Hadoop Ecosystem Exam

Arturo Quintanilla
Software Engineer

This project was done using **cloudera-quickstart** for ease of use 15/05/2018.

ALLPPT.com - Free PowerPoint Templates, Diagrams and Charts

Hands-on

Tasks to accomplish

1. Create a flume agent that transfers netcat output to a hive table.
2. Use bash and Python to generate a csv file with 1000 rows and has five columns, first column value should be unique, file name created should contain the date created
3. Use the file generated in question 2 and load into a mysql table.
4. Sqoop this table into Hive.
5. Load the same CSV file HDFS and create an external table which use this source
6. Integrate this table with HBase and show results through HBase.
7. Hive Workout
8. Spark RDD Exercises

Task #1

Create a flume agent that transfers netcat output to a hive table.

First where going to create a directory to store our logged data.

hadoop fs -mkdir /user/cloudera/flume/

We use " hadoop fs" keywords in order to create a directory inside the hadoop file system and no t the local cent OS system.

Task #1

Create a flume agent that transfers netcat output to a hive table.

A flume agent consist of the following components:

- Source – Defines where the data is coming from.
    - In this task we'll be using netcat to fetch data from a configured port.
- Channel – are pipes which establish connections between sources and sinks.
    - In this task we'll be using a memory.
- Sink – Defines the destination of the data pipelined from various sources.
    - In this task we'll be using hdfs to store our logged data.

TASK #1

Create a flume agent that transfers netcat output to a hive table.

Now let's create a flume agent which will be named "netcat.conf".

Each component will be given a name so the first lines of code in our configuration file will look something like this.

```
netcatagent.sources = netcat
netcatagent.channels = memorychannel
netcatagent.sinks = hdfs
```

TASK #1

Create a flume agent that transfers netcat output to a hive table.

Now lets configure our source component, so our source component consist of 3 attributes:

- type – The type of source to work with.
- bind – ip of source.
- Port – ip's open port to fetch data.

```
# For each one of the sources, the type is defined
netcatagent.sources.netcat.type = netcat
netcatagent.sources.netcat.bind = localhost
netcatagent.sources.netcat.port = 56565
```

Task #1

Create a flume agent that transfers netcat output to a hive table.

Coming up next we have our memory configuration which consists of 3 attributes.

- type – Type of channel
- capacity - This is the maximum capacity number of events of the channel.
- transactionCapacity - This is the max number of events stored in the channel per transaction

```
# The channel can be defined as follows.
netcatagent.channels.memorychannel.type = memory
netcatagent.channels.memorychannel.capacity = 1000
netcatagent.channels.memorychannel.transactionCapacity = 100
```

Task #1

Create a flume agent that transfers netcat output to a hive table.

We're almost done configuring our agent now we need to configure our sink. Our sink consists of three attributes which are as follows:

- Type – Type of sink to use
- hdfs.path – path to store our logged files.
- writeFormat – file format

```
# Each sink's type must be defined
#netcatagent.sinks.hdfs.fileType = Datastream
netcatagent.sinks.hdfs.type = hdfs
netcatagent.sinks.hdfs.hdfs.path =
hdfs://localhost:8022/user/cloudera/flume/logs
netcatagent.sinks.hdfs.hdfs.writeFormat = Text
```

Task #1

Create a flume agent that transfers netcat output to a hive table.

Notice that we didn't have to do any fancy configuration in order to ingest data from flume to hive. The reason why this was done this way was because as long as our hive table is referencing an specific path in our hadoop file system we will be able to access our ingested data. So that's what we did in this line of code.

```
# Each sink's type must be defined
#netcatagent.sinks.hdfs.fileType = Datastream
netcatagent.sinks.hdfs.type = hdfs
netcatagent.sinks.hdfs.hdfs.path =
hdfs://localhost:8022/user/cloudera/flume/logs
netcatagent.sinks.hdfs.hdfs.writeFormat = Text
```

Task #1

Create a flume agent that transfers netcat output to a hive table.

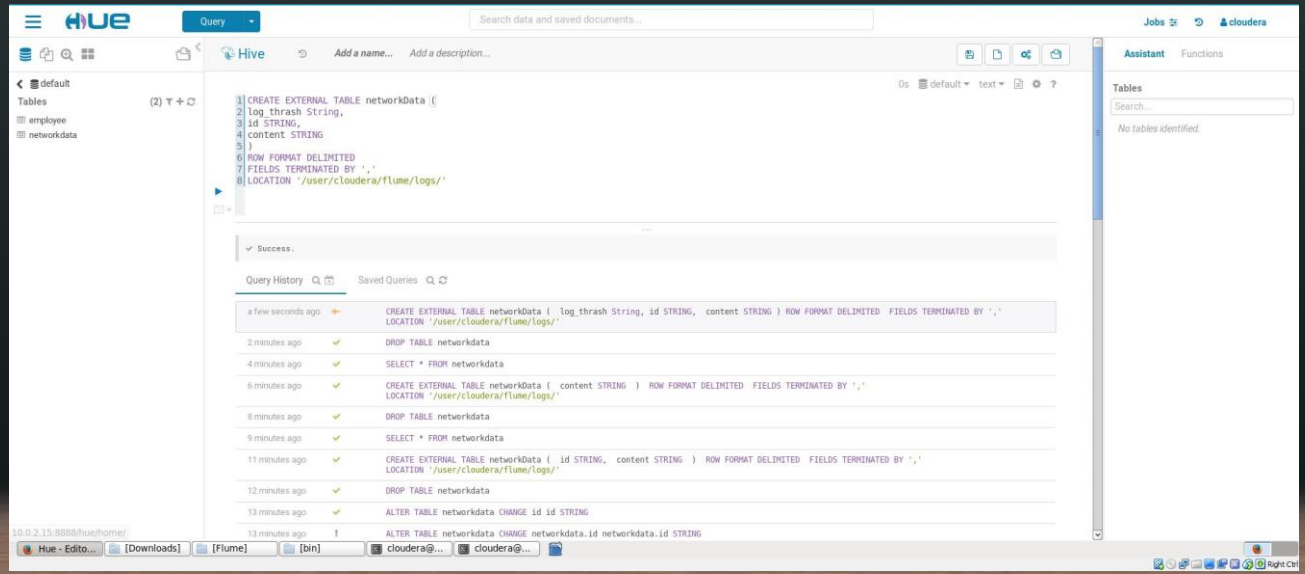
Finally lets bind all our components so our agent can work correctly.

```
#Bind the source and sink to the channel
netcatagent.sources.netcat.channels = memorychannel
netcatagent.sinks.hdfs.channel = memorychannel
```

Task #1

Create a flume agent that transfers netcat output to a hive table.

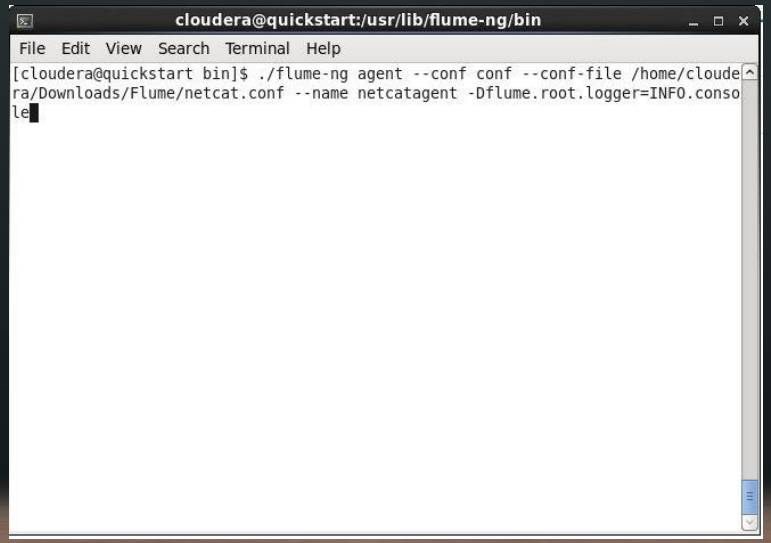
No let's create an external table in hive using HUE. Notice that we're referencing the path as our f lume agent.



Task #1

Create a flume agent that transfers netcat output to a hive table.

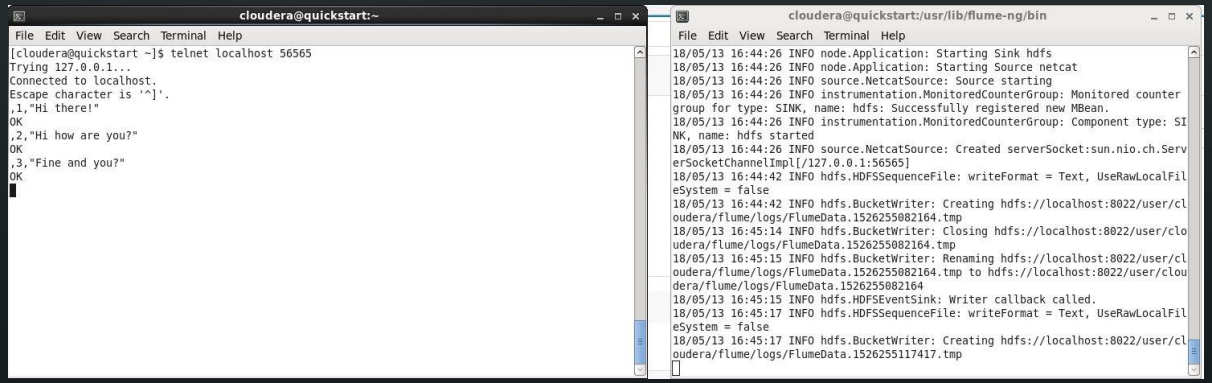
Now let's execute our agent. Take into count that we need to be inside of "/usr/lig/flume-ng" dire ctory. The command is as follows:



Task #1

Create a flume agent that transfers netcat output to a hive table.

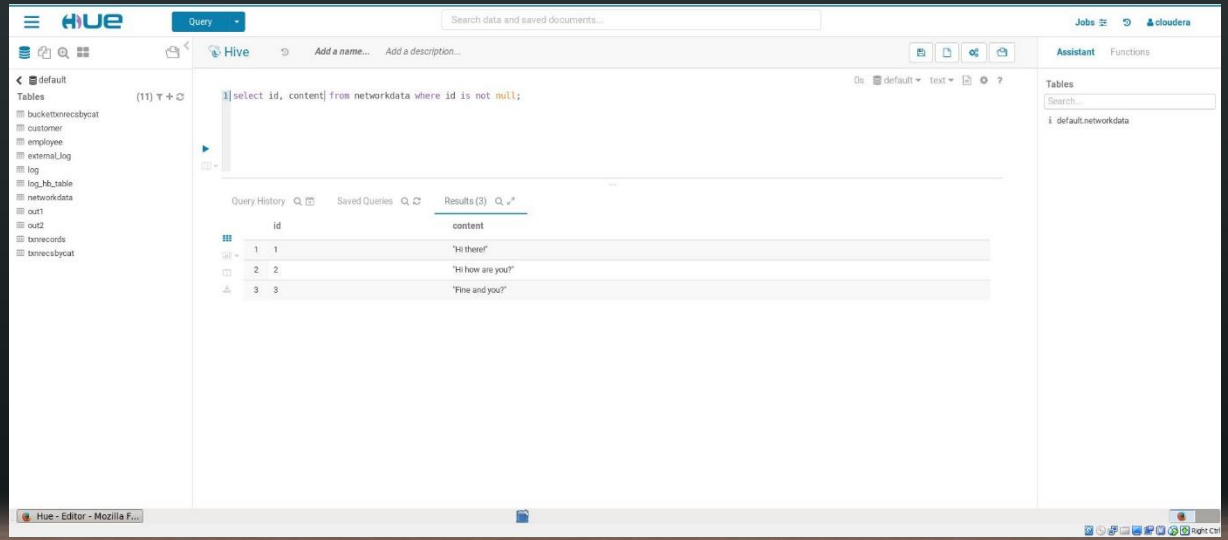
Now let's start ingesting data. In order to do that we first need to open a new terminal and type t he following command "telnet 56565". Afterwards we'll be able to ingest data to our hive table. Ju st as shown bellow.



Task #1

Create a flume agent that transfers netcat output to a hive table.

Finally let's view our ingested data from flume within our hive table.



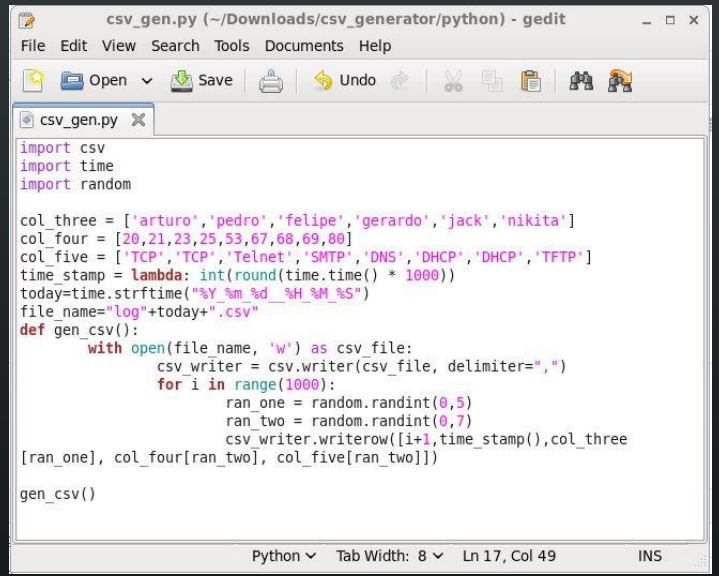
Task #2

Use bash and Python to generate a csv.

