfs@tos ~]$ yarn application -list
19/06/16 15:05:07 INFO client.RMProxy: Connecting to ResourceManager at tos.master.com/10.10.20.20:8050
19/06/16 15:05:08 INFO client.AHSProxy: Connecting to Application History server at tos.master.com/10.10.20.20:10200
Total number of applications (application-types: [], states: [SUBMITTED, ACCEPTED, RUNNING] and tags: []):1
                Application-Id      Application-Name        Application-Type          User           Queue                   State
      Final-State              Progress                        Tracking-URL
application_1560671604408_0003  HIVE-642eec72-46f6-4bba-a23e-7c1e88a923e7                      TEZ          hdfs         default
          RUNNING               UNDEFINED                   0%     http://tos.master.com:35974/ui/
[hdfs@tos ~]$
```

If you want to kill the job : yarn application -kill application_1560671604408_0003



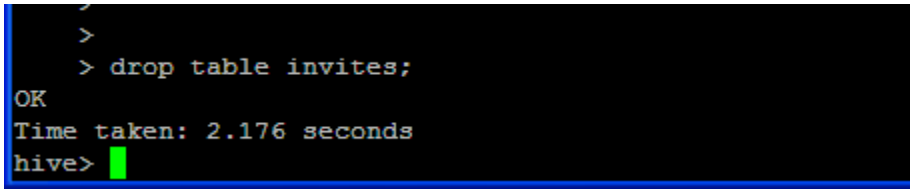
You can even execute the hadoop command from hive cli [dfs -ls -R /tmp/hdfs_out;]

#hdf dfs -cat /tmp/hdfs_out/000000_0

```
478□val_479□2008-08-15
118□val_119□2008-08-15
1□val_2□2008-08-15
89□val_90□2008-08-15
178□val_179□2008-08-15
457□val_458□2008-08-15
29□val_30□2008-08-15
242□val_243□2008-08-15
285□val_286□2008-08-15
35□val_36□2008-08-15
227□val_228□2008-08-15
395□val_396□2008-08-15
244□val_245□2008-08-15
[root@hadoopmaster bin]#
```

Dropping tables:

hive> DROP TABLE invites;



Multiples Queries Execution:

Create a database Learning

#create database Learning;

A Database, learning.db will be created in /opt/data/hive_local/ folder

Create a table and Load Data in it
#CREATE TABLE learning.pokes (foo INT, bar STRING);

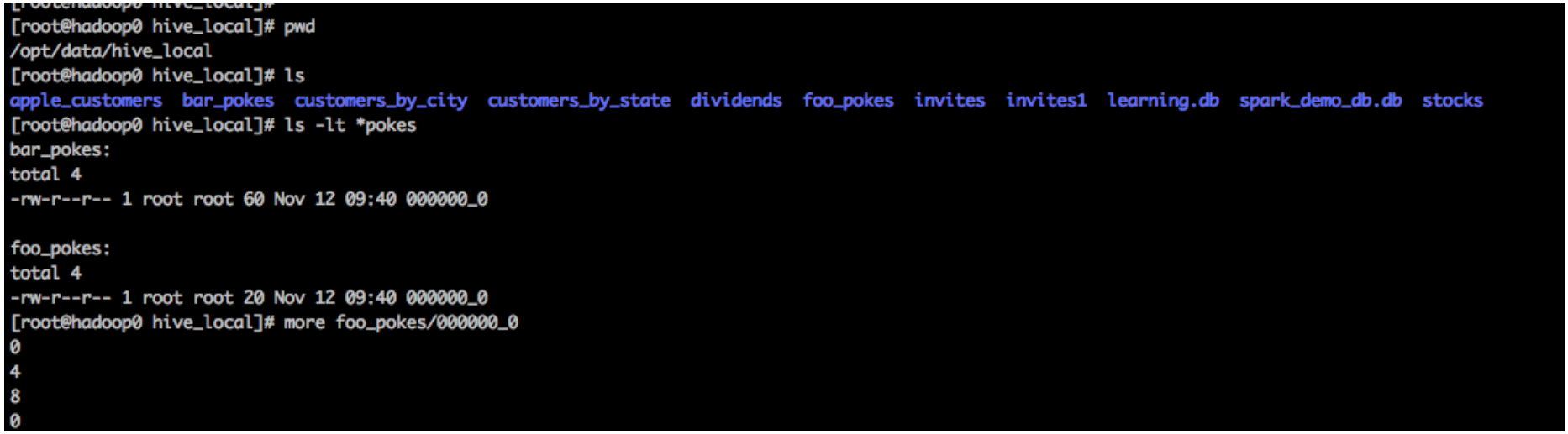
Loading data from flat files into Hive:

LOAD DATA LOCAL INPATH '/opt/hive_local/examples/files/kv1.txt' OVERWRITE INTO TABLE learning.pokes;

If **we sometimes need to extract data to multiple files, you can do with the following command.**

#FROM learning.pokes p
INSERT OVERWRITE DIRECTORY '/opt/data/hive_local/foo_pokes' SELECT foo WHERE foo < 10
INSERT OVERWRITE DIRECTORY '/opt/data/hive_local/bar_pokes' SELECT bar WHERE foo < 10;

You can verify the record with the following commands. It will create two folders, foo_pokes and bar_pokes respectively.



Exit the beeline:

> !q

Errata:

Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: Principal [name=hdfs, type=USER] does not have following privileges for operation LOAD [ADMIN] (state=42000,code=40000)

Solutions:

Go to Ambari > Hive > CONFIGS > ADVANCED > Custom hive-site and add hive.users.in.admin.role to the list of comma-separated users who require admin role authorization (such as the user hive). Restart the Hive services for the changes to take effect.

hive.users.in.admin.role = hdfs

Errata:

Caused by: org.apache.hadoop.ipc.RemoteException: Server IPC version 9 cannot communicate with client version 4
at org.apache.hadoop.ipc.Client.call(Client.java:1070)
java.lang.RuntimeException: Failed to create DataStorage
at
org.apache.pig.backend.hadoop.datastorage.HDataStorage.init(HDataSto
r age.java:75)
Answer:-- Verify that hadoop home environment is set properly as show
export HADOOP_HOME=/YARN/hadoop-2.6.0

Include all the global environment in the logon scripts: **/etc/profile**
// Error starting hive
[ERROR] Terminal initialization failed; falling back to unsupported
java.lang.IncompatibleClassChangeError: Found class jline.Terminal, but
interfac e was expected
at jline.TerminalFactory.create(TerminalFactory.java:101)

Exception in thread "main" java.lang.IncompatibleClassChangeError: Found class j
line.Terminal, but interface was expected

at jline.console.ConsoleReader.<init>(ConsoleReader.java:230)

/// remedy: remove the unwanted jar

Issue Map reduce jobs not proceeding ahead and stuck at Accepted state.
Solution : verify the yarn-site.xml file and execute yarn services with yarn user only

---------- Lab Ends Here ----------

Using Avro & ORC - 30 Minutes

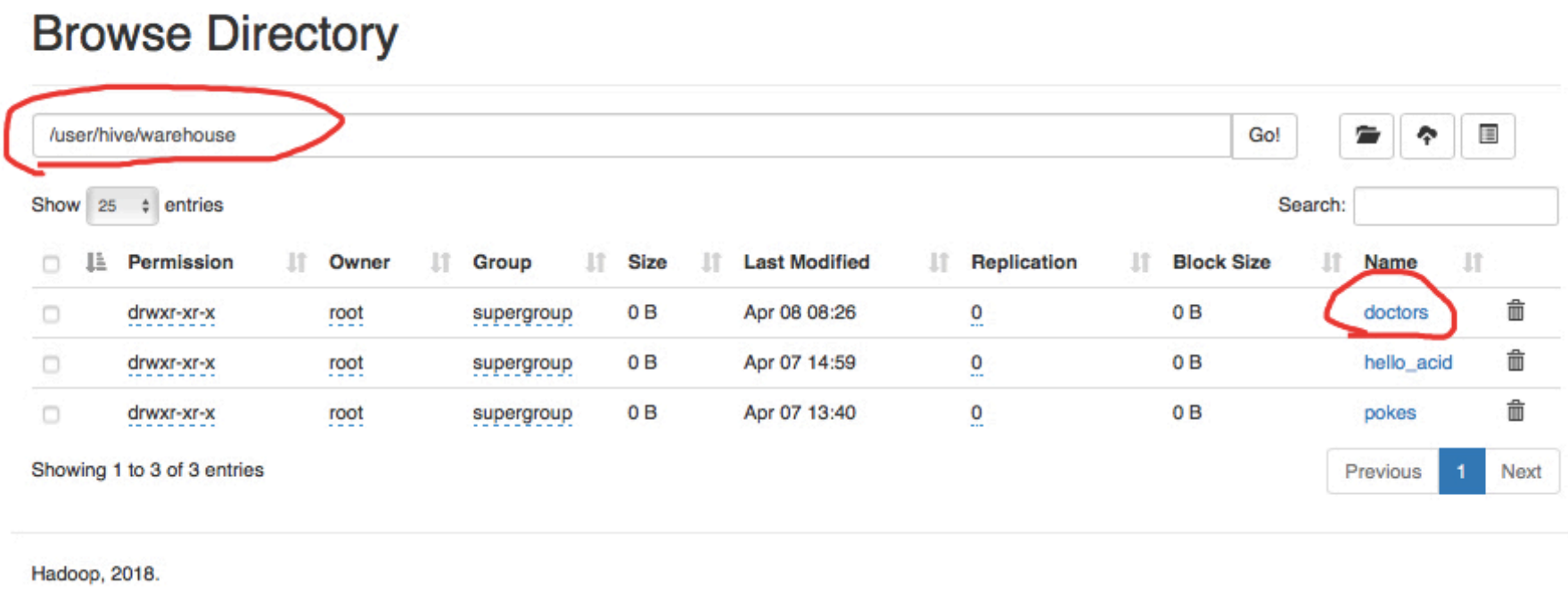
Using Avro Data Files in Hive

The following example demonstrates how to create a Hive table backed by Avro data files:

From a beeline CLI, execute the following command. It will create a table doctors with avro format.

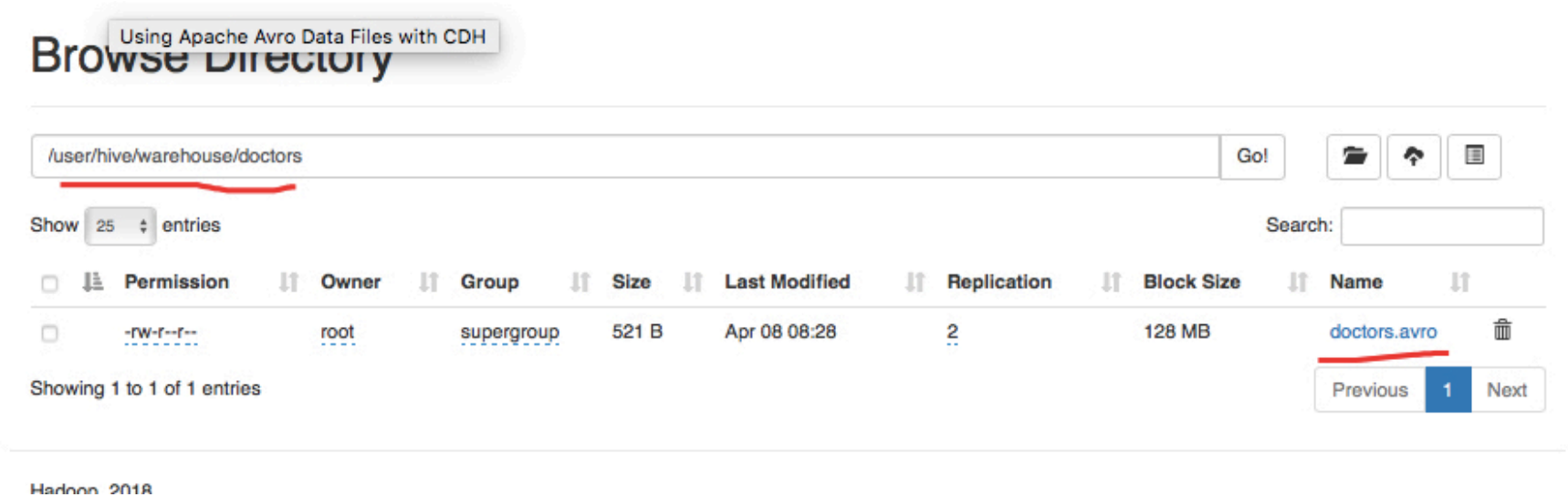
```
CREATE TABLE doctors
ROW FORMAT
SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
STORED AS
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
TBLPROPERTIES ('avro.schema.literal'='{
  "namespace": "testing.hive.avro.serde",
  "name": "doctors",
  "type": "record",
  "fields": [
    {
      "name":"number",
      "type":"int",
      "doc":"Order of playing the role"
    },
    {
      "name":"first_name",
      "type":"string",
```

```
    "doc":"first name of actor playing role"
   },
   {
    "name":"last_name",
    "type":"string",
    "doc":"last name of actor playing role"
   },
   {
    "name":"extra_field",
    "type":"string",
    "doc:":"an extra field not in the original file",
    "default":"fishfingers and custard"
   }
  ]
}');
```



Load the data to the above table.

#LOAD DATA LOCAL INPATH '/opt/hive/examples/files/doctors.avro' INTO TABLE doctors;



Verify the data using the HDFS browser as shown above and using CLI below.

On the beeline CLI.
select * from doctors;

```
+-----------------+---------------------+--------------------+--------------------------+
| doctors.number  | doctors.first_name  | doctors.last_name  |   doctors.extra_field    |
+-----------------+---------------------+--------------------+--------------------------+
| 6               | Colin               | Baker              | fishfingers and custard  |
| 3               | Jon                 | Pertwee            | fishfingers and custard  |
| 4               | Tom                 | Baker              | fishfingers and custard  |
| 5               | Peter               | Davison            | fishfingers and custard  |
| 11              | Matt                | Smith              | fishfingers and custard  |
| 1               | William             | Hartnell           | fishfingers and custard  |
| 7               | Sylvester           | McCoy              | fishfingers and custard  |
| 8               | Paul                | McGann             | fishfingers and custard  |
| 2               | Patrick             | Troughton          | fishfingers and custard  |
| 9               | Christopher         | Eccleston          | fishfingers and custard  |
| 10              | David               | Tennant            | fishfingers and custard  |
+-----------------+---------------------+--------------------+--------------------------+
11 rows selected (0.553 seconds)
0: jdbc:hive2://hadoop0:10000> select * from doctors;
```

Using ORC.

To create a table named PARQUET_TABLE that uses the Parquet format, use a command like the following, substituting your own table name, column names, and data types:

Execute all the command on the Beeline CLI.

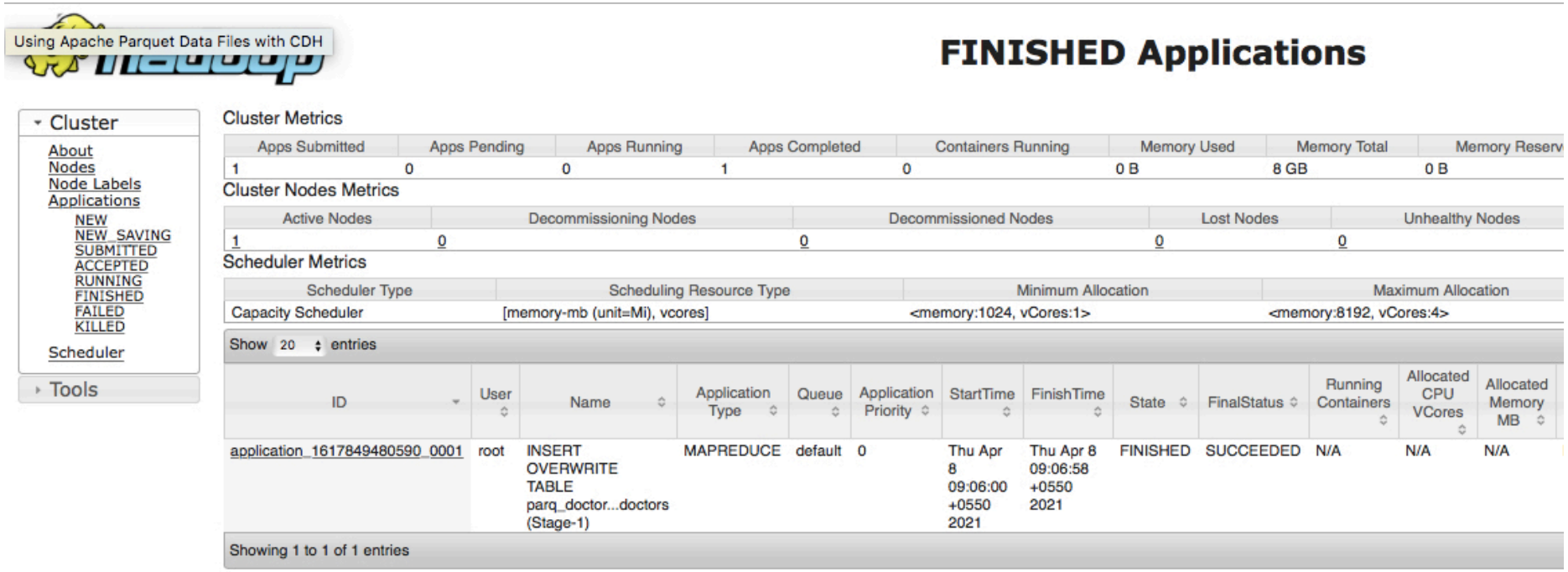
```
CREATE TABLE parq_doctor (number INT, first_name STRING, last_name STRING,extra_field STRING) STORED AS PARQUET;
```

To set the compression type to use when writing data, configure the parquet.compression property:

```
set parquet.compression=GZIP;
INSERT OVERWRITE TABLE parq_doctor SELECT * FROM doctors;
```

```
INFO  : MapReduce Total cumulative CPU time: 13 seconds 880 msec
INFO  : Ended Job = job_1617849480590_0001
INFO  : Starting task [Stage-7:CONDITIONAL] in serial mode
INFO  : Stage-4 is selected by condition resolver.
INFO  : Stage-3 is filtered out by condition resolver.
INFO  : Stage-5 is filtered out by condition resolver.
INFO  : Starting task [Stage-4:MOVE] in serial mode
INFO  : Moving data to directory hdfs://hadoop0:8020/user/hive/warehouse/parq_doctor/.hive-staging_hive_2021-0
4-08_03-35-53_941_5781618877110864858-2/-ext-10000 from hdfs://hadoop0:8020/user/hive/warehouse/parq_doctor/.h
ive-staging_hive_2021-04-08_03-35-53_941_5781618877110864858-2/-ext-10002
INFO  : Starting task [Stage-0:MOVE] in serial mode
INFO  : Loading data to table default.parq_doctor from hdfs://hadoop0:8020/user/hive/warehouse/parq_doctor/.hi
ve-staging_hive_2021-04-08_03-35-53_941_5781618877110864858-2/-ext-10000
INFO  : Starting task [Stage-2:STATS] in serial mode
INFO  : MapReduce Jobs Launched:
INFO  : Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 13.88 sec   HDFS Read: 19642 HDFS Write: 1694 SUCCE
SS
INFO  : Total MapReduce CPU Time Spent: 13 seconds 880 msec
INFO  : Completed executing command(queryId=root_20210408033553_563e8cf6-83ed-4a4c-ae30-1b88a7cce016); Time ta
ken: 66.271 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (67.693 seconds)
0: jdbc:hive2://hadoop0:10000>
```

Verify the application using the RM UI.



Verify the records.

#select * from parq_doctor;

```
INFO  : Concurrency mode is disabled, not creating a lock manager
+---------------------+-------------------------+-------------------------+--------------------------+
| parq_doctor.number  | parq_doctor.first_name  | parq_doctor.last_name   | parq_doctor.extra_field  |
+---------------------+-------------------------+-------------------------+--------------------------+
| 6                   | Colin                   | Baker                   | fishfingers and custard  |
| 3                   | Jon                     | Pertwee                 | fishfingers and custard  |
| 4                   | Tom                     | Baker                   | fishfingers and custard  |
| 5                   | Peter                   | Davison                 | fishfingers and custard  |
| 11                  | Matt                    | Smith                   | fishfingers and custard  |
| 1                   | William                 | Hartnell                | fishfingers and custard  |
| 7                   | Sylvester               | McCoy                   | fishfingers and custard  |
| 8                   | Paul                    | McGann                  | fishfingers and custard  |
| 2                   | Patrick                 | Troughton               | fishfingers and custard  |
| 9                   | Christopher             | Eccleston               | fishfingers and custard  |
| 10                  | David                   | Tennant                 | fishfingers and custard  |
+---------------------+-------------------------+-------------------------+--------------------------+
11 rows selected (2.264 seconds)
0: jdbc:hive2://hadoop0:10000> describe parq_doctor;
```

Congrats! You have successfully created hive table with AVRO and Parquet file.

---------- lab Ends Here----------

Hive DML – Using Partition – 60 Minutes

Goals: You will be able to perform various data fetching operations available in the HQL and store data using bucket and partition capabilities.

Understand the following at the end of the lab:

- Partition and Bucket
- Structure data type in Hive table.

Create table from the Hive console.

```
#CREATE TABLE invitespart (foo INT, bar STRING) PARTITIONED BY (ds STRING)
CLUSTERED BY (foo) INTO 9 BUCKETS STORED AS TEXTFILE;
```

The above created a table with partition by ds field and having 9 buckets.

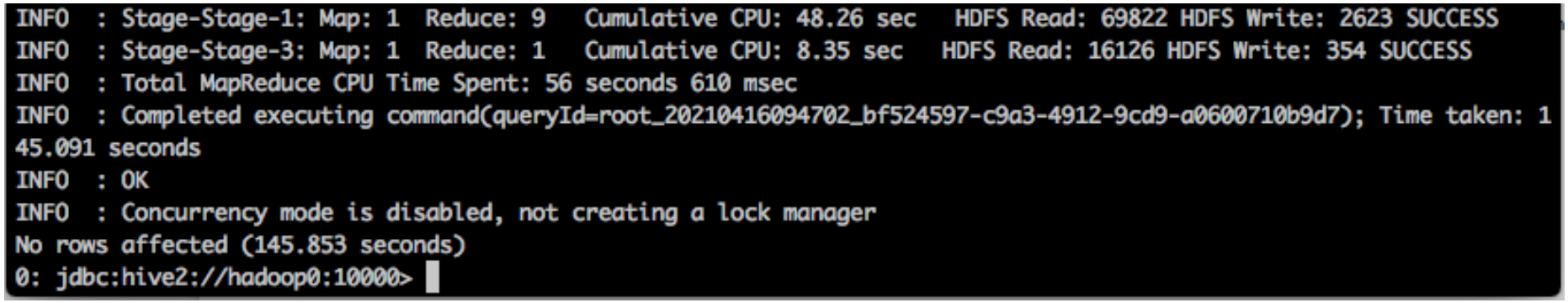
Specify the Queue if any.

