ACADGILD  
 BIG DATA HADOOP & SPARK . TRAINING

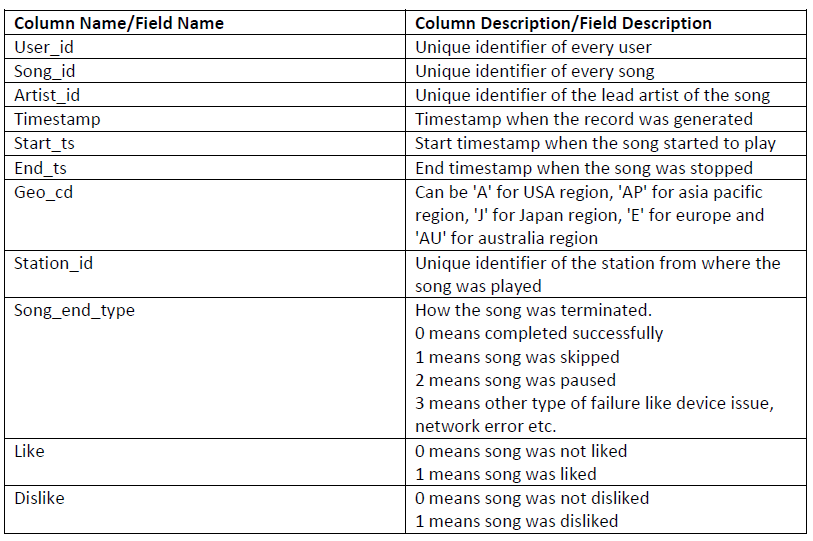
**Final PROJECT**

Music Data Analysis

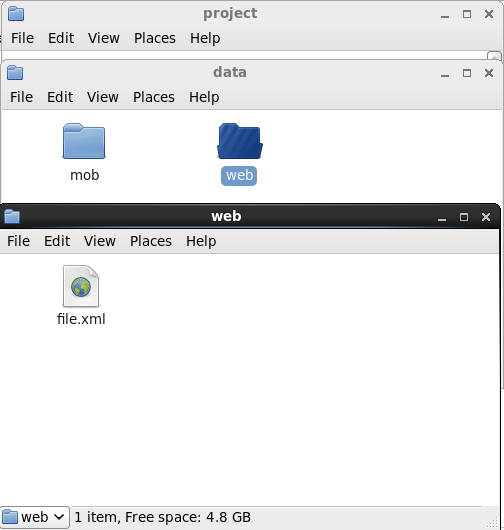
**From :**

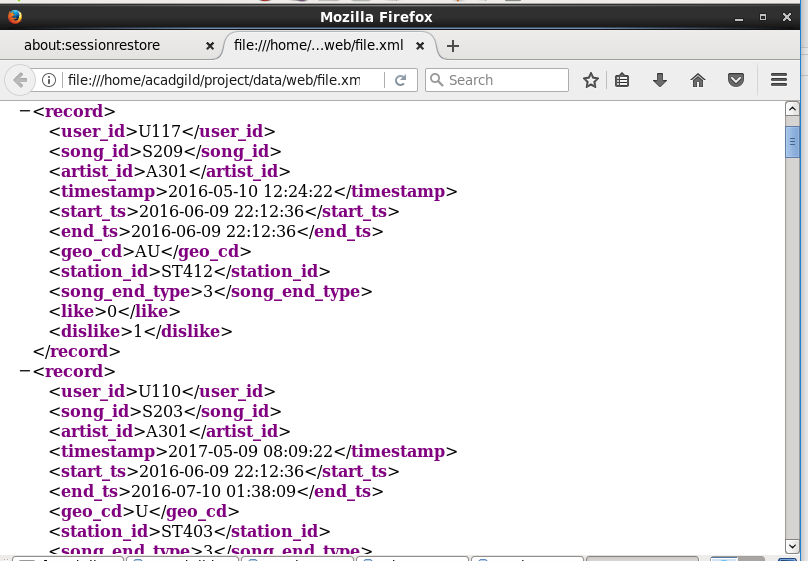
**Monisha N**

**Data Description**

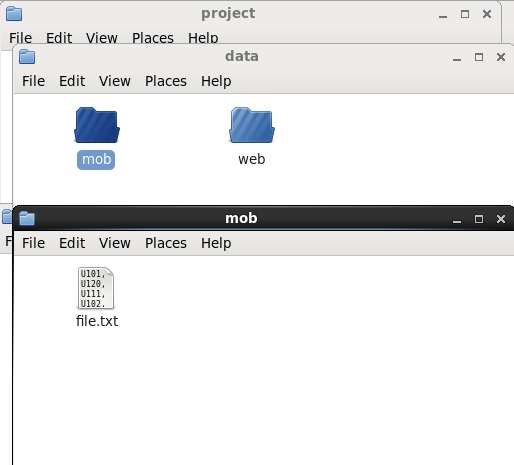


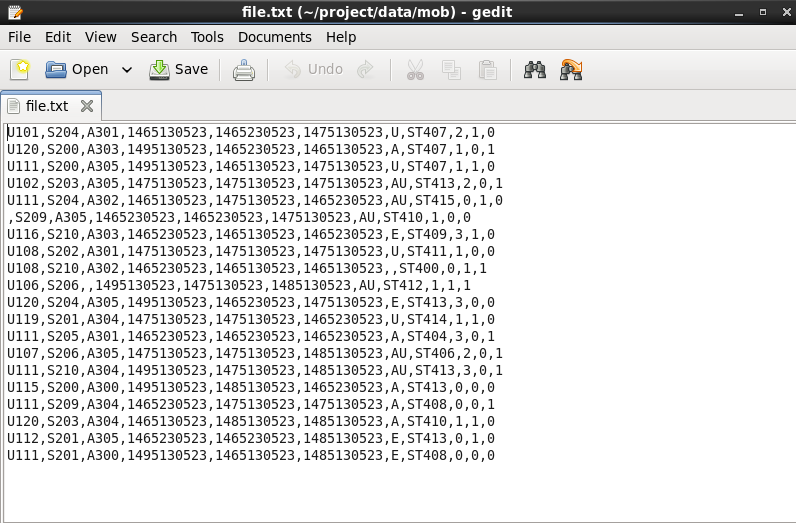
**Data Files:**Below is the data coming from web applications, that reside in /data/web and has **xml** format.