This is the python code I've developed. It's a pretty basi c code which consists of records based on the followin g structure.

- The first column is a unique identifier.
- The second column is the current date in millisecon ds.
- The three last rows are random indexes which are g oing to be used to access data from the three array s I've hard coded.

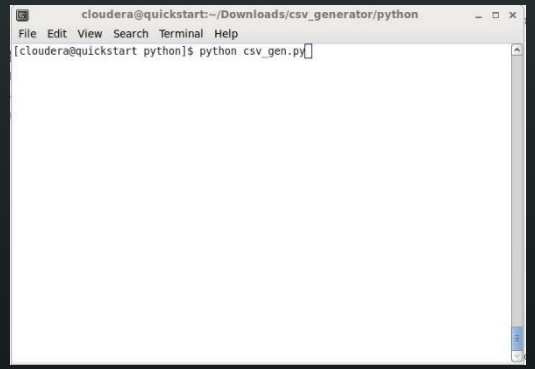
On each iteration a new record will be added to the ge nerated csv file.

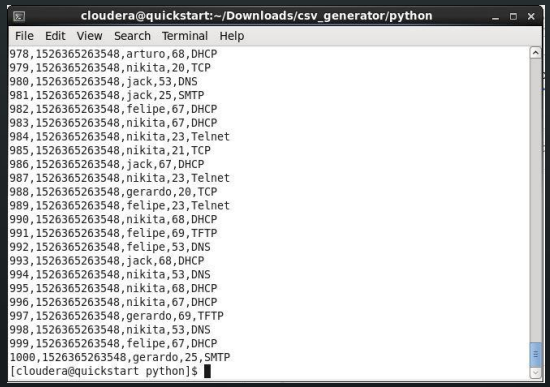


Task #2

Use bash and Python to generate a csv.

Now let's see some proof of work.





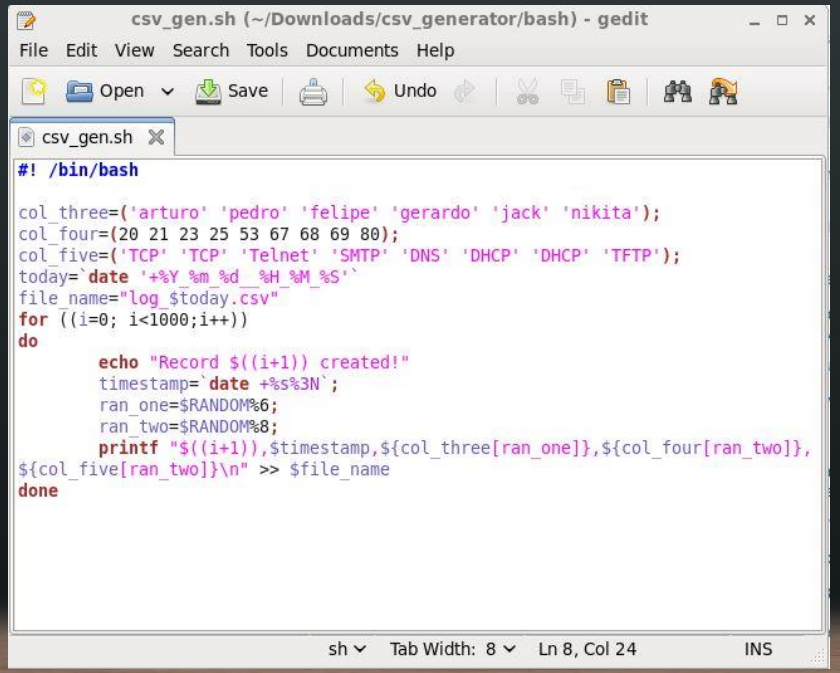
Task #2

Use bash and Python to generate a csv.

This is the bash script I've developed. It's a pretty basic script which consists of records based on th e following structure.

- The first column is a unique identifier.
- The second column is the current date in milli seconds.
- The three last rows are random indexes which are going to be used to access data from the three arrays I've hard coded.

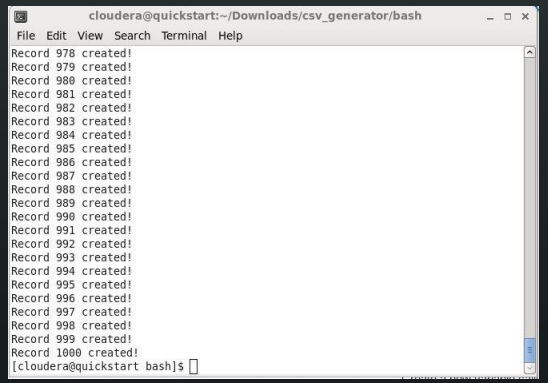
On each iteration a new record will be added to the generated csv file.

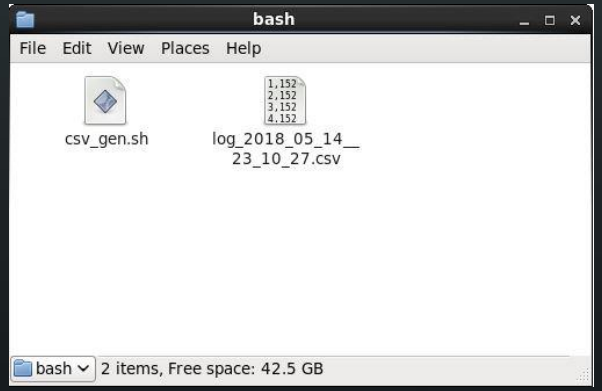


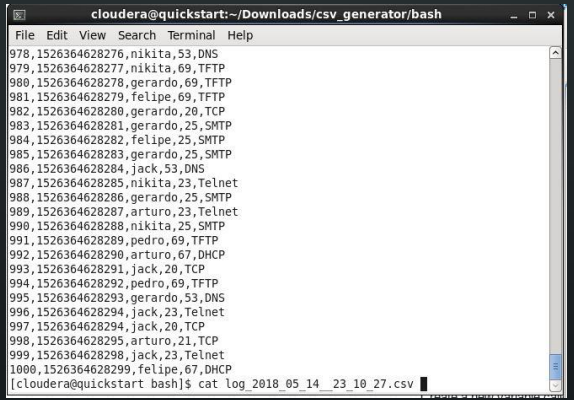
Task #2

Use bash and Python to generate a csv.

Now let's see some proof of work.



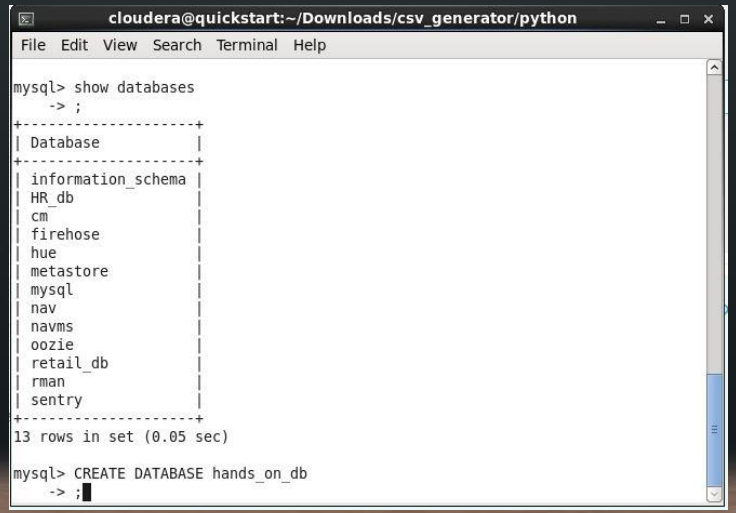


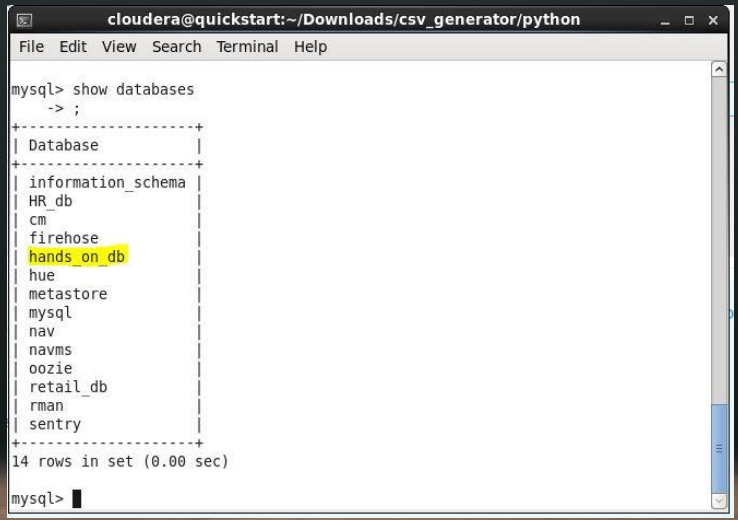


Task #3

Use the file generated in question 2 and load into a mysql table.

First of all were going to open our terminal, start mysql and create a database.
In cloudera to start mysql I used the following command: *"mysql –uroot -pcloudera"*

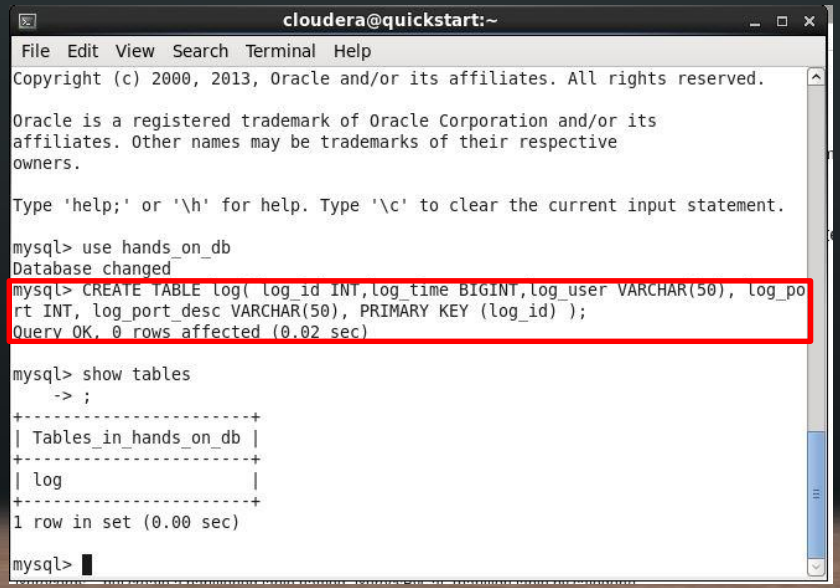




Task #3

Use the file generated in question 2 and load into a mysql table.

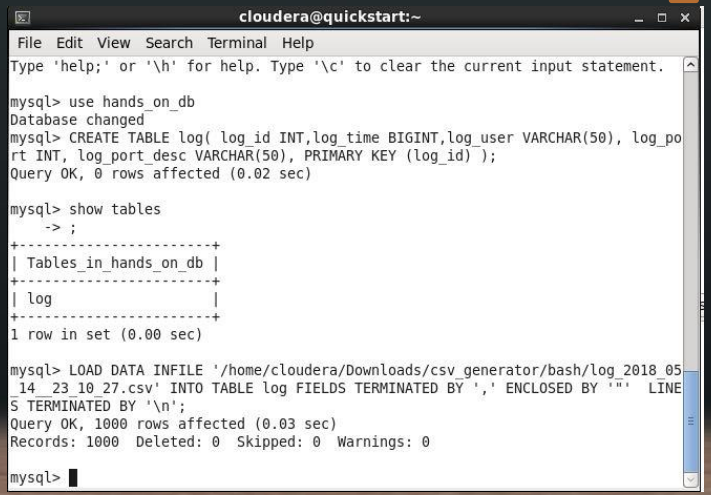
Now we'll create a table named "log" using the following command:

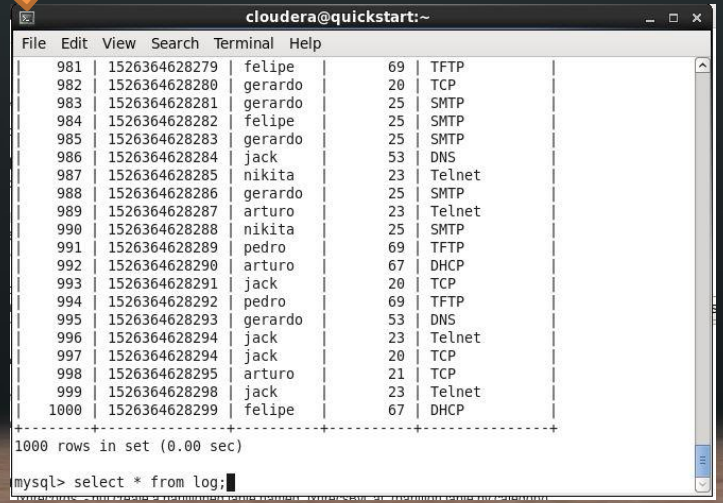


Task #3

Use the file generated in question 2 and load into a mysql table.

Finally we're going to import our data stored in our csv file to our log table in mysql and query d ata from it afterwards.





Task #4

Sqoop this table into Hive.