```
#set mapred.job.queue.name=QE;
```

#LOAD DATA LOCAL INPATH '/opt/hive/examples/files/kv2.txt' OVERWRITE INTO TABLE invitespart PARTITION (ds='2008-08-15');

#LOAD DATA LOCAL INPATH '/opt/hive/examples/files/kv3.txt' OVERWRITE INTO TABLE invitespart 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.

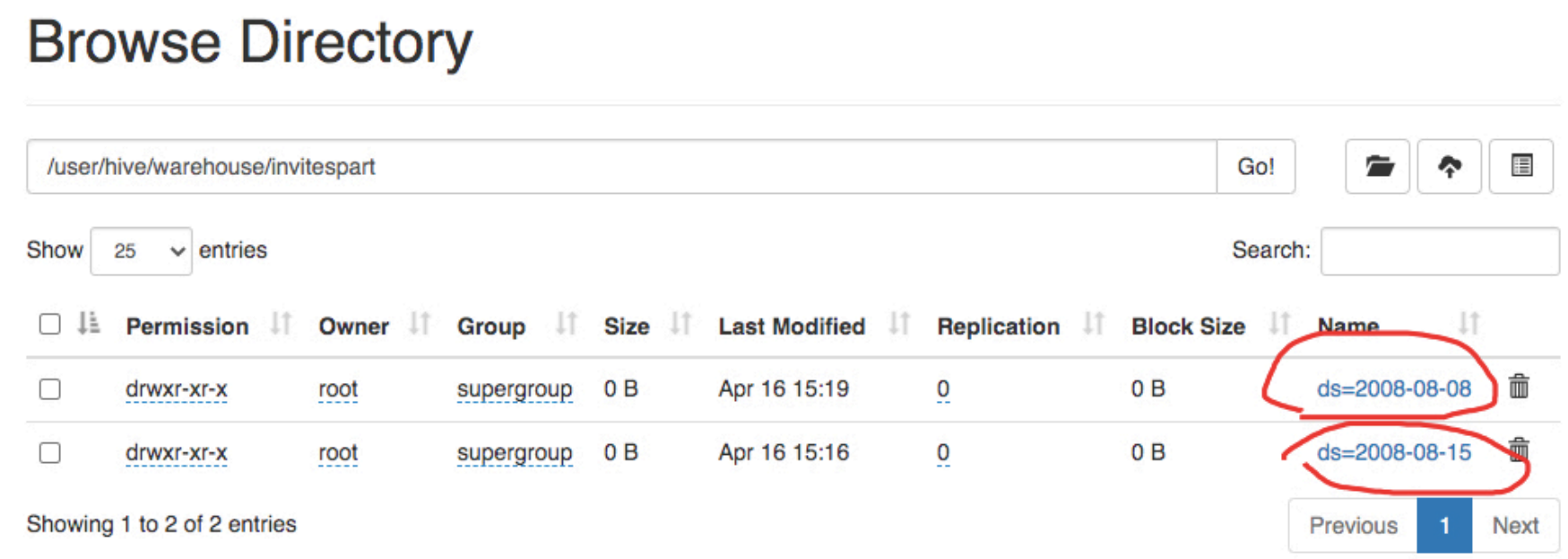


Using the Web UI console, you can browse the hive warehouse folder to view the invitespart table.

Show 25 entries Search:

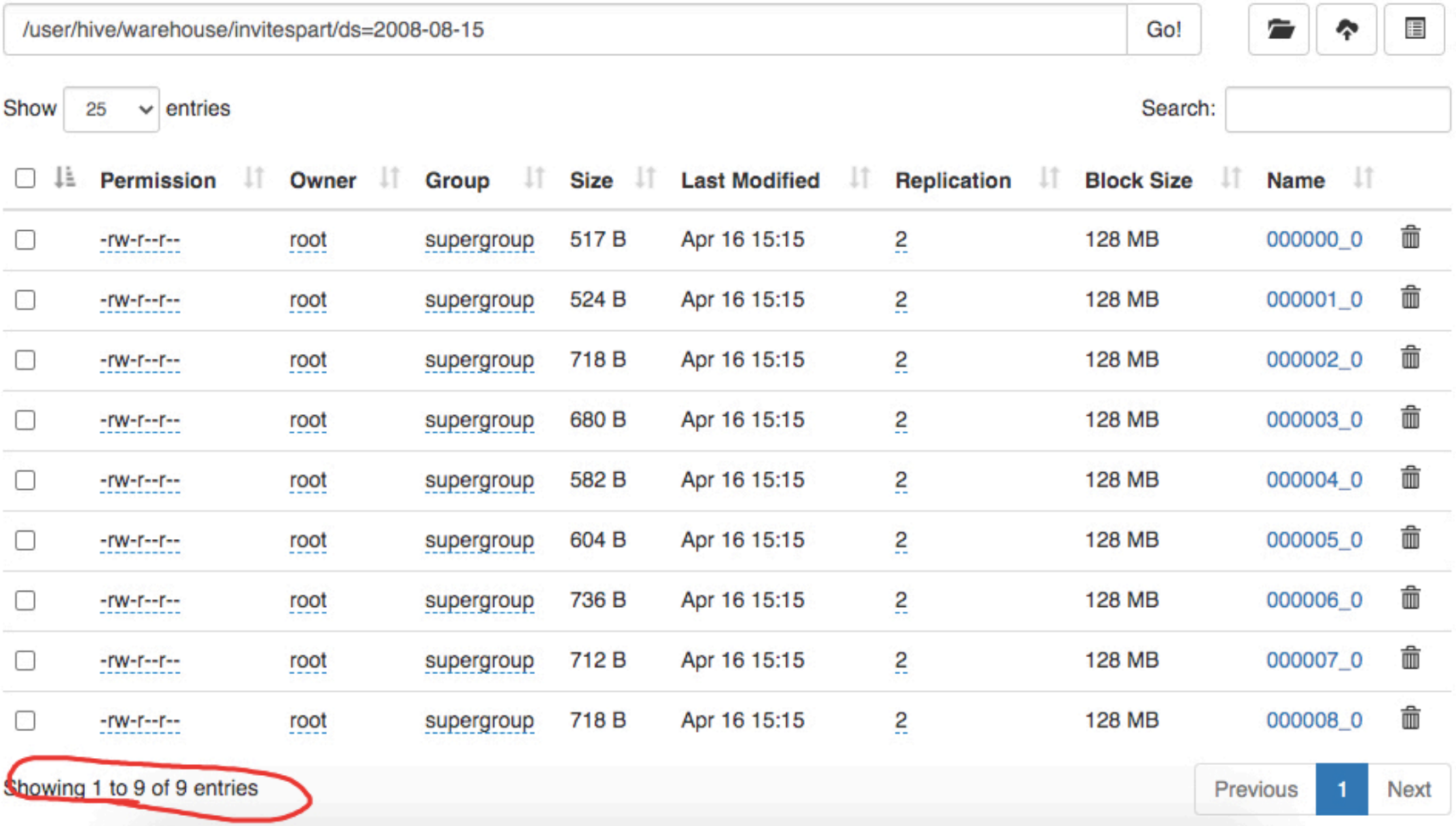
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	root	supergroup	0 B	Apr 08 15:20	0	0 B	building
drwxr-xr-x	root	supergroup	0 B	Apr 08 08:28	0	0 B	doctors
drwxr-xr-x	root	supergroup	0 B	Apr 08 12:41	0	0 B	hello_acid
drwxr-xr-x	root	supergroup	0 B	Apr 16 15:09	0	0 B	henry.db
drwxr-xr-x	root	supergroup	0 B	Apr 08 16:11	0	0 B	hvac
drwxr-xr-x	root	supergroup	0 B	Apr 08 15:51	0	0 B	hvac_orc
drwxr-xr-x	root	supergroup	0 B	Apr 16 15:17	0	0 B	invitespart
drwxr-xr-x	root	supergroup	0 B	Apr 16 12:51	0	0 B	myhive.db
drwxr-xr-x	root	supergroup	0 B	Apr 08 09:07	0	0 B	parq_doctor
drwxr-xr-x	root	supergroup	0 B	Apr 07 13:40	0	0 B	pokes

Click on invitespart



You have two sub directories by the partition value. – This is the Partition folders.

Click on one of the date i.e ds=2008-08-15.



You have 9 entries. Why? Hints (Buckets number specified in the table creation)

Or You can also verify using hadoop command:

hdfs dfs -ls /user/hive/warehouse/invitespart/ds=2008-08-15

```
[root@hadoop0 files]# hdfs dfs -ls /user/hive/warehouse/invitespart/ds=2008-08-15
2021-04-16 09:59:35,642 INFO Configuration.deprecation: No unit for dfs.client.datanode-restart.timeout(30) assuming
 SECONDS
Found 9 items
-rw-r--r--   2 root supergroup        517 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000000_0
-rw-r--r--   2 root supergroup        524 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000001_0
-rw-r--r--   2 root supergroup        718 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000002_0
-rw-r--r--   2 root supergroup        680 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000003_0
-rw-r--r--   2 root supergroup        582 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000004_0
-rw-r--r--   2 root supergroup        604 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000005_0
-rw-r--r--   2 root supergroup        736 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000006_0
-rw-r--r--   2 root supergroup        712 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000007_0
-rw-r--r--   2 root supergroup        718 2021-04-16 09:45 /user/hive/warehouse/invitespart/ds=2008-08-15/000008_0
[root@hadoop0 files]#
```

Execute a select query;
select * from invitespart limit 2;

You can even execute query using the partition

#select * from invitespart where ds='2008-08-15' limit 5;

Let us create a database and import some records in a hive table to understand various features provided by hive for data transformation.

Execute the following commands in the hive console.

#create database henry;

>use henry;

-- The Employees table that has simple fields, like name and salary,
-- but also complex fields, including an array of subordinates' names,
-- A map of names of deductions and the percentage amount to be
-- deducted at each pay period, and a struct containing the employee's
-- address.
-- Note that for the complex data types, Java-style generic type
-- arguments are used.
-- Next, if you know the table doesn't already exist, you can drop
-- the clause IF NOT EXISTS.
-- Finally, everything starting at "ROW FORMAT DELIMITED..." is OPTIONAL;
-- it's just regurgitating the default settings!

```
#CREATE TABLE IF NOT EXISTS employees (
  name      STRING,
  salary    FLOAT,subordinates ARRAY<STRING>, deductions      MAP<STRING, FLOAT>,
  address   STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
)
ROW FORMAT
DELIMITED FIELDS
TERMINATED BY '\001'
COLLECTION ITEMS TERMINATED
BY '\002' MAP KEYS TERMINATED
BY '\003'
LINES TERMINATED
BY '\n' STORED AS
TEXTFILE;

#SHOW TABLES;
```

```
hive> CREATE TABLE IF NOT EXISTS employees (
    >   name         STRING,
    >   salary       FLOAT,
    >   subordinates ARRAY<STRING>,
    >   deductions   MAP<STRING, FLOAT>,
    >   address      STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY '\001'
    > COLLECTION ITEMS TERMINATED BY '\002'
    > MAP KEYS TERMINATED BY '\003'
    > LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE;
OK
Time taken: 0.626 seconds
hive> SHOW TABLES;
OK
employees
Time taken: 0.34 seconds, Fetched: 1 row(s)
hive>
```

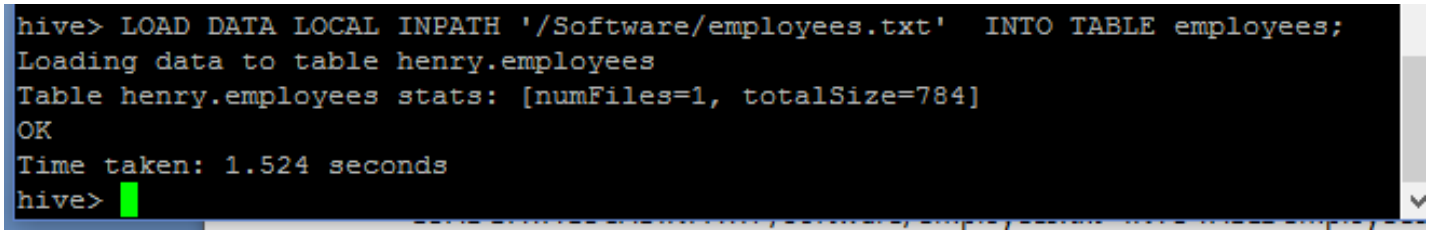
-- Now let's load data into this table, from a LOCAL directory.

-- First, confirm the directory exists and contains at least one file:

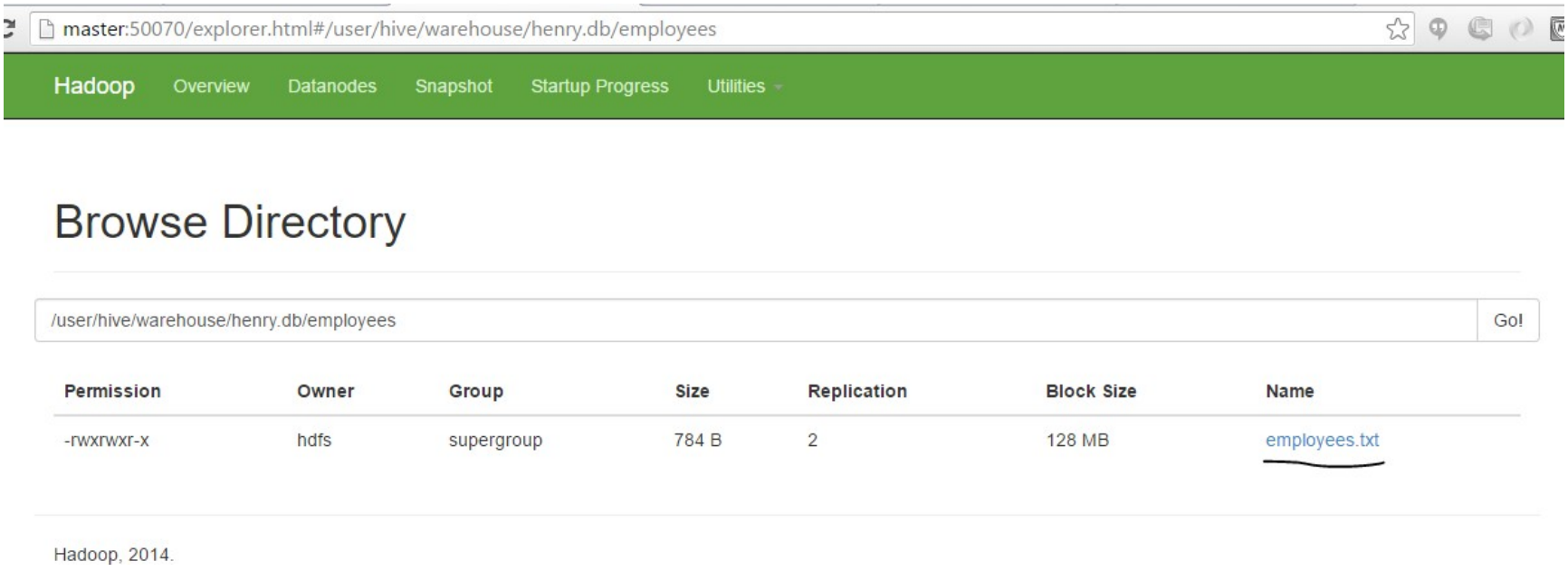
ls /opt/employees.txt

-- Now load the data!

#LOAD DATA LOCAL INPATH '/opt/employees.txt' INTO TABLE employees;



After upload on Hive, You can verify the file in the hive folder as shown below



-- How fast is the following query?
-- Notice how the complex data values are formatted

in the output.

#SELECT * FROM employees;

```
hive> SELECT * FROM employees;
OK
John Doe        100000.0        ["Mary Smith","Todd Jones"]     {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}        {"street":"1 Michigan Av
e.","city":"Chicago","state":"IL","zip":60600}
Mary Smith      80000.0 ["Bill King"]   {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}        {"street":"100 Ontario St.","city":"Chicago","st
ate":"IL","zip":60601}
Todd Jones      70000.0 []      {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}       {"street":"200 Chicago Ave.","city":"Oak Park","state":"
IL","zip":60700}
Bill King       60000.0 []      {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}       {"street":"300 Obscure Dr.","city":"Obscuria","state":"I
L","zip":60100}
Boss Man        200000.0        ["John Doe","Fred Finance"]     {"Federal Taxes":0.3,"State Taxes":0.07,"Insurance":0.05}       {"street":"1 Pretentious
 Drive.","city":"Chicago","state":"IL","zip":60500}
Fred Finance    150000.0        ["Stacy Accountant"]    {"Federal Taxes":0.3,"State Taxes":0.07,"Insurance":0.05}       {"street":"2 Pretentious Drive."
,"city":"Chicago","state":"IL","zip":60500}
Stacy Accountant        60000.0 []      {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}       {"street":"300 Main St.","city":"Naperville","st
ate":"IL","zip":60563}
Time taken: 0.141 seconds, Fetched: 7 row(s)
hive>
```

-- Try this query and a few others that "project out" some of the columns.
-- Is it slower or faster than the previous one?

SELECT name, subordinates FROM employees;

```
hive> SELECT name, subordinates FROM employees;
OK
John Doe        ["Mary Smith","Todd Jones"]
Mary Smith      ["Bill King"]
Todd Jones      []
Bill King       []
Boss Man        ["John Doe","Fred Finance"]
Fred Finance    ["Stacy Accountant"]
Stacy Accountant        []
Time taken: 0.127 seconds, Fetched: 7 row(s)
hive>
```

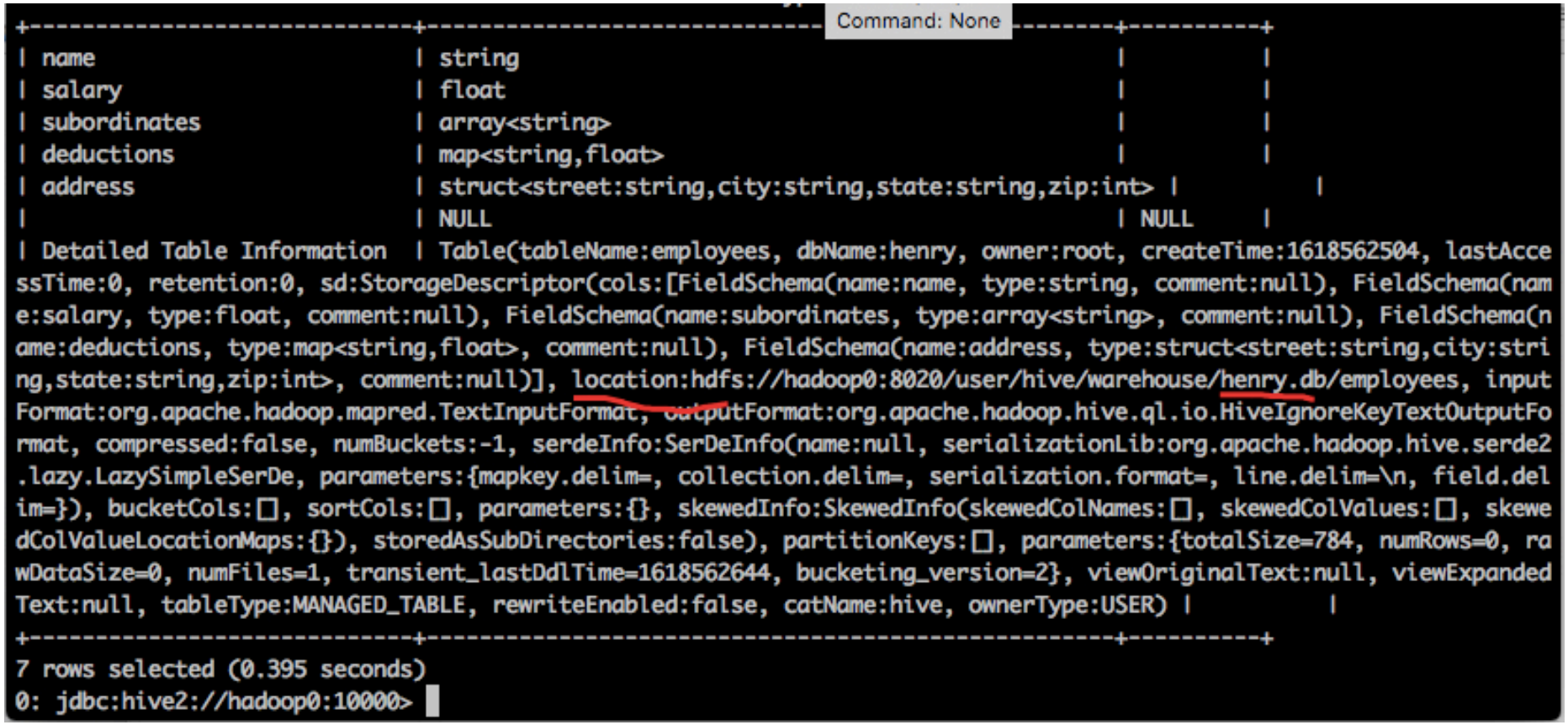
-- What if we forget the schema?

#DESCRIBE employees;

-- Want even more information?

#DESCRIBE EXTENDED employees;

-- Look for the "location" field in the output and note the path:
-- hdfs://foobar/mnt/hive_081/warehouse/${DB}.db/employees
-- Where "${DB}" should actually be you database name.
-- (Note: The default path for the Apache Hive release would be
-- hdfs://foobar/user/hive/warehouse/${DB}.db/employees)

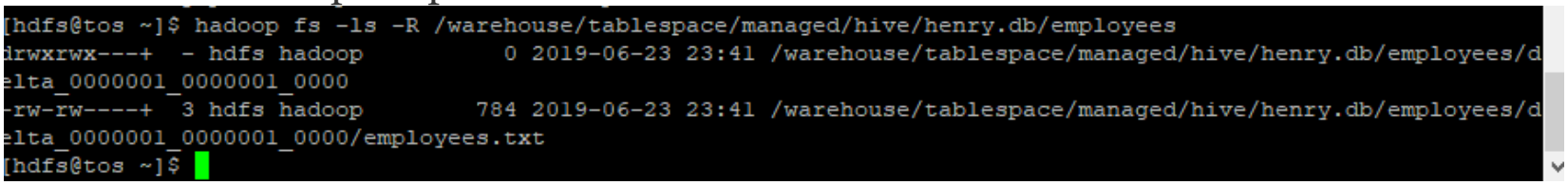


In CLI:

Substitute the file path with that of the above location.

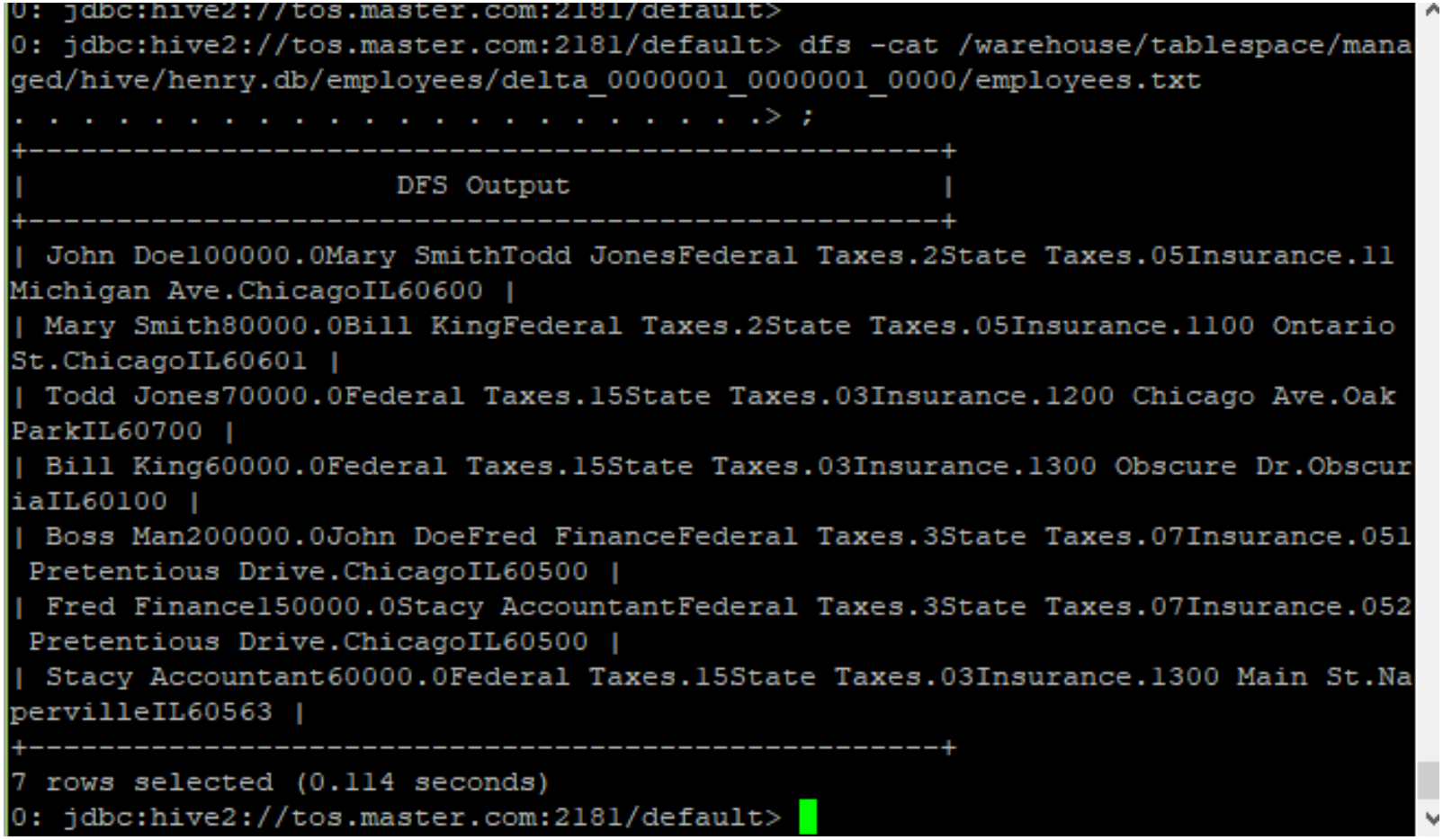
#hdfs dfs -ls –R /user/hive/warehouse/henry.db/employees

In hdfs command prompt



You can view the content of the file.

#hdfs dfs -cat /user/hive/warehouse/henry.db/employees/employees.txt



-- We can drop the table after we're done with it. DON'T DO THIS NOW!!

#DROP TABLE employees;

Actual Data file

