# ***Music analysis project***

**TOPIC** -------------------------------------------------------------------------- **Page No**

1. Abstract ------------------------------------------------------------------- 3

2. Data Generation -------------------------------------------------------- 8

3. Start Demons -------------------------------------------------------- 18

4. Populate Lookup Tables ------------------------------------------------- 21

5. Data Formatting --------------------------------------------------------- 28

6. Data Enrichment ---------------------------------------------------------- 35

7. Data Analysis with Spark ---------------------------------------------------- 40

# 1. Abstract

Big data has its own way of advantages in processing the live data. It can parse not only the structured data but also un structured and semi structured data. The main usage of big data involves in making the right decision by organizations and observing the user interests (trends ). And it involves in the health care also in predicting and analyzing the data to find out some valid outcomes and react for the future problems. Normal RDBMS have limitations in achieving it. And some other salient features of big data are it is open source and it can be run on commodity hard wares.

And the current project involves in analyzing the music data. The analysis will take the raw data of the users in mobile and web. The raw data consists of the data of the users who is listing the songs, how many they have completed listed, how many subscribed users are there. We want to analyze the data to identify the top stations, top songs which are listed by most of the users, identify the connected artists. And based on this data, loyalty will be applied for the songs which are mostly liked and completely listed by users.

And the analyzed data in used to observe the trend and taste of the users. The taste of the music lovers will change from time to time. By observing the trend, the music companies will encourage the artists which are liked by users. And it will help the music companies to provide a play lists of with the liked songs of users. In this way, they can best features according to users interests. It will help them to gain the customer satisfaction which in turn helps to increase the traffic and revenue options.

The data analysis consists of multiple steps. 1. Data Generation 2. Populate lookup tables 3. Data Formatting 4. Data enrichment 5. Data Analysis

Data formatting involves making data ready for processing. Data enrichment will play a vital role in identifying the data for the analysis. All the unnecessary data and invalid data will be discarded here only and forward the data for analysis. Data analysis will involved in running some criteria and processes to observe the trend.

Now we will go through all the steps of the data.

# 

# 2.Data Generation

In the current project, we will get data from two sources. Mobile and Web.

As we can not get the real time data, we will create the data by some scripts. Here we are using python to create the data.

And here two scripts involved to create data for web and mobile.

The following script will create the web data.

***python /home/cloudera/project/scripts/generate\_web\_data.py***

We will go indetail and dive into the script line by line

*from random import randint*

*from random import choice*

*file = open("/home/cloudera/project/data/web/file.xml", "w")*

*count = 20*

*file.write("<records>\n")*

*while (count > 0):*

*geo\_cd\_list=["A", "E", "AU", "AP", "U"]*

*song\_end\_type\_list=["0","1","2","3"]*

*timestamp\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*start\_ts\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*end\_ts\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*if (count%15 == 0):*

*user\_id = ""*

*else:*

*user\_id = "U" + str(randint(100,120))*

*song\_id = "S" + str(randint(200,210))*

*if (count%11 == 0):*

*artist\_id = ""*

*else:*

*artist\_id = "A" + str(randint(300,305))*

*timestamp = choice(timestamp\_list)*

*start\_ts = choice(start\_ts\_list)*

*end\_ts = choice(end\_ts\_list)*

*if (count%12 == 0):*

*geo\_cd = ""*

*else:*

*geo\_cd = choice(geo\_cd\_list)*

*station\_id = "ST" + str(randint(400,415))*

*song\_end\_type = choice(song\_end\_type\_list)*

*like = str(randint(0,1))*

*dislike = str(randint(0,1))*

*file.write("<record>\n")*

*file.write("<user\_id>%s</user\_id>\n" % (user\_id))*

*file.write("<song\_id>%s</song\_id>\n" % (song\_id))*

*file.write("<artist\_id>%s</artist\_id>\n" % (artist\_id))*

*file.write("<timestamp>%s</timestamp>\n" % (timestamp))*

*file.write("<start\_ts>%s</start\_ts>\n" % (start\_ts))*

*file.write("<end\_ts>%s</end\_ts>\n" % (end\_ts))*

*file.write("<geo\_cd>%s</geo\_cd>\n" % (geo\_cd))*

*file.write("<station\_id>%s</station\_id>\n" % (station\_id))*

*file.write("<song\_end\_type>%s</song\_end\_type>\n" % (song\_end\_type))*

*file.write("<like>%s</like>\n" % (like))*

*file.write("<dislike>%s</dislike>\n" % (dislike))*

*file.write("</record>\n")*

*count = count-1*

*file.write("</records>")*

*file.close()*

Following is the detialed explanation of the code

*from random import randint*

*from random import choice*

Importing the necessary python libries

*file = open("/home/cloudera/project/data/web/file.xml", "w")*

*count = 20*

It will open the file if exists. “w” means if file is not exists. It will create.

Count = 20 ( it is a variable to used to get the 20 records for one process. In the current project we can say it is batch.

*file.write("<records>\n")*

As it is xml file, first line is written to file. I.e <records>

*while (count > 0):*

*geo\_cd\_list=["A", "E", "AU", "AP", "U"]*

*song\_end\_type\_list=["0","1","2","3"]*

*timestamp\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*start\_ts\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*end\_ts\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

The above code is in while loop it will execute upto count is <= 0.

Here geo\_cd\_list, song\_end\_type are the arrays which we use to generate the data

Start\_ts\_list and end\_ts\_list are the differnet dates which are in YYYY-mm-dd h:i:s format

*if (count%15 == 0):*

*user\_id = ""*

*else:*

*user\_id = "U" + str(randint(100,120))*

In this code if count == 15, then user\_id is empty. Here we are creating some use cases where use\_id is empty. As per our project we need to discard the records whose user\_id = null.

If count != 15, some random user\_id created from number 100 to 120. It can be U110 or U U115

*song\_id = "S" + str(randint(200,210))*

Creating dynamic song id. First letter is S and appending the random numbers between 200 and 210

Example: S205, S210

*f (count%11 == 0):*

*artist\_id = ""*

*else:*

*artist\_id = "A" + str(randint(300,305))*

The above code creates the random artist\_id. And same like above when count == 11, the artist\_id is null. For remaining cases it will be like A302, A305

*timestamp = choice(timestamp\_list)*

*start\_ts = choice(start\_ts\_list)*

*end\_ts = choice(end\_ts\_list)*

The above code randomly selects the one values in the arreys provided in the starting

*if (count%12 == 0):*

*geo\_cd = ""*

*else:*

*geo\_cd = choice(geo\_cd\_list)*

It generates the geo\_cd. If geo\_cd == 12 it is empty. If it is not equal = 12, randomly one value will be from geo\_cd\_list array

*station\_id = "ST" + str(randint(400,415))*

*song\_end\_type = choice(song\_end\_type\_list)*

*like = str(randint(0,1))*

*dislike = str(randint(0,1))*

It creates the random station\_id, song\_end\_type

And like and dislike will be 1 or 0. They are randomly picked

*file.write("<record>\n")*

*file.write("<user\_id>%s</user\_id>\n" % (user\_id))*

*file.write("<song\_id>%s</song\_id>\n" % (song\_id))*

*file.write("<artist\_id>%s</artist\_id>\n" % (artist\_id))*

*file.write("<timestamp>%s</timestamp>\n" % (timestamp))*

*file.write("<start\_ts>%s</start\_ts>\n" % (start\_ts))*

*file.write("<end\_ts>%s</end\_ts>\n" % (end\_ts))*

*file.write("<geo\_cd>%s</geo\_cd>\n" % (geo\_cd))*

*file.write("<station\_id>%s</station\_id>\n" % (station\_id))*

*file.write("<song\_end\_type>%s</song\_end\_type>\n" % (song\_end\_type))*

*file.write("<like>%s</like>\n" % (like))*

*file.write("<dislike>%s</dislike>\n" % (dislike))*

*file.write("</record>\n")*

And It writes all the parameters to file in xml format

*count = count-1*

It is decremented by 1 for one while loop iteration

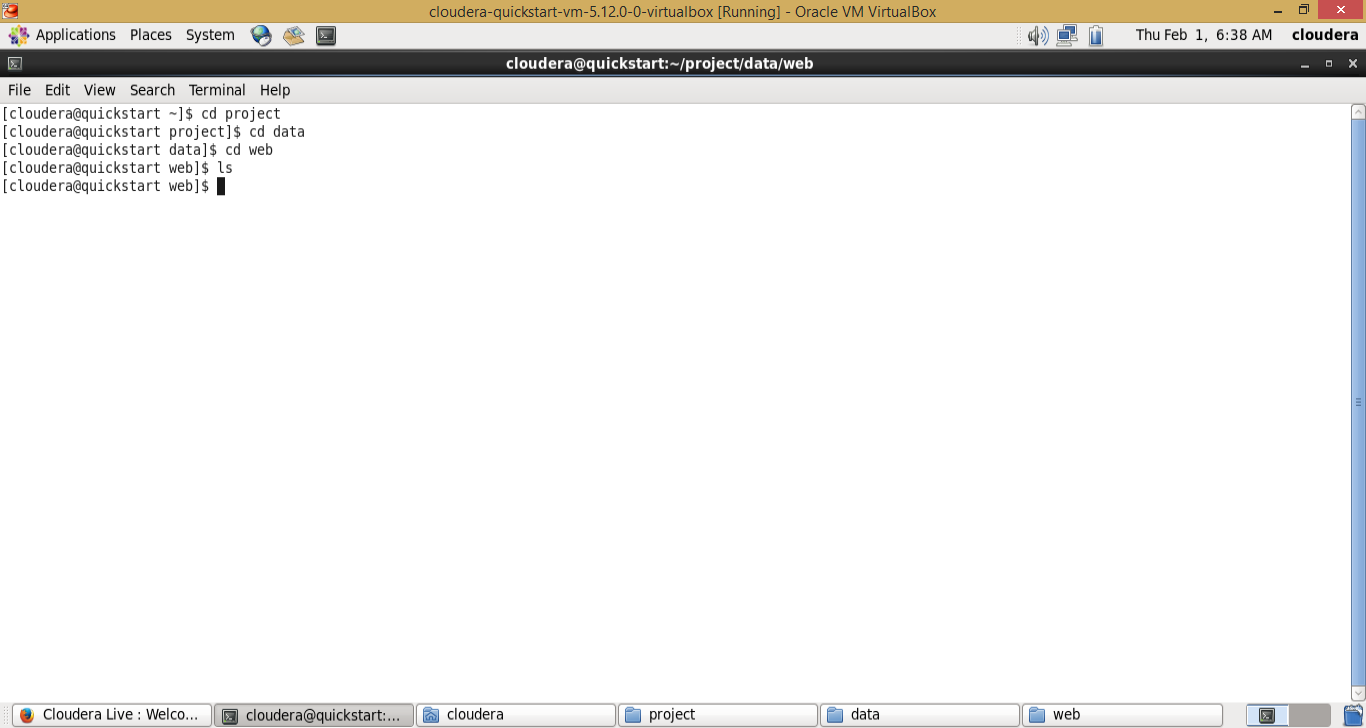
When count = 0 the loop will be stopped. It means for one process 20 records will be created.

*file.write("</records>")*

*file.close()*

Once the while loop is closed the xml file end tag is added </records> and file is closed. And in python the indentation code is treated in the loop. And the above code is normal indentation. So, they are out of the loop.

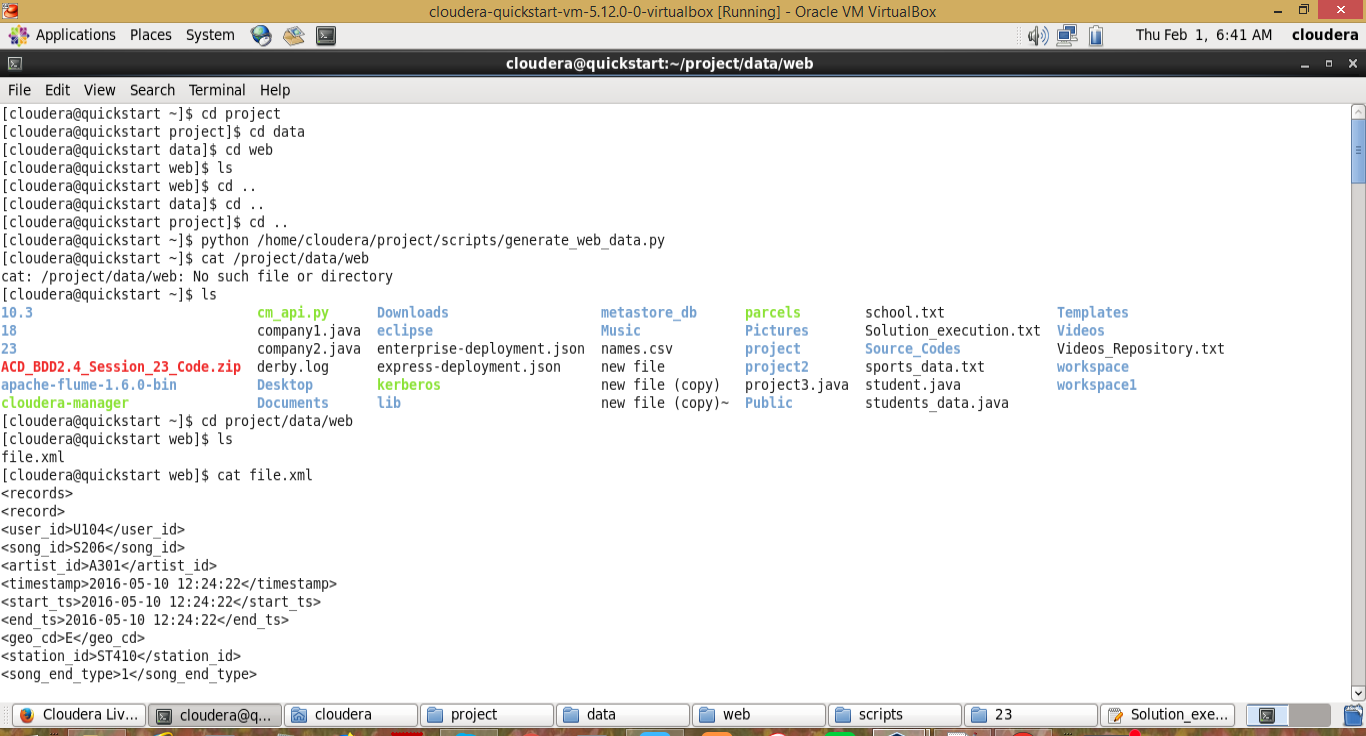
Before executing the script



Now script is executed

***python /home/cloudera/project/scripts/generate\_web\_data.py***

After script executed



So, Once the above code is executed. One file is created in the location. (/home/cloudera/project/data/web/file.xml)

Now Create the data for mobile

*from random import randint*

*from random import choice*

*file = open("/home/cloudera/project/data/mob/file.txt", "w")*

*count = 20*

*while (count > 0):*

*geo\_cd\_list=["A", "E", "AU", "AP", "U"]*

*song\_end\_type\_list=["0","1","2","3"]*

*timestamp\_list=["1465230523", "1465130523", "1475130523", "1495130523"]*

*start\_ts\_list=["1465230523", "1465130523", "1475130523", "1485130523"]*

*end\_ts\_list=["1465230523", "1465130523", "1475130523", "1485130523"]*

*if (count%15 == 0):*

*user\_id = ""*

*else:*

*user\_id = "U" + str(randint(100,120))*

*song\_id = "S" + str(randint(200,210))*

*if (count%11 == 0):*

*artist\_id = ""*

*else:*

*artist\_id = "A" + str(randint(300,305))*

*timestamp = choice(timestamp\_list)*

*start\_ts = choice(start\_ts\_list)*

*end\_ts = choice(end\_ts\_list)*

*if (count%12 == 0):*

*geo\_cd = ""*

*else:*

*geo\_cd = choice(geo\_cd\_list)*

*station\_id = "ST" + str(randint(400,415))*

*song\_end\_type = choice(song\_end\_type\_list)*

*like = str(randint(0,1))*

*dislike = str(randint(0,1))*

*file.write("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n" % (user\_id, song\_id, artist\_id, timestamp, start\_ts, end\_ts, geo\_cd, station\_id, song\_end\_type, like, dislike))*

*count = count-1*

*file.close()*

The above code creates the input data for mobile

Following is the detailed explanation for the code step by step

*file = open("/home/cloudera/project/data/mob/file.txt", "w")*

It will create the file.txt if it available. It it is not available. It will be created

*count = 20*

Creating a counter to create a data of 20 records

*while (count > 0):*

*geo\_cd\_list=["A", "E", "AU", "AP", "U"]*

*song\_end\_type\_list=["0","1","2","3"]*

*timestamp\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*start\_ts\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

*end\_ts\_list=["2016-05-10 12:24:22", "2016-06-09 22:12:36", "2016-07-10 01:38:09", "2017-05-09 08:09:22"]*

The above code is in while loop it will execute upto count is <= 0.

Here geo\_cd\_list, song\_end\_type are the arrays which we use to generate the data

Start\_ts\_list and end\_ts\_list are the differnet dates which are in YYYY-mm-dd h:i:s format

*if (count%15 == 0):*

*user\_id = ""*

*else:*

*user\_id = "U" + str(randint(100,120))*

In this code if count == 15, then user\_id is empty. Here we are creating some use cases where use\_id is empty. As per our project we need to discard the records whose user\_id = null.