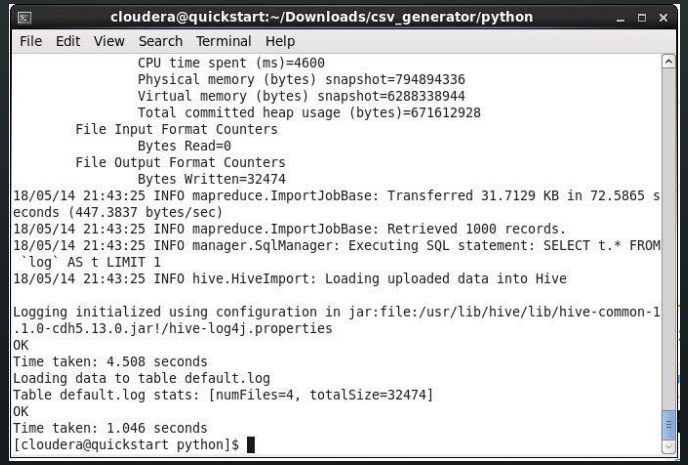
In this next task we're going to import our creat ed "log" table in mysql to hive. In order to do th at we'll execute the following command. Which consist on the following:

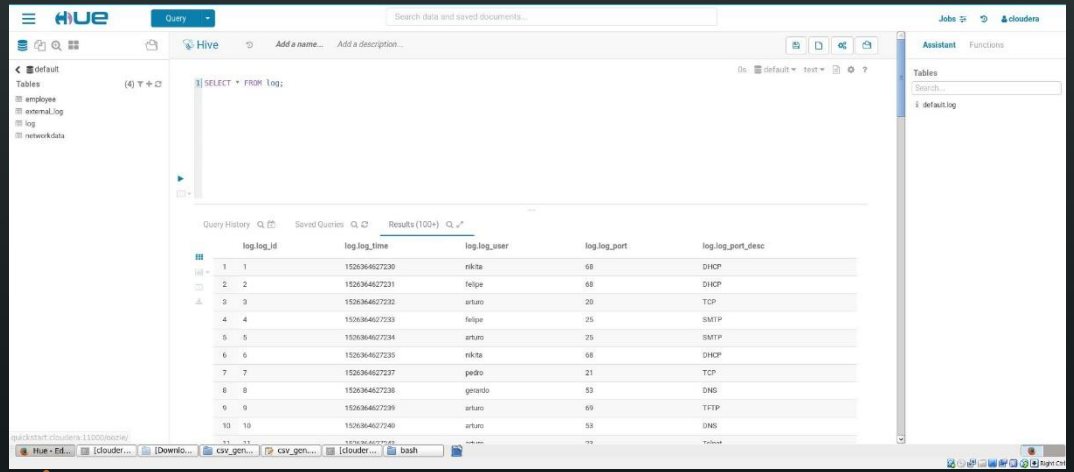
1. Sqoop import (keyword for importing)
2. --connect (to define to which database to connect)
3. --username (user name for database)
4. --password (password for database)
5. --table (table to import)
6. --hive-import (special keyword to import to hive)

Task #4

Sqoop this table into Hive.

Finally let's showcase some proof of work. We're going to see the results of our import and query data from our hive table.

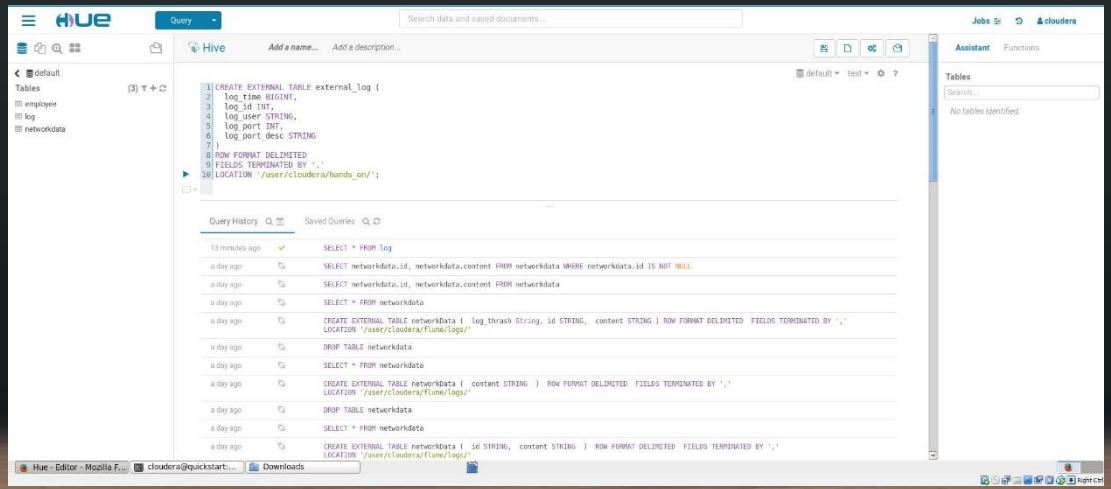




Task #5

Load the same CSV file HDFS and create an external table which use th is source.

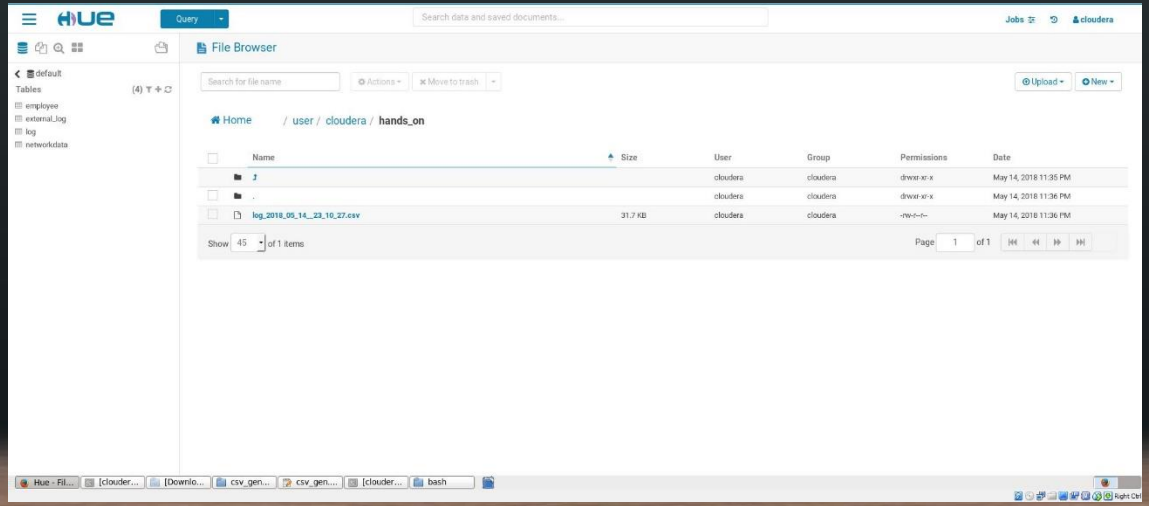
Previously we've seen how to import data from a csv to mysql and finally to a hive internal table. Now we'll be show casing how to "import" or access data stored in a csv file from a hive external table. First of all let's create an external table in hive.



Task #5

Load the same CSV file HDFS and create an external table which use th is source.

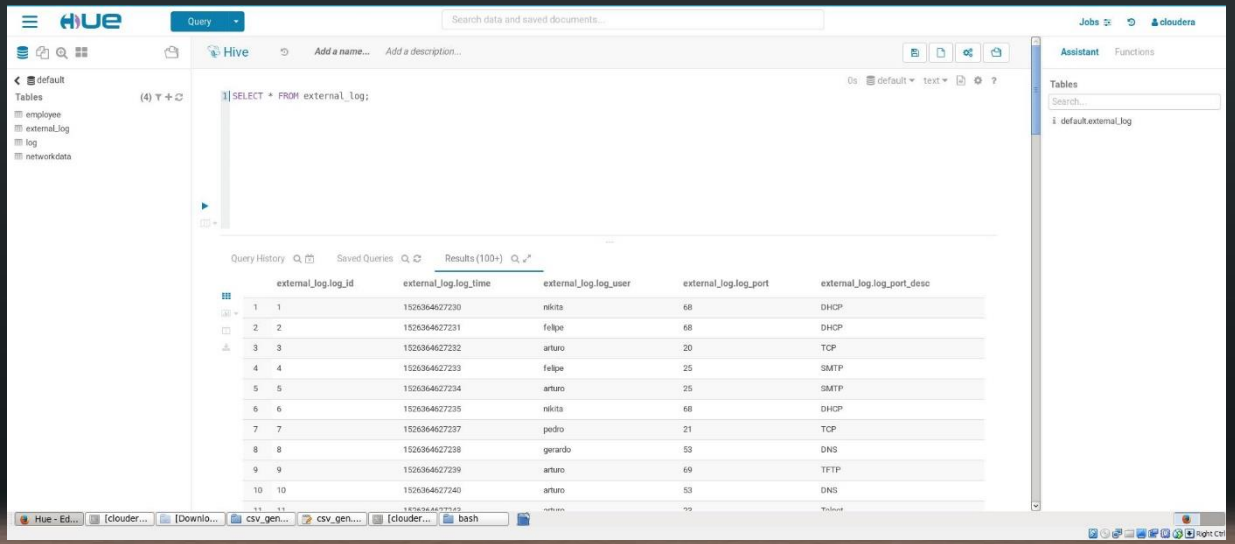
Now let's move our csv stored in our local file system to hdfs file system using HUE GUI so we ca n reference it from our recently created external table. In this case our path is "user/cloudera/han ds_on". If you don't have "hands_on" folder please create it.



Task #5

Load the same CSV file HDFS and create an external table which use th is source.

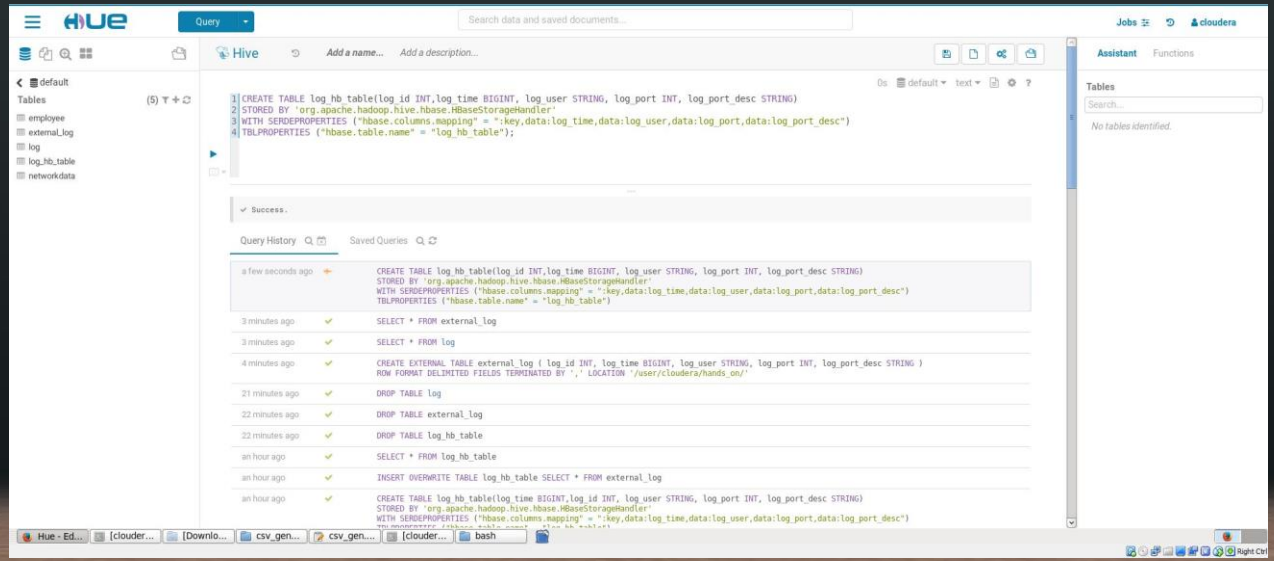
Finally let's query the data referenced in our external table named "external_log".



Task #6

Integrate this table with HBase and show results through HBase.

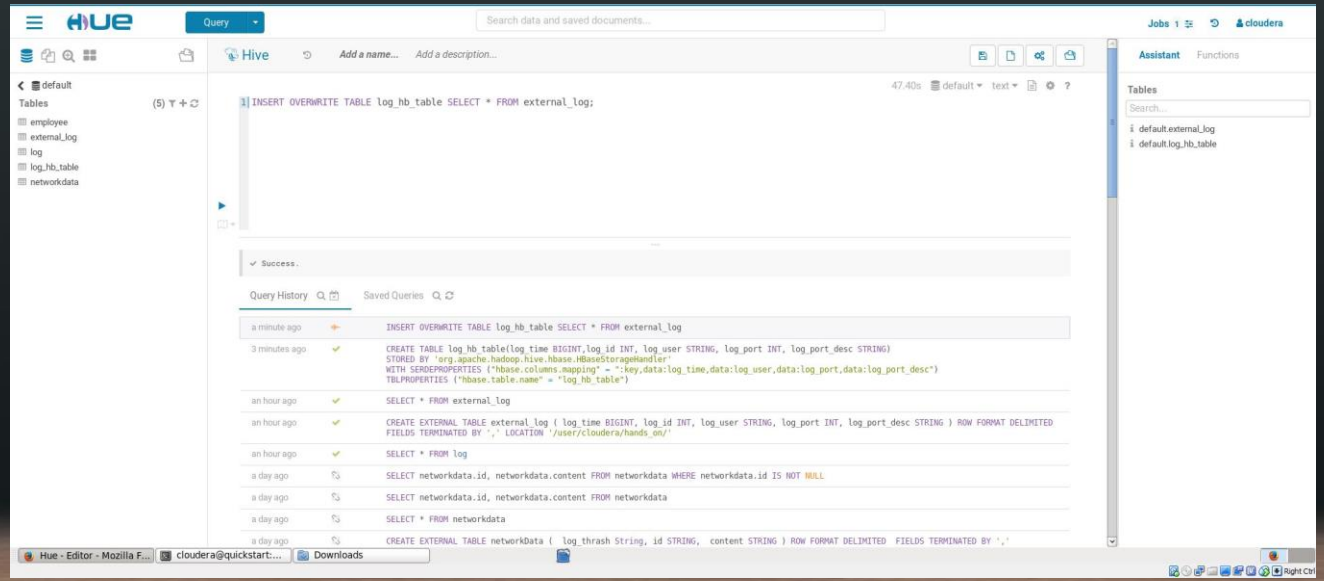
In this task for ease of use we'll be creating an hbase table from our hive editor and then transfer data from our external_log table to our newly created hbase table.