```
John Doe|100000.0|Mary Smith Todd Jones|Federal Taxes|.2 State Taxes|.05 Insurance|.1|1 Michigan Ave. Chicago IL 60600
Mary Smith|80000.0|Bill King|Federal Taxes|.2 State Taxes|.05 Insurance|.1|100 Ontario St. Chicago IL 60601
Todd Jones|70000.0||Federal Taxes|.15 State Taxes|.03 Insurance|.1|200 Chicago Ave. Oak Park IL 60700
Bill King|60000.0||Federal Taxes|.15 State Taxes|.03 Insurance|.1|300 Obscure Dr. Obscuria IL 60100
```

Represent in data model

```
John Doe        100000.0        ["Mary Smith","Todd Jones"]     {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}        {"street":"1 Michigan Av
e.","city":"Chicago","state":"IL","zip":60600}
Mary Smith      80000.0 ["Bill King"]   {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}        {"street":"100 Ontario St.","city":"Chicago","st
ate":"IL","zip":60601}
Todd Jones      70000.0 []      {"Federal Taxes":0.15,"State Taxes":0.03,"Insurance":0.1}       {"street":"200 Chicago Ave.","city":"Oak Park","state":"
IL","zip":60700}
```

---------- Ends Here ----------

External Tables with Partitions and DML – 60 Minutes.

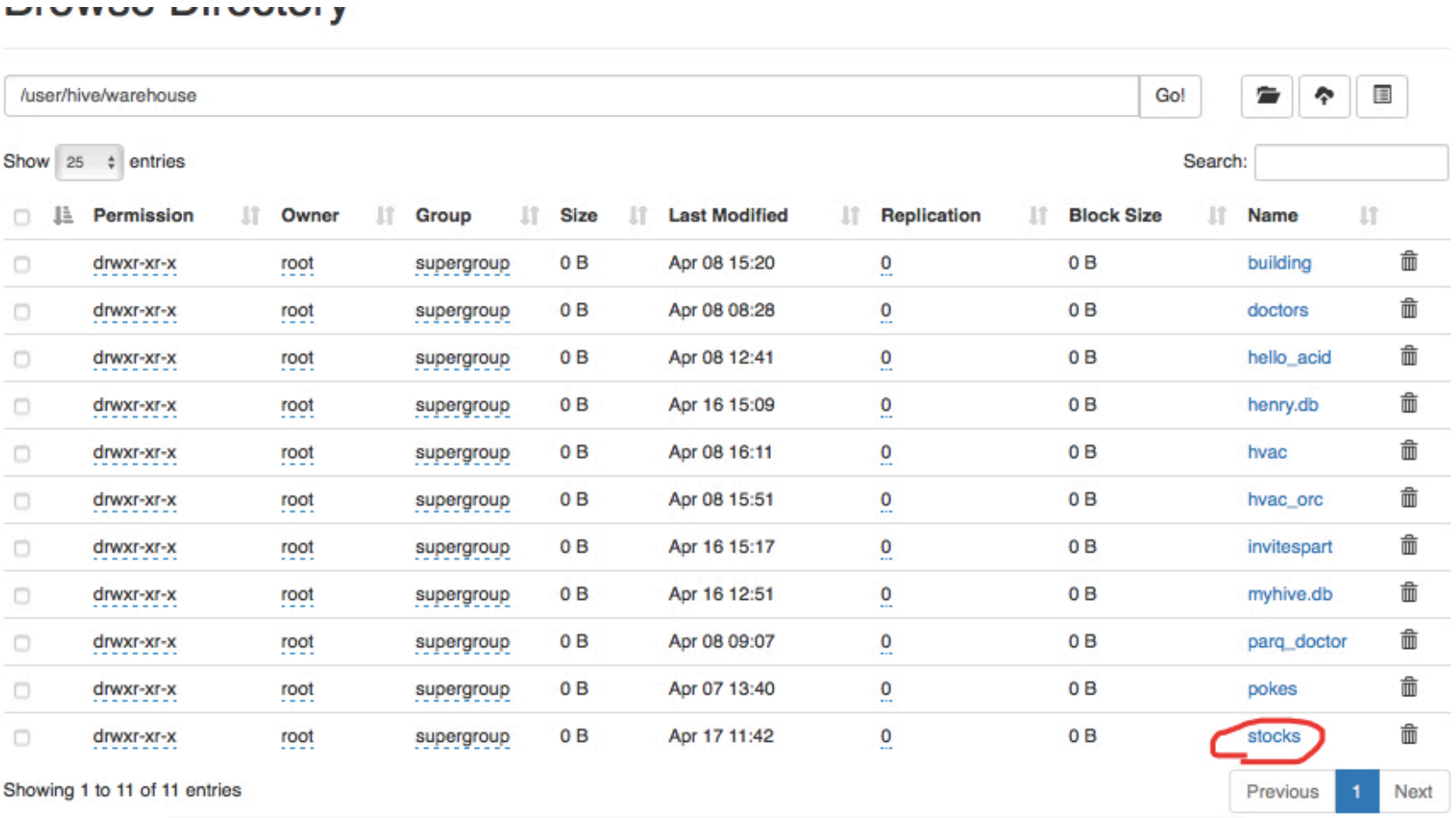
We'll demonstrate the use of two features, external (vs. managed or internal) tables and partitioning the table to speed up performance.
Recall that you can also use partitioning with managed tables.

```
-- We'll use historical stock price data from Infochimps.com:
--   NASDAQ: infochimps_dataset_4777_download_16185
--   NYSE:   infochimps_dataset_4778_download_16677

-- The EXTERNAL keyword tells Hive that the table storage will
-- be "external" to Hive, rather than the default internal
-- storage. We'll specify where the storage exists below.
-- We'll also partition the table by the exchange and the
-- stock symbol, which will speed-up queries selecting on either
-- field, because Hive will know it can skip partitions that
-- don't match the specified query values!
```

#CREATE EXTERNAL TABLE IF NOT EXISTS stocks (ymd STRING,price_open FLOAT, price_high FLOAT, price_low FLOAT, price_close FLOAT, volume INT, price_adj_close FLOAT) PARTITIONED BY (exchanged STRING, symbol STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS stocks (
    >   ymd             STRING,
    >   price_open      FLOAT,
    >   price_high      FLOAT,
    >   price_low       FLOAT,
    >   price_close     FLOAT,
    >   volume          INT,
    >   price_adj_close FLOAT )
    > PARTITIONED BY (exchanged STRING, symbol STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 0.867 seconds
hive>
```

-- We don't have any partitions yet: You can verify the folder as shown below.



Browse Directory

/user/hive/warehouse Go!

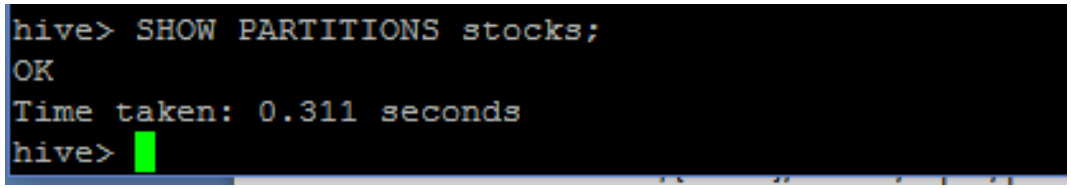
Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	root	supergroup	0 B	Apr 08 15:20	0	0 B	building
drwxr-xr-x	root	supergroup	0 B	Apr 08 08:28	0	0 B	doctors
drwxr-xr-x	root	supergroup	0 B	Apr 08 12:41	0	0 B	hello_acid
drwxr-xr-x	root	supergroup	0 B	Apr 16 15:09	0	0 B	henry.db
drwxr-xr-x	root	supergroup	0 B	Apr 08 16:11	0	0 B	hvac
drwxr-xr-x	root	supergroup	0 B	Apr 08 15:51	0	0 B	hvac_orc
drwxr-xr-x	root	supergroup	0 B	Apr 16 15:17	0	0 B	invitespart
drwxr-xr-x	root	supergroup	0 B	Apr 16 12:51	0	0 B	myhive.db
drwxr-xr-x	root	supergroup	0 B	Apr 08 09:07	0	0 B	parq_doctor
drwxr-xr-x	root	supergroup	0 B	Apr 07 13:40	0	0 B	pokes
drwxr-xr-x	root	supergroup	0 B	Apr 17 11:42	0	0 B	stocks

Showing 1 to 11 of 11 entries Previous 1 Next

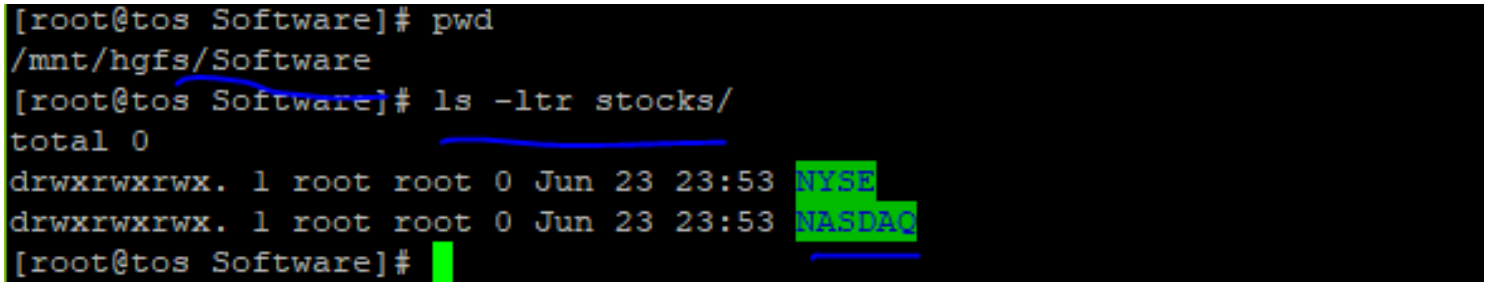
You can determine the partition information if any, here we don't have any since no data has been uploaded.

#SHOW PARTITIONS stocks;



Copy stocks.tar in a folder and un compressed it.

Copy stocks data i.e stock folder to the following folder or in any folder to upload to HDFS:



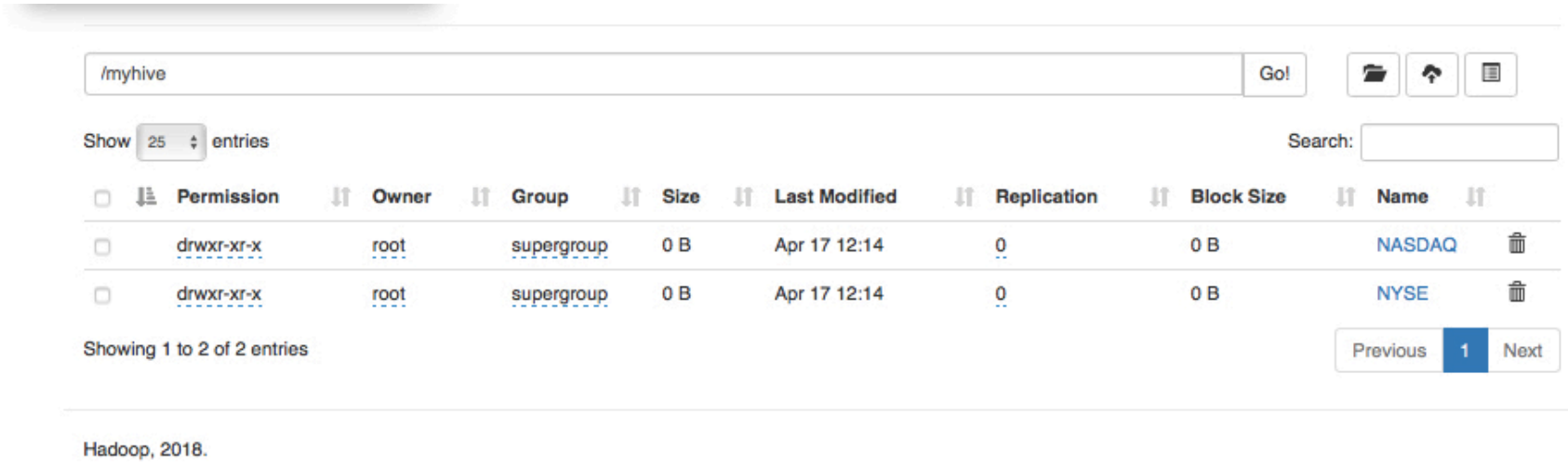
Create a folder in HDFS to dump the above stocks data which will be further map to hive table.

#hdfs dfs -mkdir -p /myhive/stocks

Go the folder where you have extracted the stocks and perform the following command.

#hdfs dfs -copyFromLocal stocks/* /myhive/stocks

You can confirm the upload using HDFS Console.



```
-- For EXTERNAL, partitioned tables, you use ALTER TABLE to add each
-- partition and specify a unique directory for its data.
-- We'll just add data for four stocks:

#ALTER TABLE stocks ADD PARTITION(exchanged = 'NASDAQ', symbol =
'AAPL') LOCATION '/myhive/stocks/NASDAQ/AAPL';

#ALTER TABLE stocks ADD PARTITION(exchanged = 'NASDAQ', symbol = 'INTC')
LOCATION '/myhive/stocks/NASDAQ/INTC';

#ALTER TABLE stocks ADD PARTITION(exchanged = 'NYSE', symbol =
'GE') LOCATION '/myhive/stocks/NYSE/GE';

#ALTER TABLE stocks ADD PARTITION(exchanged = 'NYSE', symbol = 'IBM')
LOCATION '/myhive/stocks/NYSE/IBM';
```

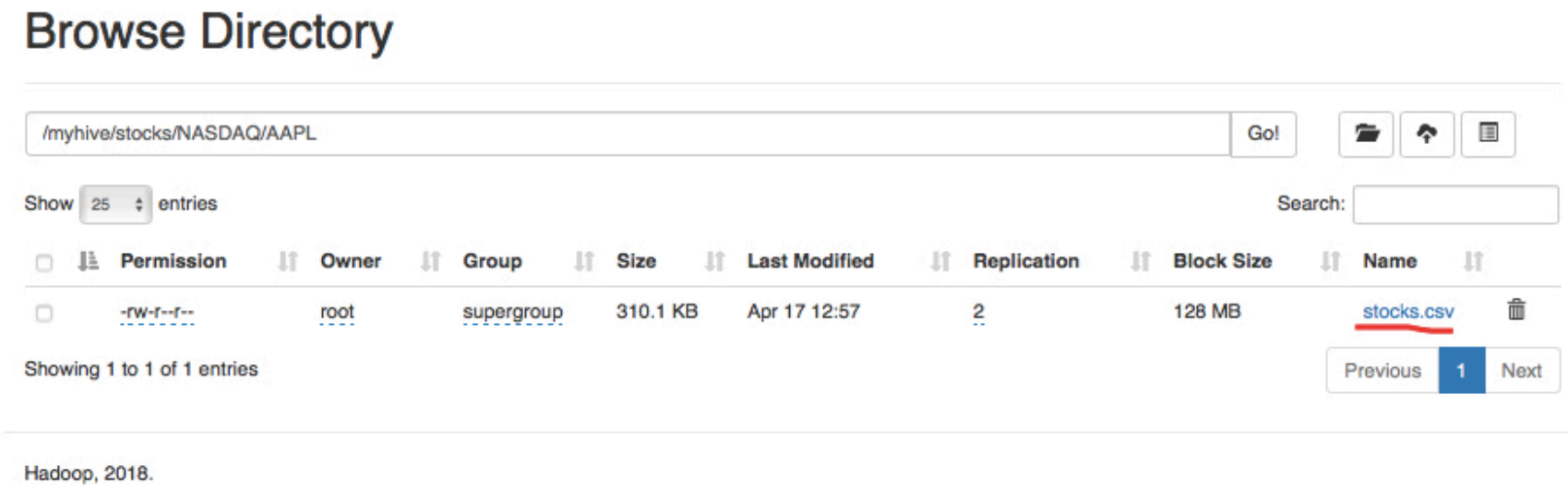
```
No rows affected (0.661 seconds)
0: jdbc:hive2://tos.master.com:2181/default>
0: jdbc:hive2://tos.master.com:2181/default> ALTER TABLE stocks ADD PARTITION(ex
changed = 'NYSE', symbol = 'IBM')
. . . . . .> LOCATION '/myhive/stocks/NYSE/IBM';

INFO  : Compiling command(queryId=hive_20190624220037_09fe02e4-aed5-4ffa-af81-ad
616e06970d): ALTER TABLE stocks ADD PARTITION(exchanged = 'NYSE', symbol = 'IBM'
)
LOCATION '/myhive/stocks/NYSE/IBM'
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=hive_20190624220037_09fe02e4-aed5-4f
fa-af81-ad616e06970d); Time taken: 0.157 seconds
INFO  : Executing command(queryId=hive_20190624220037_09fe02e4-aed5-4ffa-af81-ad
616e06970d): ALTER TABLE stocks ADD PARTITION(exchanged = 'NYSE', symbol = 'IBM'
)
LOCATION '/myhive/stocks/NYSE/IBM'
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20190624220037_09fe02e4-aed5-4f
fa-af81-ad616e06970d); Time taken: 0.323 seconds
INFO  : OK
No rows affected (0.588 seconds)
0: jdbc:hive2://tos.master.com:2181/default>
```

Now, let us determine the partitions on the table.

#SHOW PARTITIONS stocks;

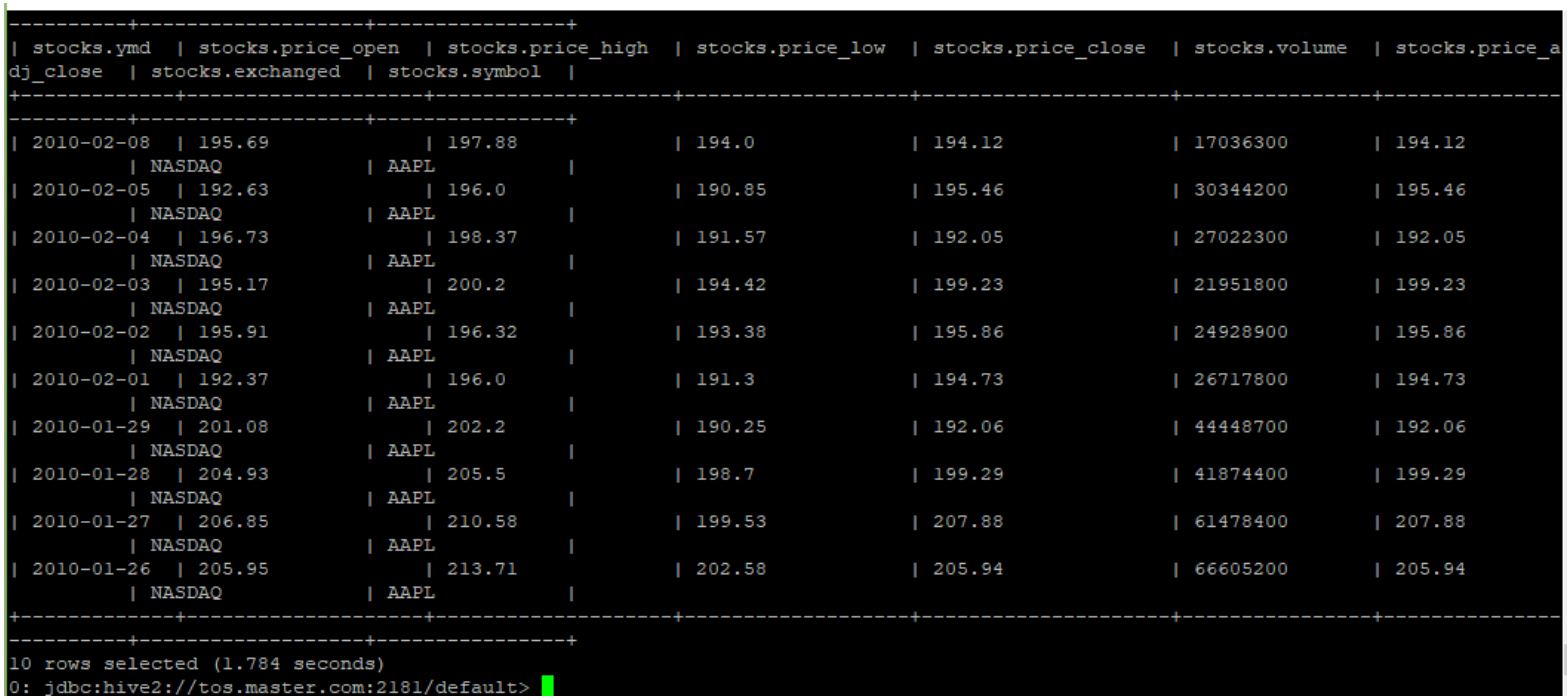
```
0: jdbc:hive2://tos.master.com:2181/default> SHOW PARTITIONS stocks;
INFO  : Compiling command(queryId=hive_20190624220101_e3eaae0e-a1b3-4dc3-b03a-0c
dc4dee3154): SHOW PARTITIONS stocks
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:partition,
type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=hive_20190624220101_e3eaae0e-a1b3-4d
c3-b03a-0cdc4dee3154); Time taken: 0.157 seconds
INFO  : Executing command(queryId=hive_20190624220101_e3eaae0e-a1b3-4dc3-b03a-0c
dc4dee3154): SHOW PARTITIONS stocks
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hive_20190624220101_e3eaae0e-a1b3-4d
c3-b03a-0cdc4dee3154); Time taken: 0.095 seconds
INFO  : OK
+------------------------------+
|           partition          |
+------------------------------+
| exchanged=NASDAQ/symbol=AAPL |
| exchanged=NASDAQ/symbol=INTC |
| exchanged=NYSE/symbol=GE     |
| exchanged=NYSE/symbol=IBM    |
+------------------------------+
4 rows selected (0.525 seconds)
0: jdbc:hive2://tos.master.com:2181/default>
```

Now you should have the following structure in HDFS.