If count != 15, some random user\_id created from number 100 to 120. It can be U110 or U U115

*song\_id = "S" + str(randint(200,210))*

Creating dynamic song id. First letter is S and appending the random numbers between 200 and 210

Example: S205, S210

*f (count%11 == 0):*

*artist\_id = ""*

*else:*

*artist\_id = "A" + str(randint(300,305))*

The above code creates the random artist\_id. And same like above when count == 11, the artist\_id is null. For remaining cases it will be like A302, A305

*timestamp = choice(timestamp\_list)*

*start\_ts = choice(start\_ts\_list)*

*end\_ts = choice(end\_ts\_list)*

The above code randomly selects the one values in the arreys provided in the starting

*if (count%12 == 0):*

*geo\_cd = ""*

*else:*

*geo\_cd = choice(geo\_cd\_list)*

It generates the geo\_cd. If geo\_cd == 12 it is empty. If it is not equal = 12, randomly one value will be from geo\_cd\_list array

*station\_id = "ST" + str(randint(400,415))*

*song\_end\_type = choice(song\_end\_type\_list)*

*like = str(randint(0,1))*

*dislike = str(randint(0,1))*

It creates the random station\_id, song\_end\_type

And like and dislike will be 1 or 0. They are randomly picked

*file.write("%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n" % (user\_id, song\_id, artist\_id, timestamp, start\_ts, end\_ts, geo\_cd, station\_id, song\_end\_type, like, dislike))*

*count = count-1*

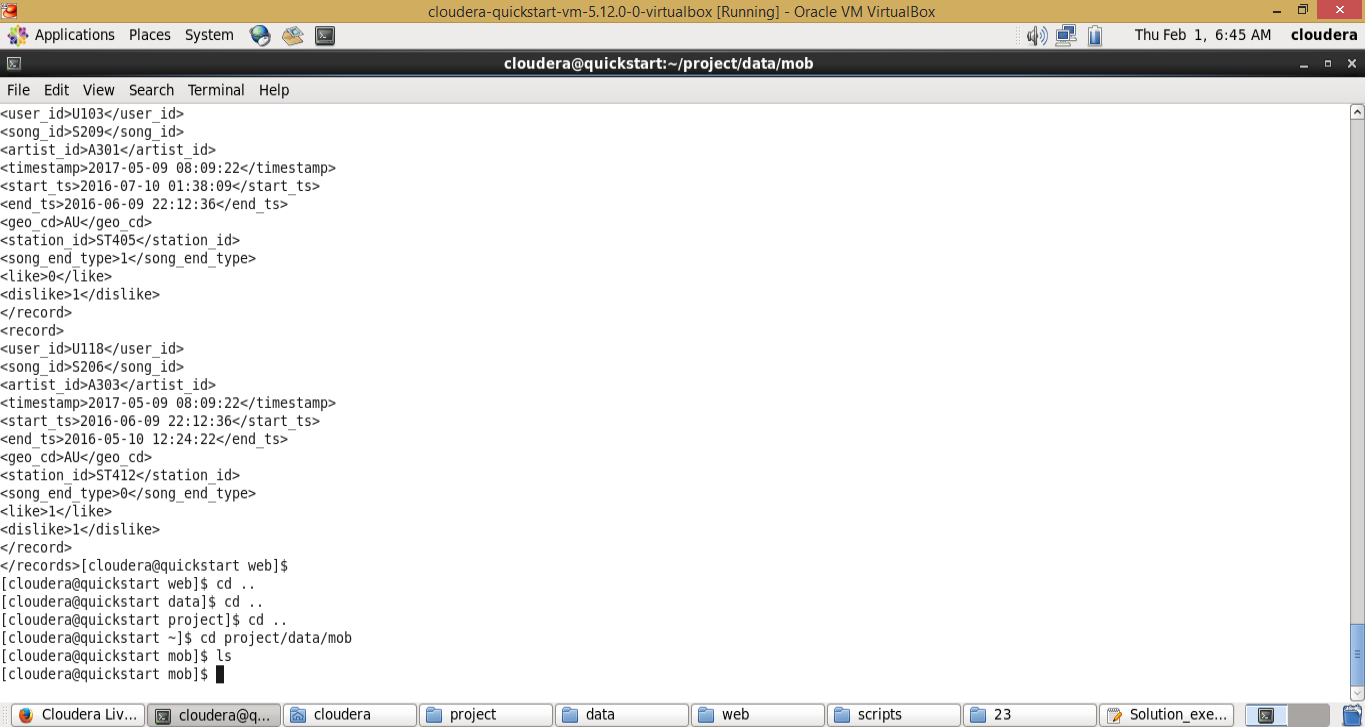
It writes all the data to file.txt

Counter is reduced by 1 for all the iteration

*file.close()*

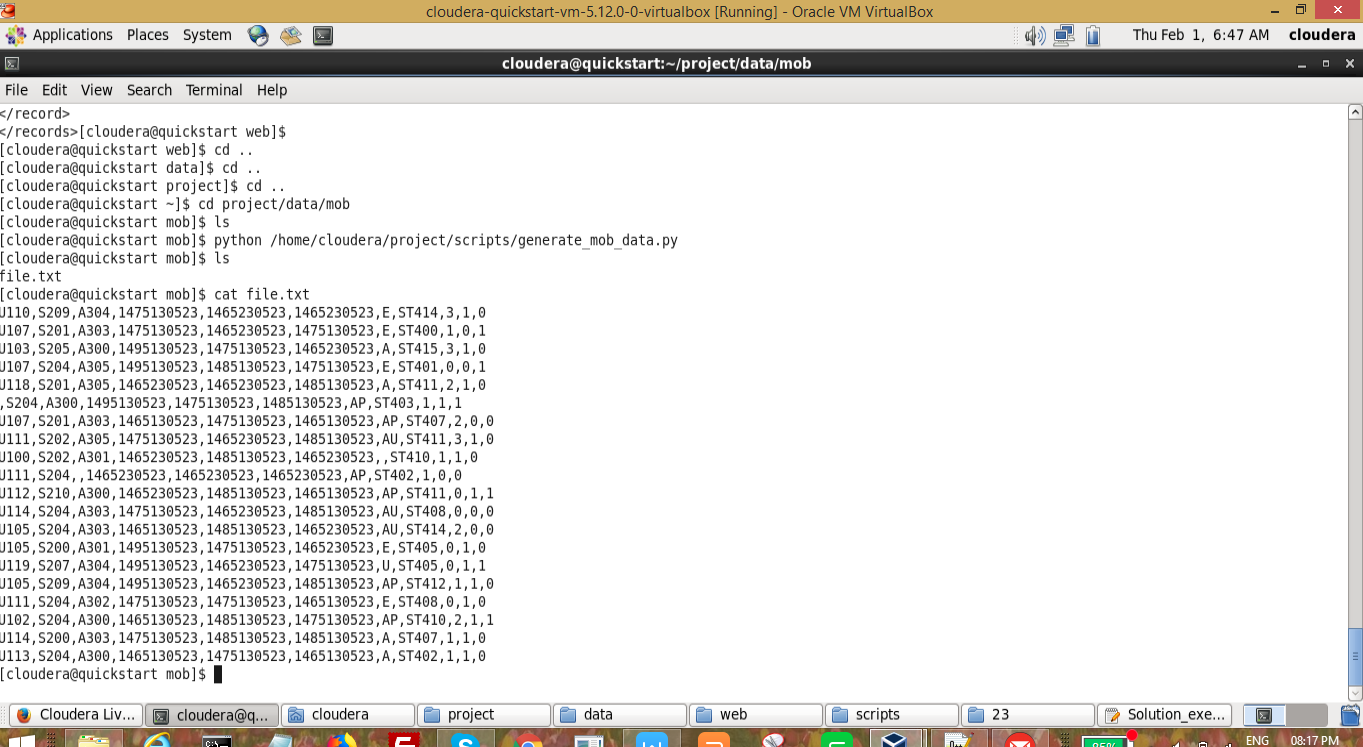
File is closed after the while loop

Before code is executed



Now script is executed

***python /home/cloudera/project/scripts/generate\_mob\_data.py***



# 3. Start Demons

***sh /home/cloudera/project/scripts/start-daemons.sh***

This script starts all the necessary demons

*#!/bin/bash*

*if [ -f "/home/cloudera/project/logs/current-batch.txt" ]*

*then*

*echo "Batch File Found!"*

*else*

*echo -n "1" > "/home/cloudera/project/logs/current-batch.txt"*

*fi*

*chmod 775 /home/cloudera/project/logs/current-batch.txt*

*batchid=`cat /home/cloudera/project/logs/current-batch.txt`*

*LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid*

*echo "Starting daemons" >> $LOGFILE*

*start-all.sh*

*start-hbase.sh*

*mr-jobhistory-daemon.sh start historyserver*

The above checks for batch id and starts all the demons. Now go through the all the stpes

*if [ -f "/home/cloudera/project/logs/current-batch.txt" ]*

*then*

*echo "Batch File Found!"*

*else*

*echo -n "1" > "/home/cloudera/project/logs/current-batch.txt"*

*fi*

It will check for he current-batch.txt file available or not.

If it is not available, it will keep the value 1 in it. It means it is the first batch

*chmod 775 /home/cloudera/project/logs/current-batch.txt*

Providing the 755 permission to the file for read and writes to update the batch id

*batchid=`cat /home/cloudera/project/logs/current-batch.txt`*

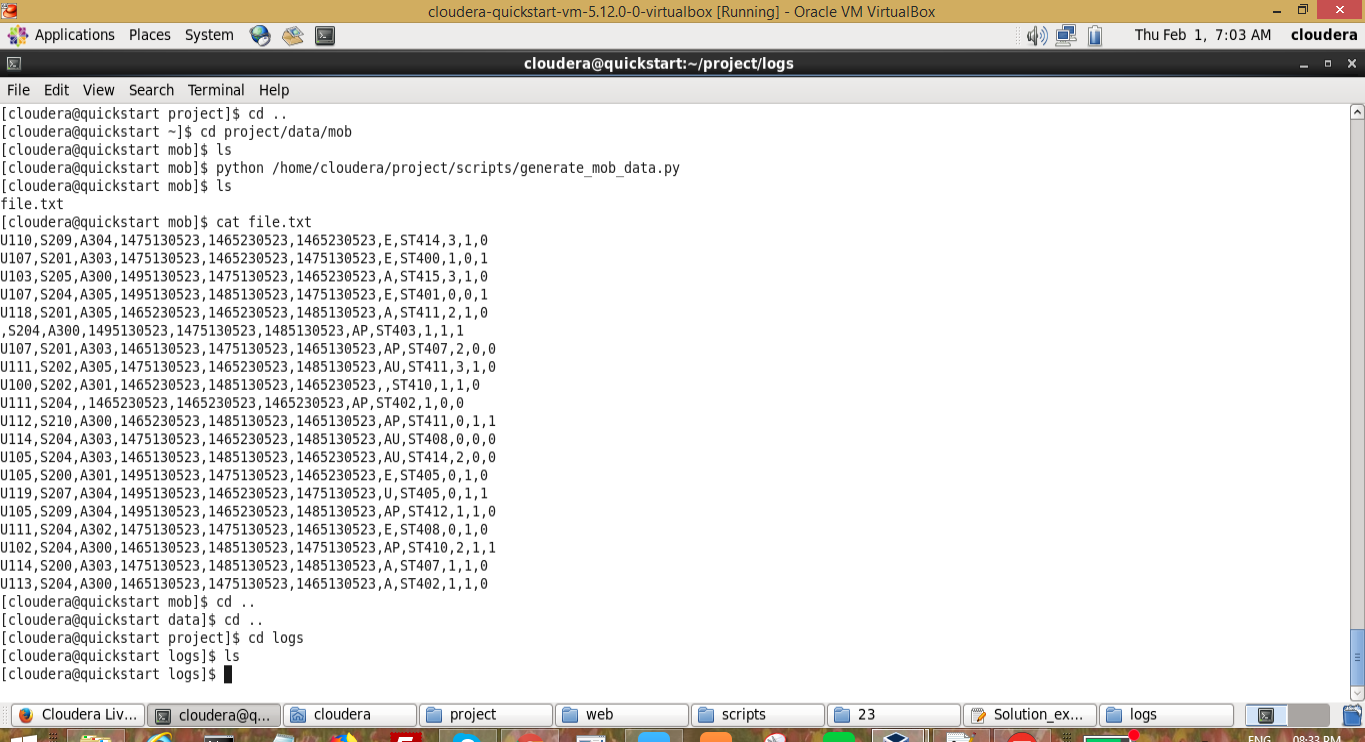
It reads the batchid for from current-batch.txt file

*LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid*

*echo "Starting daemons" >> $LOGFILE*

Creates the log\_batch\_1 file ( for first batch ) and adding the content “Starting the daemons”

Before executing this script logs folder is empty



Now execute the script

***sh /home/cloudera/project/scripts/start-daemons.sh***

After executing the script two files are created on logs folder. Following is the screenshot and data for it



# 

# 4. Populate Lookup Tables

Here we are dumping all the tables to hbase. The tables are used for reference if we get any live data is empty or null.

***sh /home/cloudera/project/scripts/populate-lookup.sh***

*#!/bin/bash*

*batchid=`cat /home/cloudera/project/logs/current-batch.txt`*

*LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid*

*echo "Creating LookUp Tables" >> $LOGFILE*

*echo "create 'station-geo-map', 'geo'" | hbase shell*

*echo "create 'subscribed-users', 'subscn'" | hbase shell*

*echo "create 'song-artist-map', 'artist'" | hbase shell*

*echo "Populating LookUp Tables" >> $LOGFILE*

*file="/home/cloudera/project/lookupfiles/stn-geocd.txt"*

*while IFS= read -r line*

*do*

*stnid=`echo $line | cut -d',' -f1`*

*geocd=`echo $line | cut -d',' -f2`*

*echo "put 'station-geo-map', '$stnid', 'geo:geo\_cd', '$geocd'" | hbase shell*

*done <"$file"*

*file="/home/cloudera/project/lookupfiles/song-artist.txt"*

*while IFS= read -r line*

*do*

*songid=`echo $line | cut -d',' -f1`*

*artistid=`echo $line | cut -d',' -f2`*

*echo "put 'song-artist-map', '$songid', 'artist:artistid', '$artistid'" | hbase shell*

*done <"$file"*

*file="/home/cloudera/project/lookupfiles/user-subscn.txt"*

*while IFS= read -r line*

*do*

*userid=`echo $line | cut -d',' -f1`*

*startdt=`echo $line | cut -d',' -f2`*

*enddt=`echo $line | cut -d',' -f3`*

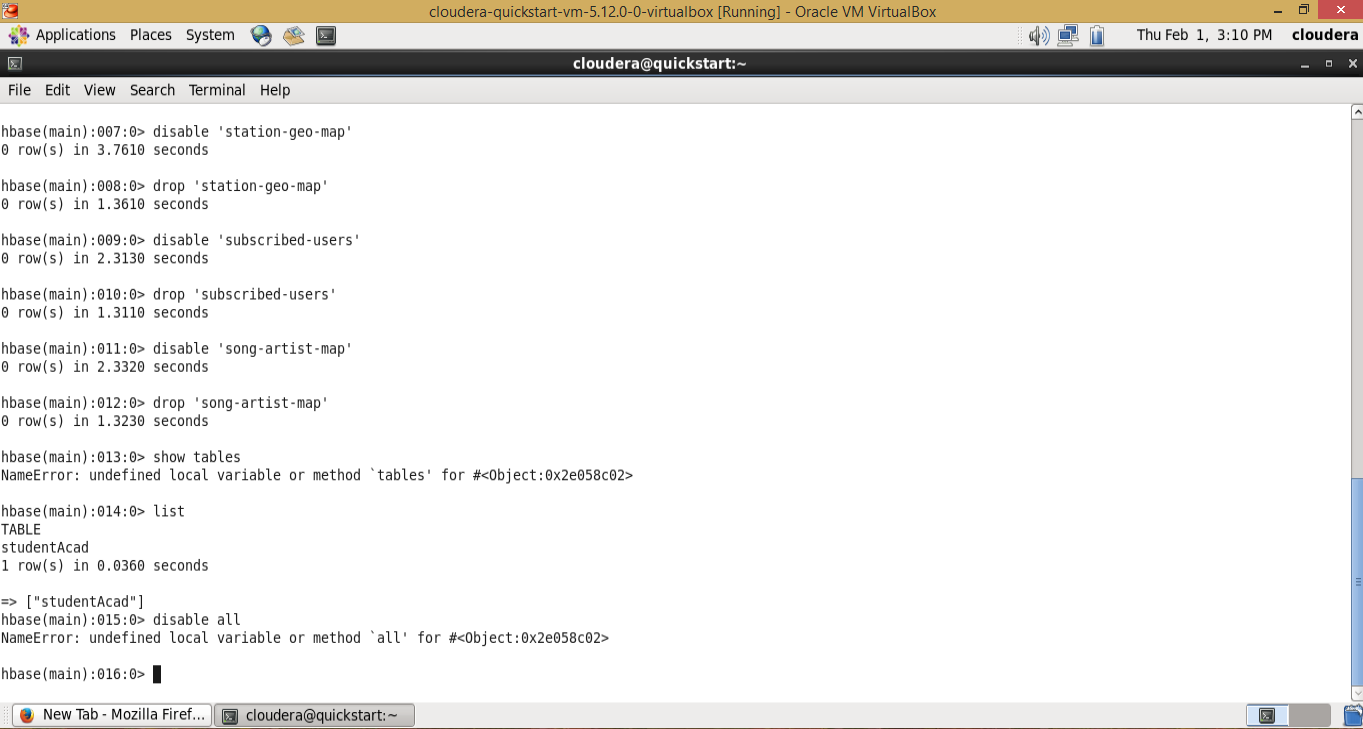
*echo "put 'subscribed-users', '$userid', 'subscn:startdt', '$startdt'" | hbase shell*

*echo "put 'subscribed-users', '$userid', 'subscn:enddt', '$enddt'" | hbase shell*

*done <"$file"*

*hive -f /home/cloudera/project/scripts/user-artist.hql*

Before running this script, no tables available in habase.