Task #6

Integrate this table with HBase and show results through HBase.

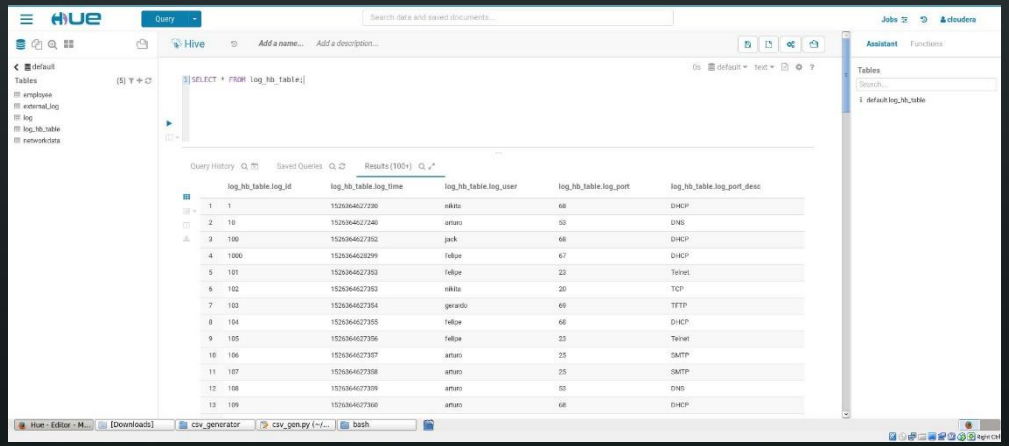
Now to transfer data from our external hive table to our hbase table we'll use the following com mand.

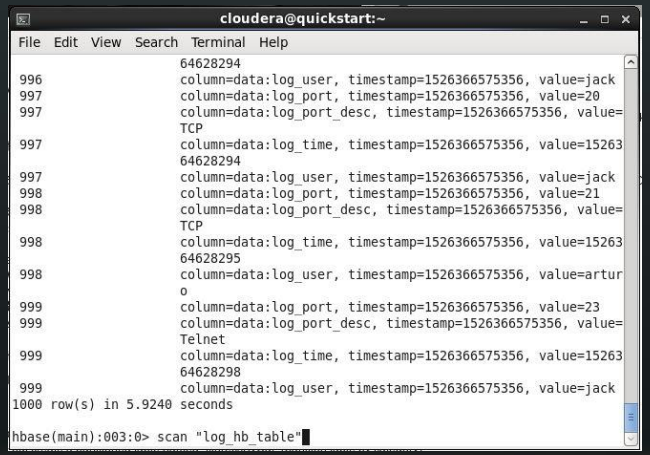


Task #6

Integrate this table with HBase and show results through HBase.

Finally let's query out some data from our new table "log_hb_table" using hive editor and our hba se CLI.





Task #7

Hive Workout

- Create a internal/managed table structure named "txnrecords" with below column/datatype for the data file txns.txt:

txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, state STRING, spendby STRING

- Load the table with data file placed in HDFS

Note : Below 2 tables are to be created as external tables with some user defined location

create another table structure with similar columns like table "txnrecords" - but create a partitioned table named "txnrecsByCat" (partition table by category)

create another table structure with similar columns like table 'txnrecords" - but create a table with partition and 10 buckets named "buckettxnrecsByCat"(partition by category & buckets by state)

Note : For dynamic partition - you need to execcute two below hive properties before inserts

set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.dynamic.partition=true;

Insert table "txnrecsByCat" with data from previously created table "txnrecords" (Dynamic partitioning)

Insert table "buckettxnrecsByCat" with data from previously created table "txnrecords" (Dynamic partitioning)

Task #7

Hive Workout

- Create a internal/managed table structure named "customer" with below column/datatype for the data cust.txt :

custno string, firstname string, lastname string, age int,profession string

- Load the table with data file placed in HDFS
- Create a internal/managed table "out1" with below structure :

custno int,firstname string,age int,profession string,amount double,product string

- Insert the result of join between tables "txnrecords" & "customer" into table "out1"
- Create a internal/managed table "out2" with below structure :

custno int,firstname string,age int,profession string,amount double,product string, level string

- Populate first 6 columns with similar join results achieved in table "out1"

level column has to be populated based on below logic:

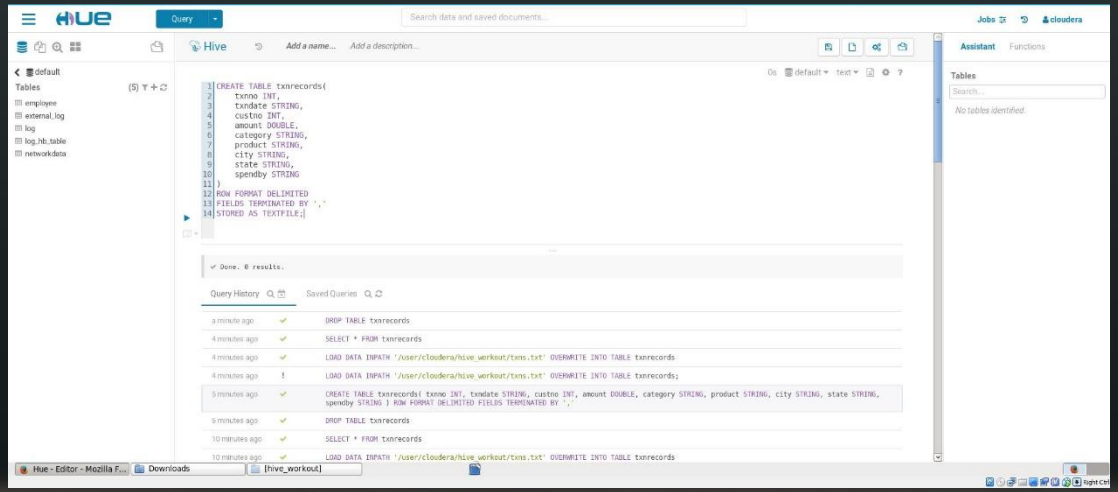
  - If age less than 30, level value = low
  - If age is between 30 & 50, level value = middle
  - If age is greater than 50, level value = old

Task #7

Hive workout

- Create a internal/managed table structure named "txnrecords" with below column/datatype for the data file txns.txt:

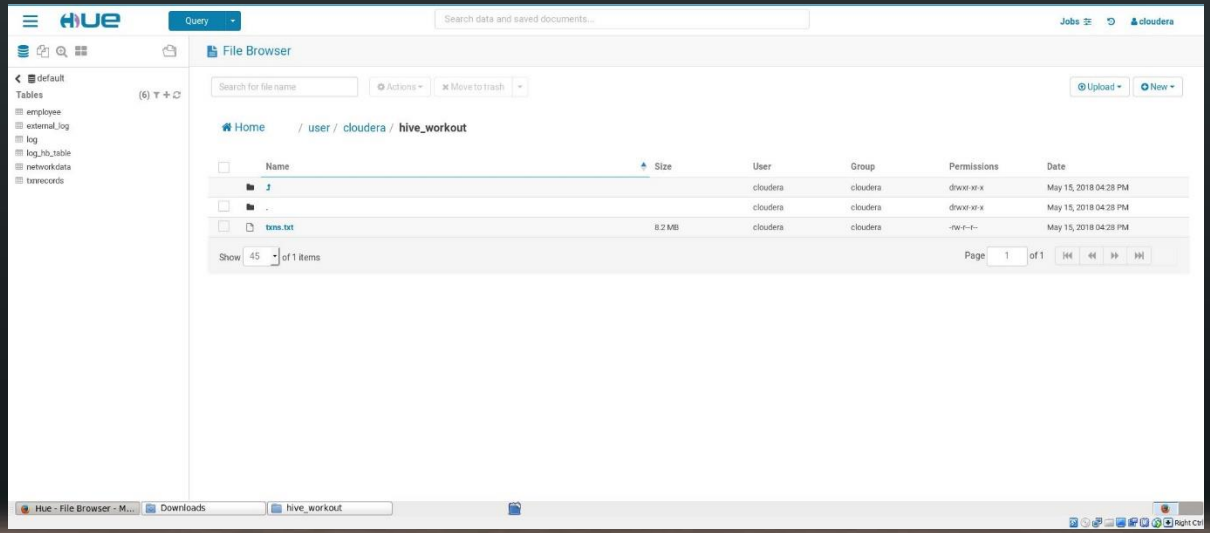
txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, state STRING, spendby STRING



Task #7

Hive workout

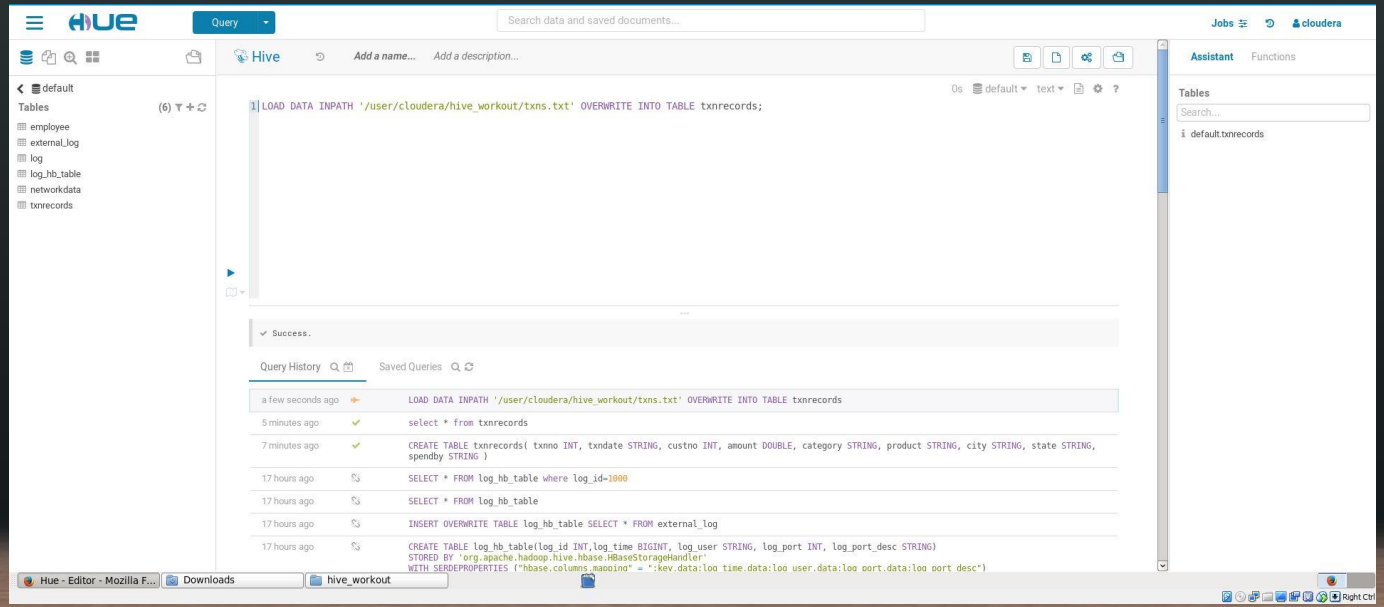
Loading our text file to hdfs



Task #7

Hive workout

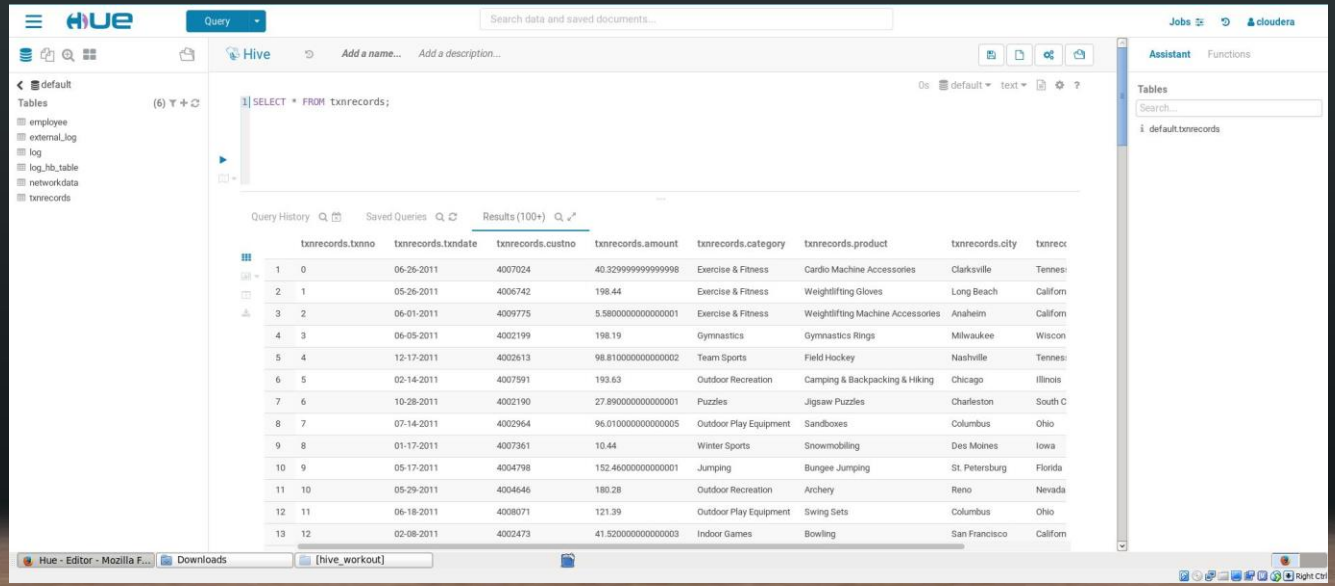
Dump data from csv file to internal hive table.