-- Try a test query. How fast are these two queries??

#SELECT * FROM stocks WHERE exchanged = 'NASDAQ' AND symbol = 'AAPL' LIMIT 10;



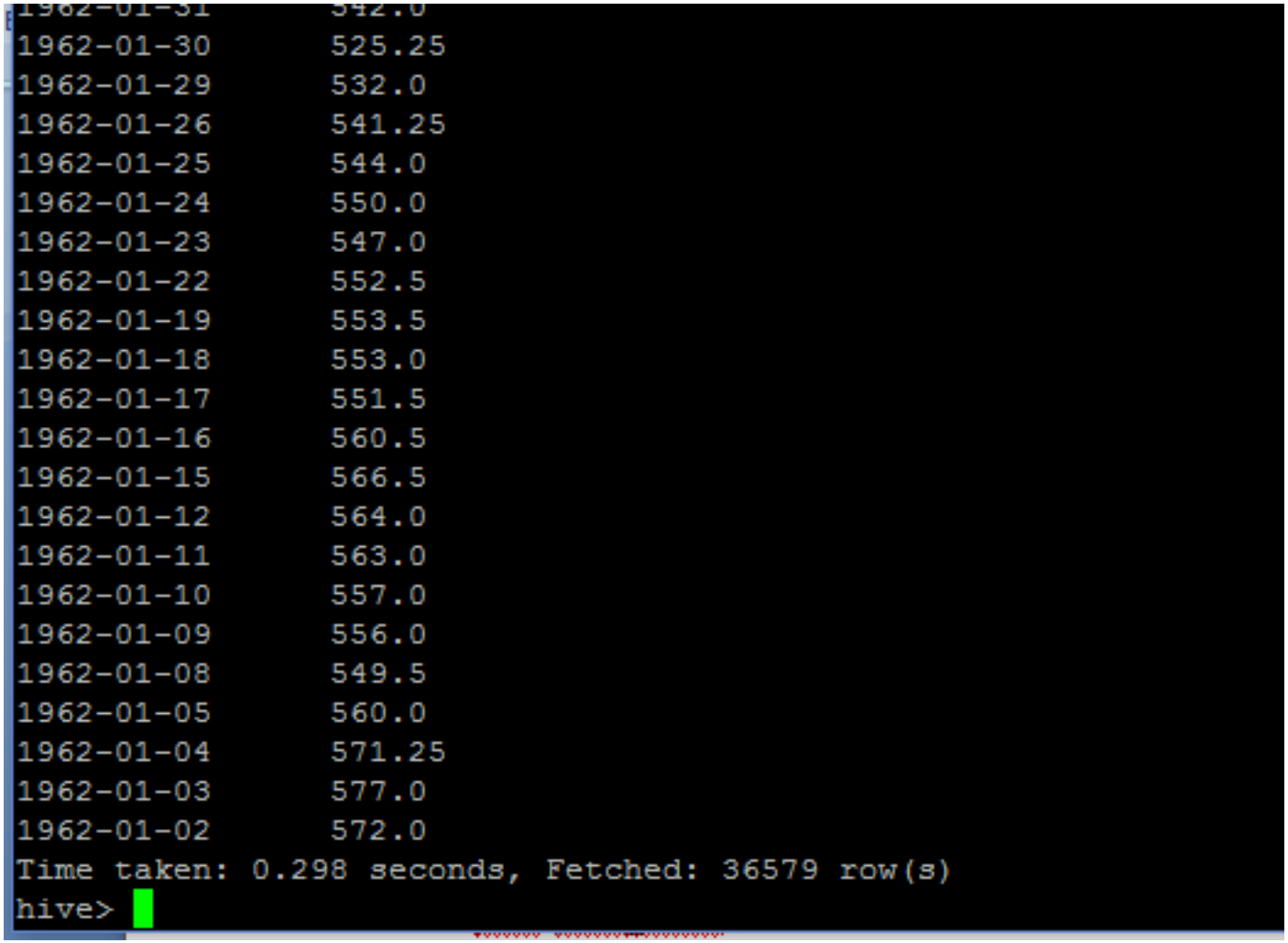
```
+-------------+-------------------+-------------------+------------------+--------------------+----------------+---------------+
| stocks.ymd  | stocks.price_open | stocks.price_high | stocks.price_low | stocks.price_close | stocks.volume  | stocks.price_adj_close  | stocks.exchanged  | stocks.symbol  |
+-------------+-------------------+-------------------+------------------+--------------------+----------------+---------------+
| 2010-02-08  | 195.69            | 197.88            | 194.0            | 194.12             | 17036300       | 194.12        | NASDAQ            | AAPL           |
| 2010-02-05  | 192.63            | 196.0             | 190.85           | 195.46             | 30344200       | 195.46        | NASDAQ            | AAPL           |
| 2010-02-04  | 196.73            | 198.37            | 191.57           | 192.05             | 27022300       | 192.05        | NASDAQ            | AAPL           |
| 2010-02-03  | 195.17            | 200.2             | 194.42           | 199.23             | 21951800       | 199.23        | NASDAQ            | AAPL           |
| 2010-02-02  | 195.91            | 196.32            | 193.38           | 195.86             | 24928900       | 195.86        | NASDAQ            | AAPL           |
| 2010-02-01  | 192.37            | 196.0             | 191.3            | 194.73             | 26717800       | 194.73        | NASDAQ            | AAPL           |
| 2010-01-29  | 201.08            | 202.2             | 190.25           | 192.06             | 44448700       | 192.06        | NASDAQ            | AAPL           |
| 2010-01-28  | 204.93            | 205.5             | 198.7            | 199.29             | 41874400       | 199.29        | NASDAQ            | AAPL           |
| 2010-01-27  | 206.85            | 210.58            | 199.53           | 207.88             | 61478400       | 207.88        | NASDAQ            | AAPL           |
| 2010-01-26  | 205.95            | 213.71            | 202.58           | 205.94             | 66605200       | 205.94        | NASDAQ            | AAPL           |
+-------------+-------------------+-------------------+------------------+--------------------+----------------+---------------+
10 rows selected (1.784 seconds)
0: jdbc:hive2://tos.master.com:2181/default>
```

#SELECT ymd, price_close FROM stocks WHERE exchanged = 'NASDAQ' AND symbol = 'AAPL' LIMIT 10;

```
hive> SELECT ymd, price_close FROM stocks WHERE exchanged = 'NASDAQ' AND symbol = 'AAPL' LIMIT 10;
OK
2010-02-08      194.12
2010-02-05      195.46
2010-02-04      192.05
2010-02-03      199.23
2010-02-02      195.86
2010-02-01      194.73
2010-01-29      192.06
2010-01-28      199.29
2010-01-27      207.88
2010-01-26      205.94
Time taken: 0.296 seconds, Fetched: 10 row(s)
hive>
```

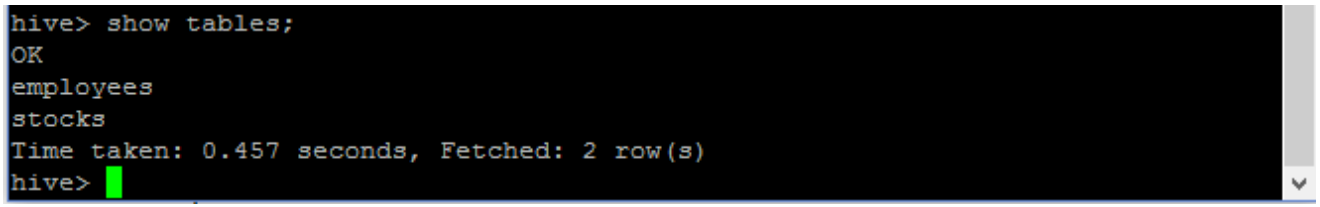
-- Try a few other queries to play with the data.

#SELECT ymd, price_close FROM stocks;



-- Select statements.

Confirm that stocks table is there as follow:



-- First, let's look at 20 days-worth of APPL stock results,
-- just the open and close prices.

#SELECT ymd, price_open, price_close
FROM stocks WHERE symbol = 'AAPL' AND exchanged = 'NASDAQ' LIMIT 20;

```
hive> SELECT ymd, price_open, price_close
    > FROM stocks WHERE symbol = 'AAPL' AND exchanged = 'NASDAQ' LIMIT 20;
OK
2010-02-08      195.69  194.12
2010-02-05      192.63  195.46
2010-02-04      196.73  192.05
2010-02-03      195.17  199.23
2010-02-02      195.91  195.86
2010-02-01      192.37  194.73
2010-01-29      201.08  192.06
2010-01-28      204.93  199.29
2010-01-27      206.85  207.88
2010-01-26      205.95  205.94
2010-01-25      202.51  203.07
2010-01-22      206.78  197.75
2010-01-21      212.08  208.07
2010-01-20      214.91  211.73
2010-01-19      208.33  215.04
2010-01-15      210.93  205.93
2010-01-14      210.11  209.43
2010-01-13      207.87  210.65
2010-01-12      209.19  207.72
2010-01-11      212.8   210.11
Time taken: 2.292 seconds, Fetched: 20 row(s)
```

#SELECT ymd, price_open, price_close from stocks where price_open >= 195.69 and symbol = 'AAPL' AND exchanged = 'NASDAQ' limit 12;

```
hive> SELECT ymd, price_open, price_close from stocks where price_open >= 195.69
 and symbol = 'AAPL' AND exchanged = 'NASDAQ' limit 12;
OK
2010-02-08      195.69  194.12
2010-02-04      196.73  192.05
2010-02-02      195.91  195.86
2010-01-29      201.08  192.06
2010-01-28      204.93  199.29
2010-01-27      206.85  207.88
2010-01-26      205.95  205.94
2010-01-25      202.51  203.07
2010-01-22      206.78  197.75
2010-01-21      212.08  208.07
2010-01-20      214.91  211.73
2010-01-19      208.33  215.04
Time taken: 0.315 seconds, Fetched: 12 row(s)
hive>
```

#SELECT ymd, price_open, price_close from stocks where price_open >= 195.69 limit 12;

```
hive> SELECT ymd, price_open, price_close from stocks where price_open = 195.69
 limit 12;
OK
2010-02-08      195.69  194.12
Time taken: 0.152 seconds, Fetched: 1 row(s)
hive>
```

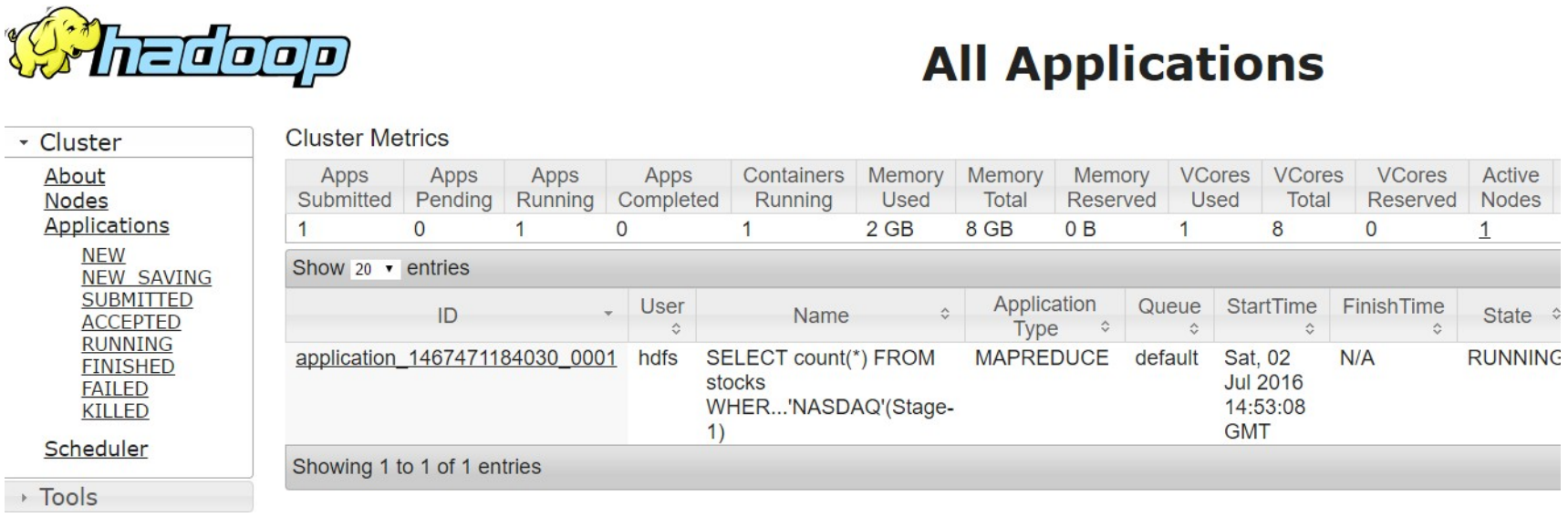
```
-- EXERCISE: What looks different in the output when you use "*"
-- instead of a list of columns in the select clause? Is this query
-- faster?

-- The previous query uses a "partition filter" to limit the
-- directories it scans to only the SINGLE directory for AAPL under
-- NASDAQ.


-- Built-in Functions:
-- Hive has many built-in functions, although they are often called
-- User Defined Functions (UDFs), because we can implement and add
-- our own functions using the same mechanisms that Hive uses!

-- One kind of UDF is the user defined aggregate function (UDAF),
-- which aggregates multiple records into a single output. For example,
-- let's count the number of records for AAPL: (6412)
```

Set a specific Queue for the Hive if you have configured Queue else skip.

```
#set mapred.job.queue.name=QE;


#SELECT count(*) FROM stocks WHERE symbol = 'AAPL' AND exchanged = 'NASDAQ';
```

You can verify the Job using the following URL:

http://localhost:8088/cluster



```
INFO  :    SPILLED_RECORDS: 0
INFO  : TaskCounter_Reducer_2_INPUT_Map_1:
INFO  :    FIRST_EVENT_RECEIVED: 1475
INFO  :    INPUT_RECORDS_PROCESSED: 1
INFO  :    LAST_EVENT_RECEIVED: 1475
INFO  :    NUM_FAILED_SHUFFLE_INPUTS: 0
INFO  :    NUM_SHUFFLED_INPUTS: 1
INFO  :    SHUFFLE_BYTES: 27
INFO  :    SHUFFLE_BYTES_DECOMPRESSED: 13
INFO  :    SHUFFLE_BYTES_DISK_DIRECT: 27
INFO  :    SHUFFLE_BYTES_TO_DISK: 0
INFO  :    SHUFFLE_BYTES_TO_MEM: 0
INFO  :    SHUFFLE_PHASE_TIME: 1612
INFO  : TaskCounter_Reducer_2_OUTPUT_out_Reducer_2:
INFO  :    OUTPUT_RECORDS: 0
INFO  : org.apache.hadoop.hive.ql.exec.tez.HiveInputCounters:
INFO  :    GROUPED_INPUT_SPLITS_Map_1: 1
INFO  :    INPUT_DIRECTORIES_Map_1: 1
INFO  :    INPUT_FILES_Map_1: 1
INFO  :    RAW_INPUT_SPLITS_Map_1: 1
INFO  : Completed executing command(queryId=hive_20190624220424_12d3e999-62c2-4d9c-b559-ccd319ba69cc); Time taken: 64.973 seconds
INFO  : OK
+--------+
|  _c0   |
+--------+
| 6412   |
+--------+
1 row selected (67.551 seconds)
0: jdbc:hive2://tos.master.com:2181/default>
```

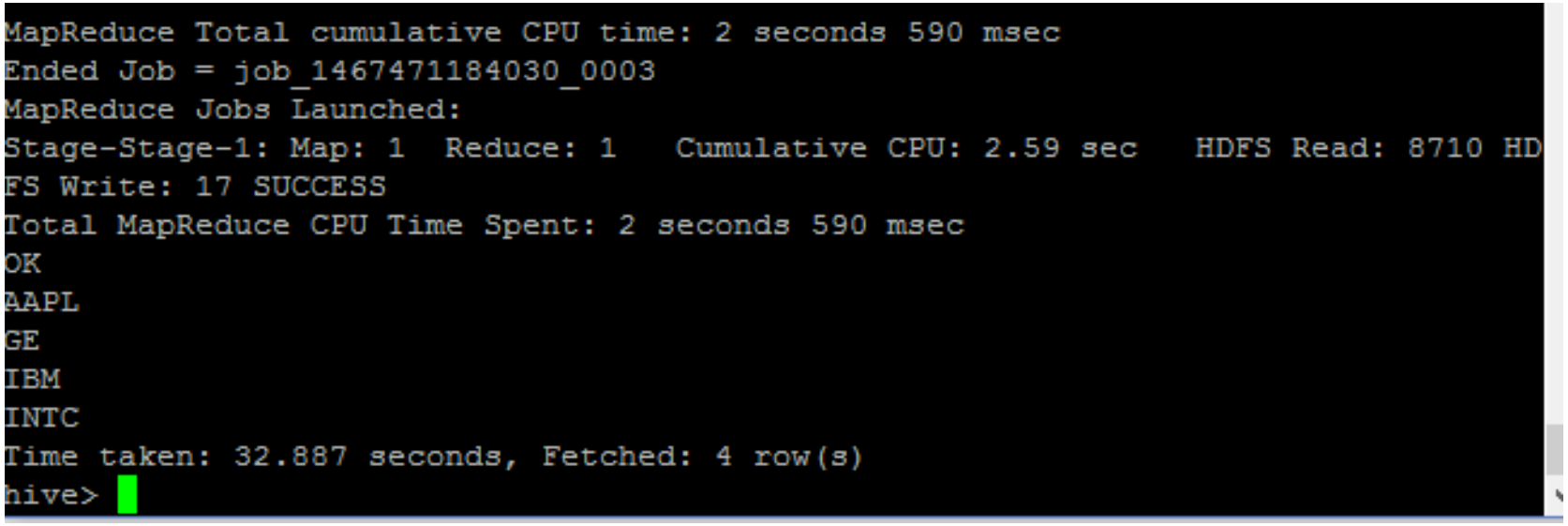
-- EXERCISE: Count the number of GE records. Which has more, AAPL or GE?
-- Average the closing price for AAPL: ($51.75)

#SELECT avg(price_close) FROM stocks WHERE symbol = 'AAPL' AND exchanged = 'NASDAQ';

```
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2016-07-02 20:27:24,672 Stage-1 map = 0%,  reduce = 0%
2016-07-02 20:27:35,736 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.82 se
c
2016-07-02 20:27:48,160 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.81
sec
MapReduce Total cumulative CPU time: 3 seconds 810 msec
Ended Job = job_1467471184030_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.81 sec   HDFS Read: 325399
HDFS Write: 19 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 810 msec
OK
51.745206035529534
Time taken: 37.516 seconds, Fetched: 1 row(s)
hive>
```

-- DISTINCT
-- You can find distinct things, but what happens in this query?

SELECT DISTINCT symbol FROM stocks;



-- How do you query with ARRAY, MAP, and STRUCT elements?
-- Let's use the employees table to see.

-- Select a value, given a key, from a MAP;
-- You specify the key inside array brackets.
-- This query also does a floating-point comparison. Is the answer
-- what you expect??

#use henry;
#SELECT name, deductions['Federal Taxes'] FROM employees WHERE deductions['Federal Taxes'] > 0.2;

```
hive> SELECT name, deductions['Federal Taxes'] FROM employees
    > WHERE deductions['Federal Taxes'] > 0.2;
OK
John Doe        0.2
Mary Smith      0.2
Boss Man        0.3
Fred Finance    0.3
Time taken: 0.129 seconds, Fetched: 4 row(s)
hive>
```

```
hive> SELECT * FROM employees
    > WHERE deductions['Federal Taxes'] > 0.2;
OK
John Doe        100000.0        ["Mary Smith","Todd Jones"]     {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}  {
"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
Mary Smith      80000.0 ["Bill King"]   {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}        {"street":"100 Onta
rio St.","city":"Chicago","state":"IL","zip":60601}
Boss Man        200000.0        ["John Doe","Fred Finance"]     {"Federal Taxes":0.3,"State Taxes":0.07,"Insurance":0.05} {
"street":"1 Pretentious Drive.","city":"Chicago","state":"IL","zip":60500}
Fred Finance    150000.0        ["Stacy Accountant"]    {"Federal Taxes":0.3,"State Taxes":0.07,"Insurance":0.05}       {"s
treet":"2 Pretentious Drive.","city":"Chicago","state":"IL","zip":60500}
Time taken: 0.151 seconds, Fetched: 4 row(s)
hive>
    >
```

-- Now try this variant. Is the answer different and what you expect?

#SELECT name, deductions['Federal Taxes'] FROM employees WHERE deductions['Federal Taxes'] > cast(0.2 as float);

```
hive> SELECT name, deductions['Federal Taxes'] FROM employees
    > WHERE deductions['Federal Taxes'] > cast(0.2 as float);
OK
Boss Man        0.3
Fred Finance    0.3
Time taken: 0.134 seconds, Fetched: 2 row(s)
hive>
```

-- Select elements in an ARRAY:
-- You provide an integer index, starting at ZERO.
-- The query returns one name from the ARRAY. Note that we're asking
-- if "Todd Jones" is the SECOND person who is in the subordinate
-- array of a manager.

#SELECT name FROM employees WHERE subordinates[1] = 'Todd Jones';

```
hive> SELECT name FROM employees WHERE subordinates[1] = 'Todd Jones';
OK
John Doe
Time taken: 0.087 seconds, Fetched: 1 row(s)
hive>
```

-- Who is a manager?

#SELECT name FROM employees WHERE size(subordinates) > 0;

-- EXERCISE: Who isn't a manager?

```
hive> SELECT name FROM employees WHERE size(subordinates) > 0;
OK
John Doe
Mary Smith
Boss Man
Fred Finance
Time taken: 0.104 seconds, Fetched: 4 row(s)
hive>
```

```
hive> SELECT * FROM employees WHERE size(subordinates) > 0 limit 2;
OK
John Doe        100000.0        ["Mary Smith","Todd Jones"]     {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}  {
"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
Mary Smith      80000.0 ["Bill King"]   {"Federal Taxes":0.2,"State Taxes":0.05,"Insurance":0.1}        {"street":"100 Onta
rio St.","city":"Chicago","state":"IL","zip":60601}
Time taken: 0.09 seconds, Fetched: 2 row(s)
hive>
```

-- Select elements in a STRUCT:
-- Use the "dot" notation: "struct.element".
-- Who lives in the 60500 Zip Code?

#SELECT name FROM employees WHERE address.zip = 60500;

```
hive> SELECT name FROM employees WHERE address.zip = 60500;
OK
Boss Man
Fred Finance
Time taken: 0.157 seconds, Fetched: 2 row(s)
hive>
```

-- Hive supports the LIKE operator.