Now we will go indetail of the code

*batchid=`cat /home/cloudera/project/logs/current-batch.txt`*

*LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid*

*echo "Creating LookUp Tables" >> $LOGFILE*

Fetches the batch id and updating the logs file with the appropriate comments for the debugging later

*echo "create 'station-geo-map', 'geo'" | hbase shell*

*echo "create 'subscribed-users', 'subscn'" | hbase shell*

*echo "create 'song-artist-map', 'artist'" | hbase shell*

Creates the hbase tables

*file="/home/cloudera/project/lookupfiles/stn-geocd.txt"*

*while IFS= read -r line*

*do*

*stnid=`echo $line | cut -d',' -f1`*

*geocd=`echo $line | cut -d',' -f2`*

*echo "put 'station-geo-map', '$stnid', 'geo:geo\_cd', '$geocd'" | hbase shell*

*done <"$file"*

Copy the stn-geocd.txt data to hbase table

Put is the command in hbase to insert the data

*file="/home/cloudera/project/lookupfiles/song-artist.txt"*

*while IFS= read -r line*

*do*

*songid=`echo $line | cut -d',' -f1`*

*artistid=`echo $line | cut -d',' -f2`*

*echo "put 'song-artist-map', '$songid', 'artist:artistid', '$artistid'" | hbase shell*

*done <"$file"*

Copy the song-artist.txt data to song-artist-map hbase table

*file="/home/cloudera/project/lookupfiles/user-subscn.txt"*

*while IFS= read -r line*

*do*

*userid=`echo $line | cut -d',' -f1`*

*startdt=`echo $line | cut -d',' -f2`*

*enddt=`echo $line | cut -d',' -f3`*

*echo "put 'subscribed-users', '$userid', 'subscn:startdt', '$startdt'" | hbase shell*

*echo "put 'subscribed-users', '$userid', 'subscn:enddt', '$enddt'" | hbase shell*

*done <"$file"*

Copy the user-subscn.txt data to subscribed-users hbas table

*hive -f /home/cloudera/project/scripts/user-artist.hql*

THe user-artist.hql file contains the below hive code

*CREATE DATABASE IF NOT EXISTS project;*

*USE project;*

*CREATE TABLE users\_artists*

*(*

*user\_id STRING,*

*artists\_array ARRAY<STRING>*

*)*

*ROW FORMAT DELIMITED*

*FIELDS TERMINATED BY ','*

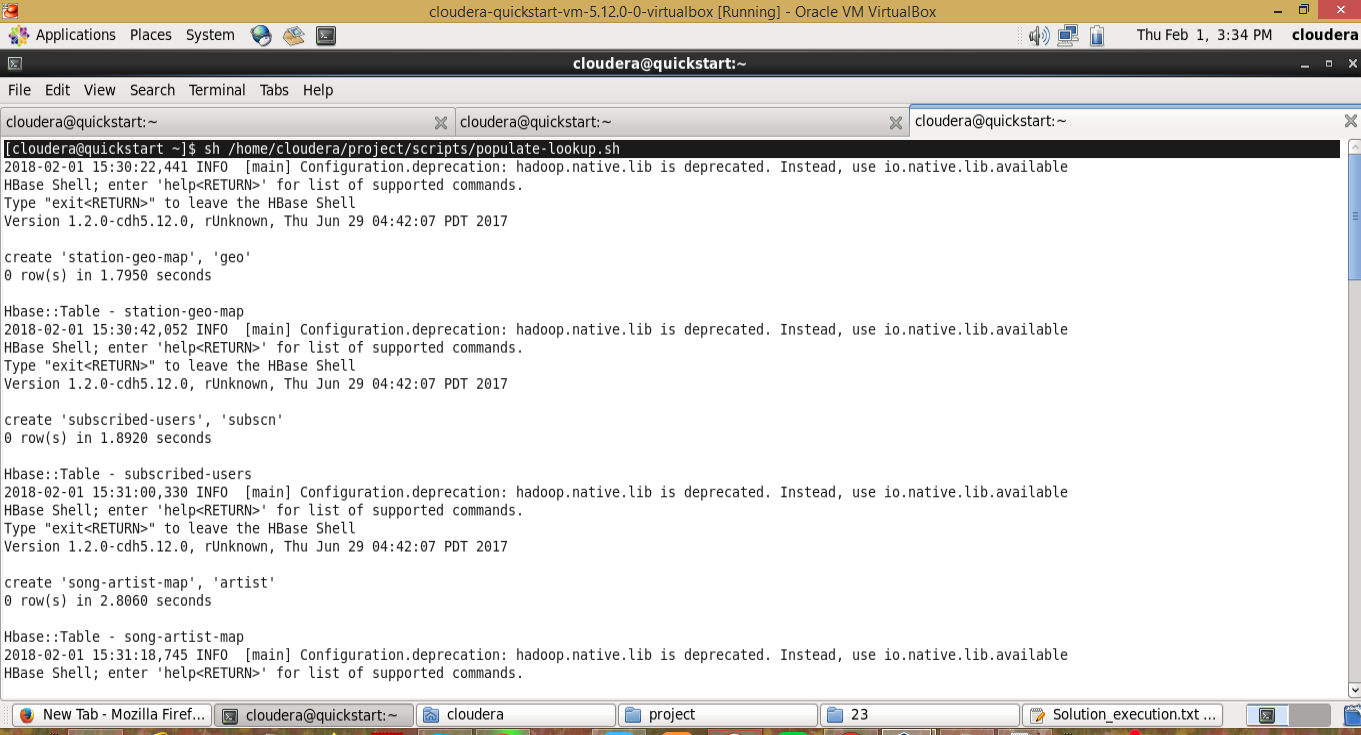
*COLLECTION ITEMS TERMINATED BY '&';*

*LOAD DATA LOCAL INPATH '/home/cloudera/project/lookupfiles/user-artist.txt'*

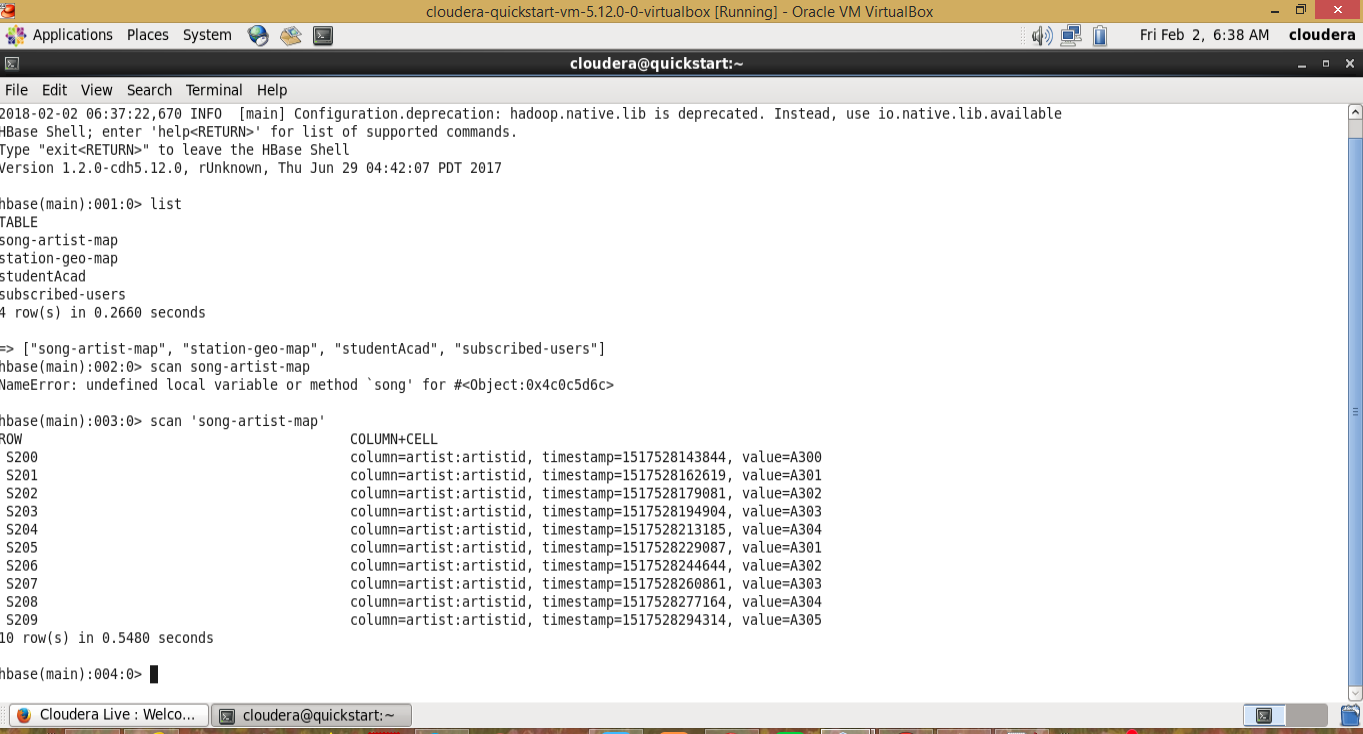
*OVERWRITE INTO TABLE users\_artists;*

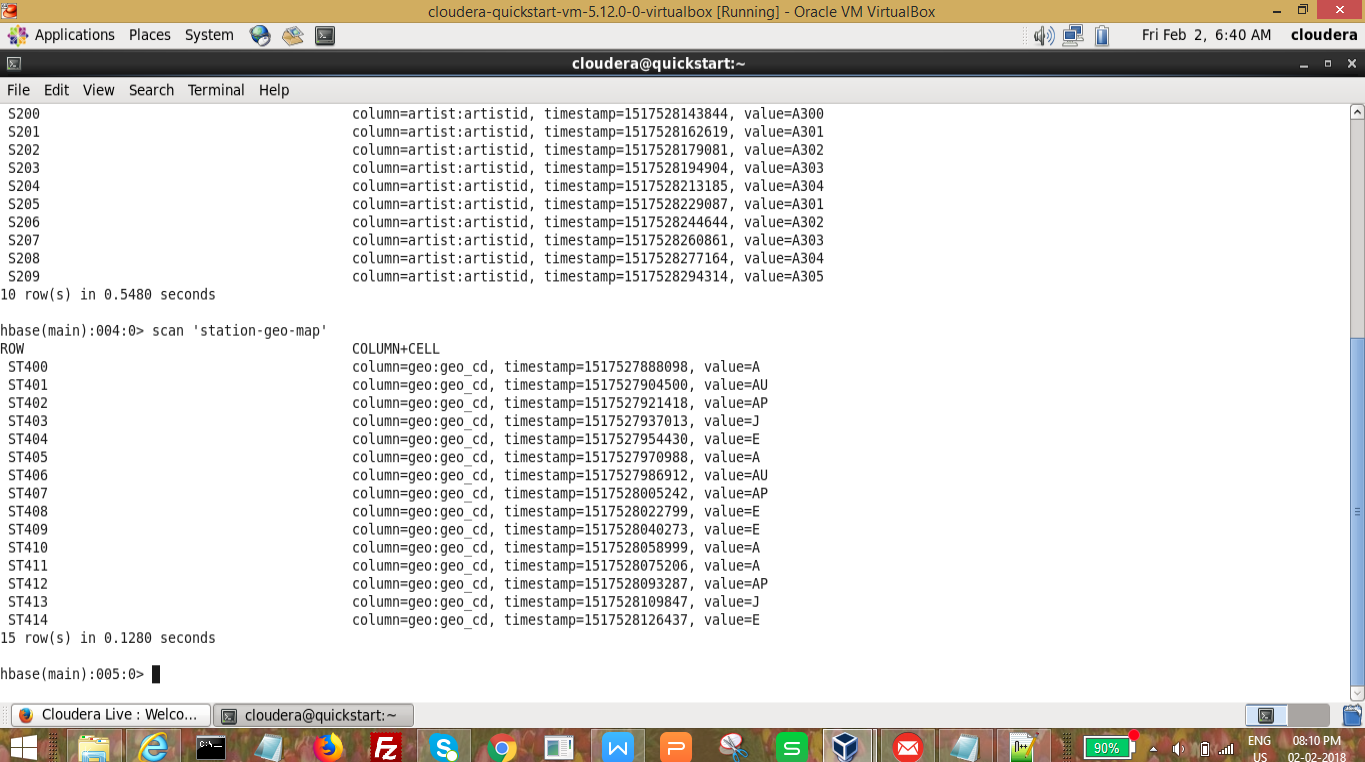
It creates the users\_artists hive table in project database and add the data from user-artist.txt file

Script executed



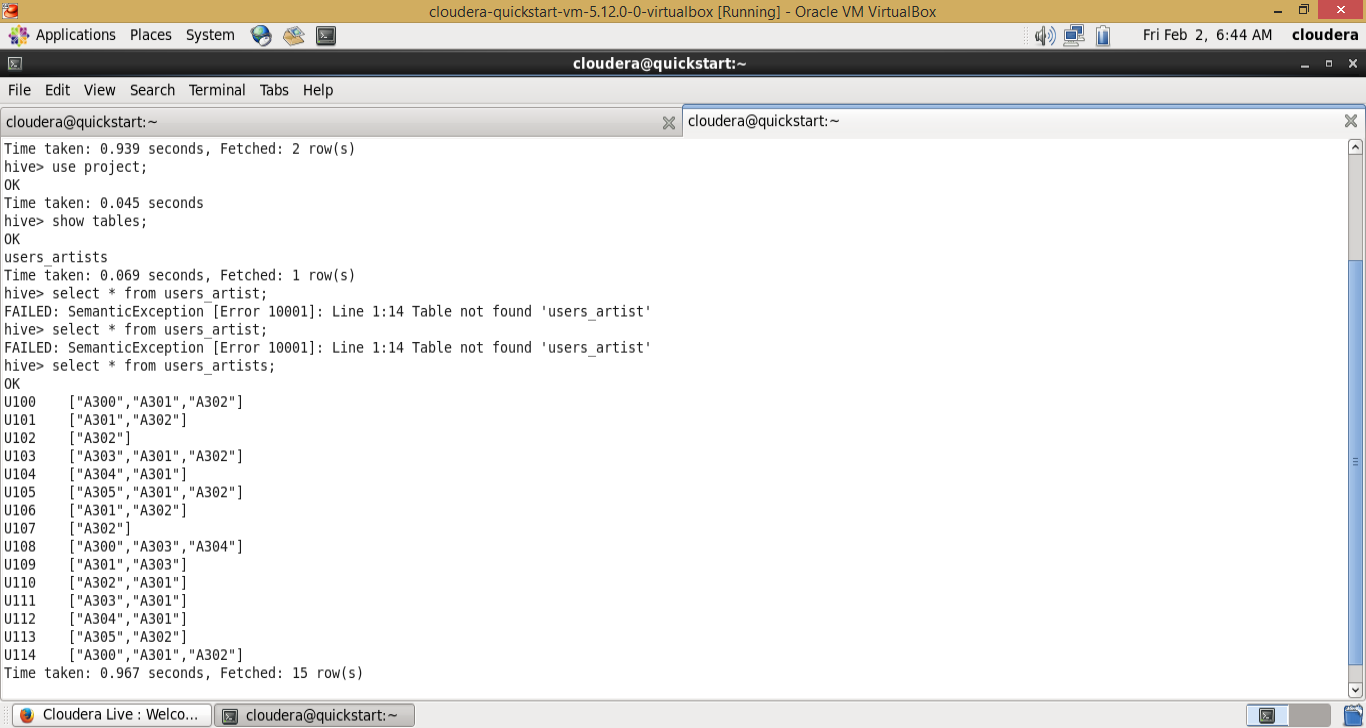
Tables are created on hbase and hive







Hive table created and data is also loaded



# 5. Data Formatting

Data formatting involves moving the input raw data to hdfs. In this way we will make it available for the data enrichment process.