Task #7

Hive workout

Finally query data from internal table.



	txnrecords.txnno	txnrecords.txndate	txnrecords.custno	txnrecords.amount	txnrecords.category	txnrecords.product	txnrecords.city	txnrec
1	0	06-26-2011	4007024	40.329999999999998	Exercise & Fitness	Cardio Machine Accessories	Clarksville	Tennes
2	1	05-26-2011	4006742	198.44	Exercise & Fitness	Weightlifting Gloves	Long Beach	Californ
3	2	06-01-2011	4009775	5.5800000000000001	Exercise & Fitness	Weightlifting Machine Accessories	Anaheim	Californ
4	3	06-05-2011	4002199	198.19	Gymnastics	Gymnastics Rings	Milwaukee	Wiscon
5	4	12-17-2011	4002613	98.810000000000002	Team Sports	Field Hockey	Nashville	Tennes
6	5	02-14-2011	4007591	193.63	Outdoor Recreation	Camping & Backpacking & Hiking	Chicago	Illinois
7	6	10-28-2011	4002190	27.890000000000001	Puzzles	Jigsaw Puzzles	Charleston	South C
8	7	07-14-2011	4002964	96.010000000000005	Outdoor Play Equipment	Sandboxes	Columbus	Ohio
9	8	01-17-2011	4007361	10.44	Winter Sports	Snowmobiling	Des Moines	Iowa
10	9	05-17-2011	4004798	152.46000000000001	Jumping	Bungee Jumping	St. Petersburg	Florida
11	10	05-29-2011	4004646	180.28	Outdoor Recreation	Archery	Reno	Nevada
12	11	06-18-2011	4008071	121.39	Outdoor Play Equipment	Swing Sets	Columbus	Ohio
13	12	02-08-2011	4002473	41.520000000000003	Indoor Games	Bowling	San Francisco	Californ

Task #7

Hive workout

Load the table with data file placed in HDFS.

Note : Below 2 tables are to be created as external tables with some user defined location

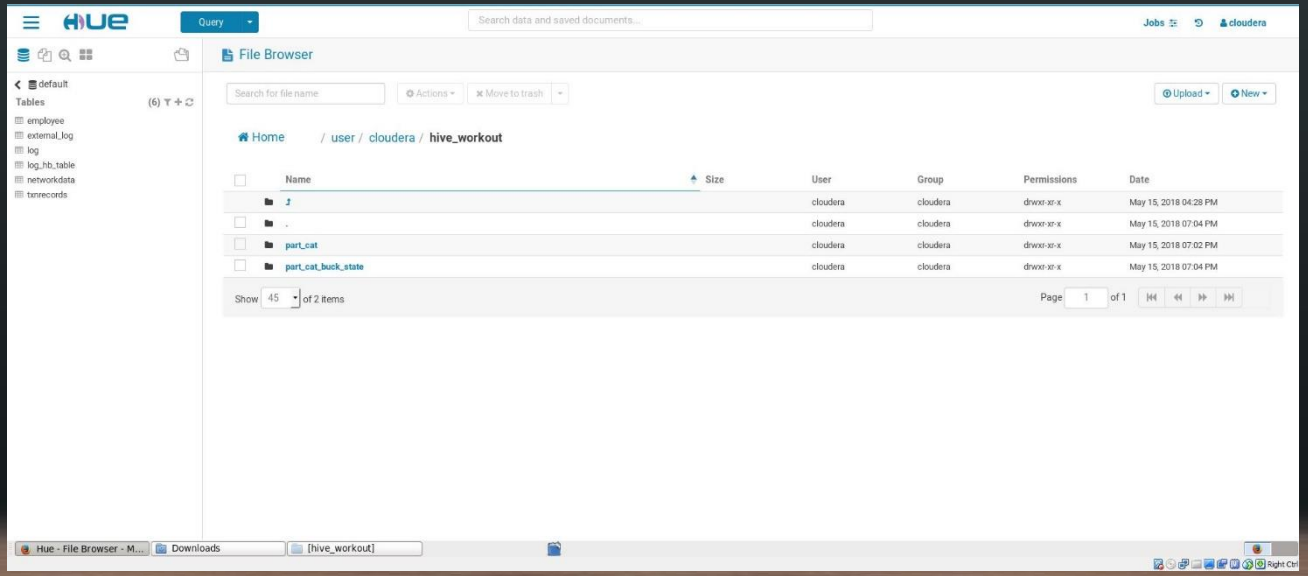
create another table structure with similar columns like table "txnrecords" - but create a partitione d table named "txnrecsByCat" (partition table by category)

create another table structure with similar columns like table 'txnrecords" - but create a table with partition and 10 buckets named "buckettxnrecsByCat"(partition by category & buckets by state)

Task #7

Hive workout

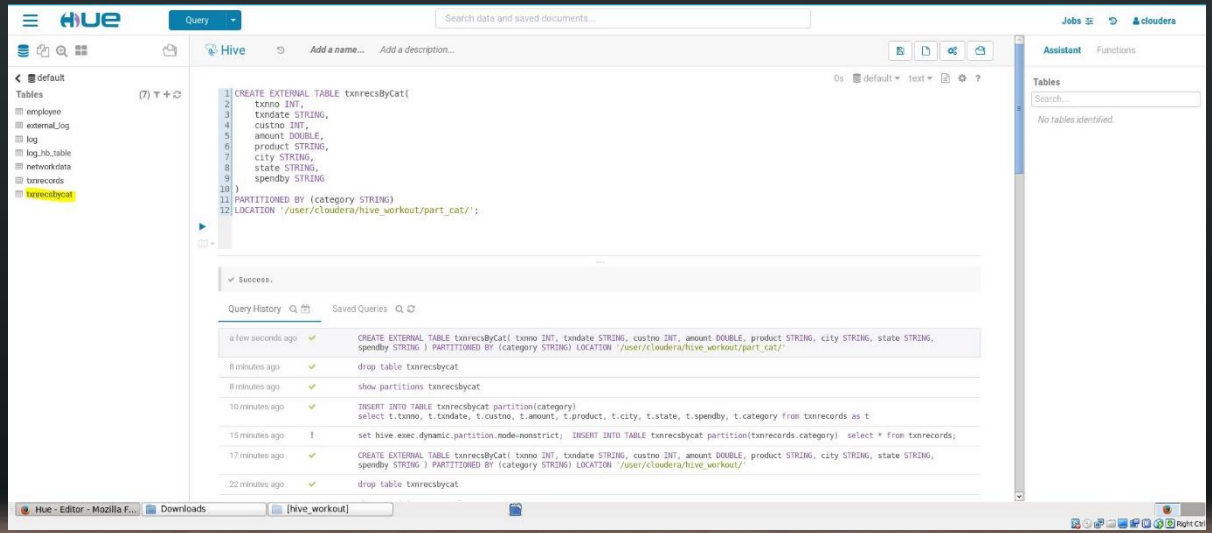
This is the directory in which we're going to store both of our table partitions.



Task #7

Hive workout

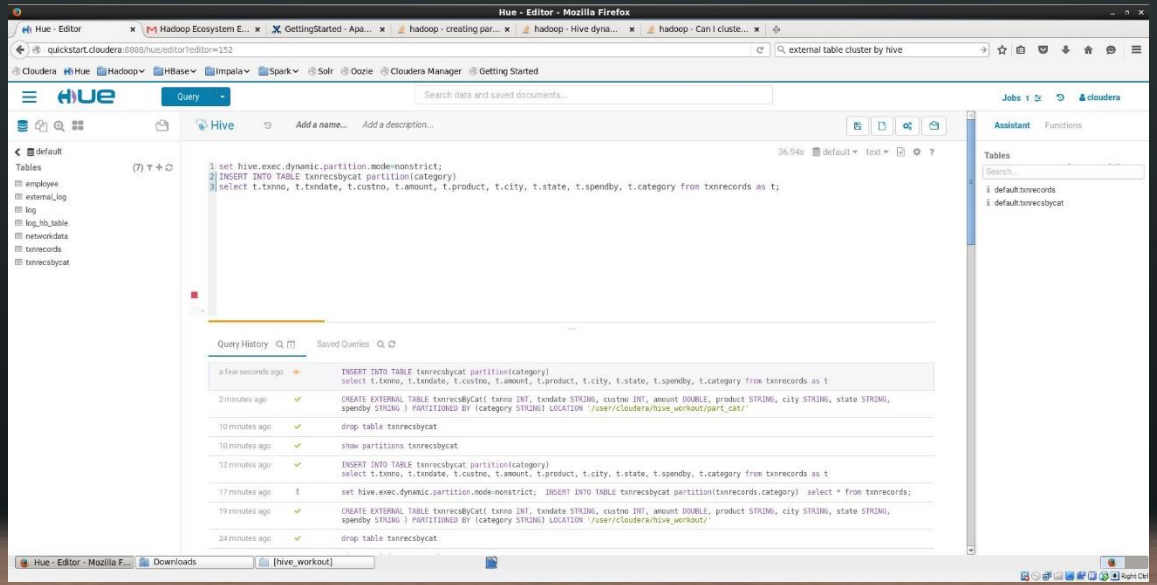
Let’s create our "txnrecsByCat" table (partition table by category).



Task #7

Hive workout

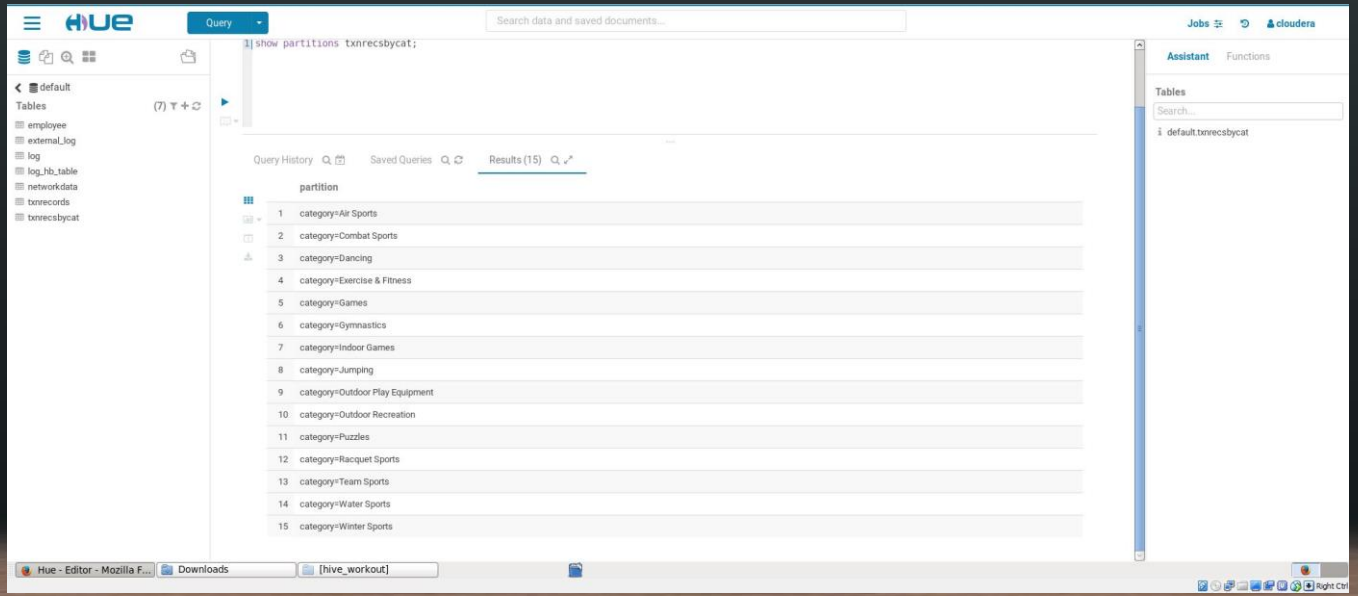
Let’s insert our data from our source table (txnrecords) to our external partition table (txnrecsByCat).