#SELECT name, address FROM employees WHERE address.city LIKE 'C%';

```
hive> SELECT name, address FROM employees WHERE address.city LIKE 'C%';
OK
John Doe        {"street":"1 Michigan Ave.","city":"Chicago","state":"IL","zip":60600}
Mary Smith      {"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
Boss Man        {"street":"1 Pretentious Drive.","city":"Chicago","state":"IL","zip":60500}
Fred Finance    {"street":"2 Pretentious Drive.","city":"Chicago","state":"IL","zip":60500}
Time taken: 0.16 seconds, Fetched: 4 row(s)
hive>
```

-- Hive also supports matching on Java-style regular expressions using
-- an extension, the RLIKE operator.
-- The expression at the end of this query says to match streets that
-- contain either "Ontario" or "Chicago". The "|" indicates "or".

#SELECT name, address FROM employees WHERE address.street RLIKE 'Ontario|Chicago';

```
hive> SELECT name, address FROM employees
    > WHERE address.street RLIKE 'Ontario|Chicago';
OK
Mary Smith      {"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
Todd Jones      {"street":"200 Chicago Ave.","city":"Oak Park","state":"IL","zip":60700}
Time taken: 0.233 seconds, Fetched: 2 row(s)
hive>
```

-- This variation is effectively the same, but it matches the entire
-- street string. "^" and "$" match the beginning and end, respectively,
-- of a string. The ".*" match zero or more characters of any kind.
-- The parentheses are necessary make it clear that the expression is
-- "^.*(something).*$", where "something" is "Ontario|Chicago".

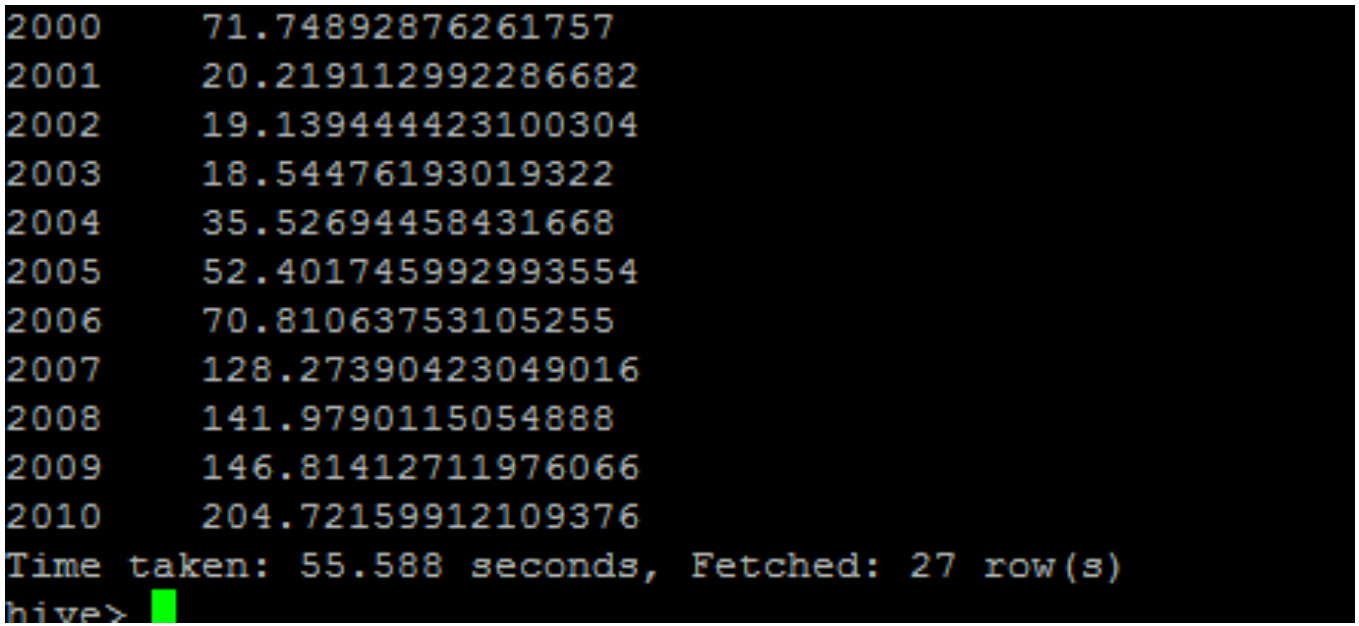
#SELECT name, address FROM employees WHERE address.street RLIKE '^.*(Ontario|Chicago).*$';

```
hive> SELECT name, address FROM employees
    > WHERE address.street RLIKE '^.*(Ontario|Chicago).*$';
OK
Mary Smith      {"street":"100 Ontario St.","city":"Chicago","state":"IL","zip":60601}
Todd Jones      {"street":"200 Chicago Ave.","city":"Oak Park","state":"IL","zip":60700}
Time taken: 0.143 seconds, Fetched: 2 row(s)
hive>
```

-- EXERCISE: Who lives in a Zip Code greater than 60500?
-- EXERCISE: Who pays 15% Federal Income Tax? (Besides Mitt Romney ;^)
-- EXERCISE: Who lives on a street that is a "Drive" or a "Park"?

-- GROUP BY divides the resulting data into groups, based on the
-- specified criteria. NOTE: the GROUP BY can only refer to fields
-- (or their equivalents) referenced in the SELECT.

```
#use default;
# SELECT year(ymd), avg(price_close) FROM stocks
  WHERE symbol = 'AAPL' AND exchanged = 'NASDAQ' GROUP BY year(ymd);
```



-- EXERCISE: Compute the average for GE.

-- EXERCISE: Compute the average for each month.
-- GROUP BY ... HAVING is a way of further filtering the output.

#SELECT year(ymd), avg(price_close) FROM stocks WHERE symbol = 'AAPL' AND exchanged = 'NASDAQ' GROUP BY year(ymd)
HAVING avg(price_close) > 50.0 AND avg(price_close) < 100.0;

```
2016-07-02 20:52:51,973 Stage-1 map = 0%,  reduce = 0%
2016-07-02 20:53:23,760 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.79 sec
2016-07-02 20:53:41,019 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.2 sec
MapReduce Total cumulative CPU time: 9 seconds 200 msec
Ended Job = job_1467471184030_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 9.2 sec   HDFS Read: 326527 HDFS Write: 162 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 200 msec
OK
1987    53.88968399108163
1991    52.49553383386182
1992    54.80338610251119
1999    57.77071460844979
2000    71.74892876261757
2005    52.401745992993554
2006    70.81063753105255
Time taken: 71.171 seconds, Fetched: 7 row(s)
hive>
```

---------- Ends Here ----------

Hive Join – 30 Minutes

Let's also now create a new table of dividends, using the same partitioning scheme. We have already set up the external files in the distributed file system, so we won't go through the same sequence of steps we used for "stocks".
Run the following sequence of commands to create the table and add the required partitions.

#CREATE EXTERNAL TABLE IF NOT EXISTS dividends (ymd STRING, dividend FLOAT) PARTITIONED BY (exchanged STRING, symbol STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

```
hive>
    > CREATE EXTERNAL TABLE IF NOT EXISTS dividends (
    >   ymd             STRING,
    >   dividend        FLOAT
    > )
    > PARTITIONED BY (exchanged STRING, symbol STRING)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 1.087 seconds
hive>
```

Ensure to copy all the necessary data on HDFS cluster:

Download and un compress the dividends.tar

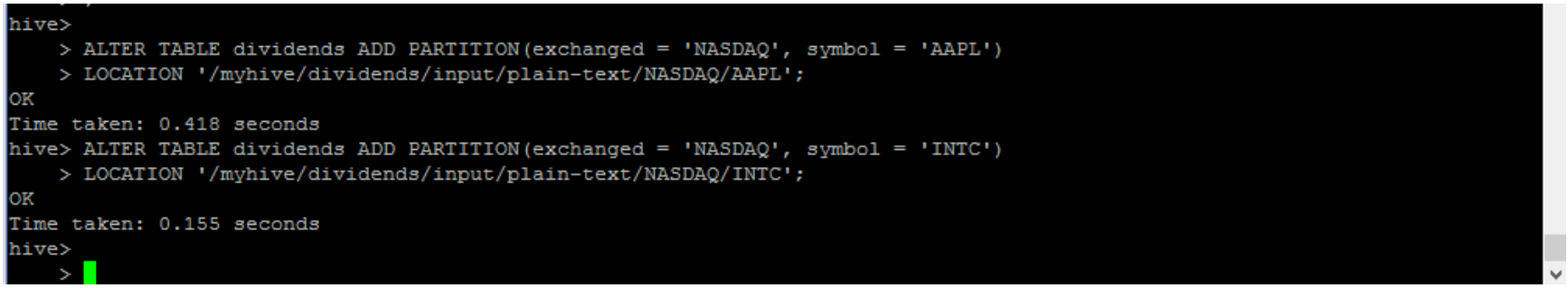
Copy the dividends folder to HDFS /myhive folder.

#hdfs dfs -copyFromLocal dividends/ /myhive/

Load the record using the following command from the Hive CLI.

#ALTER TABLE dividends ADD PARTITION(exchanged = 'NASDAQ', symbol = 'AAPL') LOCATION '/myhive/dividends/input/plain-text/NASDAQ/AAPL';

#ALTER TABLE dividends ADD PARTITION(exchanged = 'NASDAQ', symbol = 'INTC') LOCATION '/myhive/dividends/input/plain-text/NASDAQ/INTC';



ALTER TABLE dividends ADD PARTITION(exchanged = 'NYSE', symbol = 'GE') LOCATION '/myhive/dividends/input/plain-text/NYSE/GE';

ALTER TABLE dividends ADD PARTITION(exchanged = 'NYSE', symbol = 'IBM') LOCATION '/myhive/dividends/input/plain-text/NYSE/IBM';

```
hive>
    > ALTER TABLE dividends ADD PARTITION(exchanged = 'NYSE', symbol = 'GE')
    > LOCATION '/myhive/dividends/input/plain-text/NYSE/GE';
OK
Time taken: 0.251 seconds
hive>
    > ALTER TABLE dividends ADD PARTITION(exchanged = 'NYSE', symbol = 'IBM')
    > LOCATION '/myhive/dividends/input/plain-text/NYSE/IBM';
OK
Time taken: 0.162 seconds
hive>
```

-- Try a self join:

#SELECT a.ymd, a.symbol, a.price_close, b.symbol, b.price_close FROM stocks a JOIN stocks b ON a.ymd = b.ymd AND a.symbol = 'AAPL' AND b.symbol = 'IBM' WHERE a.ymd > '2010-01-01' LIMIT 20;

```
hive> SELECT a.ymd, a.symbol, a.price_close, b.symbol, b.price_close
    > FROM stocks a
    > JOIN stocks b
    > ON a.ymd    = b.ymd AND
    >    a.symbol = 'AAPL' AND
    >    b.symbol = 'IBM'
    > WHERE a.ymd > '2010-01-01'
    > LIMIT 20;
Query ID = hdfs_20160702212525_ae5e4da4-8689-48ac-b015-a6906d8ebd8f
Total jobs = 1
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/YARN/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/Static
LoggerBinder.class]
SLF4J: Found binding in [jar:file:/YARN/hive-1.1.0/lib/hive-jdbc-1.1.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.cl
ass]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
16/07/02 21:25:48 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
Execution log at: /tmp/hdfs/hdfs_20160702212525_ae5e4da4-8689-48ac-b015-a6906d8ebd8f.log
2016-07-02 09:25:51     Starting to launch local task to process map join;      maximum memory = 518979584
```

```
2016-07-02 21:26:10,048 Stage-3 map = 0%,  reduce = 0%
2016-07-02 21:26:25,564 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU 2.61 sec
MapReduce Total cumulative CPU time: 2 seconds 610 msec
Ended Job = job_1467471184030_0006
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 2.61 sec   HDFS Read: 10786 HDFS Write: 677 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 610 msec
OK
2010-02-08      AAPL    194.12  IBM     121.88
2010-02-05      AAPL    195.46  IBM     123.52
2010-02-04      AAPL    192.05  IBM     123.0
2010-02-03      AAPL    199.23  IBM     125.66
2010-02-02      AAPL    195.86  IBM     125.53
2010-02-01      AAPL    194.73  IBM     124.67
2010-01-29      AAPL    192.06  IBM     122.39
2010-01-28      AAPL    199.29  IBM     123.75
2010-01-27      AAPL    207.88  IBM     126.33
2010-01-26      AAPL    205.94  IBM     125.75
2010-01-25      AAPL    203.07  IBM     126.12
2010-01-22      AAPL    197.75  IBM     125.5
2010-01-21      AAPL    208.07  IBM     129.0
2010-01-20      AAPL    211.73  IBM     130.25
2010-01-19      AAPL    215.04  IBM     134.14
2010-01-15      AAPL    205.93  IBM     131.78
2010-01-14      AAPL    209.43  IBM     132.31
2010-01-13      AAPL    210.65  IBM     130.23
2010-01-12      AAPL    207.72  IBM     130.51
2010-01-11      AAPL    210.11  IBM     129.48
Time taken: 44.932 seconds, Fetched: 20 row(s)
```

-- JOIN Performance.

-- These two inner join queries are identical except for the MAPJOIN
-- directive in the second one. Hive will try to perform the whole join
-- in the map phase, without a reduce phase. It will succeed because the
-- dividends table is small enough to be cached in memory! How much
-- faster is the second query?

#SELECT s.ymd, s.symbol, s.price_close, d.dividend FROM dividends d
JOIN stocks s ON s.ymd = d.ymd AND s.symbol = d.symbol WHERE s.symbol = 'AAPL';

```
hive> SELECT s.ymd, s.symbol, s.price_close, d.dividend
    > FROM dividends d
    > JOIN stocks s ON s.ymd = d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'AAPL';
Query ID = hdfs_20160702212727_0da6af69-edc9-4673-8296-89d01ad092c1
Total jobs = 1
```

```
1992-02-14      AAPL    64.12   0.03
1991-11-18      AAPL    52.13   0.03
1991-08-19      AAPL    50.5    0.03
1991-05-20      AAPL    44.25   0.03
1991-02-15      AAPL    57.63   0.03
1990-11-16      AAPL    35.13   0.03
1990-08-20      AAPL    36.75   0.0275
1990-05-21      AAPL    39.5    0.0275
1990-02-16      AAPL    33.75   0.0275
1989-11-17      AAPL    44.75   0.0275
1989-08-21      AAPL    42.25   0.025
1989-05-22      AAPL    46.0    0.025
1989-02-17      AAPL    36.75   0.025
1988-11-21      AAPL    36.63   0.025
1988-08-15      AAPL    41.25   0.02
1988-05-16      AAPL    41.25   0.02
1988-02-12      AAPL    41.0    0.02
1987-11-17      AAPL    35.0    0.02
1987-08-10      AAPL    48.25   0.015
1987-05-11      AAPL    77.0    0.015
Time taken: 51.83 seconds, Fetched: 35 row(s)
hive>
```

#SELECT /*+ MAPJOIN(d) */ s.ymd, s.symbol, s.price_close, d.dividend FROM dividends d JOIN stocks s ON s.ymd = d.ymd AND s.symbol = d.symbol WHERE s.symbol = 'AAPL';

```
hive> SELECT /*+ MAPJOIN(d) */ s.ymd, s.symbol, s.price_close, d.dividend
    > FROM dividends d
    > JOIN stocks s ON s.ymd = d.ymd AND s.symbol = d.symbol
    > WHERE s.symbol = 'AAPL';
Query ID = hdfs_20160702212828_30c6e766-8190-443c-bd07-468fe378cf1c
Total jobs = 1
```

```
1990-11-16      AAPL    35.13   0.03
1990-08-20      AAPL    36.75   0.0275
1990-05-21      AAPL    39.5    0.0275
1990-02-16      AAPL    33.75   0.0275
1989-11-17      AAPL    44.75   0.0275
1989-08-21      AAPL    42.25   0.025
1989-05-22      AAPL    46.0    0.025
1989-02-17      AAPL    36.75   0.025
1988-11-21      AAPL    36.63   0.025
1988-08-15      AAPL    41.25   0.02
1988-05-16      AAPL    41.25   0.02
1988-02-12      AAPL    41.0    0.02
1987-11-17      AAPL    35.0    0.02
1987-08-10      AAPL    48.25   0.015
1987-05-11      AAPL    77.0    0.015
Time taken: 56.784 seconds, Fetched: 35 row(s)
hive>
```

-- Remember, this is an optimization. Are the results EXACTLY the same?
-- Why or why not? If they are different, how are they different and
-- is this a bad thing?

---------- Ends Here ----------

Complex Data with Hive – 30 Minutes

Demo Array Structure -Create a customers' phones table.

```
CREATE TABLE customers_phones  (cust_id STRING,
name STRING,
phones ARRAY<STRING>)  ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '|';
```

Create a Data File. (/opt/data/cust_phones.txt)

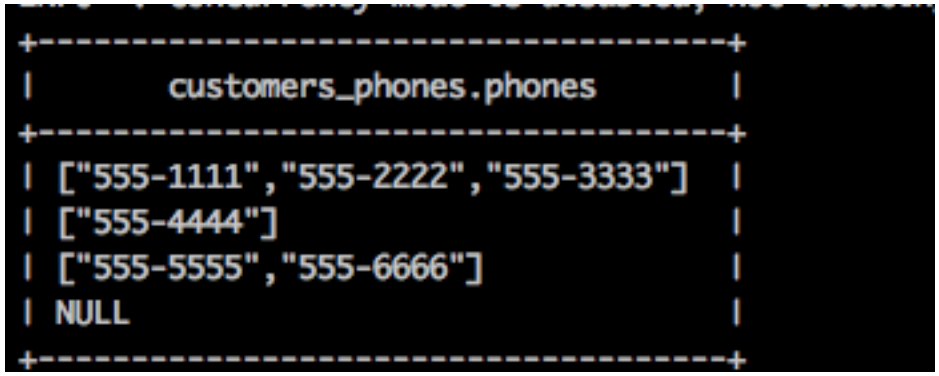
```
a,Alice,555-1111|555-2222|555-3333
b,Bob,555-4444
c,Carlos,555-5555|555-6666
```

Upload the data in the table.

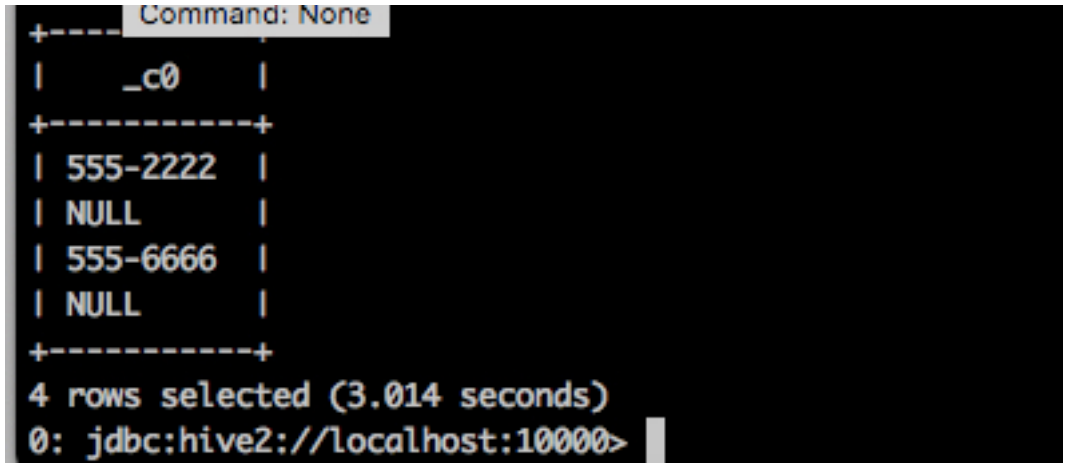
```
LOAD DATA LOCAL INPATH
'/opt/data/cust_phones.txt' OVERWRITE INTO TABLE customers_phones ;
```

Different ways to retrieve Data from Hive Table having Array attribute.

select * FROM customers_phones;
select customers_phones.phones FROM customers_phones;



select customers_phones.phones[1] FROM customers_phones;



Demo – Map Structure.

```
CREATE TABLE customers_phonesm  (cust_id STRING,
name STRING,
phones MAP<STRING,STRING>)  ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '|'  MAP KEYS TERMINATED BY ':';
```

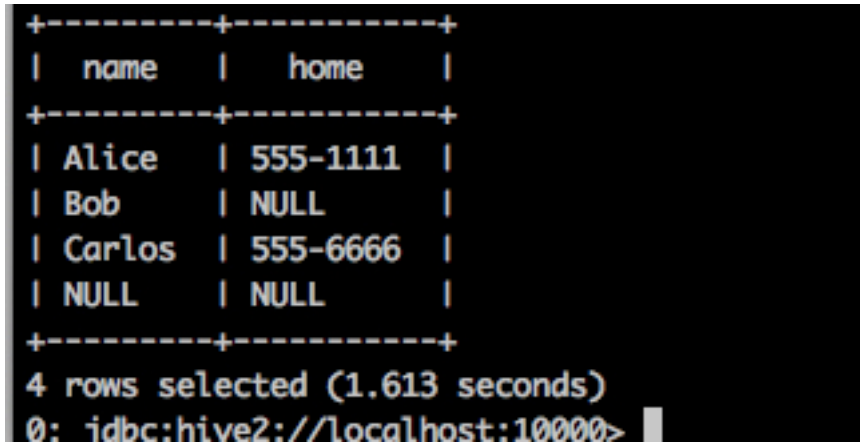
Data File (/opt/data/phones_m.txt)

```
a,Alice,home:555-1111|work:555-2222|mobile:555-3333
b,Bob,mobile:555-4444
c,Carlos,work:555-5555|home:555-6666
```

Load the data file in the table.

```
LOAD DATA LOCAL INPATH
'/opt/data/phones_m.txt' OVERWRITE INTO TABLE customers_phonesm ;
```

```
SELECT name,  phones['home'] AS home
FROM customers_phonesm;
```



Demo Structure.