***sh /home/cloudera/project/scripts/dataformatting.sh***

***#!/bin/bash***

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

***echo "Placing data files from local to HDFS..." >> $LOGFILE***

***hadoop fs -rm -r /user/cloudera/project/batch${batchid}/web/***

***hadoop fs -rm -r /user/cloudera/project/batch${batchid}/formattedweb/***

***hadoop fs -rm -r /user/cloudera/project/batch${batchid}/mob/***

***hadoop fs -mkdir -p /user/cloudera/project/batch${batchid}/web/***

***hadoop fs -mkdir -p /user/cloudera/project/batch${batchid}/mob/***

***hadoop fs -put /home/cloudera/project/data/web/\* /user/cloudera/project/batch${batchid}/web/***

***hadoop fs -put /home/cloudera/project/data/mob/\* /user/cloudera/project/batch${batchid}/mob/***

***echo "Running pig script for data formatting..." >> $LOGFILE***

***pig -param batchid=$batchid /home/cloudera/project/scripts/dataformatting.pig***

***echo "Running hive script for formatted data load..." >> $LOGFILE***

***hive -hiveconf batchid=$batchid -f /home/cloudera/project/scripts/formatted\_hive\_load.hql***

The above script moves the data to hdfs and format the data web and mobile

Now we will go through the code by line by line

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

***echo "Placing data files from local to HDFS..." >> $LOGFILE***

Fetch the batchid and update the comments on log\_batch\_1 (if batchid is 1 ) file

***hadoop fs -rm -r /user/cloudera/project/batch${batchid}/web/***

***hadoop fs -rm -r /user/cloudera/project/batch${batchid}/formattedweb/***

***hadoop fs -rm -r /user/cloudera/project/batch${batchid}/mob/***

Remove the folders if exists on the hdfs environment

***hadoop fs -mkdir -p /user/cloudera/project/batch${batchid}/web/***

***hadoop fs -mkdir -p /user/cloudera/project/batch${batchid}/mob/***

Creates the folders on hdfs location for web and mobile

***hadoop fs -put /home/cloudera/project/data/web/\* /user/cloudera/project/batch${batchid}/web/***

***hadoop fs -put /home/cloudera/project/data/mob/\* /user/cloudera/project/batch${batchid}/mob/***

Copies the data from local directory to hdfs environment

***pig -param batchid=$batchid /home/cloudera/project/scripts/dataformatting.pig***

This pig code formats the web data

***REGISTER /home/cloudera/project/lib/piggybank.jar;***

***DEFINE XPath org.apache.pig.piggybank.evaluation.xml.XPath();***

***A = LOAD '/user/cloudera/project/batch${batchid}/web/' using org.apache.pig.piggybank.storage.XMLLoader('record') as (x:chararray);***

***B = FOREACH A GENERATE TRIM(XPath(x, 'record/user\_id')) AS user\_id,***

***TRIM(XPath(x, 'record/song\_id')) AS song\_id,***

***TRIM(XPath(x, 'record/artist\_id')) AS artist\_id,***

***ToUnixTime(ToDate(TRIM(XPath(x, 'record/timestamp')),'yyyy-MM-dd HH:mm:ss')) AS timestamp,***

***ToUnixTime(ToDate(TRIM(XPath(x, 'record/start\_ts')),'yyyy-MM-dd HH:mm:ss')) AS start\_ts,***

***ToUnixTime(ToDate(TRIM(XPath(x, 'record/end\_ts')),'yyyy-MM-dd HH:mm:ss')) AS end\_ts,***

***TRIM(XPath(x, 'record/geo\_cd')) AS geo\_cd,***

***TRIM(XPath(x, 'record/station\_id')) AS station\_id,***

***TRIM(XPath(x, 'record/song\_end\_type')) AS song\_end\_type,***

***TRIM(XPath(x, 'record/like')) AS like,***

***TRIM(XPath(x, 'record/dislike')) AS dislike;***

***STORE B INTO '/user/cloudera/project/batch${batchid}/formattedweb/' USING PigStorage(',');***

***REGISTER /home/cloudera/project/lib/piggybank.jar;***

Register the piggybank to process the xml file

***DEFINE XPath org.apache.pig.piggybank.evaluation.xml.XPath();***

Initiating the Xpath

***A = LOAD '/user/cloudera/project/batch${batchid}/web/' using org.apache.pig.piggybank.storage.XMLLoader('record') as (x:chararray);***

Load the xml file to A variable. Here record is one one xml attribute to process

***B = FOREACH A GENERATE TRIM(XPath(x, 'record/user\_id')) AS user\_id,***

***TRIM(XPath(x, 'record/song\_id')) AS song\_id,***

***TRIM(XPath(x, 'record/artist\_id')) AS artist\_id,***

***ToUnixTime(ToDate(TRIM(XPath(x, 'record/timestamp')),'yyyy-MM-dd HH:mm:ss')) AS timestamp,***

***ToUnixTime(ToDate(TRIM(XPath(x, 'record/start\_ts')),'yyyy-MM-dd HH:mm:ss')) AS start\_ts,***

***ToUnixTime(ToDate(TRIM(XPath(x, 'record/end\_ts')),'yyyy-MM-dd HH:mm:ss')) AS end\_ts,***

***TRIM(XPath(x, 'record/geo\_cd')) AS geo\_cd,***

***TRIM(XPath(x, 'record/station\_id')) AS station\_id,***

***TRIM(XPath(x, 'record/song\_end\_type')) AS song\_end\_type,***

***TRIM(XPath(x, 'record/like')) AS like,***

***TRIM(XPath(x, 'record/dislike')) AS dislike;***

Iterate the A variable and get the data. The fields are user\_id, song\_id, artist\_id, timestamp, starts\_ts, end\_ts, geo\_cd, station\_id, song\_end\_type, like and dislike

***STORE B INTO '/user/cloudera/project/batch${batchid}/formattedweb/' USING PigStorage(',');***

Store the Data to hdfs in formattedweb directory

***hive -hiveconf batchid=$batchid -f /home/cloudera/project/scripts/formatted\_hive\_load.hql***

Hive script formats the data of mobile data

***USE project;***

***CREATE TABLE IF NOT EXISTS formatted\_input***

***(***

***User\_id STRING,***

***Song\_id STRING,***

***Artist\_id STRING,***

***Timestamp STRING,***

***Start\_ts STRING,***

***End\_ts STRING,***

***Geo\_cd STRING,***

***Station\_id STRING,***

***Song\_end\_type INT,***

***Like INT,***

***Dislike INT***

***)***

***PARTITIONED BY***

***(batchid INT)***

***ROW FORMAT DELIMITED***

***FIELDS TERMINATED BY ',';***

***LOAD DATA INPATH '/user/cloudera/project/batch${hiveconf:batchid}/formattedweb/'***

***INTO TABLE formatted\_input PARTITION (batchid=${hiveconf:batchid});***

***LOAD DATA INPATH '/user/cloudera/project/batch${hiveconf:batchid}/mob/'***

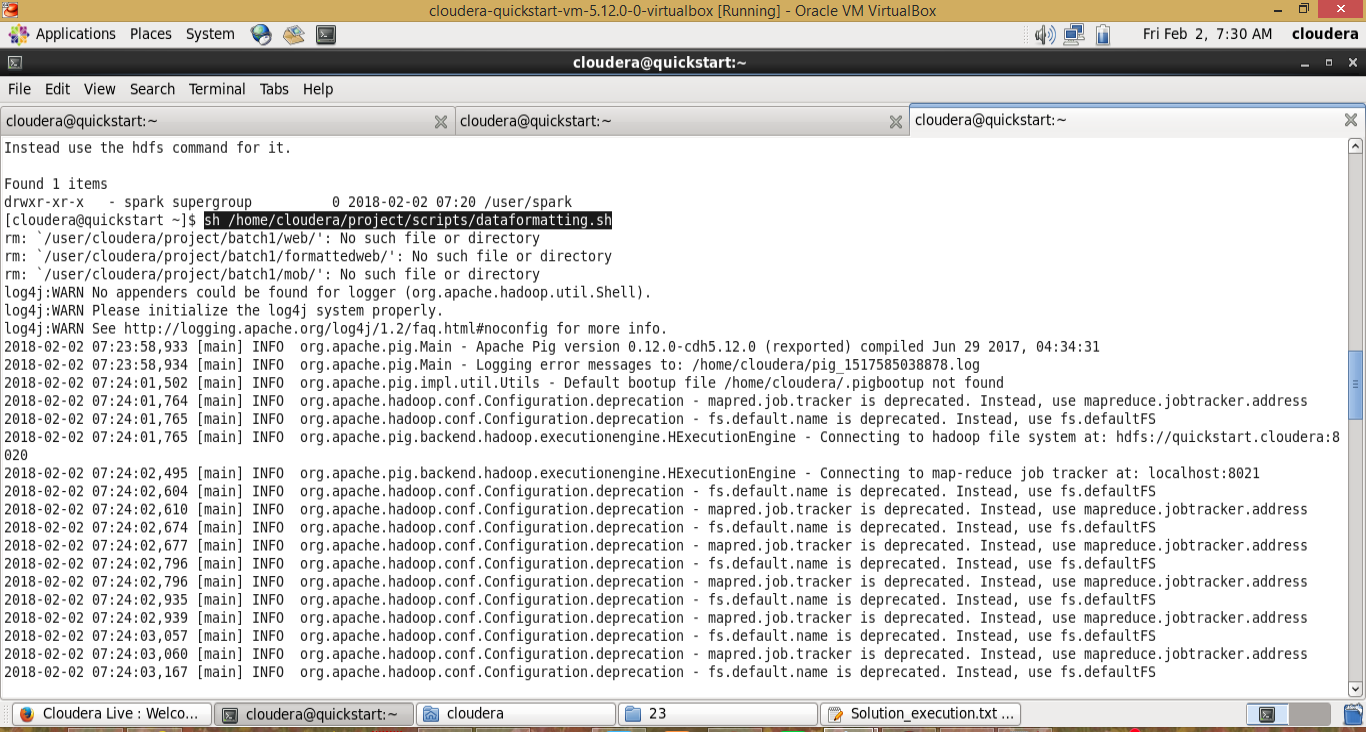
***INTO TABLE formatted\_input PARTITION (batchid=${hiveconf:batchid});***

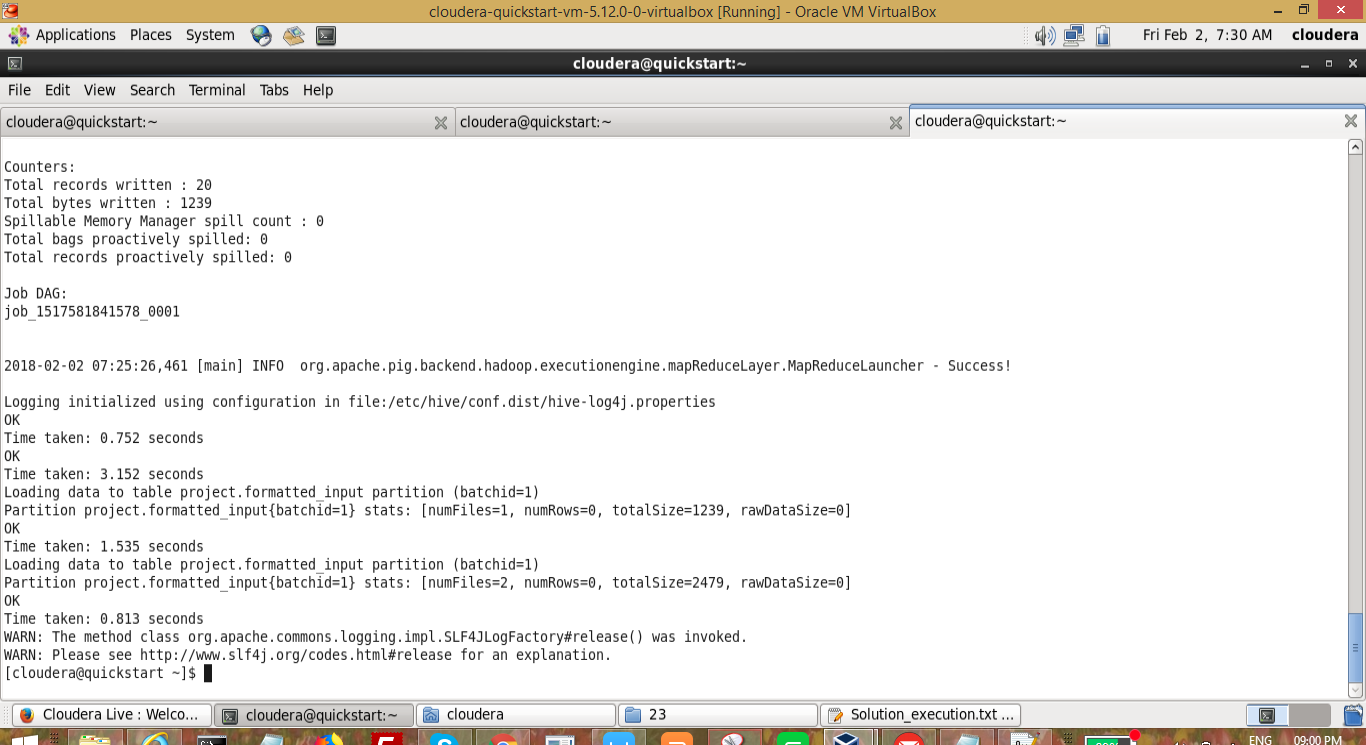
The above hive code creates the table formatted\_input in project database.

And dumps the data of mobile and web to formatted\_input table.

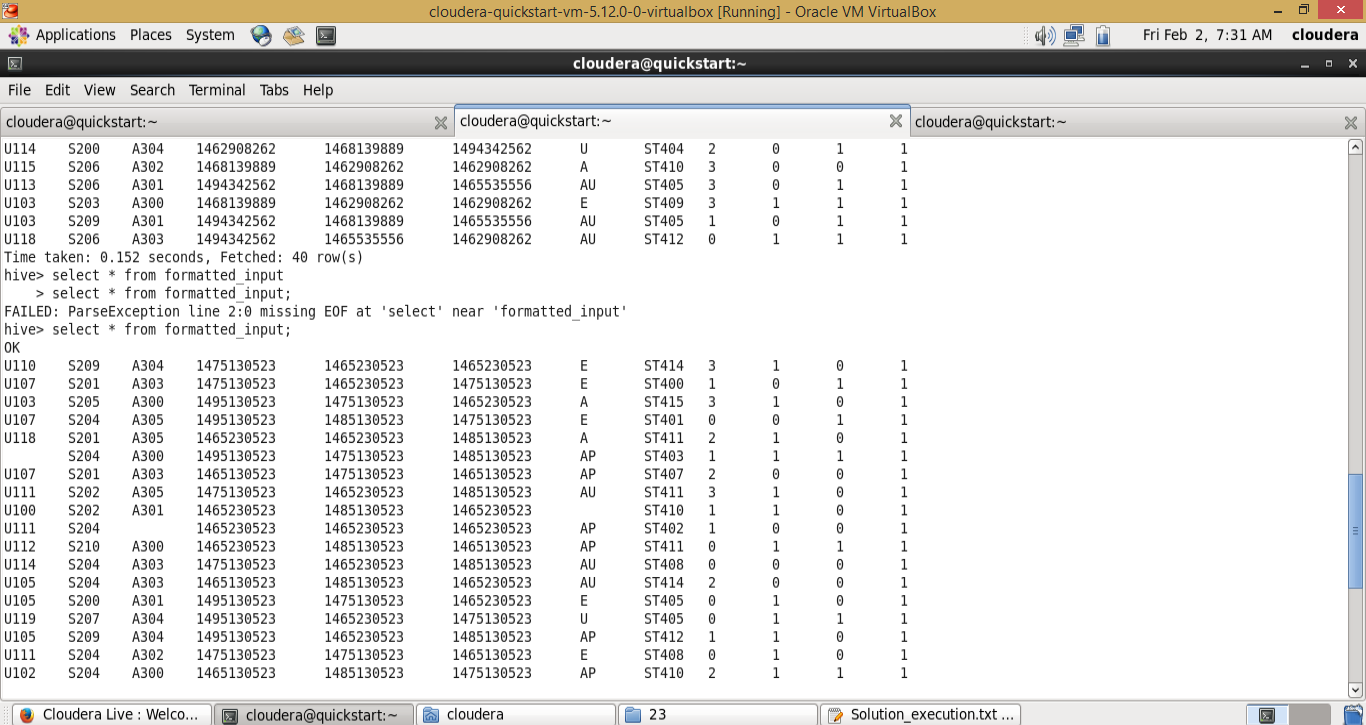
Now the script executed

***sh /home/cloudera/project/scripts/dataformatting.sh***





Data in hive table



Next we will execute the following script which moves the data from hbase to hive

***hive -f /home/cloudera/project/scripts/create\_hive\_hbase\_lookup.hql***

***USE project;***

***create external table if not exists station\_geo\_map***

***(***

***station\_id String,***

***geo\_cd string***

***)***

***STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'***

***with serdeproperties***

***("hbase.columns.mapping"=":key,geo:geo\_cd")***

***tblproperties("hbase.table.name"="station-geo-map");***

***create external table if not exists subscribed\_users***

***(***

***user\_id STRING,***

***subscn\_start\_dt STRING,***

***subscn\_end\_dt STRING***

***)***

***STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'***

***with serdeproperties***

***("hbase.columns.mapping"=":key,subscn:startdt,subscn:enddt")***

***tblproperties("hbase.table.name"="subscribed-users");***

***create external table if not exists song\_artist\_map***

***(***

***song\_id STRING,***

***artist\_id STRING***

***)***

***STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'***

***with serdeproperties***

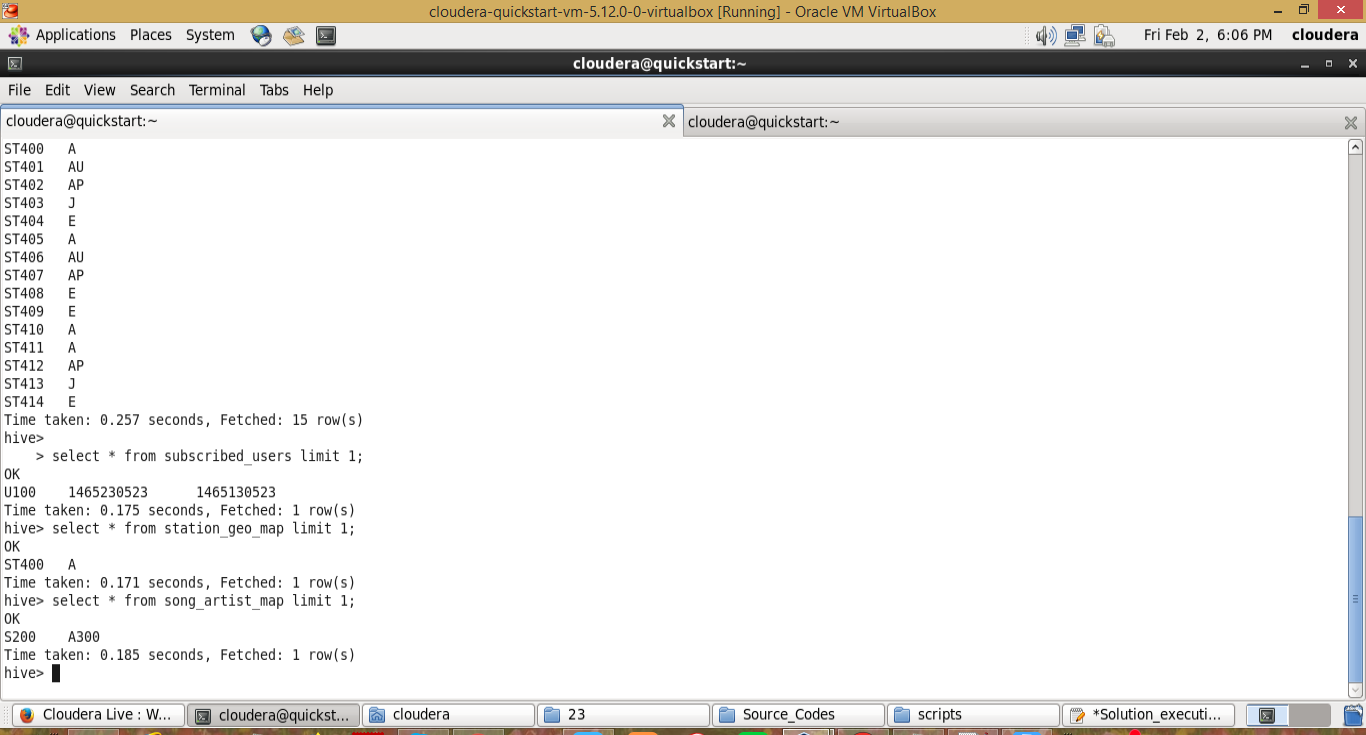
***("hbase.columns.mapping"=":key,artist:artistid")***

***tblproperties("hbase.table.name"="song-artist-map");***

It will create new tables station\_geo\_map, subscribed\_users, song\_artist\_map in the project database



Tables created with data in hive



# 

# 6. Data Enrichment

Data enrichment is a general term that refers to processes used to enhance, refine or otherwise improve raw data. This idea and other similar concepts contribute to making data a valuable asset for almost any modern business or enterprise. It also shows the common imperative of proactively using this data in various ways.