Task #7

Hive workout

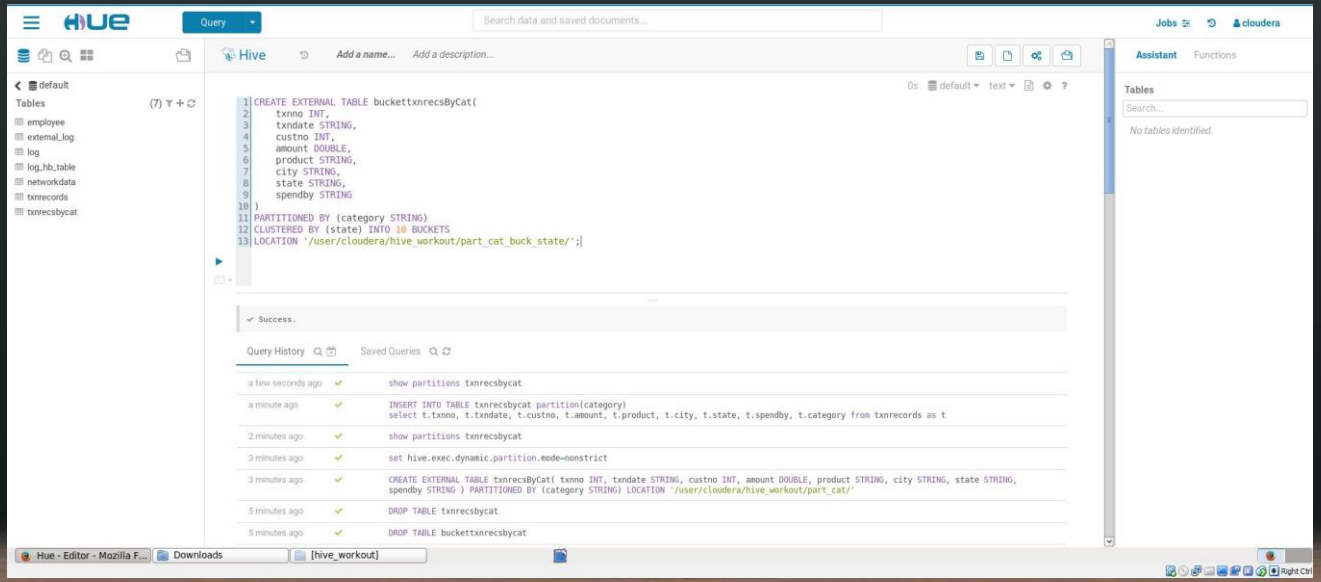
Finally let's query the different partitions that our table contains.



Task #7

Hive workout

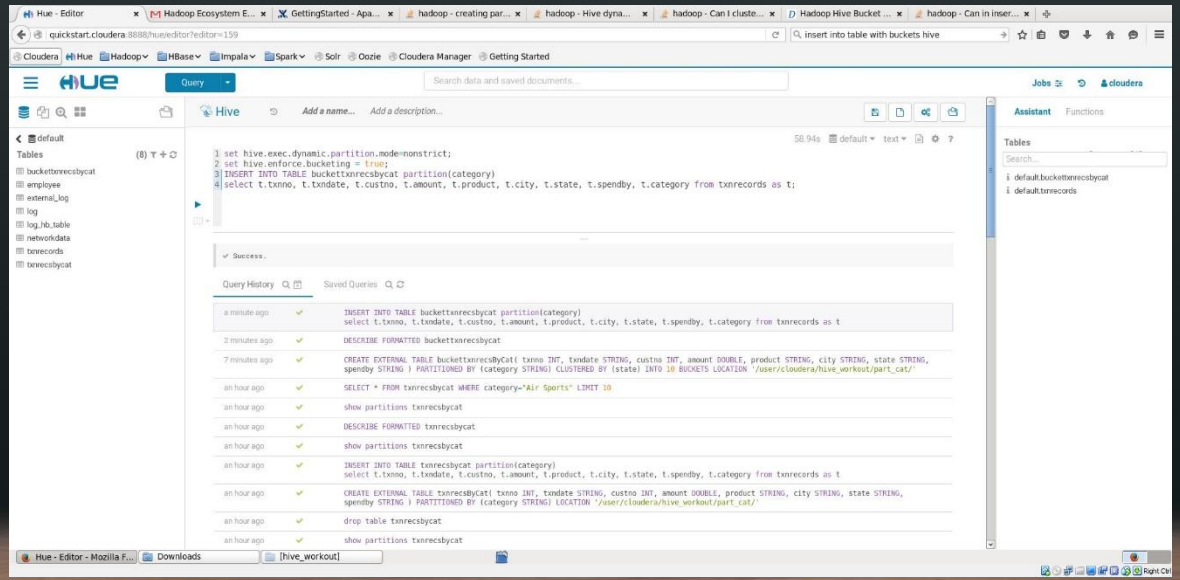
Let’s create our "buckettxnrecsByCat” table (partition by category & buckets by state)



Task #7

Hive workout

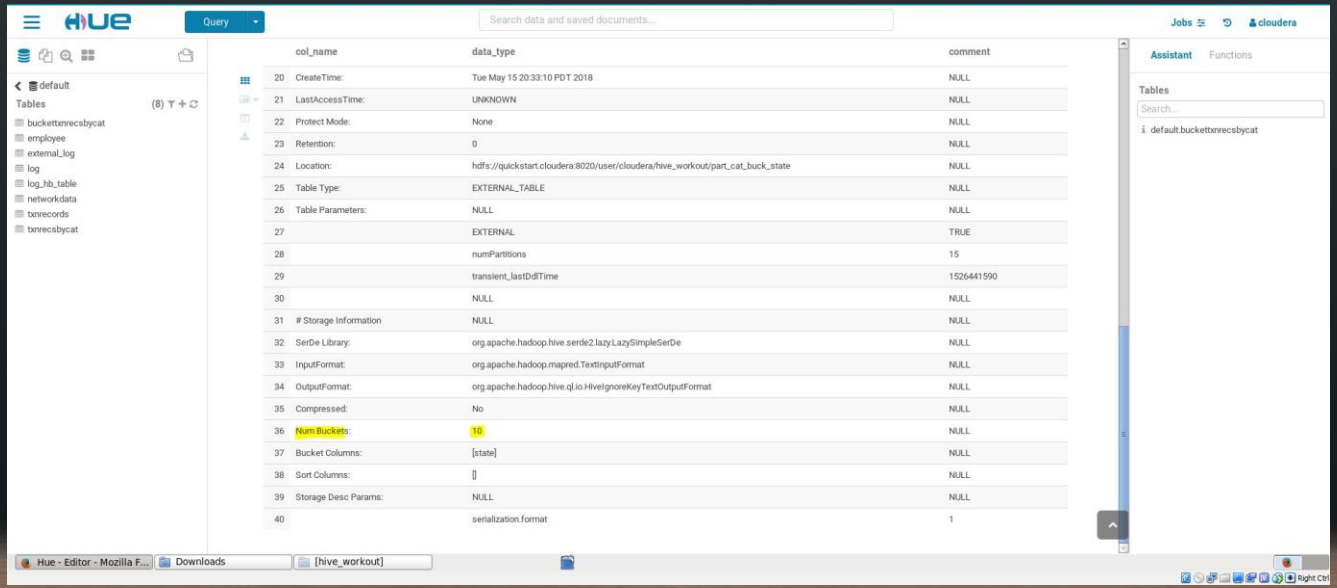
Now let's insert data from our source table (txnrecords) to our external table (buckettxnrecsByCat)



Task #7

Hive workout

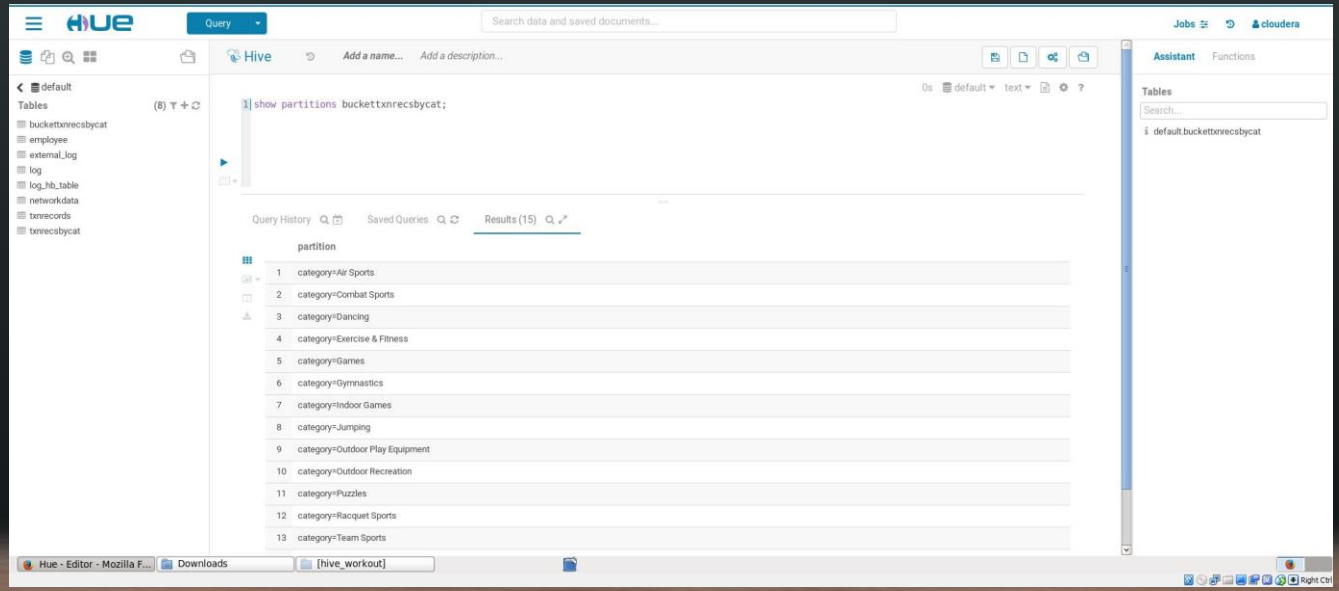
Now let's see the number of buckets our table has.



Task #7

Hive workout

Finally let's see our generated partitions.



Task #7

Hive workout

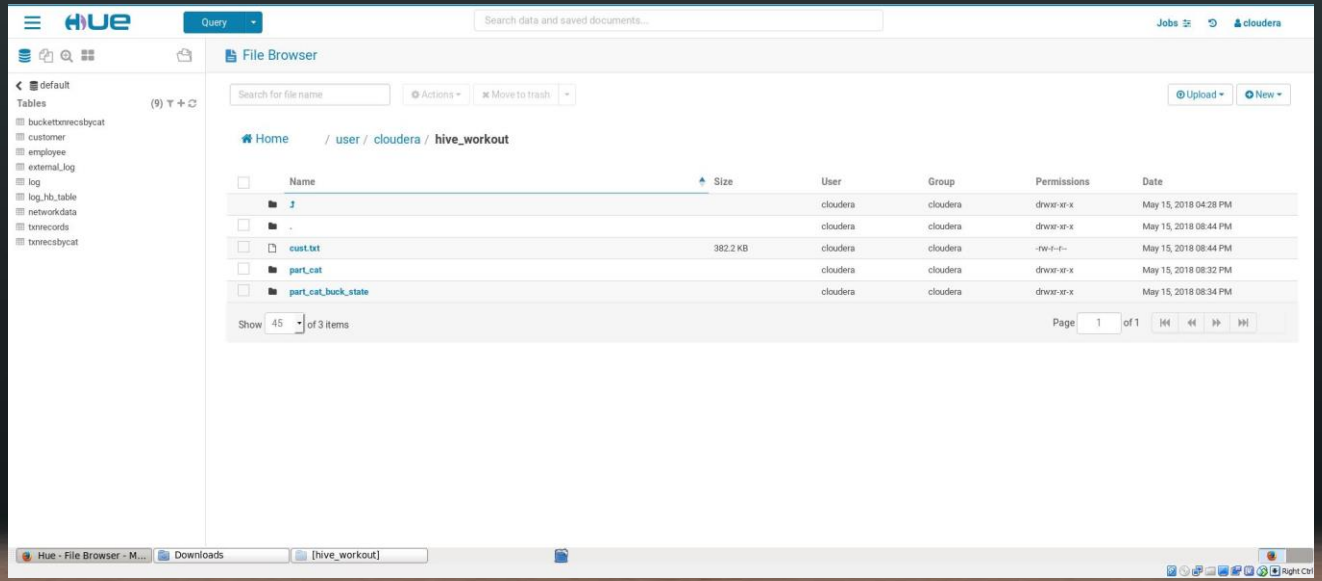
Create a internal/managed table structure named "customer" with below column/datatype for the data cust.txt :

custno string, firstname string, lastname string, age int,profession string

Task #7

Hive workout

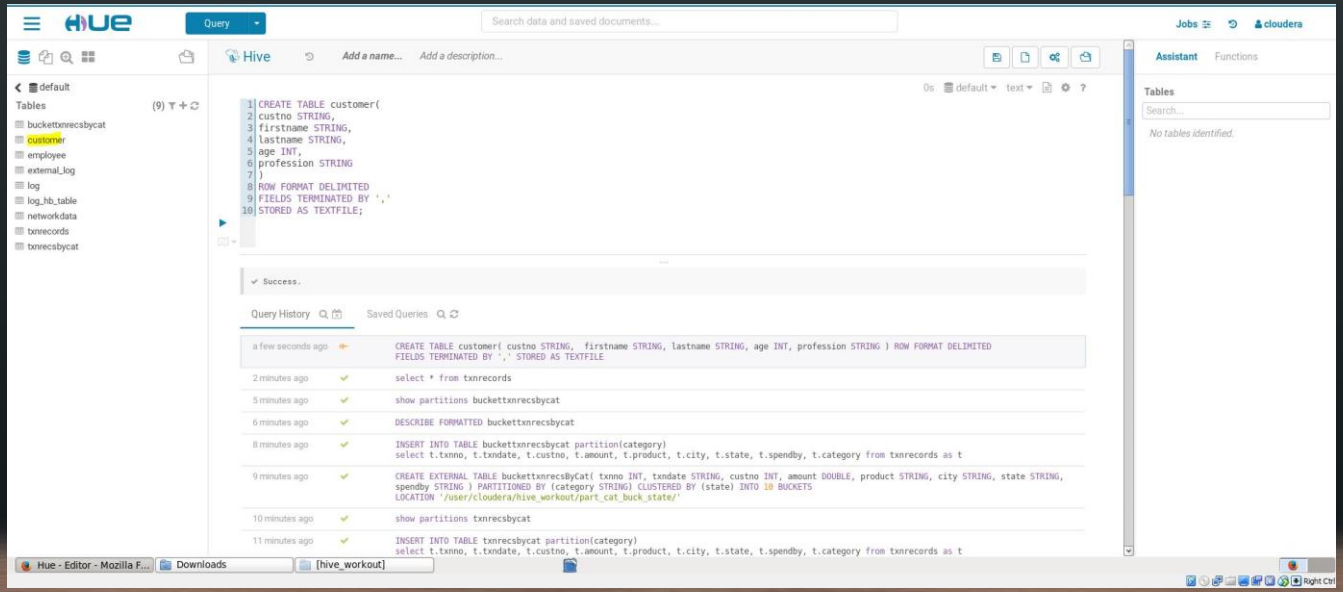
Uploading our customer text file to HDFS.