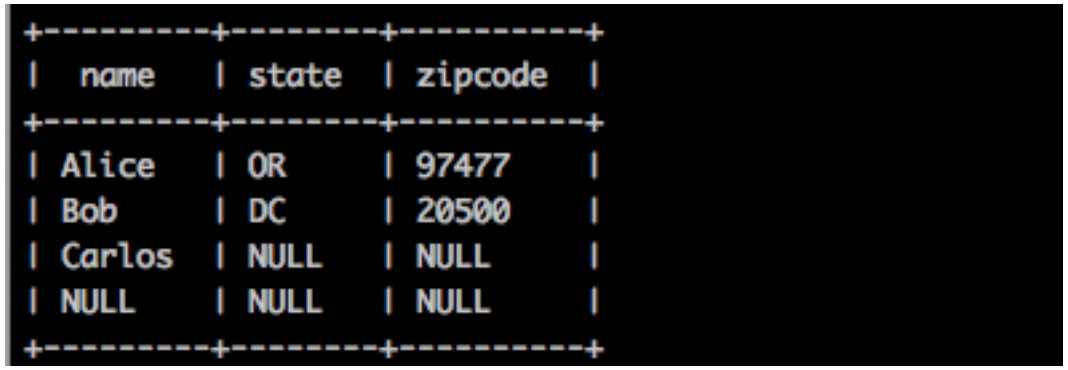
```
CREATE TABLE customers_addr  (cust_id STRING,
name STRING,
address STRUCT<street:STRING,
city:STRING,  state:STRING,  zipcode:STRING>)
ROW FORMAT DELIMITED  FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY '|';
```

Data file : /opt/data/address.txt

```
a,Alice,742 Evergreen Terrace|Springfield|OR|97477
b,Bob,1600 Pennsylvania Ave NW|Washington|DC|20500
c,Carlos,342 Gravelpit Terrace|Bedrock
```

Load the data.
LOAD DATA LOCAL INPATH
'/opt/data/address.txt' OVERWRITE INTO TABLE customers_addr ;

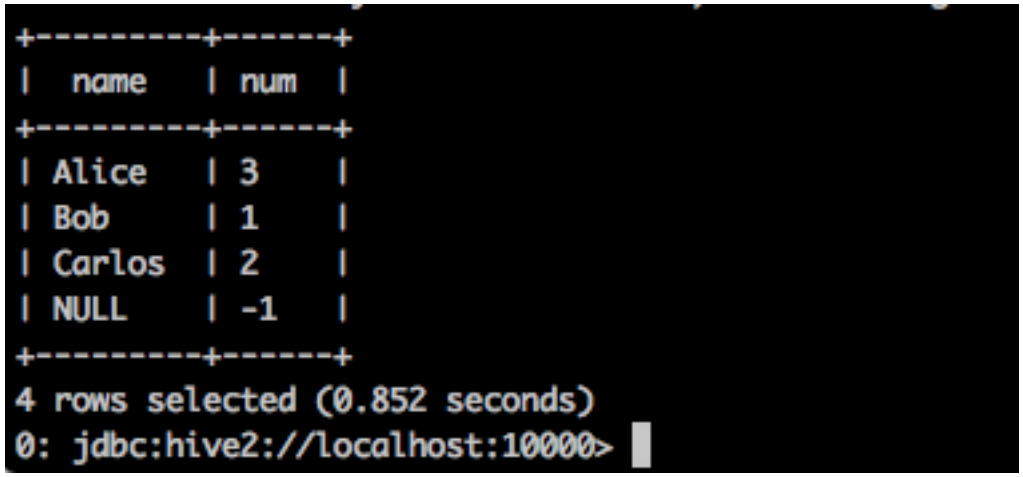
SELECT name, address.state, address.zipcode
FROM customers_addr;



name	state	zipcode
Alice	OR	97477
Bob	DC	20500
Carlos	NULL	NULL
NULL	NULL	NULL

The size function returns the number of items in an ARRAY or MAP

SELECT name, size(phones) AS num FROM customers_phones



The explode function creates a record for each element in an ARRAY
– An example of a *table-generating function*

SELECT explode(phones) AS phone FROM customers_phones;

```
+-----------+
|   phone   |
+-----------+
| 555-1111  |
| 555-2222  |
| 555-3333  |
| 555-4444  |
| 555-5555  |
| 555-6666  |
+-----------+
6 rows selected (0.663 seconds)
```

No other columns can be included in the SELECT list with explode

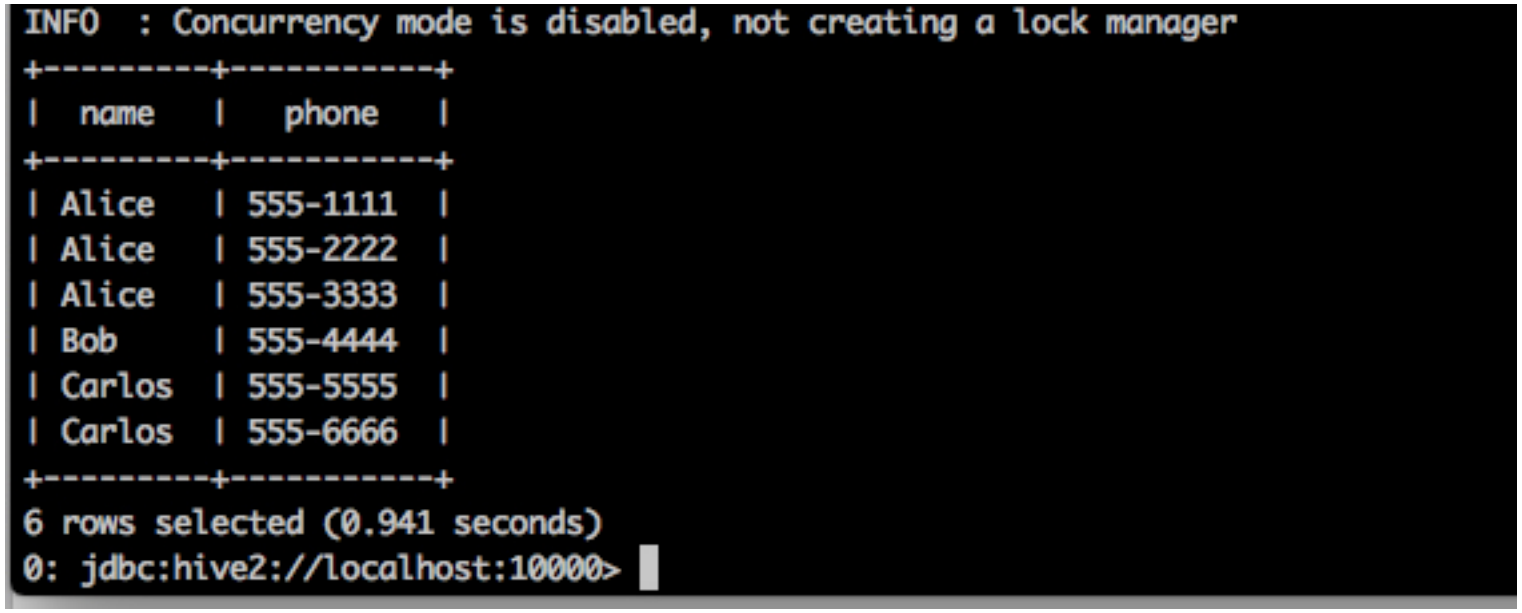
SELECT explode(phones) AS phone , name FROM customers_phones;

```
0: jdbc:hive2://localhost:10000> SELECT explode(phones) AS phone , name  FROM customers_phones;
Error: Error while compiling statement: FAILED: SemanticException 1:34 Only a single expression in the SELECT cl
ause is supported with UDTF's. Error encountered near token 'name' (state=42000,code=40000)
0: jdbc:hive2://localhost:10000>
```

As observe, you can't refer any column other then that of the explode expression.

Alternative using Lateral_View

SELECT name, phone
FROM customers_phones LATERAL VIEW
explode(phones) p AS phone;



---------- Lab Ends here ----------

HIVE transactional & Buckets– 45 Minutes

Let's start by creating a transactional table. Only transactional tables can support updates and deletes.

Hive transaction manager must be set to org.apache.hadoop.hive.ql.lockmgr.DbTxnManager in order to work with ACID tables.

Execute the following in beeline CLI:

```
SET hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
set hive.support.concurrency=true;
```

```
CREATE TABLE hello_acid (key int, value int)
PARTITIONED BY (load_date date)
CLUSTERED BY(key) INTO 3 BUCKETS
STORED AS ORC TBLPROPERTIES ('transactional'='true');
```

```
0: jdbc:hive2://hadoop0:10000> CREATE TABLE hello_acid (key int, value int)
. . . . . . . . . . . . . . .> PARTITIONED BY (load_date date)
. . . . . . . . . . . . . . .> CLUSTERED BY(key) INTO 3 BUCKETS
. . . . . . . . . . . . . . .> STORED AS ORC TBLPROPERTIES ('transactional'='true');
INFO  : Compiling command(queryId=root_20210407083106_4b272515-43c1-4466-ae56-213ad643cdcd): CREATE T
ABLE hello_acid (key int, value int)
PARTITIONED BY (load_date date)
CLUSTERED BY(key) INTO 3 BUCKETS
STORED AS ORC TBLPROPERTIES ('transactional'='true')
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:null, properties:null)
INFO  : Completed compiling command(queryId=root_20210407083106_4b272515-43c1-4466-ae56-213ad643cdcd)
; Time taken: 0.446 seconds
INFO  : Executing command(queryId=root_20210407083106_4b272515-43c1-4466-ae56-213ad643cdcd): CREATE T
ABLE hello_acid (key int, value int)
PARTITIONED BY (load_date date)
CLUSTERED BY(key) INTO 3 BUCKETS
STORED AS ORC TBLPROPERTIES ('transactional'='true')
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=root_20210407083106_4b272515-43c1-4466-ae56-213ad643cdcd)
; Time taken: 0.252 seconds
INFO  : OK
No rows affected (1.198 seconds)
0: jdbc:hive2://hadoop0:10000>
```

Here is an example that inserts some records, deletes one record and updates one record.

```
INSERT INTO hello_acid partition (load_date='2016-03-01') VALUES (1, 1);
INSERT INTO hello_acid partition (load_date='2016-03-02') VALUES (2, 2);
INSERT INTO hello_acid partition (load_date='2016-03-03') VALUES (3, 3);
SELECT * FROM hello_acid;
```

Next, let's delete and update data in the same window execution:

```
DELETE FROM hello_acid WHERE key = 2;
UPDATE hello_acid SET value = 10 WHERE key = 3;
SELECT * FROM hello_acid;
```

```
INFO  : Completed executing command(queryId=root_20210408071213_dfad54ec-d8ff-4f8e-9d49-d96e04003110); Time taken: 0
.0 seconds
INFO  : OK
+-----------------+-------------------+-----------------------+
| hello_acid.key  | hello_acid.value  | hello_acid.load_date  |
+-----------------+-------------------+-----------------------+
| 1               | 1                 | 2016-03-01            |
| 3               | 10                | 2016-03-03            |
+-----------------+-------------------+-----------------------+
2 rows selected (1.004 seconds)
0: jdbc:hive2://hadoop0:10000>
```

```
show transactions;
```

ACID transactions create a number of locks during the course of their operation. Transactions and their locks can be viewed using a number of tools within Hive.

```
.038 seconds
INFO  : OK
+----------------+-------------------+---------------+----------------------+-------+-----------+
|      txnid     |       state       |  startedtime  |  lastheartbeattime   | user  |   host    |
+----------------+-------------------+---------------+----------------------+-------+-----------+
| Transaction ID | Transaction State | Started Time  | Last Heartbeat Time  | User  | Hostname  |
| 2              | ABORTED           | 1617784758708 | 1617786113292        | root  | hadoop0   |
| 3              | ABORTED           | 1617786199580 | 1617787776974        | root  | hadoop0   |
| 4              | ABORTED           | 1617787797525 | 1617789730611        | root  | hadoop0   |
| 7              | ABORTED           | 1617792472620 | 1617794679525        | root  | hadoop0   |
+----------------+-------------------+---------------+----------------------+-------+-----------+
5 rows selected (0.175 seconds)
0: jdbc:hive2://hadoop0:10000>
```

show locks;

This command shows locks, along with their associated transaction IDs. Example:

ACID tables have a hidden column called row__id. You should consider this column a system internal and assume that its name or even its presence may change at any time without warning.

With that out of the way, this column records:

1. The transactionid that was active when the data was inserted or updated.
2. The bucketid, the bucket number where the data lives.
3. The rowid, the rowid within this transaction/bucket combo.

select row__id from hello_acid;

```
INFO  : OK
+-----------------------------------------------+
|                    row__id                    |
+-----------------------------------------------+
| {"writeid":5,"bucketid":536936448,"rowid":0}  |
| {"writeid":9,"bucketid":537001984,"rowid":0}  |
+-----------------------------------------------+
2 rows selected (0.843 seconds)
0: jdbc:hive2://hadoop0:10000> select row__id from hello_acid;
```

A common need is to confirm that all records were ingested. Let's say your upstream provider insists data is missing in Hive. Your provider (e.g. Storm Bolt) can tell you the transaction ID used to insert data. You can count the actual records using the transactionid. Replace X with your transactionid:

```
set hive.optimize.ppd=false;
select count(*) from hello_acid where row__id.writeid = 5;
```

```
INFO  : Stage-Stage-1: Map: 4  Reduce: 1   Cumulative CPU: 29.18 sec   HDFS Read: 45308 HDFS Write: 101 SUCCESS
INFO  : Total MapReduce CPU Time Spent: 29 seconds 180 msec
INFO  : Completed executing command(queryId=root_20210408072529_be29dca2-7ac1-482a-a15a-08332bd3b0de); Time taken: 7
3.776 seconds
INFO  : OK
+------+
| _c0  |
+------+
| 1    |
+------+
1 row selected (74.8 seconds)
0: jdbc:hive2://hadoop0:10000> select count(*) from hello_acid where row__id.writeid = 5;
```

Let us create an explicit bucket Table.

Creates a table supporting 20 buckets based on foo column
Each bucket should contain roughly 5% of the table's data

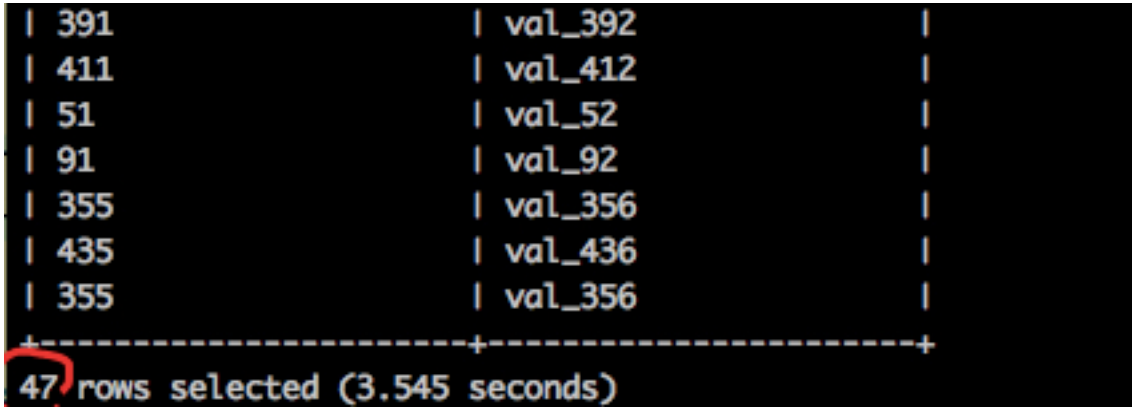
#CREATE TABLE invites_bucketed (foo INT,
bar string)
CLUSTERED BY (foo) INTO 20 BUCKETS;

Bucketing is not automatically enforced when inserting data. Enforce the bucketing and insert that data from the Invites table.

```
SET hive.enforce.bucketing=true;
INSERT OVERWRITE TABLE invites_bucketed SELECT * FROM invites;
```

Use the following syntax to sample data from a bucketed table
– This example selects one of every ten records (10%)

```
SELECT * FROM invites_bucketed TABLESAMPLE (BUCKET 1 OUT OF 10 ON foo);
```



– This example selects one of every hundred records (100%)

SELECT * FROM invites_bucketed TABLESAMPLE (BUCKET 1 OUT OF 100 ON foo);

```
TABLESAMPLE (BUCKET 1 OUT OF 100 ON foo)
INFO  : Completed executing command(queryId=root_20211116095254_843b4dce-dc2b-4ce5-837b-605be848f543); Time take
n: 0.001 seconds
INFO  : OK
+----------------------+----------------------+
| invites_bucketed.foo  | invites_bucketed.bar  |
+----------------------+----------------------+
| 410                  | val_411              |
| 21                   | val_22               |
| 21                   | val_22               |
| 21                   | val_22               |
| 21                   | val_22               |
+----------------------+----------------------+
5 rows selected (2.968 seconds)
```

No of records in total.

#SELECT count(*) FROM invites_bucketed ;

```
INFO  : OK
+------+
| _c0  |
+------+
| 500  |
+------+
1 row selected (1.302 seconds)
0: jdbc:hive2://localhost:10000>
```

Verify the table – File structures.
It created a physical file of 20 buckets.

```
[root@hadoop0 invites_bucketed]# pwd
/opt/data/hive_local/invites_bucketed
[root@hadoop0 invites_bucketed]# ls
000000_0  000002_0  000004_0  000006_0  000008_0  000010_0  000012_0  000014_0  000016_0  000018_0
000001_0  000003_0  000005_0  000007_0  000009_0  000011_0  000013_0  000015_0  000017_0  000019_0
[root@hadoop0 invites_bucketed]#
```

---------- Lab Ends Here ----------

Apache Hive Performance Tuning (Tez & CBO) – 90 Minutes

In this lab, we will demonstrate how to configure tez runtime instead of Map Reduce.
We will also understand how to analyse Hive Query along will collections of table statistics.

- Cost-Based Optimization and Statistics
- Bloom Filters
- Execution and Resource Plans

Before configuration of Tez execute the following for comparison.

Let's use the above two csv files (HVAC.csv & building.csv) to create two new tables using the following step.

Create the following two tables using beeline CLI.

The following queries create the the tables:

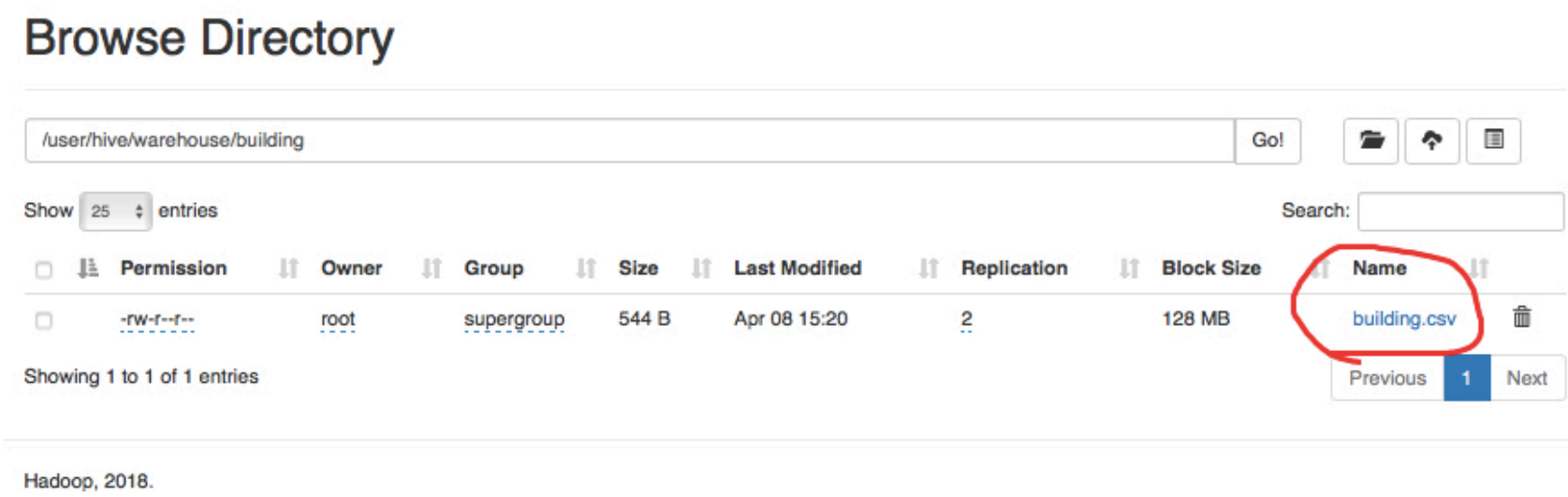
```
#create table hvac (stage STRING,dtime String ,targettemp BIGINT, actualtemp BIGINT, system BIGINT, systemage BIGINT, building_id BIGINT) row format delimited  fields terminated by ',' stored as textfile tblproperties ("skip.header.line.count"="1");
```

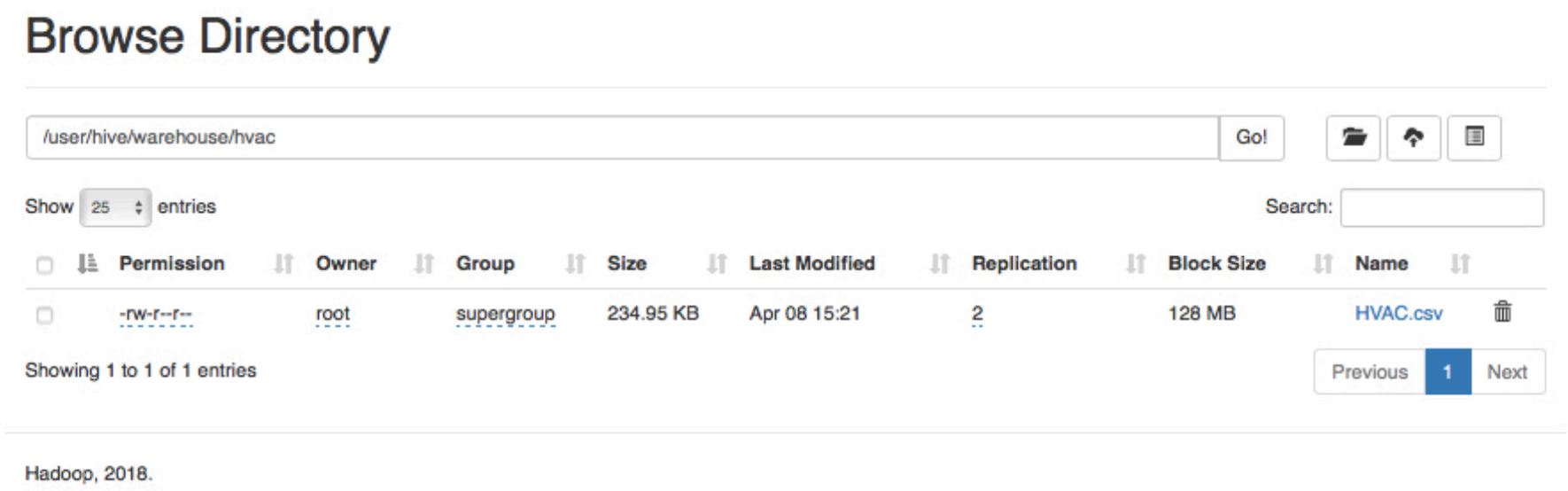
```
#create table building (building_id BIGINT, building_mgr STRING, building_age BIGINT,
hvacproduct STRING, country STRING)
row format delimited
fields terminated by ','
stored as textfile
tblproperties ("skip.header.line.count"="1");
```

Now, your database explorer should show the two tables building and hvac.
To load the data from the csv files into the tables, we execute the following queries.

```
#LOAD DATA LOCAL INPATH '/opt/data/building.csv' OVERWRITE INTO TABLE building;

#LOAD DATA local INPATH '/opt/data/HVAC.csv' OVERWRITE INTO TABLE hvac;
```

To test if the data was loaded correctly,


Browse Directory
/user/hive/warehouse/building
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
-rw-r--r--
root
supergroup
544 B
Apr 08 15:20
2
128 MB
building.csv
Showing 1 to 1 of 1 entries
Previous
1
Next
Hadoop, 2018.



Speed Improvements.

To take a look at the speed improvements of Hive on Tez, we can run some sample queries. For this we will use the above two tables – hvac and building.

Step 1 :

We will run first Hive without Tez.

Please note that Hive is running using MapReduce Framework from the log output on your screen.