In our current project, the data is treated id invalid, if it not meet certain conditions. They are if song is liked and disliked both, user\_is is null, song\_id is null, timestep is null, start time or end time is empty, user\_id is empty, song\_id is empty, artist\_id is empty. If it meets all the conditions, we will update it with pass otherwise we will make it fail. And we will only consider the pass records for the data analysis

***sh /home/cloudera/project/scripts/data\_enrichment.sh***

***#!/bin/bash***

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

***VALIDDIR=/home/cloudera/project/processed\_dir/valid/batch\_$batchid***

***INVALIDDIR=/home/cloudera/project/processed\_dir/invalid/batch\_$batchid***

***echo "Running hive script for data enrichment and filtering..." >> $LOGFILE***

***hive -hiveconf batchid=$batchid -f /home/cloudera/project/scripts/data\_enrichment.hql***

***if [ ! -d "$VALIDDIR" ]***

***then***

***mkdir -p "$VALIDDIR"***

***fi***

***if [ ! -d "$INVALIDDIR" ]***

***then***

***mkdir -p "$INVALIDDIR"***

***fi***

***echo "Copying valid and invalid records in local file system..." >> $LOGFILE***

***hadoop fs -get /user/hive/warehouse/project.db/enriched\_data/batchid=$batchid/status=pass/\* $VALIDDIR***

***hadoop fs -get /user/hive/warehouse/project.db/enriched\_data/batchid=$batchid/status=fail/\* $INVALIDDIR***

***echo "Deleting older valid and invalid records from local file system..." >> $LOGFILE***

***find /home/cloudera/project/processed\_dir/ -mtime +7 -exec rm {} \;***

The above code is the heart of the project. It enriches or get the appropriate data for the data analysis. Following are the step by step detailed description

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

***VALIDDIR=/home/cloudera/project/processed\_dir/valid/batch\_$batchid***

***INVALIDDIR=/home/cloudera/project/processed\_dir/invalid/batch\_$batchid***

***echo "Running hive script for data enrichment and filtering..." >> $LOGFILE***

Fetch the current batch id and defined the validdir and invaliddir folders. And added the log message to log file

***hive -hiveconf batchid=$batchid -f /home/cloudera/project/scripts/data\_enrichment.hql***

Running hql file which contains the logic. Here batchid is the input param to the hql file

Data\_enrichment.hql file

***SET hive.auto.convert.join=false;***

***SET hive.exec.dynamic.partition.mode=nonstrict;***

***USE project;***

***CREATE TABLE IF NOT EXISTS enriched\_data***

***(***

***User\_id STRING,***

***Song\_id STRING,***

***Artist\_id STRING,***

***Timestamp STRING,***

***Start\_ts STRING,***

***End\_ts STRING,***

***Geo\_cd STRING,***

***Station\_id STRING,***

***Song\_end\_type INT,***

***Like INT,***

***Dislike INT***

***)***

***PARTITIONED BY***

***(batchid INT,***

***status STRING)***

***STORED AS ORC;***

***INSERT OVERWRITE TABLE enriched\_data***

***PARTITION (batchid, status)***

***SELECT***

***i.user\_id,***

***i.song\_id,***

***sa.artist\_id,***

***i.timestamp,***

***i.start\_ts,***

***i.end\_ts,***

***sg.geo\_cd,***

***i.station\_id,***

***IF (i.song\_end\_type IS NULL, 3, i.song\_end\_type) AS song\_end\_type,***

***IF (i.like IS NULL, 0, i.like) AS like,***

***IF (i.dislike IS NULL, 0, i.dislike) AS dislike,***

***i.batchid,***

***IF((i.like=1 AND i.dislike=1)***

***OR i.user\_id IS NULL***

***OR i.song\_id IS NULL***

***OR i.timestamp IS NULL***

***OR i.start\_ts IS NULL***

***OR i.end\_ts IS NULL***

***OR i.geo\_cd IS NULL***

***OR i.user\_id=''***

***OR i.song\_id=''***

***OR i.timestamp=''***

***OR i.start\_ts=''***

***OR i.end\_ts=''***

***OR i.geo\_cd=''***

***OR sg.geo\_cd IS NULL***

***OR sg.geo\_cd=''***

***OR sa.artist\_id IS NULL***

***OR sa.artist\_id='', 'fail', 'pass') AS status***

***FROM formatted\_input i***

***LEFT OUTER JOIN station\_geo\_map sg ON i.station\_id = sg.station\_id***

***LEFT OUTER JOIN song\_artist\_map sa ON i.song\_id = sa.song\_id***

***WHERE i.batchid=${hiveconf:batchid};***

***if [ ! -d "$VALIDDIR" ]***

***then***

***mkdir -p "$VALIDDIR"***

***fi***

***if [ ! -d "$INVALIDDIR" ]***

***then***

***mkdir -p "$INVALIDDIR"***

***fi***

It Creates the valid and invalid directories if not exists

***hadoop fs -get /user/hive/warehouse/project.db/enriched\_data/batchid=$batchid/status=pass/\* $VALIDDIR***

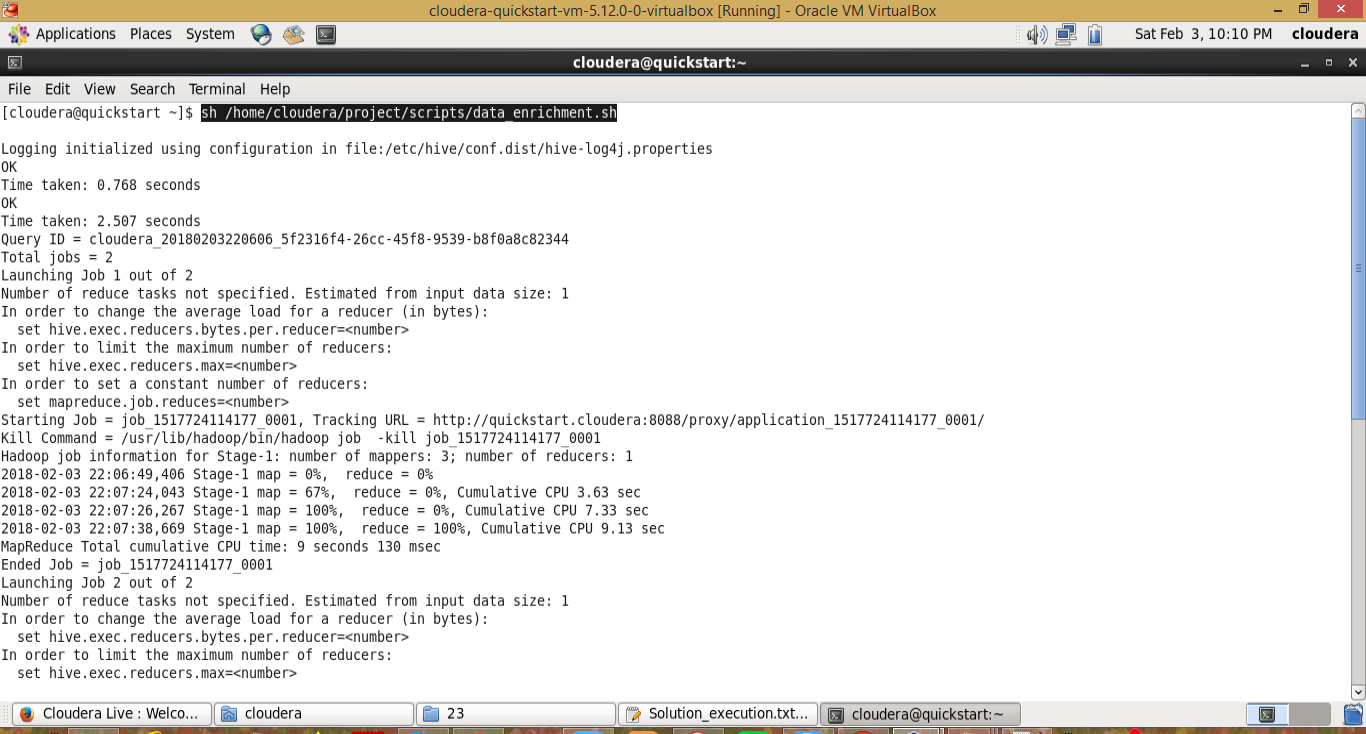
***hadoop fs -get /user/hive/warehouse/project.db/enriched\_data/batchid=$batchid/status=fail/\* $INVALIDDIR***

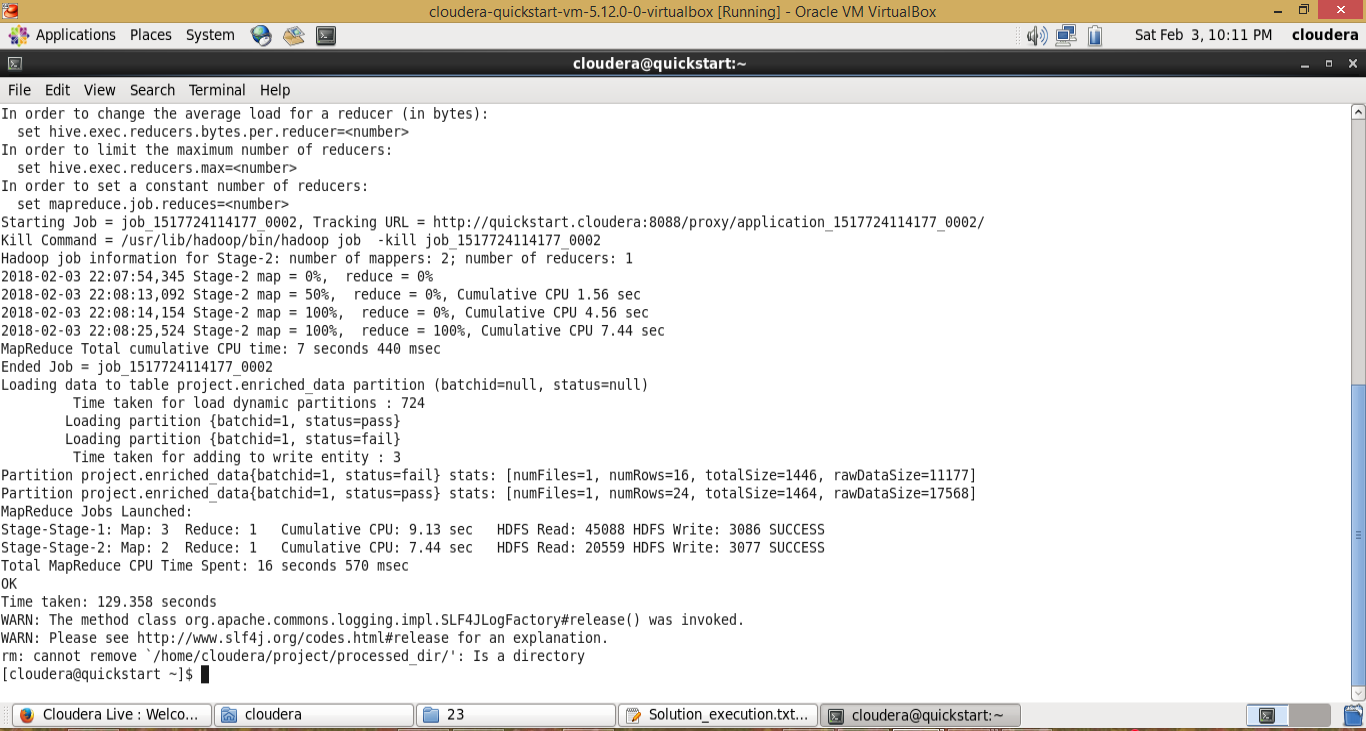
Move the enriched data ( success and failure) to the local directory.

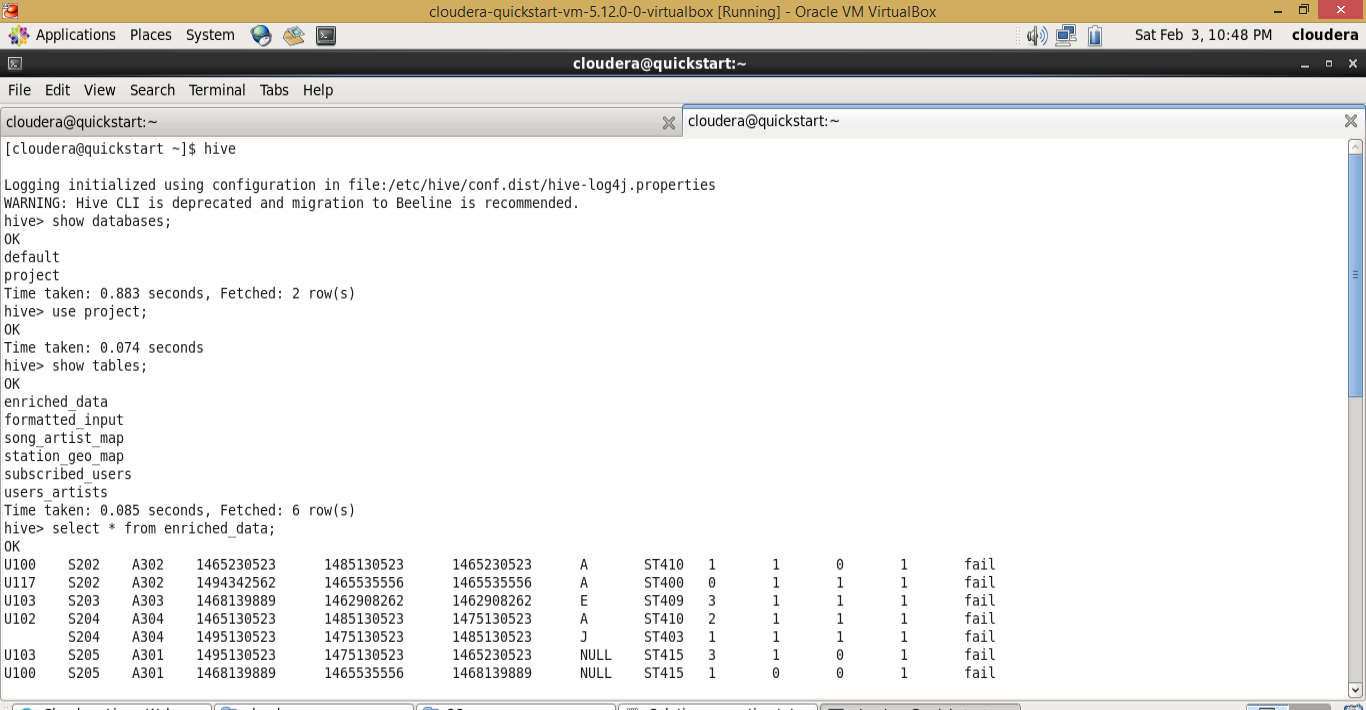
***find /home/cloudera/project/processed\_dir/ -mtime +7 -exec rm {} \;***

Delete the 7 days old records of valid and invalid in the local directory. The cleaning process has to be done all the time to save the system from memory exhaust issues.

We kept 7 days records for reference.