Task #7

Hive workout

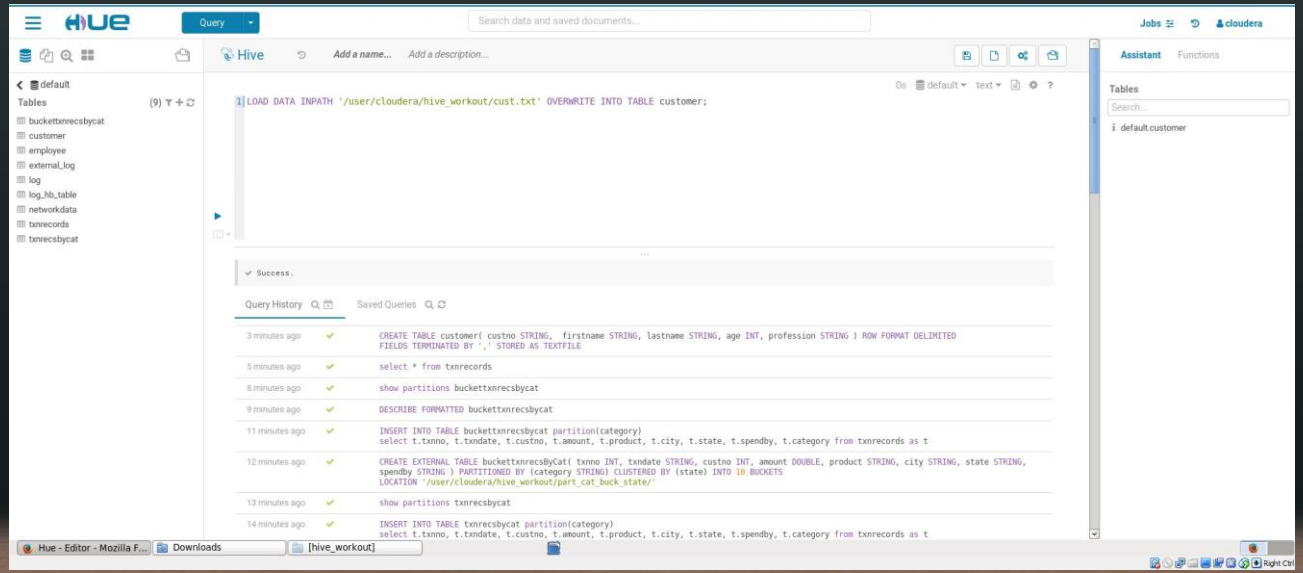
Creating our internal customer table.



Task #7

Hive workout

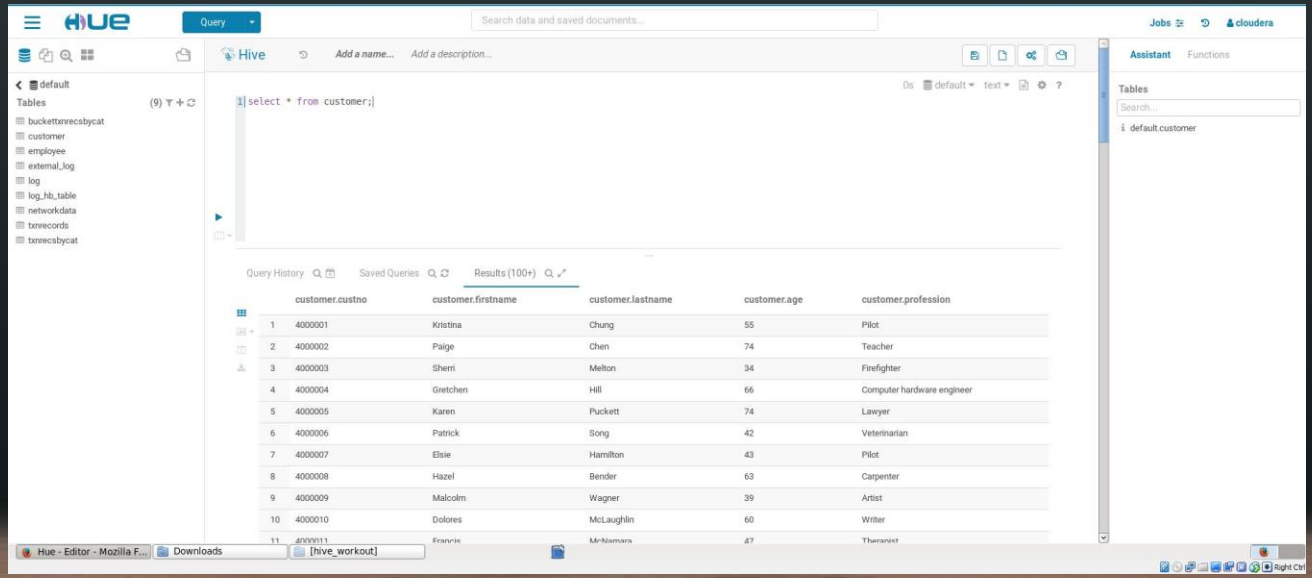
Loading data to our customer table.



Task #7

Hive workout

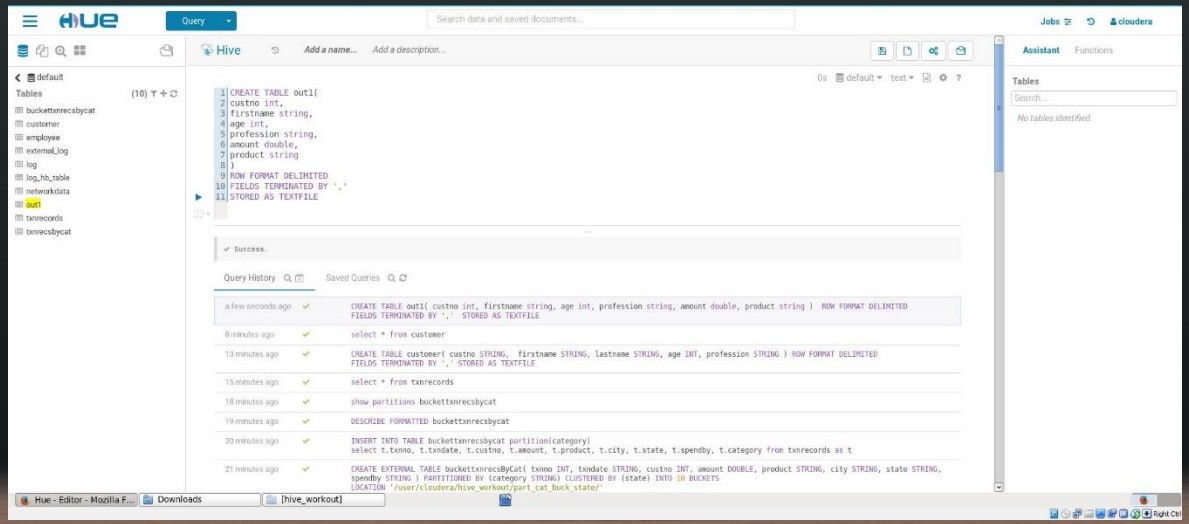
Querying data from our customer table.



Task #7

Hive workout

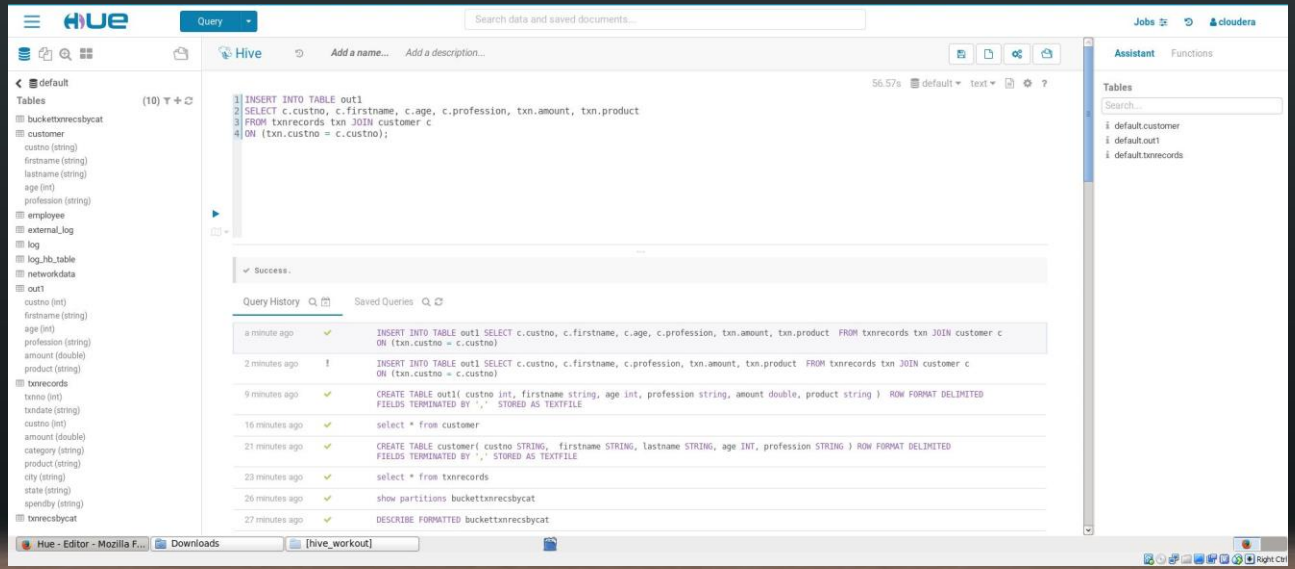
Create a internal/managed table "out1" with below structure :
custno int,firstname string,age int,profession string,amount double,product string



Task #7

Hive workout

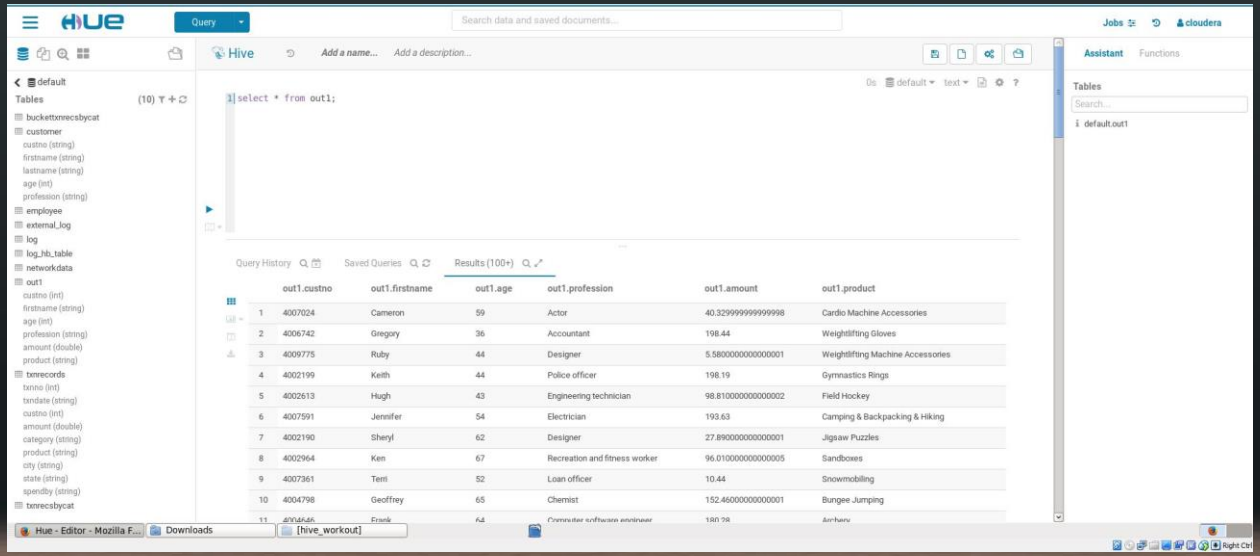
Insert the result of join between tables "txnrecords" & "customer" into table "out1".