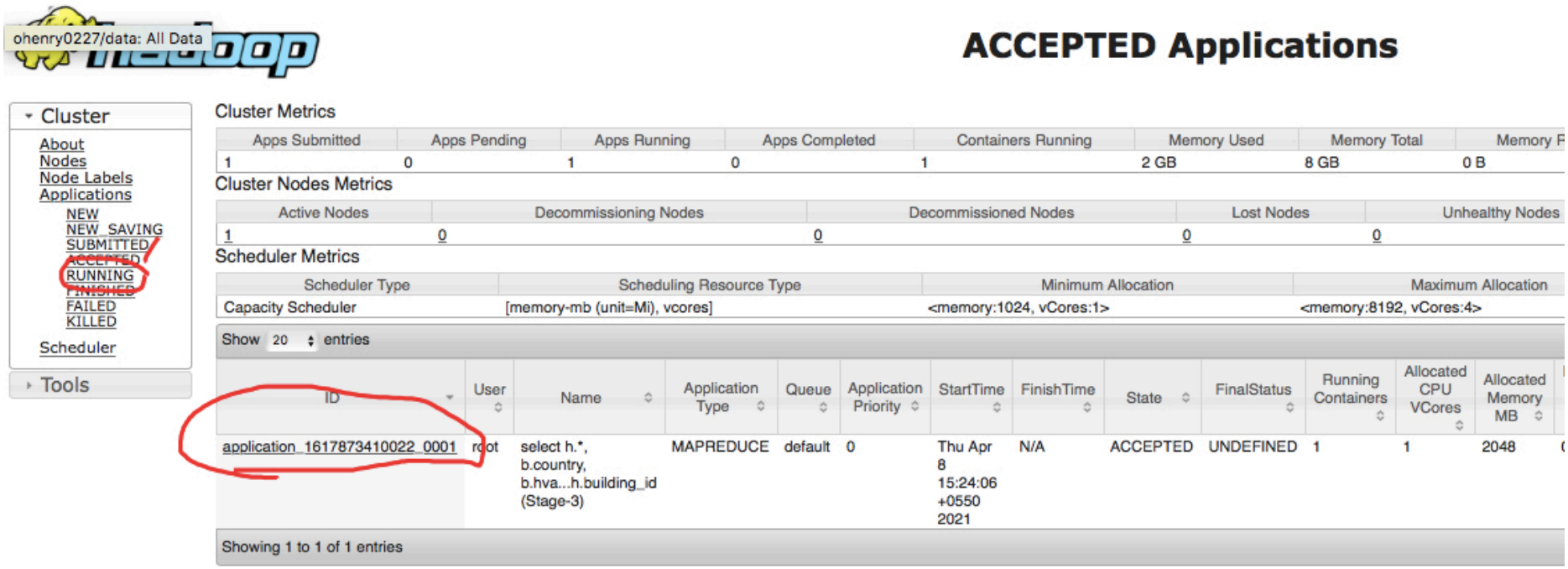
```
#set hive.execution.engine=mr;
```

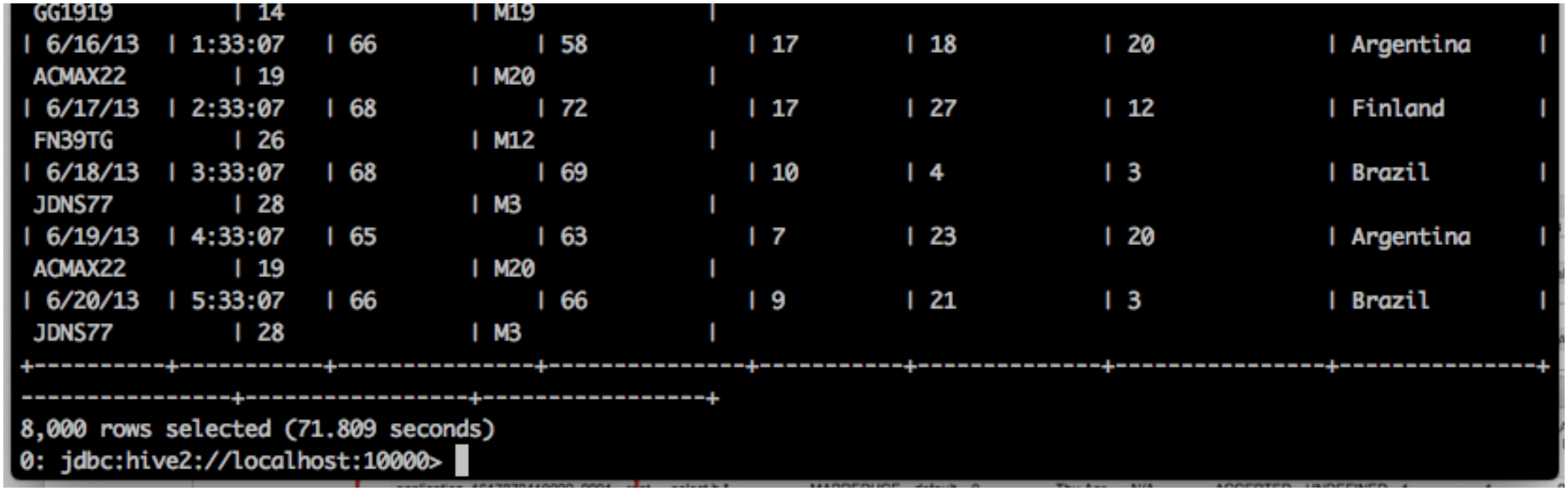
Then, let's execute the hiveql as below.

Set a specific Queue for the Hive if you have configured Queue else skip.

```
#set mapred.job.queue.name=QE;
```

```
#select h.*, b.country, b.hvacproduct, b.building_age, b.building_mgr
from building b join hvac h
on b.building_id = h.building_id;
```


ohenry0227/data: All Data
ACCEPTED Applications
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Cluster Metrics
Apps Submitted
Apps Pending
Apps Running
Apps Completed
Containers Running
Memory Used
Memory Total
Memory F
1
0
1
0
1
2 GB
8 GB
0 B
Cluster Nodes Metrics
Active Nodes
Decommissioning Nodes
Decommissioned Nodes
Lost Nodes
Unhealthy Nodes
1
0
0
0
0
Scheduler Metrics
Scheduler Type
Scheduling Resource Type
Minimum Allocation
Maximum Allocation
Capacity Scheduler
[memory-mb (unit=Mi), vcores]
<memory:1024, vCores:1>
<memory:8192, vCores:4>
Show 20 entries
ID
User
Name
Application Type
Queue
Application Priority
StartTime
FinishTime
State
FinalStatus
Running Containers
Allocated CPU VCores
Allocated Memory MB
application_1617873410022_0001
root
select h.*, b.country, b.hva...h.building_id (Stage-3)
MAPREDUCE
default
0
Thu Apr 8 15:24:06 +0550 2021
N/A
ACCEPTED
UNDEFINED
1
1
2048
Showing 1 to 1 of 1 entries



This query was run using the MapReduce framework. Note the time it takes to execute the query. In the example above it took 71.8 seconds.

Now let us configure TEz.

Download the following tez.

https://downloads.apache.org/tez/0.9.2/

```
apache-tez-0.9.2-bin.tar.gz
```

Extract the compress file using tar -xvf
#mv apa* tez

Perform the following on the Server side i.e (Master Node)

Copy full tarball on HDFS cluster with the following command.

#cd /opt

```
$ hdfs dfs -mkdir -p /apps/tez
$ hdfs dfs -copyFromLocal tez/share/tez.tar.gz /apps/tez
```

Note: apache-tez-0.9.2-bin.tar.gz is not the one to upload but ${TEZ_HOME}/share/tez.tar.gz is the one. see here for the detail.

Workaround of HDFS-12920
To avoid the issue, following configuration was required on ${HADOOP_HOME}/conf/hdfs-site.xml.

```
<property>
  <name>dfs.namenode.decommission.interval</name>
  <value>30</value>
</property>
<property>
  <name>dfs.client.datanode-restart.timeout</name>
  <value>30</value>
```

```
</property>
```

Perform the TEZ_HOME configuration.
On the CLI, in which hiver server will be started.

```
export TEZ_HOME=/opt/tez
```

Set Hive configuration as follows at the last line of the file and start hiveserver2. Stop the Hive if started earlier.

```
${HIVE_HOME}/conf/hive-env.sh
```

```
export
HADOOP_CLASSPATH=${TEZ_HOME}/conf:${TEZ_HOME}/*:${TEZ_HOME}/lib/*:${HADO
OP_CLASSPATH}
```

And specify the above from the CLI too.

```
#hiverserver2
```

On the Client side : Since we have only one Nod (Configure it on the Master terminal only)

To try tez applications, following configurations are required on a client.

- Create tez-site.xml on ${TEZ_HOME}/conf

```
<configuration>
  <property>
    <name>tez.lib.uris</name>
    <value>${fs.defaultFS}/apps/tez/tez.tar.gz</value>
  </property>
</configuration>
```

Note: tez-default-template.xml is a template of configuration file.

Connect using beeline – Open a terminal.
#export TEZ_HOME=/opt/tez

```
#export
HADOOP_CLASSPATH=${TEZ_HOME}/conf:${TEZ_HOME}/*:${TEZ_HOME}/lib/*:${HADOOP_CLASSPATH}
```

#beeline -u jdbc:hive2://hadoop0:10000

```
[root@hadoop0 /]# export HADOOP_CLASSPATH=${TEZ_HOME}/conf:${TEZ_HOME}/*:${TEZ_HOME}/lib/*
[root@hadoop0 /]# echo $HADOOP_CLASSPATH
/opt/tez/conf:/opt/tez/*:/opt/tez/lib/*
[root@hadoop0 /]# beeline -u jdbc:hive2:/hadoop0:10000
SLF4J: Class path contains multiple SLF4J bindings.
```

```
[root@hadoop0 /]# beeline -u jdbc:hive2://hadoop0:10000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinde
r.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/tez/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinde
r.class]
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerB
inder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Connecting to jdbc:hive2://hadoop0:10000
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 3.1.2 by Apache Hive
0: jdbc:hive2://hadoop0:10000>
```

Hive on Tez

1. Enable Hive to use Tez DAG APIs. On the Hive client machine, add the following to your Hive script or execute it in the Hive shell:

```
set hive.execution.engine=tez;
```

Step 2 :

Now we can enable Hive on Tez execution and take advantage of Directed Acyclic Graph (DAG) execution representing the query instead of multiple stages of MapReduce program which involved a lot of synchronization, barriers and IO overheads. This is improved in Tez, by writing intermediate data set into memory instead of hard disk.
Use the following step to set the execution engine to Tez:

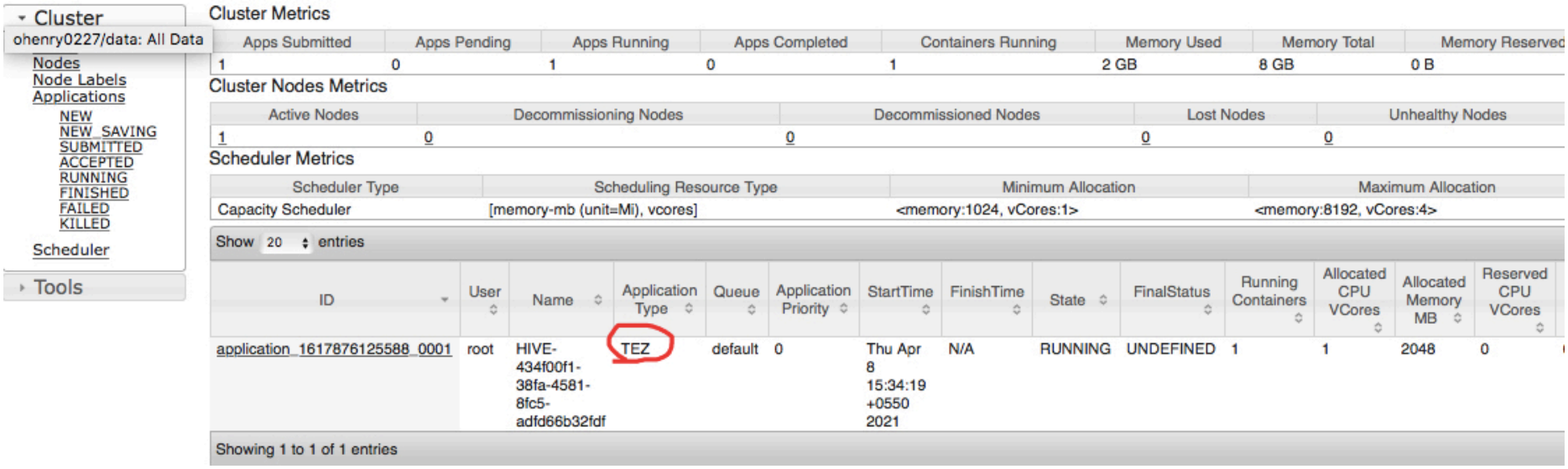
#set hive.execution.engine=tez;

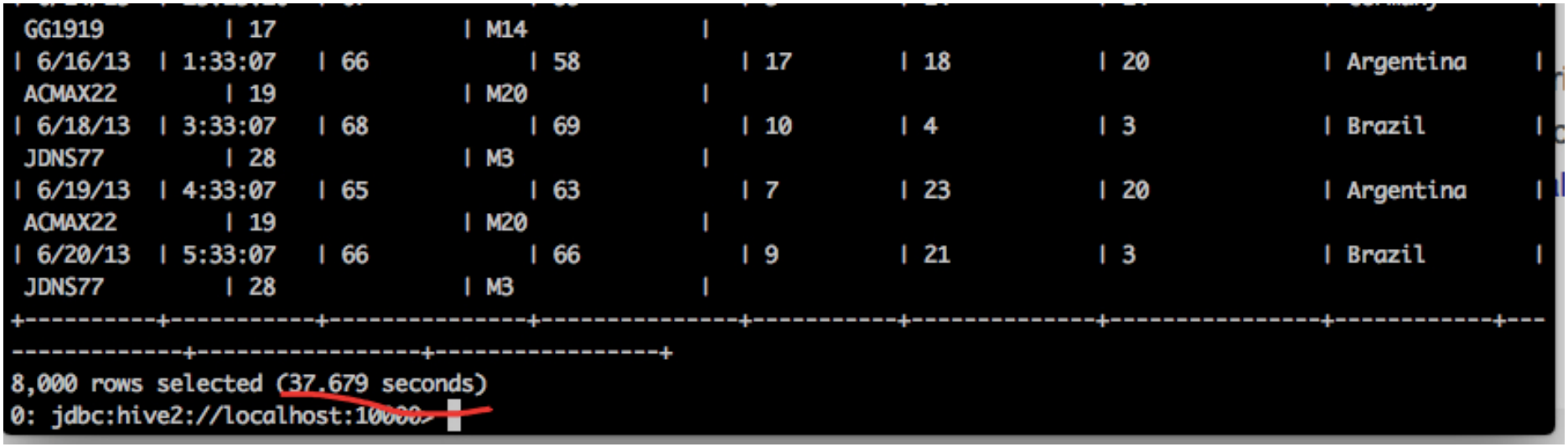
#set mapred.job.queue.name=QE;

Run the same query as we had run earlier in Step 1, to see if the speed has improved or not.

```
#select h.*, b.country, b.hvacproduct, b.building_age, b.building_mgr
from building b join hvac h
on b.building_id = h.building_id;
```

Check the output of this job. It shows the usage of the containers.





This query was run using the Tez framework. Note the time it takes by the query to execute. In the example above it took 37.7 seconds.

Let's create a table with ORC file format as follows:

#create table hvac_orc stored as orc as select * from hvac;

Step 2:

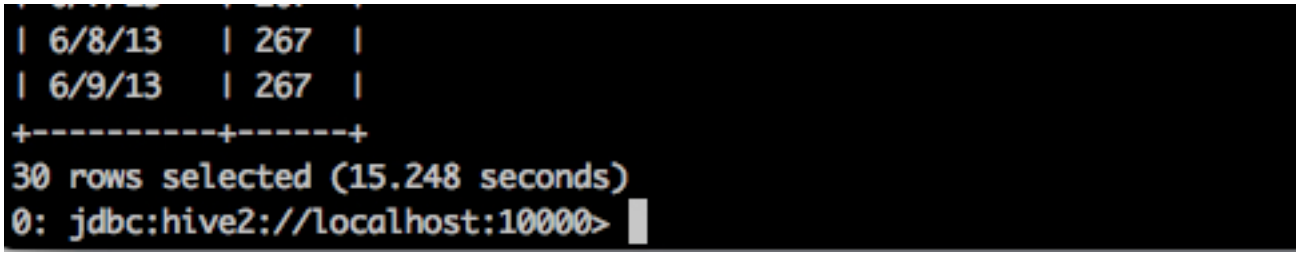
Run the following statement to enable Tez.

set hive.execution.engine=tez;

Step 3:
Run the following query.

#select stage, count(building_id) from hvac group by stage;

Note down the time taken.

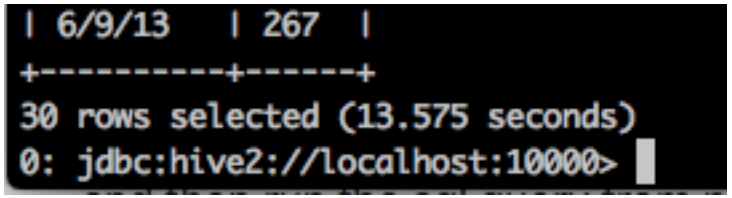


Step 4:

Now let's run the same sql query as before:

#select stage, count(building_id) from hvac_orc group by stage;

Note down the time taken and compare to step 3.



Step 5:

Now let's run the following steps to enable vectorization:

#set hive.vectorized.execution.enabled;

and then run the sql query from previous step

#select stage, count(building_id) from hvac_orc group by stage;

This time it runs with a vectorized query plan, which scales very well especially with large datasets.

```
| 6/7/13    | 267  |
| 6/8/13    | 267  |
| 6/9/13    | 267  |
+-----------+------+
30 rows selected (14.017 seconds)
0: jdbc:hive2://localhost:10000> select stage, count(building_id) from hvac_orc group by stage;
```

rerun again.

```
| 6/9/13    | 267  |
+-----------+------+
30 rows selected (1.817 seconds)
0: jdbc:hive2://localhost:10000>
```

As you can see, it reached up till 1.8 sec aprox.

Step 6:

Let's look at the 'explain' plan to confirm that it is indeed using a vectorized query plan:

#explain select stage, count(building_id) from hvac_orc group by stage;

```
INFO  : Concurrency mode is disabled, not creating a lock manager
+----------------------------------------------------+
|                      Explain                       |
+----------------------------------------------------+
| Plan optimized by CBO.                             |
|                                                    |
| Vertex dependency in root stage                    |
| Reducer 2 <- Map 1 (SIMPLE_EDGE)                   |
|                                                    |
| Stage-0                                            |
|   Fetch Operator                                   |
|     limit:-1                                       |
|     Stage-1                                        |
|       Reducer 2 vectorized                         |
|       File Output Operator [FS_11]                 |
|         Group By Operator [GBY_10] (rows=4000 width=221) |
|           Output:["_col0","_col1"],aggregations:["count(VALUE._col0)"],keys:KEY._col0 |
|         <-Map 1 [SIMPLE_EDGE] vectorized           |
|           SHUFFLE [RS_9]                           |
|             PartitionCols:_col0                    |
|             Group By Operator [GBY_8] (rows=8000 width=221) |
|               Output:["_col0","_col1"],aggregations:["count(building_id)"],keys:stage |
|               Select Operator [SEL_7] (rows=8000 width=221) |
|                 Output:["stage","building_id"]     |
|                 TableScan [TS_0] (rows=8000 width=221) |
|                   default@hvac_orc,hvac_orc,Tbl:COMPLETE,Col:NONE,Output:["stage","building_id"] |
|                                                    |
+----------------------------------------------------+
23 rows selected (0.492 seconds)
0: jdbc:hive2://localhost:10000> explain select stage, count(building_id) from hvac_orc group by stage;
```

Please note that in the explain plan, the Execution mode is “vectorized”. When this feature is switched off, you will not see the same line in the plan.

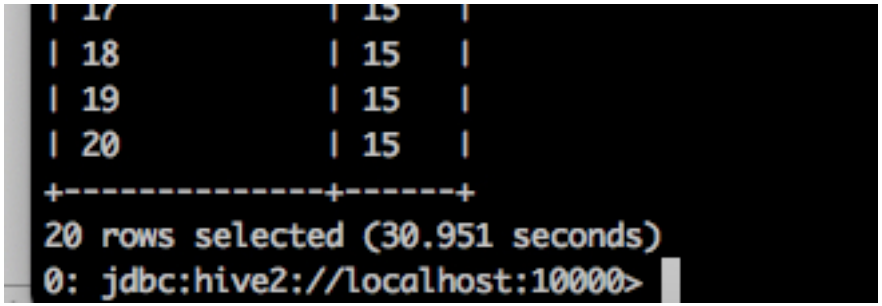
Stats & Cost Based Optimization (CBO)

Step 1:

Let's do a simple exercise. Let's run the following query and see how long it takes.

#select building_id, max(targettemp-actualtemp) from hvac group by building_id;

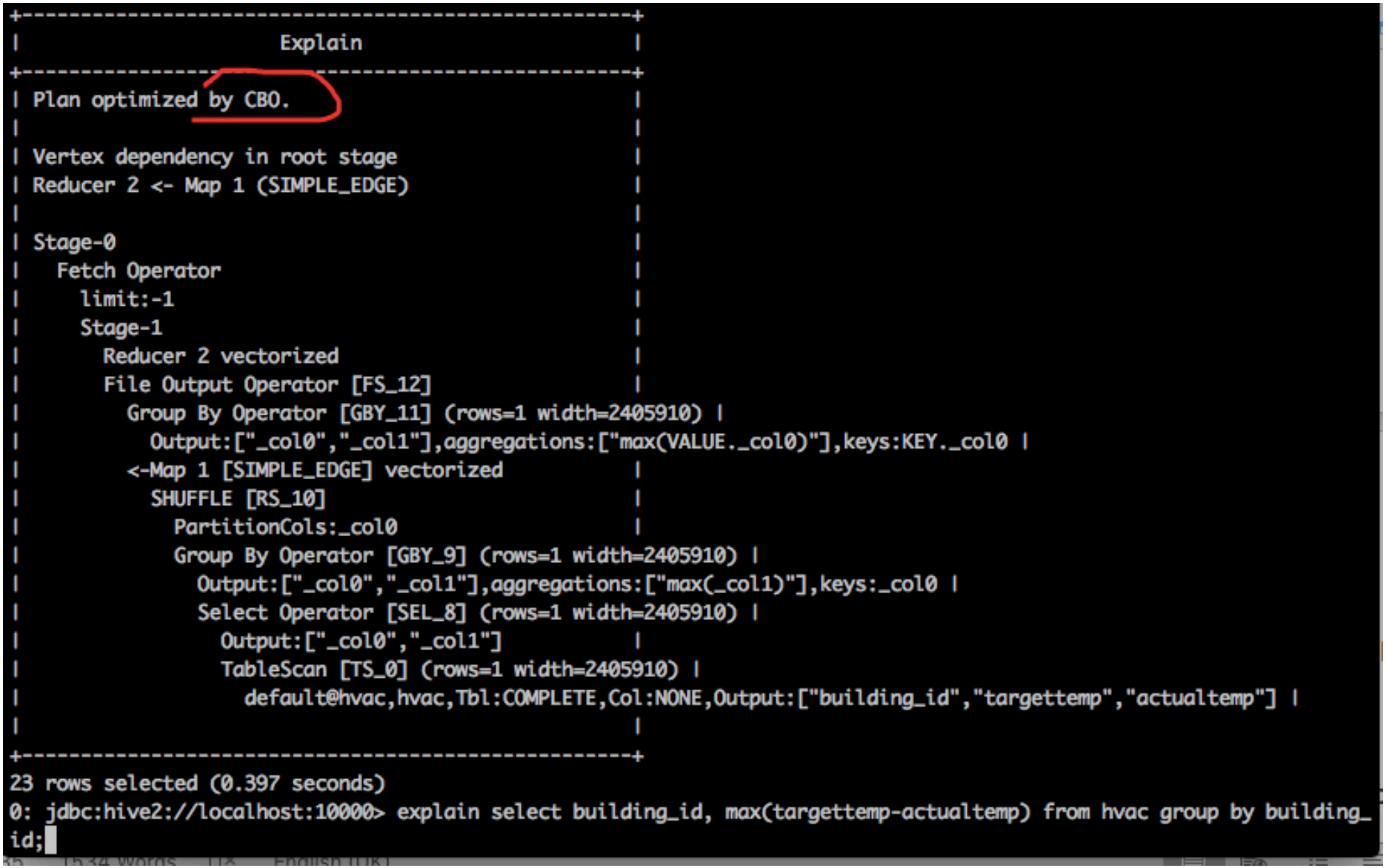
Please note down the time taken.