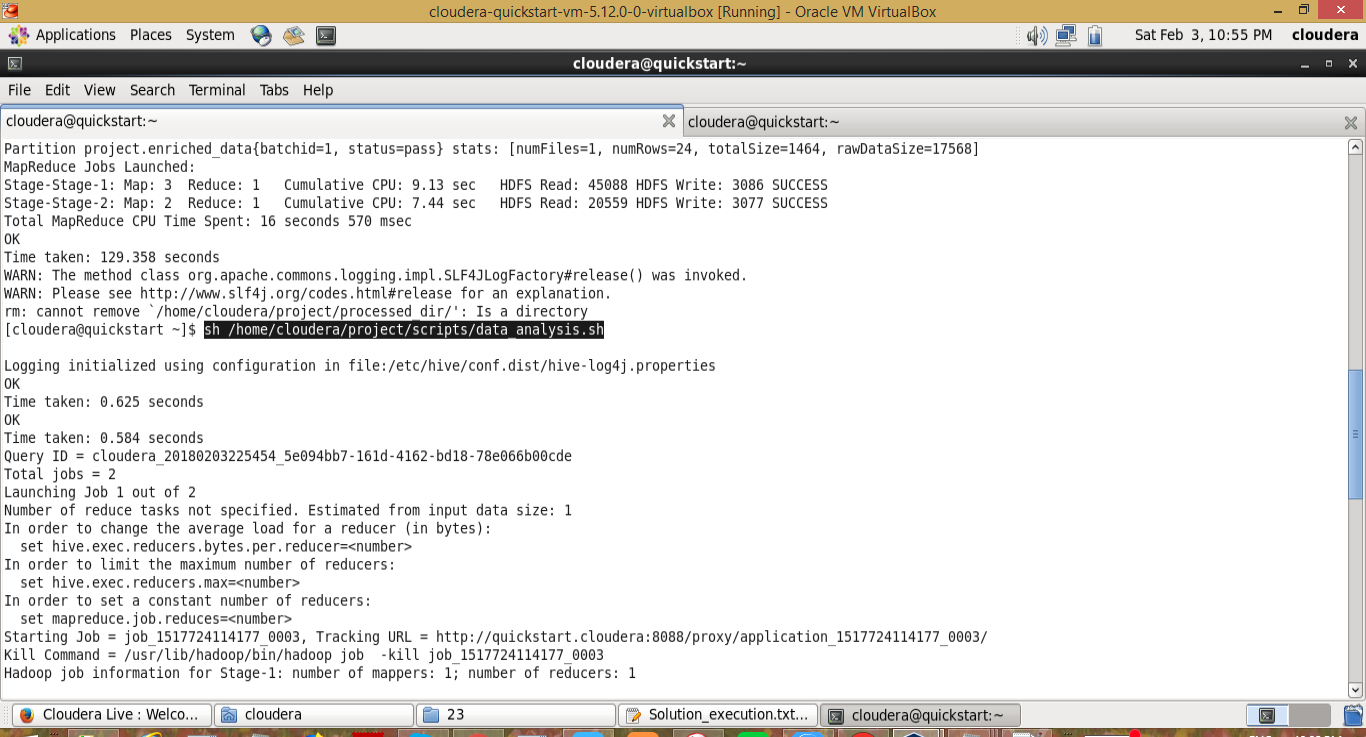


# 

# 7. Data Analysis with Spark

The data which is generated by data enrichment will be considered for the data analysis. In data analysis, we will run some queries and criterias to get the output. Right now in this project, we are using spark to analyze the data. Spark consists of actions and transformations to process the data.

***sh /home/cloudera/project/scripts/data\_analysis.sh***



The script do the analysis of the data.

***#!/bin/bash***

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

***echo "Running hive script for data analysis..." >> $LOGFILE***

***sh /home/cloudera/project/scripts/test2.sh***

***sh /home/cloudera/project/scripts/data\_export.sh***

***echo "Incrementing batchid..." >> $LOGFILE***

***batchid=`expr $batchid + 1`***

***echo -n $batchid > /home/cloudera/project/logs/current-batch.txt***

The above scripts do the analysis by the queries and exports the data to mysql.

*batchid=`cat /home/cloudera/project/logs/current-batch.txt`*

*LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid*

*echo "Running hive script for data analysis..." >> $LOGFILE*

Read the current batchId and update the logfile with proper comments which useful for the debugging

*sh /home/cloudera/project/scripts/test2.sh*

The above code consists of spark project and the process to execute the project by sbt. So, ensure sbt is installed and running already.

Test.sh file sourcecode

***#!/bin/bash***

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

***echo "current Batchid is $batchid"  >> $LOGFILE***

***echo "Running Spark Application for data analysis..." >> $LOGFILE***

***cd /home/cloudera/workspace/SparkProject/***

***sbt package***

***spark-submit --class "SparkProject" --master local[2] target/scala-2.10/hello-sparkapp-4\_2.10-1.0.jar $batchid***

***batchid=`cat /home/cloudera/project/logs/current-batch.txt`***

***LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid***

Fetch the batchid from the current-batch.txt file

***echo "current Batchid is $batchid"  >> $LOGFILE***

***echo "Running Spark Application for data analysis..." >> $LOGFILE***

Provide the log message in log file

***cd /home/cloudera/workspace/SparkProject/***

Move to sparkproject directory

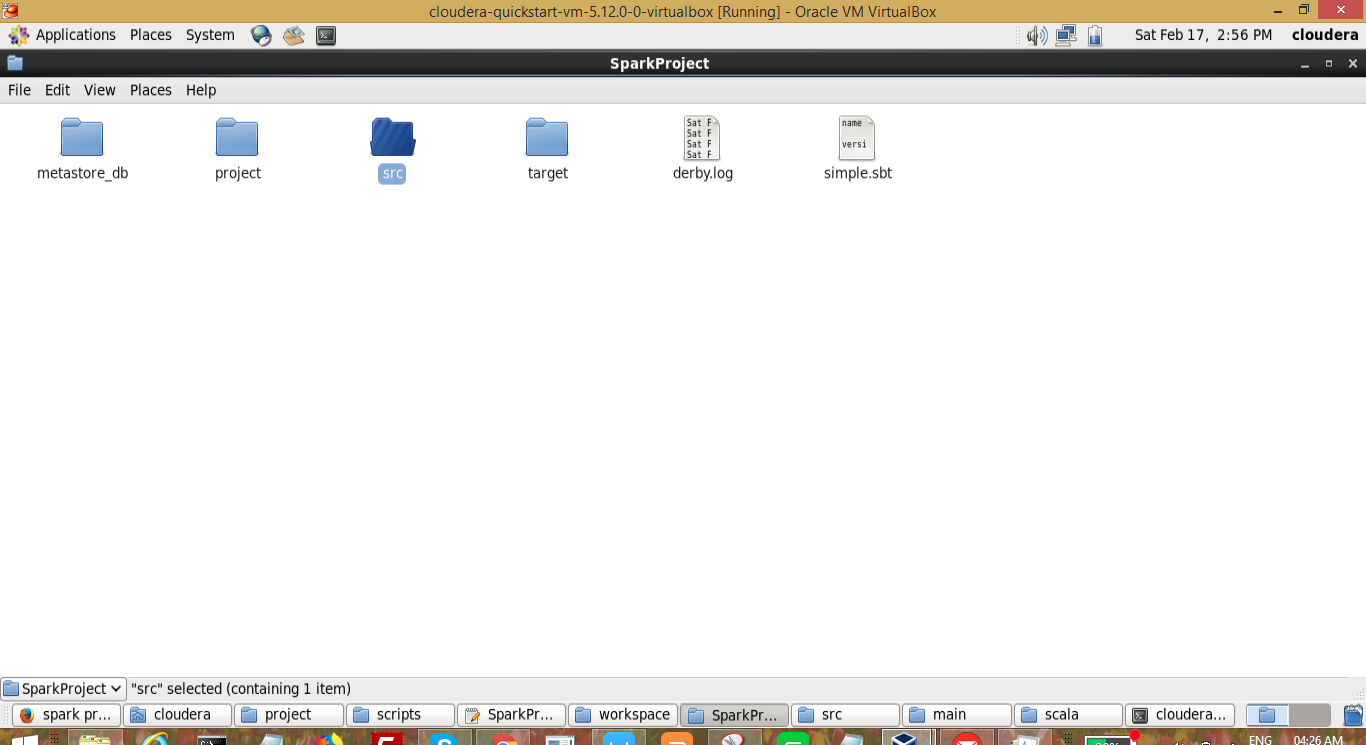
***sbt package***

It dumps all the packages to the project. The simple.sbt files consists of packages which are required for the project. Now we are connecting to hive from spark. So, we have added the hive package in the .sbt file

Once the sbt pacakge command is executed, it will create a target, project, metastore\_db folders.

Target folder consists of the jar file of the current package.

The below screenshot shows the folders after the sbt package is executed.



We can run the spark project in multiple ways. Right now we are running by spark submit by using the jar file. The below command is used to execute the spark file

*spark-submit --class "SparkProject" --master local[2] target/scala-2.10/hello-sparkapp-4\_2.10-1.0.jar $batchid*

It will execute the jar file exists in target folder. And pass the $batchid as argument.

Below is the source code of simple.sbt file

*name := "Hello SparkApp 4"*

*version := "1.0"*

*scalaVersion := "2.10.4"*

*libraryDependencies += "org.apache.spark" %% "spark-hive" % "1.5.2"*

Now we will go through the spark project. It is in the folder (/home/cloudera/workspace/SparkProject/src/main/scala)

The souce code of SparkProject.scala

*import org.apache.spark.SparkContext*

*import org.apache.spark.SparkContext.\_*

*import org.apache.spark.SparkConf*

*import org.apache.spark.sql.hive.HiveContext*

*import scala.io.Source*

*object SparkProject{*

*def main(args: Array[String]) {*

*var batch\_id = args(0);*

*println(batch\_id);*

*val conf = new SparkConf()*

*.setAppName("SparkProject")*

*.setMaster("local")*

*val sc = new SparkContext(conf)*

*val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)*

*hiveContext.setConf("hive.auto.convert.join", "false")*

*hiveContext.setConf("hive.support.sql11.reserved.keywords","false")*

*hiveContext.setConf("hive.exec.reducers.max","4")*

*hiveContext.setConf("mapreduce.job.reduces","4")*

*hiveContext.setConf("hive.metastore.warehouse.dir", "hdfs://quickstart.cloudera:8020/user/hive/warehouse")*

*hiveContext.sql("use project")*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS top\_10\_stations (station\_id STRING, total\_distinct\_songs\_played INT,distinct\_user\_count INT)PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE top\_10\_stations PARTITION(batchid="""+batch\_id+""") SELECT station\_id, COUNT(DISTINCT song\_id) AS total\_distinct\_songs\_played, COUNT(DISTINCT user\_id) AS distinct\_user\_count FROM enriched\_data WHERE status='pass' AND batchid="""+batch\_id+""" AND like=1 GROUP BY station\_id ORDER BY total\_distinct\_songs\_played DESC LIMIT 10""");*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS users\_behaviour (user\_type STRING,duration INT) PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED*

*BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE users\_behaviour PARTITION(batchid="""+batch\_id+""") SELECT CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestamp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED' WHEN (su.user\_id IS NOT NULL AND CAST(ed.timestamp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END AS user\_type,SUM(ABS(CAST(ed.end\_ts AS DECIMAL(20,0))-CAST(ed.start\_ts AS DECIMAL(20,0)))) AS duration FROM enriched\_data ed*

*LEFT OUTER JOIN subscribed\_users su ON ed.user\_id=su.user\_id WHERE ed.status='pass' AND ed.batchid="""+batch\_id+""" GROUP BY CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestamp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED' WHEN (su.user\_id IS NOT NULL AND CAST(ed.timestamp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END""");*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS connected\_artists (artist\_id STRING,user\_count INT) PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE connected\_artists PARTITION(batchid="""+batch\_id+""") SELECT ua.artist\_id, COUNT(DISTINCT ua.user\_id) AS user\_count FROM*

*( SELECT user\_id, artist\_id FROM users\_artists LATERAL VIEW explode(artists\_array) artists AS artist\_id ) ua INNER JOIN (SELECT artist\_id, song\_id, user\_id FROM enriched\_data WHERE status='pass' AND batchid="""+batch\_id+""") ed ON ua.artist\_id=ed.artist\_id AND ua.user\_id=ed.user\_id GROUP BY ua.artist\_id ORDER BY user\_count DESC LIMIT 10""");*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS top\_10\_royalty\_songs (song\_id STRING,duration INT) PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE top\_10\_royalty\_songs PARTITION(batchid="""+batch\_id+""") SELECT song\_id, SUM(ABS(CAST(end\_ts AS DECIMAL(20,0))-CAST(start\_ts AS DECIMAL(20,0)))) AS duration FROM enriched\_data WHERE status='pass' AND batchid="""+batch\_id+""" AND (like=1 OR song\_end\_type=0) GROUP BY song\_id ORDER BY duration DESC LIMIT 10""");*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS top\_10\_unsubscribed\_users (user\_id STRING,duration INT)PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE top\_10\_unsubscribed\_users PARTITION(batchid="""+batch\_id+""") SELECT ed.user\_id,SUM(ABS(CAST(ed.end\_ts AS DECIMAL(20,0))-CAST(ed.start\_ts AS DECIMAL(20,0)))) AS duration FROM enriched\_data ed LEFT OUTER JOIN subscribed\_users su ON ed.user\_id=su.user\_id WHERE ed.status='pass' AND ed.batchid="""+batch\_id+""" AND (su.user\_id IS NULL OR (CAST(ed.timestamp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0)))) GROUP BY ed.user\_id*

*ORDER BY duration DESC LIMIT 10""");*

*}*

*}*

Now we will dive into the code by step by step

*import org.apache.spark.SparkContext*

*import org.apache.spark.SparkContext.\_*

*import org.apache.spark.SparkConf*

*import org.apache.spark.sql.hive.HiveContext*

*import scala.io.Source*

Specifying the all the required packages.

*var batch\_id = args(0);*

*println(batch\_id);*

Read the batchid which is passed as argument

*val conf = new SparkConf()*

*.setAppName("SparkProject")*

*.setMaster("local")*

*val sc = new SparkContext(conf)*

*val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)*

Initializing the sc and hiveContext objects

*hiveContext.setConf("hive.auto.convert.join", "false")*

*hiveContext.setConf("hive.support.sql11.reserved.keywords","false")*

*hiveContext.setConf("hive.exec.reducers.max","4")*

*hiveContext.setConf("mapreduce.job.reduces","4")*

*hiveContext.setConf("hive.metastore.warehouse.dir", "hdfs://quickstart.cloudera:8020/user/hive/warehouse")*

Defining the configurations of the project

*hiveContext.sql("use project")*

Selecting the hive datbase project

1. Determine top 10 station\_id(s) where maximum number of songs were played, which were liked by unique users.

*Explanation: We need to get the top 10 stations. So, group by done on station id. Fetched the unique songs played for unique users for a station. And like=1 to get the liked records.*

*Status=pass which satisfies all the conditions we added the data enrichment process*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS top\_10\_stations (station\_id STRING, total\_distinct\_songs\_played INT,distinct\_user\_count INT)PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE top\_10\_stations PARTITION(batchid="""+batch\_id+""") SELECT station\_id, COUNT(DISTINCT song\_id) AS total\_distinct\_songs\_played, COUNT(DISTINCT user\_id) AS distinct\_user\_count FROM enriched\_data WHERE status='pass' AND batchid="""+batch\_id+""" AND like=1 GROUP BY station\_id ORDER BY total\_distinct\_songs\_played DESC LIMIT 10""");*

1. Determine total duration of songs played by each type of user, where type of user can be 'subscribed' or 'unsubscribed'. An unsubscribed user is the one whose record is either not present in Subscribed\_users lookup table or has subscription\_end\_date earlier than the timestamp of the song played by him.

*Explanation: Here first and foremost we are identifying the user is subscribed or not . For subscribed users, the subscription\_end\_date is future date and user\_id is available on subscribed\_users lookup table. So, if the subscription\_end\_date is old date it will be unsubscribed user. After that we will calculate the total duration of time which the user listed. It is done by sum of all the songs by endtime - starttim. And we are grouping the data for subscribed and un subscribed users.*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS users\_behaviour (user\_type STRING,duration INT) PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED*

*BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE users\_behaviour PARTITION(batchid="""+batch\_id+""") SELECT CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestamp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED' WHEN (su.user\_id IS NOT NULL AND CAST(ed.timestamp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END AS user\_type,SUM(ABS(CAST(ed.end\_ts AS DECIMAL(20,0))-CAST(ed.start\_ts AS DECIMAL(20,0)))) AS duration FROM enriched\_data ed*

*LEFT OUTER JOIN subscribed\_users su ON ed.user\_id=su.user\_id WHERE ed.status='pass' AND ed.batchid="""+batch\_id+""" GROUP BY CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestamp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED' WHEN (su.user\_id IS NOT NULL AND CAST(ed.timestamp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END""");*

1. Determine top 10 connected artists. Connected artists are those whose songs are most listened by the unique users who follow them.

*Explanation: Connected artists means the users is following the artists. It means the user and artist combination will exist on users\_artists table. Next we are idenfying the distinct users associated to artists who are following and group by by artist. To identify the top artists, we will order by artists*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS connected\_artists (artist\_id STRING,user\_count INT) PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE connected\_artists PARTITION(batchid="""+batch\_id+""") SELECT ua.artist\_id, COUNT(DISTINCT ua.user\_id) AS user\_count FROM*

*( SELECT user\_id, artist\_id FROM users\_artists LATERAL VIEW explode(artists\_array) artists AS artist\_id ) ua INNER JOIN (SELECT artist\_id, song\_id, user\_id FROM enriched\_data WHERE status='pass' AND batchid="""+batch\_id+""") ed ON ua.artist\_id=ed.artist\_id AND ua.user\_id=ed.user\_id GROUP BY ua.artist\_id ORDER BY user\_count DESC LIMIT 10""");*

1. Determine top 10 songs who have generated the maximum revenue. Royalty applies to a song only if it was liked or was completed successfully or both.