Task #7

Hive workout

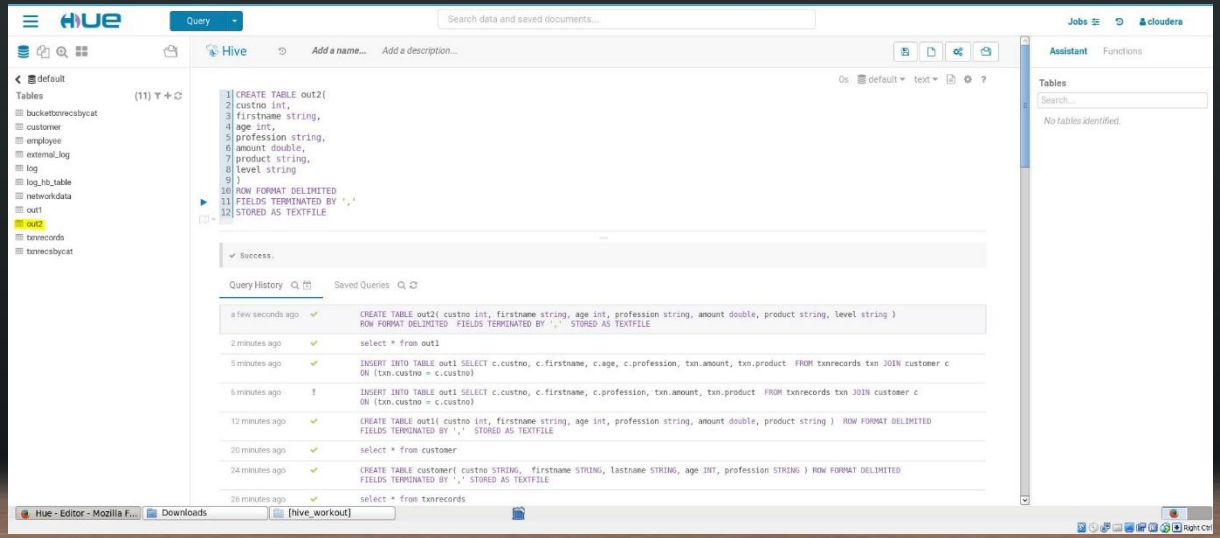
Querying data from out1 table.



Task #7

Hive workout

Create a internal/managed table "out2" with below structure :
custno int,firstname string,age int,profession string,amount double,product string, level string



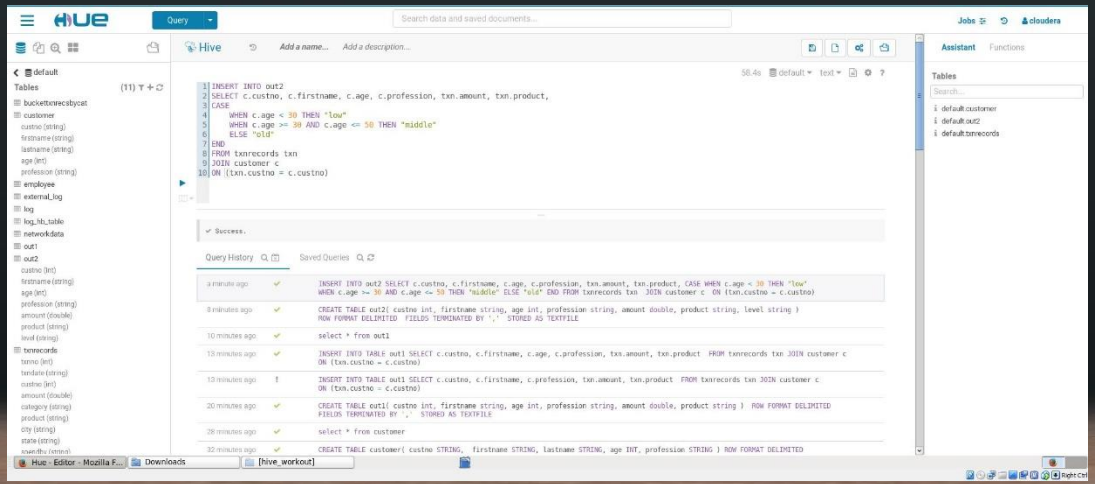
Task #7

Hive workout

- Populate first 6 columns with similar join results achieved in table "out1"

level column has to be populated based on below logic:

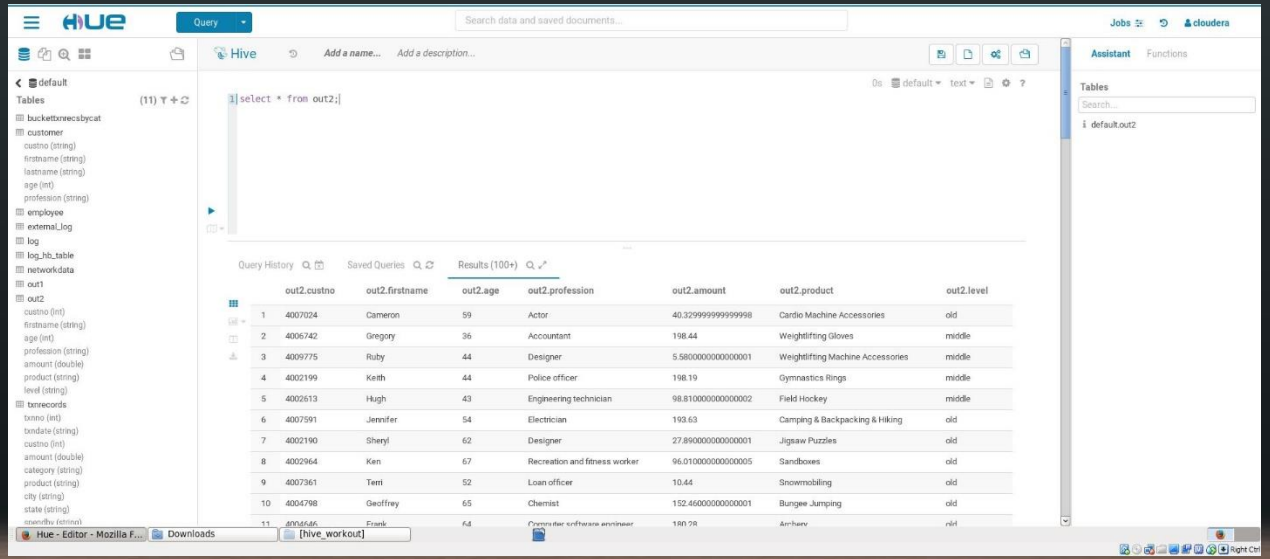
- If age less than 30, level value = low
- If age is between 30 & 50, level value = middle
- If age is greater than 50, level value = old



Task #7

Hive workout

Querying data from out2 table.

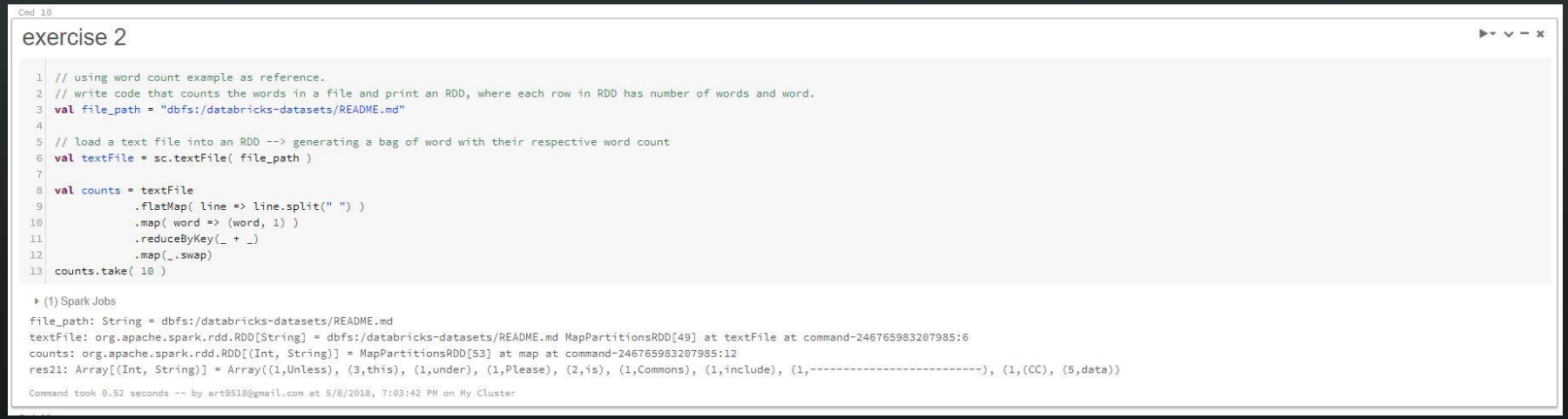


Task #8

Spark RDD Exercises

Task #8

Spark RDD Exercises



Cmd 10

exercise 2

```
// using word count example as reference.
// write code that counts the words in a file and print an RDD, where each row in RDD has number of words and word.
val file_path = "dbfs:/databricks-datasets/README.md"

// load a text file into an RDD --> generating a bag of word with their respective word count
val textFile = sc.textFile( file_path )

val counts = textFile
            .flatMap( line => line.split(" ") )
            .map( word => (word, 1) )
            .reduceByKey(_ + _)
            .map(_.swap)
counts.take( 10 )
```

▸ (1) Spark Jobs

```
file_path: String = dbfs:/databricks-datasets/README.md
textFile: org.apache.spark.rdd.RDD[String] = dbfs:/databricks-datasets/README.md MapPartitionsRDD[49] at textFile at command-246765983207985:6
counts: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[53] at map at command-246765983207985:12
res21: Array[(Int, String)] = Array((1,Unless), (3,this), (1,under), (1,Please), (2,is), (1,Commons), (1,include), (1,--------------------------), (1,(CC), (5,data))
```

Command took 0.52 seconds -- by art9518@gmail.com at 5/8/2018, 7:03:42 PM on My Cluster

Task #8

Spark RDD Exercises

Cmd 11

exercise 3

```
// create an rdd that keep all the words that appear two times.
// tip: read filter transformation
// http://spark.apache.org/docs/2.1.1/programming-guide.html#transformations
val file_path = "dbfs:/databricks-datasets/README.md"

// load a text file into an RDD --> generating a bag of word with their respective word count
val textFile = sc.textFile( file_path )

val counts = textFile
            .flatMap( line => line.split(" ") )
            .map( word => (word, 1) )
            .reduceByKey(_ + _)
            .filter((item) => item._2 > 2)

counts.take( 10 )
```

▸ (2) Spark Jobs

```
file_path: String = dbfs:/databricks-datasets/README.md
textFile: org.apache.spark.rdd.RDD[String] = dbfs:/databricks-datasets/README.md MapPartitionsRDD[21] at textFile at command-246765983207986:7
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[24] at reduceByKey at command-246765983207986:12
filterCounts: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[25] at filter at command-246765983207986:14
res11: Array[(String, Int)] = Array((this,3), (data,5), ("",5), (for,3), (the,11), (to,5), (and,4))
```

Command took 0.76 seconds -- by art9518@gmail.com at 5/8/2018, 6:50:32 PM on My Cluster

Task #8

Spark RDD Exercises

Cmd 12

exercise 4

```
// create an RDD that groups the words by number of times they occur.
// each row must has number_of_times and a collection of words.
// tip read groupByKey
// http://spark.apache.org/docs/2.1.1/programming-guide.html#transformations
val file_path = "dbfs:/databricks-datasets/README.md"

// load a text file into an RDD --> generating a bag of word with their respective word count
val textFile = sc.textFile( file_path )

val counts = textFile
            .flatMap( line => line.split(" ") )
            .map( word => (word, 1) )
            .reduceByKey(_ + _)
            .map(_.swap)
            .groupByKey()
            .map(x => (x._1, x._2.toArray))

counts.take( 10 )
```

▸ (2) Spark Jobs

```
file_path: String = dbfs:/databricks-datasets/README.md
textFile: org.apache.spark.rdd.RDD[String] = dbfs:/databricks-datasets/README.md MapPartitionsRDD[1] at textFile at command-246765983207987:8
counts: org.apache.spark.rdd.RDD[(Int, Array[String])] = MapPartitionsRDD[7] at map at command-246765983207987:16
res0: Array[(Int, Array[String])] = Array((4,Array(and)), (2,Array(is, file, Databricks, within, be, by, hosted)), (11,Array(the)), (1,Array(Unless, under, Please, Commons, include, --------------------------, (CC, otherwise, hosted-datasets@databricks.com., using, send, Make, viewed, email, new, README, International, Databricks., allows, consumed, (e.g., can, ==========, build, data,, Apache, how, 4.0, When, get, information, please, ensure, information., noted, users, Requests, want, repository,, Spark, The, it, url:, about, public., license,, -------, set),, at, following, pipelines, sure, which, Creative, README.md, To, Attribution, to:, includes, directory, 4.0),, you, BY, request,, a, given, contribute, source, datasets, Datasets, License, Contributions, contained, Hosted, or, license, of, request, an, licensed, making, additional, publish., [http://creativecommons.org/licenses/by/4.0/legalcode](http://creativecommons.org/licenses/by/4.0/legalcode))), (3,Array(this, for)), (5,Array(data, "", to)))
```

Command took 9.99 seconds -- by art9518@gmail.com at 5/15/2018, 11:32:55 PM on My Cluster

Resources

Useful Links

- https://github.com/bobfreitas/flume-logs
- https://stackoverflow.com/questions/30908641/save-flume-output-to-hive-table-with-hive-sink?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- https://stackoverflow.com/questions/18501551/how-to-use-flume-creating-a-task-to-load-data-from-hdfs-to-hive-automatic-by-tim?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- https://www.quora.com/Why-do-we-use-row-format-delimited-and-fields-terminated-by-in-table-creation-in-Hive-and-in-Hadoop

Resources

Useful Links

- http://www.lopakalogic.com/articles/hadoop-articles/log-files-flume-hive/
- https://blogs.oracle.com/datawarehousing/flume-and-hive-for-log-analytics
- https://mevivs.wordpress.com/2010/11/24/hivehbase-integration/
- https://www.quora.com/How-can-I-transfer-data-from-Hive-external-table-to-Hbase
- https://stackoverflow.com/questions/3635166/how-to-import-csv-file-to-mysql-table?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- https://stackoverflow.com/questions/32580356/creating-partition-in-external-table-in-hive?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

Github

Repository

You can find this slides and more in my github repo:
https://github.com/Arturou/Hadoop-Exercises/tree/master/hadoop_ecosystem