Step 2:

Now, let's explain the above query in Step 1.

#explain select building_id, max(targettemp-actualtemp) from hvac group by building_id;



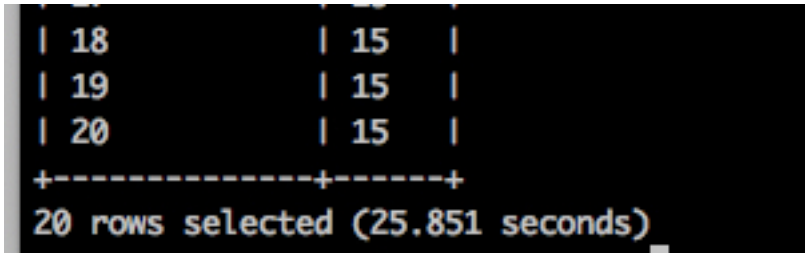
```
+----------------------------------------------------+
|                      Explain                       |
+----------------------------------------------------+
| Plan optimized by CBO.                             |
|                                                    |
| Vertex dependency in root stage                    |
| Reducer 2 <- Map 1 (SIMPLE_EDGE)                   |
|                                                    |
| Stage-0                                            |
|   Fetch Operator                                   |
|     limit:-1                                       |
|     Stage-1                                        |
|       Reducer 2 vectorized                         |
|       File Output Operator [FS_12]                 |
|         Group By Operator [GBY_11] (rows=1 width=2405910) |
|           Output:["_col0","_col1"],aggregations:["max(VALUE._col0)"],keys:KEY._col0 |
|         <-Map 1 [SIMPLE_EDGE] vectorized           |
|           SHUFFLE [RS_10]                          |
|             PartitionCols:_col0                    |
|             Group By Operator [GBY_9] (rows=1 width=2405910) |
|               Output:["_col0","_col1"],aggregations:["max(_col1)"],keys:_col0 |
|               Select Operator [SEL_8] (rows=1 width=2405910) |
|                 Output:["_col0","_col1"]           |
|                 TableScan [TS_0] (rows=1 width=2405910) |
|                   default@hvac,hvac,Tbl:COMPLETE,Col:NONE,Output:["building_id","targettemp","actualtemp"] |
|                                                    |
+----------------------------------------------------+
23 rows selected (0.397 seconds)
0: jdbc:hive2://localhost:10000> explain select building_id, max(targettemp-actualtemp) from hvac group by building_
id;
```

Please note the the CBO feature is used.

Now let us disable the CBO feature.

```
#set hive.cbo.enable=false;
```

Execute the query now.
#select building_id, max(targettemp-actualtemp) from hvac group by building_id;



and perform the explain to the above query.

#explain select building_id, max(targettemp-actualtemp) from hvac group by building_id;

```
+----------------------------------------------------+
|                      Explain                       |
+----------------------------------------------------+
| Vertex dependency in root stage                    |
| Reducer 2 <- Map 1 (SIMPLE_EDGE)                   |
|                                                    |
| Stage-0                                            |
|   Fetch Operator                                   |
|     limit:-1                                       |
|     Stage-1                                        |
|       Reducer 2 vectorized                         |
|       File Output Operator [FS_11]                 |
|         Group By Operator [GBY_10] (rows=1 width=2405910) |
|           Output:["_col0","_col1"],aggregations:["max(VALUE._col0)"],keys:KEY._col0 |
|         <-Map 1 [SIMPLE_EDGE] vectorized           |
|           SHUFFLE [RS_9]                           |
|             PartitionCols:_col0                    |
|             Group By Operator [GBY_8] (rows=1 width=2405910) |
|               Output:["_col0","_col1"],aggregations:["max((targettemp - actualtemp))"],keys:building_id |
|               Select Operator [SEL_7] (rows=1 width=2405910) |
|                 Output:["targettemp","actualtemp","building_id"] |
|                 TableScan [TS_0] (rows=1 width=2405910) |
|                   default@hvac,hvac,Tbl:COMPLETE,Col:NONE,Output:["targettemp","actualtemp","building_id"] |
|                                                    |
+----------------------------------------------------+
21 rows selected (0.229 seconds)
0: jdbc:hive2://localhost:10000>
```

No CBO use.

Step 3:
Now, we will tune the same query so that it uses Cost Based Optimization (CBO). Let's collect statistics on the table hvac.

#analyze table hvac compute statistics;

You are using Tez execution engine.

```
0: jdbc:hive2://localhost:10000> analyze table hvac compute statistics;
INFO  : Compiling command(queryId=root_20210408104037_2e6cf5c5-1895-4a2f-96af-7ce4849c6496): analyze table hvac comp
ute statistics
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:hvac.stage, type:string, comment:null), FieldSc
hema(name:hvac.dtime, type:string, comment:null), FieldSchema(name:hvac.targettemp, type:bigint, comment:null), Fiel
dSchema(name:hvac.actualtemp, type:bigint, comment:null), FieldSchema(name:hvac.system, type:bigint, comment:null),
FieldSchema(name:hvac.systemage, type:bigint, comment:null), FieldSchema(name:hvac.building_id, type:bigint, comment
:null)], properties:null)
INFO  : Completed compiling command(queryId=root_20210408104037_2e6cf5c5-1895-4a2f-96af-7ce4849c6496); Time taken: 0
.175 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20210408104037_2e6cf5c5-1895-4a2f-96af-7ce4849c6496): analyze table hvac comp
ute statistics
INFO  : Query ID = root_20210408104037_2e6cf5c5-1895-4a2f-96af-7ce4849c6496
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-0:MAPRED] in serial mode
INFO  : Subscribed to counters: [] for queryId: root_20210408104037_2e6cf5c5-1895-4a2f-96af-7ce4849c6496
INFO  : Session is already open
INFO  : Dag name: analyze table hvac compute statistics (Stage-0)
INFO  : Status: Running (Executing on YARN cluster with App id application_1617876125588_0008)

INFO  : Starting task [Stage-2:STATS] in serial mode
INFO  : Completed executing command(queryId=root_20210408104037_2e6cf5c5-1895-4a2f-96af-7ce4849c6496); Time taken: 9
.837 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (10.048 seconds)
0: jdbc:hive2://localhost:10000>
```

Step 4:
Let's collect statistics of a few columns in this table hvac. To use CBO, column level statistics are required.
#analyze table hvac compute statistics for columns targettemp, actualtemp, building_id;

```
INFO  : Compiling command(queryId=root_20210408104120_ca756f6f-2c5b-4dd5-b4b9-098668746535): analyze table hvac comp
ute statistics for columns targettemp, actualtemp, building_id
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:_c0, type:struct<columntype:string,min:bigint,m
ax:bigint,countnulls:bigint,numdistinctvalues:bigint,ndvbitvector:binary>, comment:null), FieldSchema(name:_c1, type
:struct<columntype:string,min:bigint,max:bigint,countnulls:bigint,numdistinctvalues:bigint,ndvbitvector:binary>, com
ment:null), FieldSchema(name:_c2, type:struct<columntype:string,min:bigint,max:bigint,countnulls:bigint,numdistinctv
alues:bigint,ndvbitvector:binary>, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=root_20210408104120_ca756f6f-2c5b-4dd5-b4b9-098668746535); Time taken: 0
.288 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20210408104120_ca756f6f-2c5b-4dd5-b4b9-098668746535): analyze table hvac comp
ute statistics for columns targettemp, actualtemp, building_id
INFO  : Query ID = root_20210408104120_ca756f6f-2c5b-4dd5-b4b9-098668746535
INFO  : Total jobs = 1
INFO  : Launching Job 1 out of 1
INFO  : Starting task [Stage-0:MAPRED] in serial mode
INFO  : Subscribed to counters: [] for queryId: root_20210408104120_ca756f6f-2c5b-4dd5-b4b9-098668746535
INFO  : Session is already open
INFO  : Dag name: analyze table hvac compute sta...building_id (Stage-0)
INFO  : Status: Running (Executing on YARN cluster with App id application_1617876125588_0008)

INFO  : Starting task [Stage-2:STATS] in serial mode
INFO  : Completed executing command(queryId=root_20210408104120_ca756f6f-2c5b-4dd5-b4b9-098668746535); Time taken: 1
2.355 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
No rows affected (12.672 seconds)
0: jdbc:hive2://localhost:10000>
```

Step 5:
Now let's set the 4 settings in hive as follows and run explain on the query.

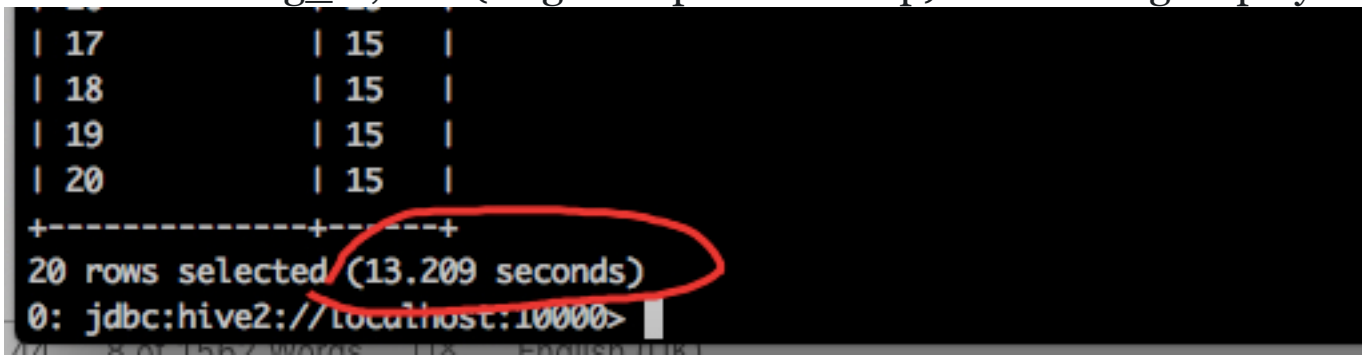
```
#set hive.compute.query.using.stats=true;
#set hive.cbo.enable=true;
#set hive.stats.fetch.column.stats=true;

#explain select building_id, max(targettemp-actualtemp)
from hvac group by building_id;
```

Note that the Plan says that it is using stats now.

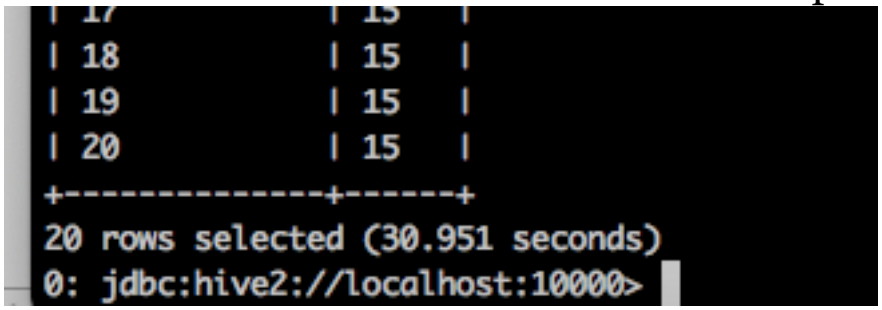
```
+----------------------------------------------------+
|                      Explain                       |
+----------------------------------------------------+
| Plan optimized by CBO.                             |
|                                                    |
| Vertex dependency in root stage                    |
| Reducer 2 <- Map 1 (SIMPLE_EDGE)                   |
|                                                    |
| Stage-0                                            |
|   Fetch Operator                                   |
|     limit:-1                                       |
|     Stage-1                                        |
|       Reducer 2 vectorized                         |
|       File Output Operator [FS_12]                 |
|         Group By Operator [GBY_11] (rows=20 width=16) |
|           Output:["_col0","_col1"],aggregations:["max(VALUE._col0)"],keys:KEY._col0 |
|         <-Map 1 [SIMPLE_EDGE] vectorized           |
|           SHUFFLE [RS_10]                          |
|             PartitionCols:_col0                    |
|             Group By Operator [GBY_9] (rows=20 width=16) |
|               Output:["_col0","_col1"],aggregations:["max(_col1)"],keys:_col0 |
|               Select Operator [SEL_8] (rows=8000 width=24) |
|                 Output:["_col0","_col1"]           |
|                 TableScan [TS_0] (rows=8000 width=24) |
|                   default@hvac,hvac,Tbl:COMPLETE,Col:COMPLETE,Output:["building_id","targettemp","actualtemp"] |
|                                                    |
+----------------------------------------------------+
23 rows selected (0.46 seconds)
0: jdbc:hive2://localhost:10000>
```

Step 6:
Let's rerun the query now and observe if it runs faster. You will see better gain with a good volume of dataset than the one we are working with.

#select building_id, max(targettemp-actualtemp) from hvac group by building_id;



Please note down total time taken and compare the result of Step 1.



Step 1.

In my case, the step 1 took 30 secs approx. and after CBO it took about 13 seconds, a considerable improvement which will be tangible when data size is huge.

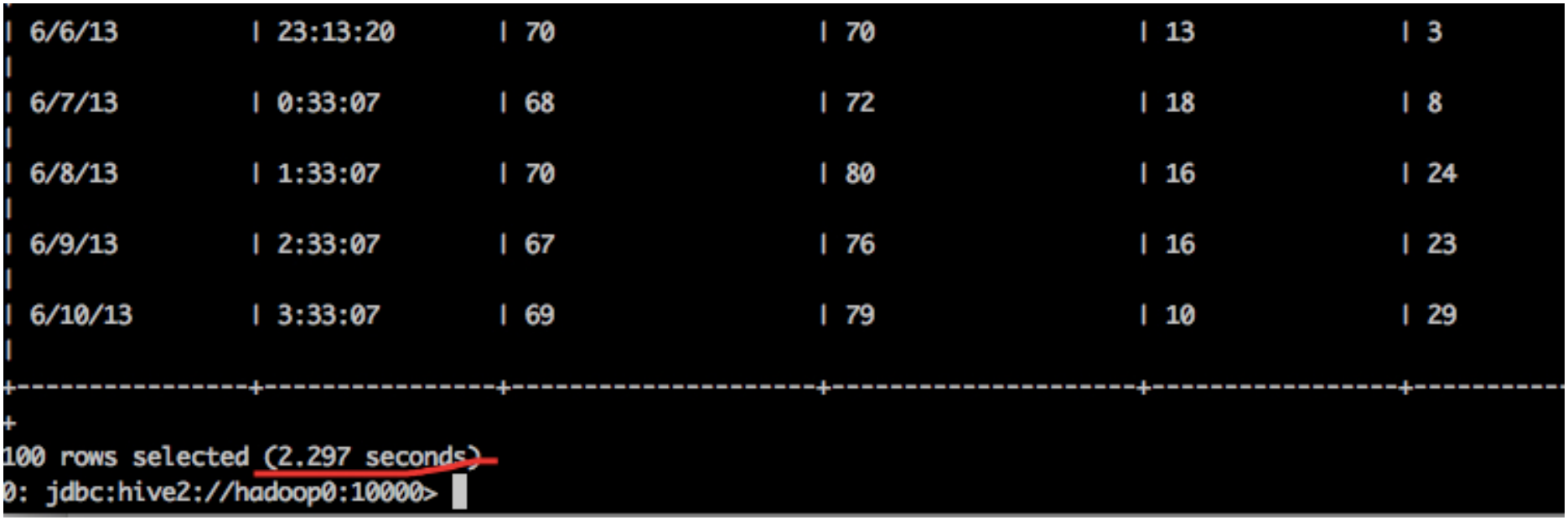
Let us define a table with Boomfilter.

Let's create a table with ORC file format having bloomfilter as follows:

```
#create table hvac_bf stored as orc TBLPROPERTIES (
            "orc.compress"="SNAPPY",
            "orc.bloom.filter.columns"="system",
            "orc.create.index" = "true"
           ) as select * from hvac;
```

Execute a select Query on the above table.

```
# select * from  hvac_bf limit 100;
```



To get more stats on the execution.

```
# set hive.tez.exec.print.summary=true;
```

At the end profile is a shown below:

```
export HADOOP_HOME=/opt/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
export HDFS_NAMENODE_USER="root"
export HDFS_DATANODE_USER="root"
```

```
export HDFS_SECONDARYNAMENODE_USER="root"
export YARN_RESOURCEMANAGER_USER="root"
export YARN_NODEMANAGER_USER="root"
export JAVA_HOME=/opt/jdk

export JAVA_HOME=/opt/jdk
export HADOOP_HOME=/opt/hadoop
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/sbin:$HADOOP_HOME/bin

export HIVE_HOME=/opt/hive
export PATH=$HIVE_HOME/bin:$PATH
export HIVE_CONF_DIR=/opt/hive/conf
export HIVE_HOME=/opt/hive
export PATH=$HIVE_HOME/bin:$PATH
export HIVE_CONF_DIR=/opt/hive/conf
export DERBY_INSTALL=/opt/derby
export DERBY_HOME=/opt/derby

export TEZ_HOME=/opt/tez
export HADOOP_CONF_DIR=/opt/hadoop/etc/hadoop
```

Reference:

https://www.keisuke.dev/2020/06/11/setup_hive3.html

https://gist.github.com/yawi/d81512c1ca185dd6a9cc
https://sungsoo.github.io/2014/04/12/tez-tutorials.html
https://dwgeek.com/apache-hive-explain-command-example.html/
https://towardsdatascience.com/apache-hive-optimization-techniques-2-e60b6200eeca

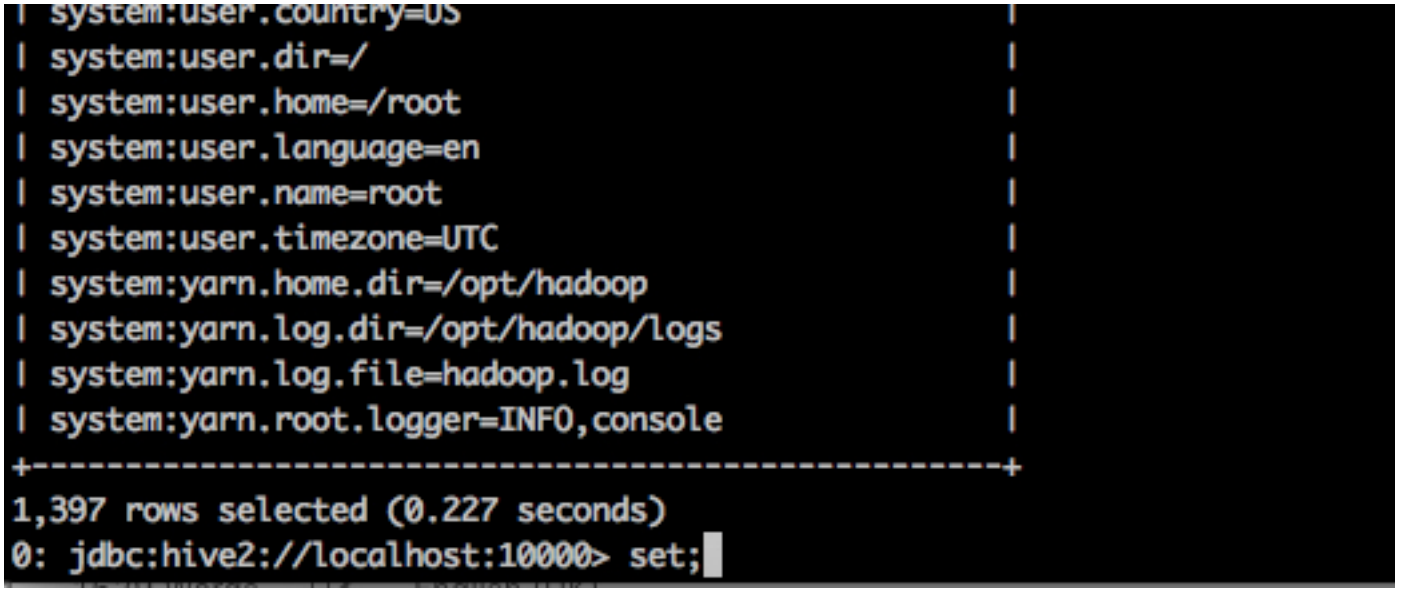
---------- Lab Ends Here ----------

Hive config

Enable Hive to use Tez DAG APIs. On the Hive client machine, add the following to your Hive script or execute it in the Hive shell:

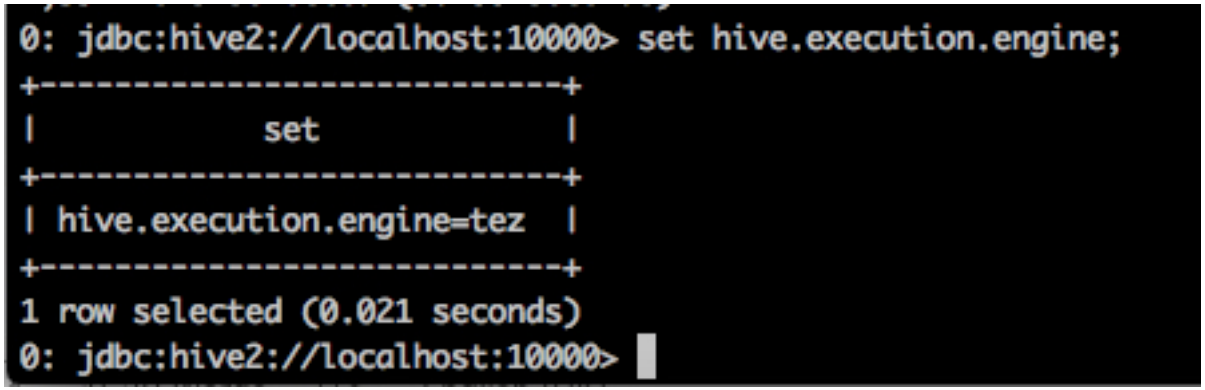
```
set hive.execution.engine=tez;
```

To get all the parameters:
#set;



To get a specific parameter.

#set hive.execution.engine;



https://github.com/apache/incubator-tez/blob/branch-0.2.0/INSTALL.txt

Executing Hive in Local Mode

Use /opt/hive_local folder.

Set the HADOOP_HOME
export HADOOP_HOME=/opt/hadoop

Update hive-site.xml

----- Config Begin ------------

```
<?xml version="1.0"?> <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
     <property>
       <name>hive.metastore.schema.verification</name>
       <value>false</value>
     </property>
    <property>
     <!-- this should eventually be deprecated since the metastore should supply this -->
       <name>hive.metastore.warehouse.dir</name>
       <value>file:///opt/data/hive_local</value>
       <description></description>
    </property>
    <property>
       <name>fs.default.name</name>
       <value>file:///opt/data/hive_local</value>
    </property>
</configuration>
```

----- Config Ends ------------

Execute all the command from the above folder only.

Create the schema
#bin/schematool -dbType derby –initSchema

Start the Hive server.

#bin/hiveserver2

Connect using beeline in local mode
#bin/beeline -u jdbc:hive2://localhost:10000

Enable local Mode with the following command
#SET mapreduce.framework.name=local;