*Explanation: Here we are identifying the songs which are producing heighest revenue. Royalty will be considered for the songs which are liked by users (like=1) or the songs which are completely listed by users (song\_end\_type=0)*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS top\_10\_royalty\_songs (song\_id STRING,duration INT) PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE top\_10\_royalty\_songs PARTITION(batchid="""+batch\_id+""") SELECT song\_id, SUM(ABS(CAST(end\_ts AS DECIMAL(20,0))-CAST(start\_ts AS DECIMAL(20,0)))) AS duration FROM enriched\_data WHERE status='pass' AND batchid="""+batch\_id+""" AND (like=1 OR song\_end\_type=0) GROUP BY song\_id ORDER BY duration DESC LIMIT 10""");*

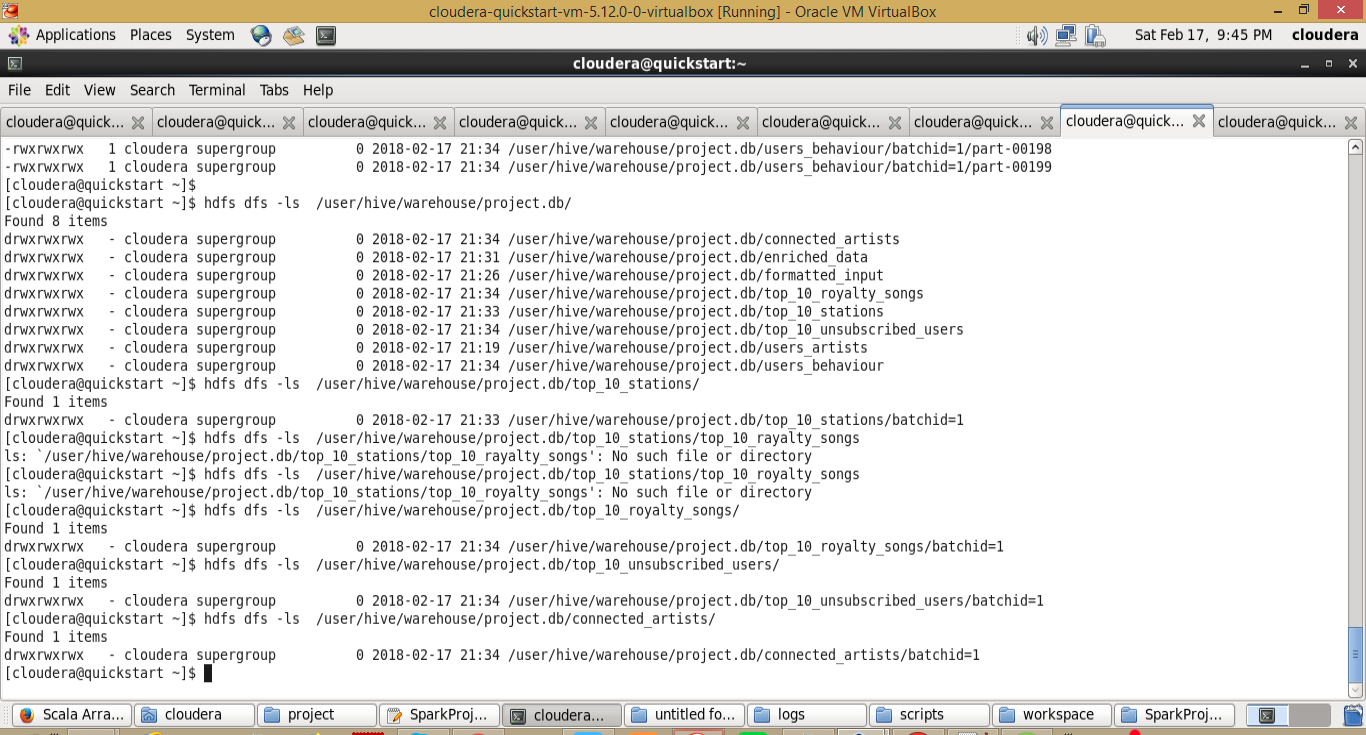
1. Determine top 10 unsubscribed users who listened to the songs for the longest duration.

*Explanation: Here we are going to identify the unsubscribed users who have listed the maximum duration of songs. Here users subscription will be identified which is provided earlier ( For subscribed users, the subscription\_end\_date is future date and user\_id is available on subscribed\_users lookup table. ). And do the order by users to identify the top users count*

*hiveContext.sql("""CREATE TABLE IF NOT EXISTS top\_10\_unsubscribed\_users (user\_id STRING,duration INT)PARTITIONED BY (batchid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE""");*

*hiveContext.sql("""INSERT OVERWRITE TABLE top\_10\_unsubscribed\_users PARTITION(batchid="""+batch\_id+""") SELECT ed.user\_id,SUM(ABS(CAST(ed.end\_ts AS DECIMAL(20,0))-CAST(ed.start\_ts AS DECIMAL(20,0)))) AS duration FROM enriched\_data ed LEFT OUTER JOIN subscribed\_users su ON ed.user\_id=su.user\_id WHERE ed.status='pass' AND ed.batchid="""+batch\_id+""" AND (su.user\_id IS NULL OR (CAST(ed.timestamp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0)))) GROUP BY ed.user\_id*

*ORDER BY duration DESC LIMIT 10""");*



Now second part of data analysis is to move the hdfs data created to mysql tables which is handled by the below command of data\_analysis.sh file

*sh /home/acadgild/project/scripts/data\_export.sh*

*#!/bin/bash*

*batchid=`cat /home/cloudera/project/logs/current-batch.txt`*

*LOGFILE=/home/cloudera/project/logs/log\_batch\_$batchid*

*echo "Creating mysql tables if not present..." >> $LOGFILE*

*echo "current Batchid is $batchid"  >> $LOGFILE*

*mysql -uroot -pcloudera < /home/cloudera/project/scripts/create\_schema.sql*

*echo "Running sqoop job for data export..." >> $LOGFILE*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table top\_10\_stations --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/top\_10\_stations/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table users\_behaviour --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/users\_behaviour/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table connected\_artists --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/connected\_artists/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table top\_10\_royalty\_songs --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/top\_10\_royalty\_songs/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table top\_10\_unsubscribed\_users --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/top\_10\_unsubscribed\_users/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

*mysql -uroot -pcloudera < /home/cloudera/project/scripts/create\_schema.sql*

*echo "Running sqoop job for data export..." >> $LOGFILE*

Creates mysql database and tables if not exists

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table top\_10\_stations --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/top\_10\_stations/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table users\_behaviour --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/users\_behaviour/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

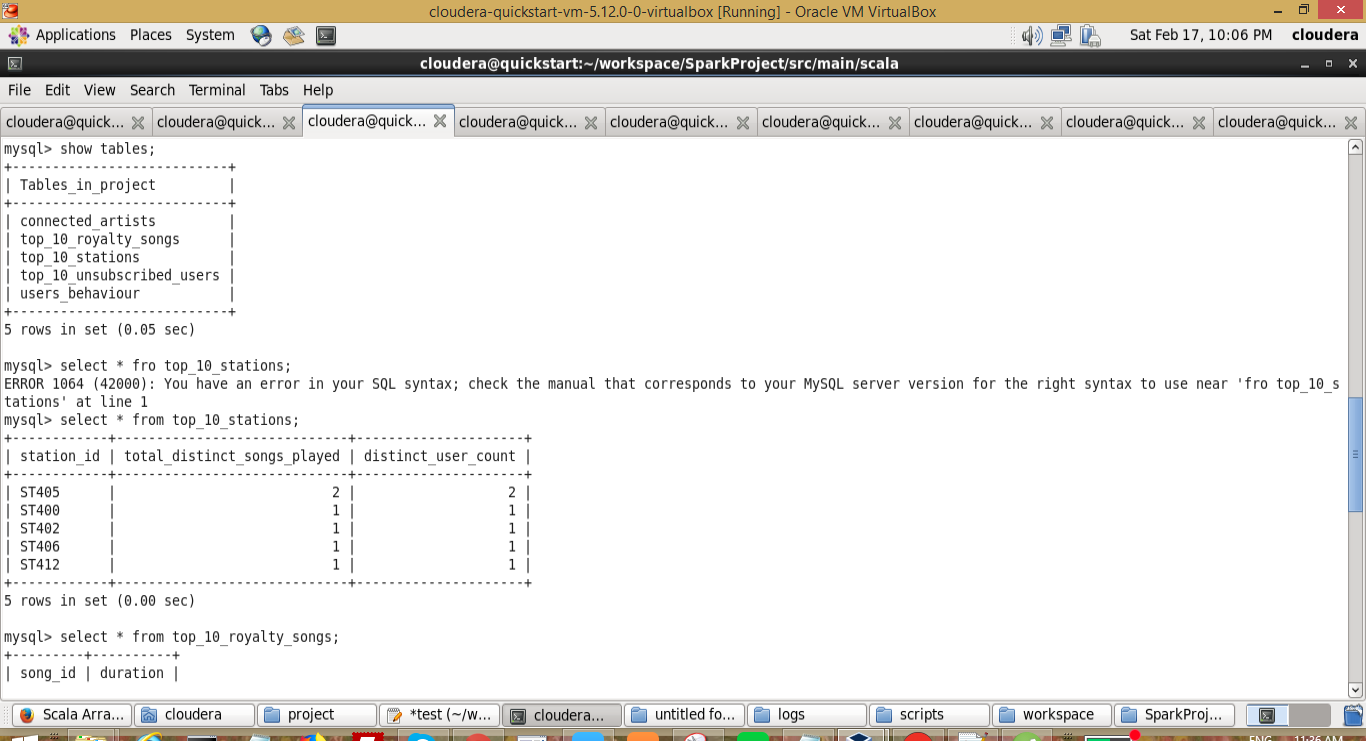
*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table connected\_artists --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/connected\_artists/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

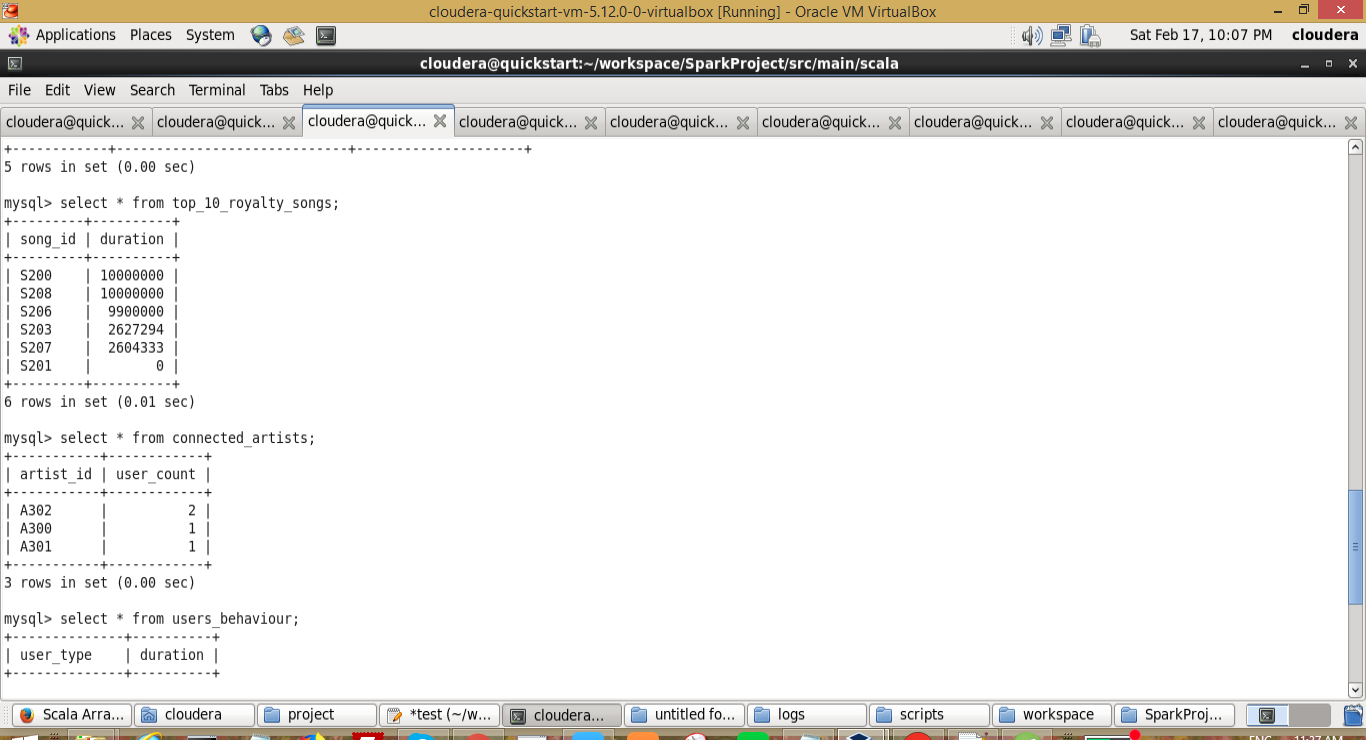
*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table top\_10\_royalty\_songs --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/top\_10\_royalty\_songs/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

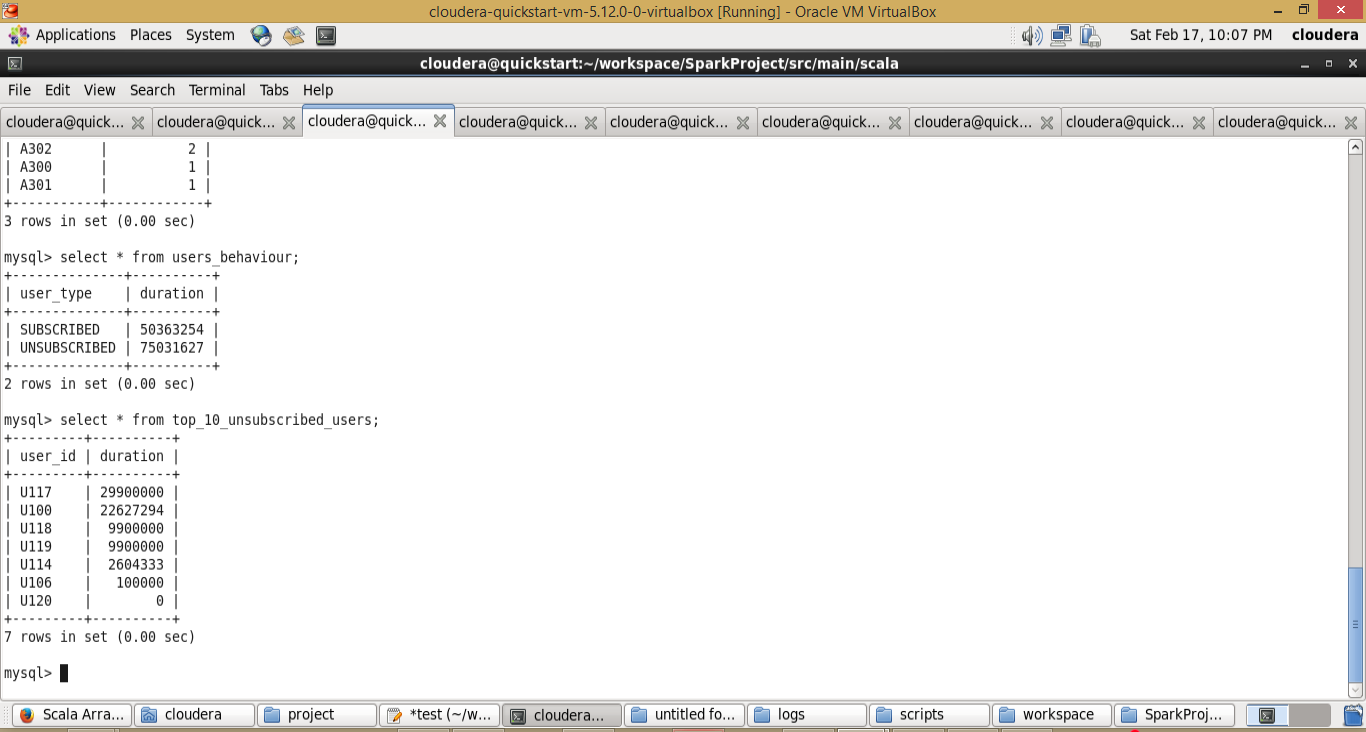
*sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table top\_10\_unsubscribed\_users --export-dir hdfs://quickstart.cloudera:8020/user/hive/warehouse/project.db/top\_10\_unsubscribed\_users/batchid=$batchid --input-fields-terminated-by ',' -m 1;*

By using sqoop export we will copy the data from hive tables to mysql tables. Once the above commends are executed, we can find the data in mysql tables.

Final records in mysql database







Wrapper.sh file soucecode

*#!/bin/bash*

*python /home/cloudera/project/scripts/generate\_web\_data.py*

*python /home/cloudera/project/scripts/generate\_mob\_data.py*

*sh /home/cloudera/project/scripts/start-daemons.sh*

*sh /home/cloudera/project/scripts/populate-lookup.sh*

*sh /home/cloudera/project/scripts/dataformatting.sh*

*sh /home/cloudera/project/scripts/data\_enrichment.sh*

*sh /home/cloudera/project/scripts/data\_analysis.sh*

Wrapper.sh contains all the files to be executed in one go. Right now we have executed them one by one. But on real time, its impossible. So, we will schedule the job by using cronjobs. Current wrapper.sh is scheduled for 3 hours

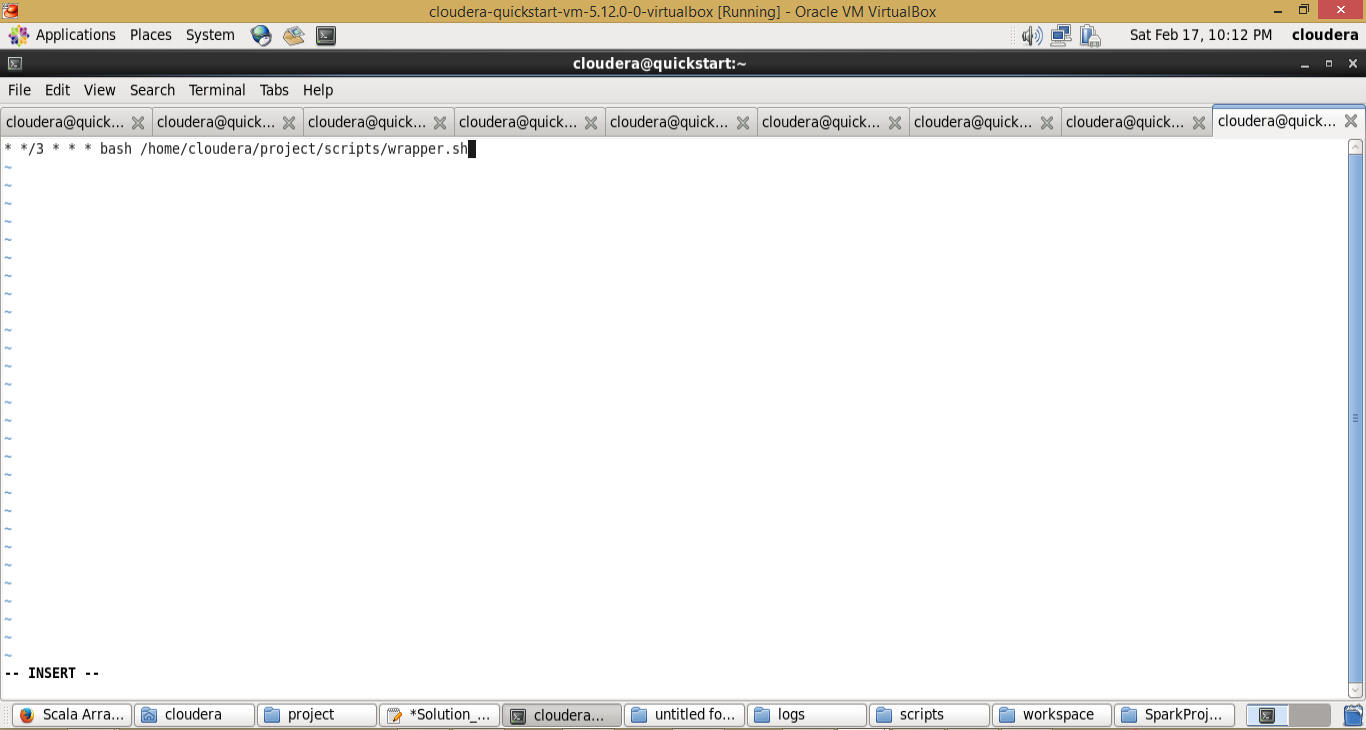
sudo crontab -e

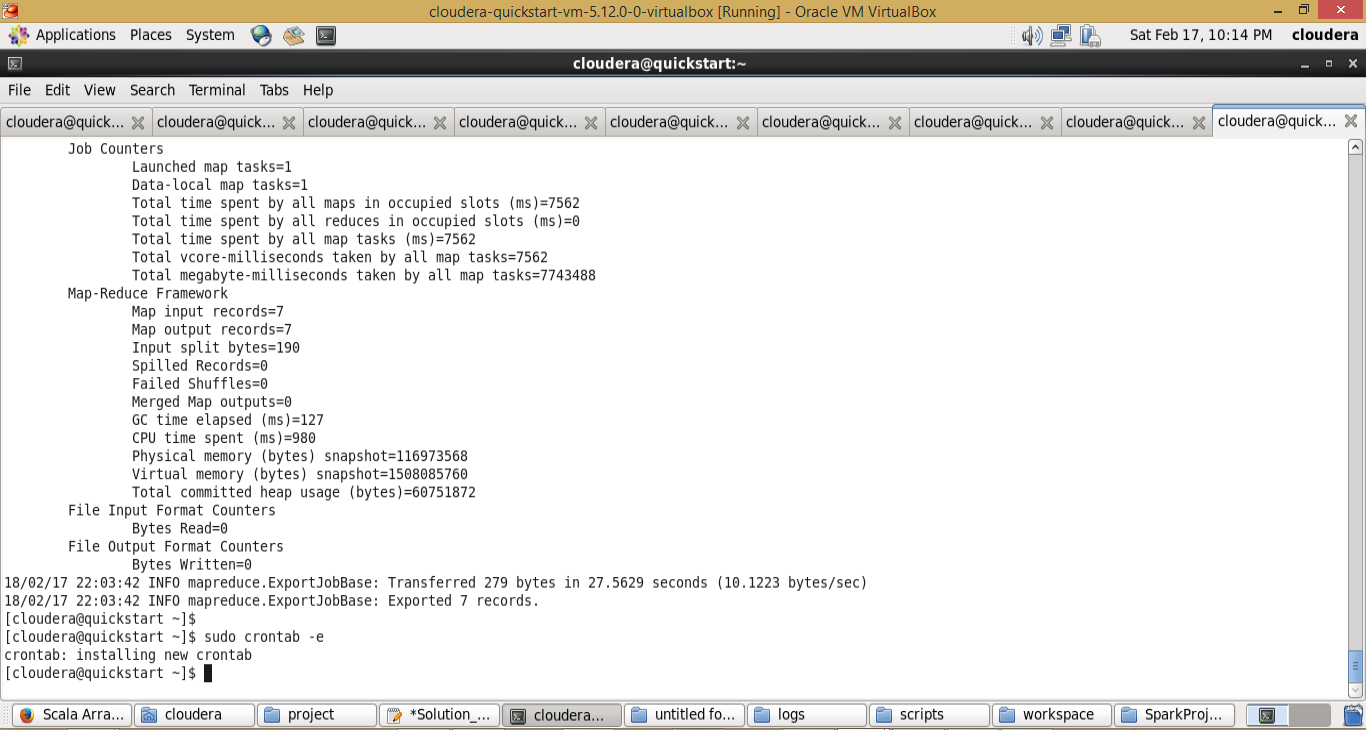
(Press i to enter insert mode)

***\* \*/3 \* \* \* bash /home/cloudera/project/scripts/wrapper.sh***

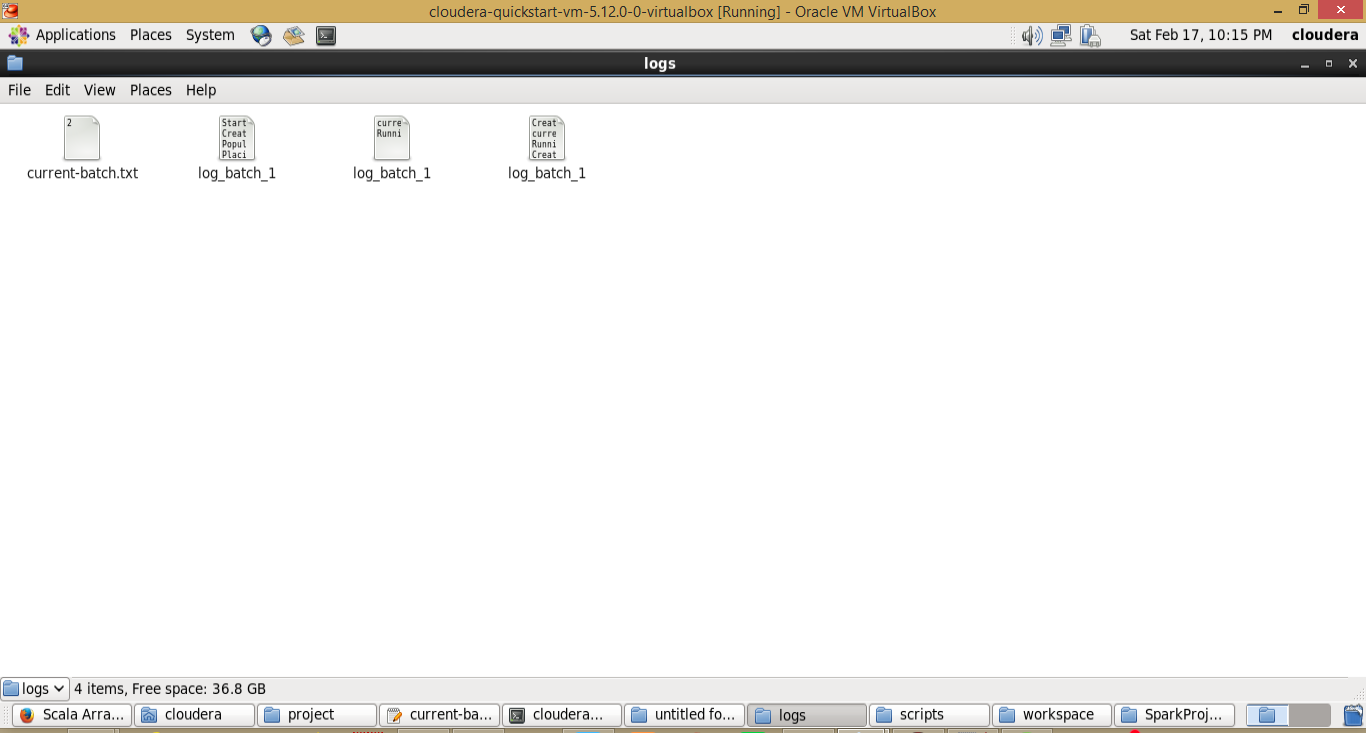
(Press Esc and then type :wq! and press Enter)

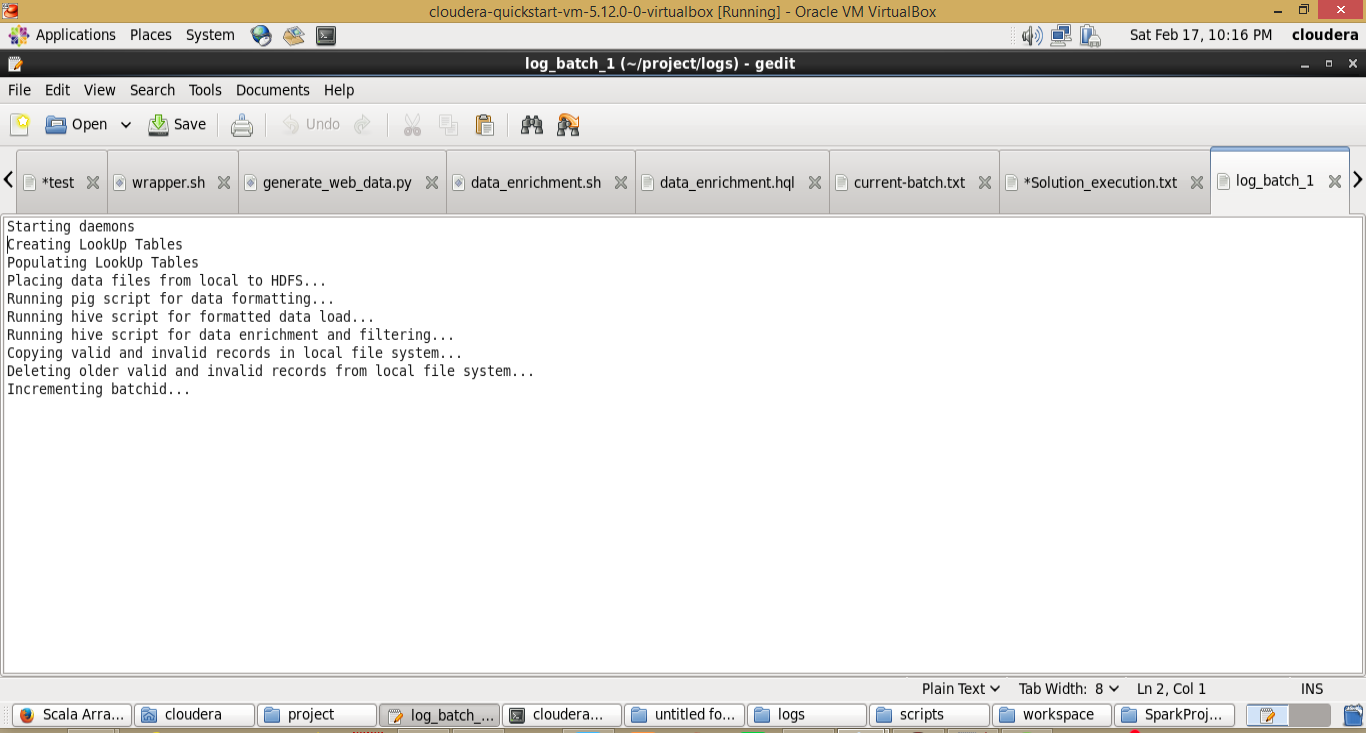
Setup the cron for 3 hours to repeat all the process

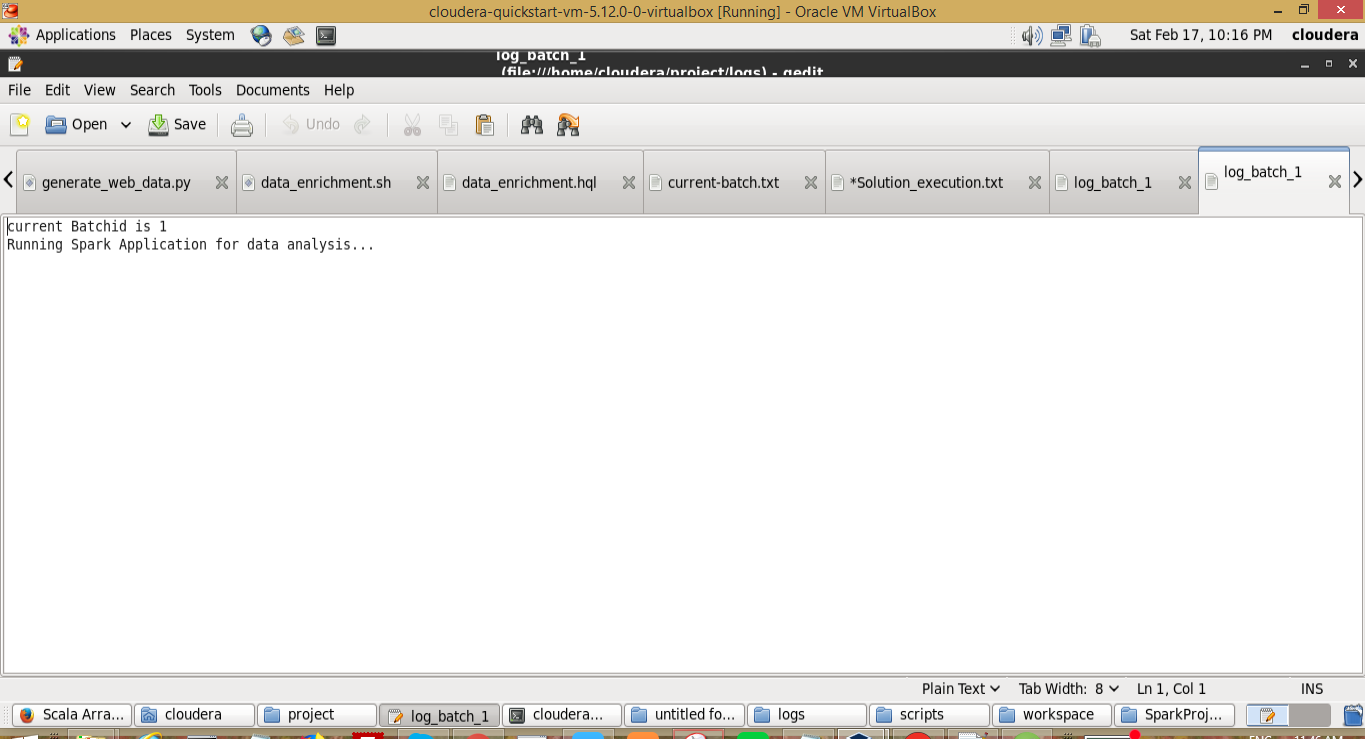


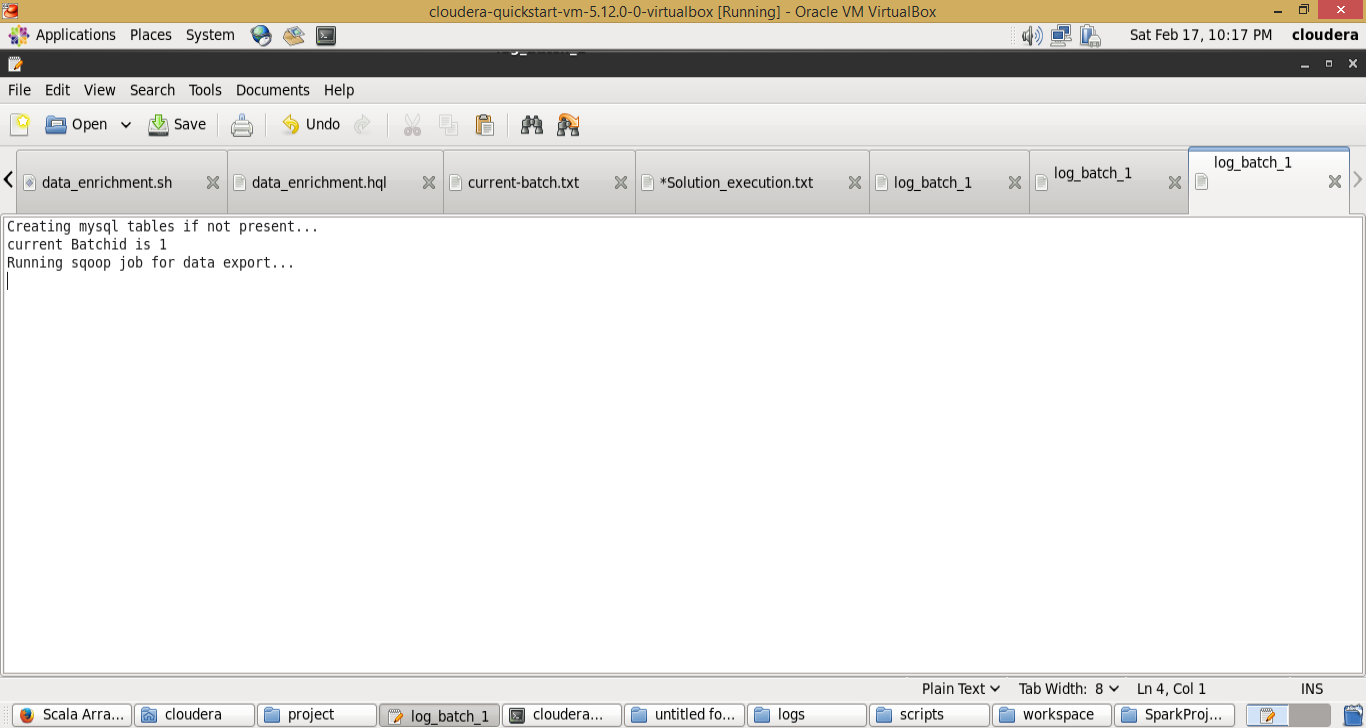


Batch files created while executing the cron job









In this way the complete process will be repeated for every 3 hours and data will be dump to mysql tables.