**Below is a sample of the data coming from mobile applications, that reside in /data/mob and has csv format**

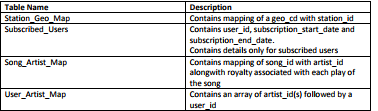
****

****

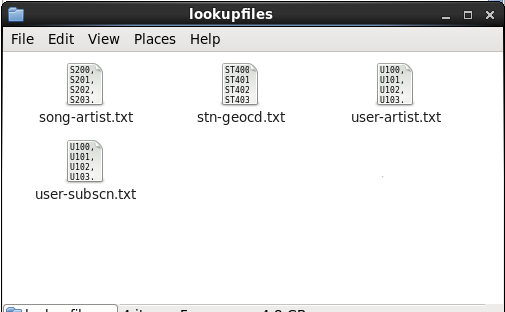
**Look-Up Tables Files:**

There are some existing lookup tables present in NoSQL Databases that play an important role in data enrichment and analysis.

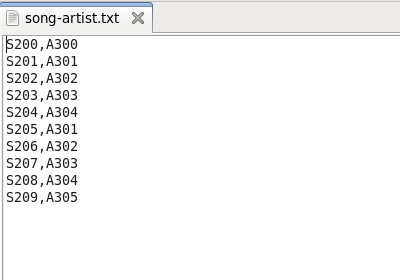
This data is present in lookup directory and loaded in HBase.



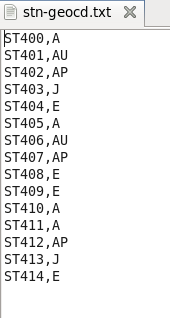
Below are the data to be present in the lookup tables. There are 4 tables:

****

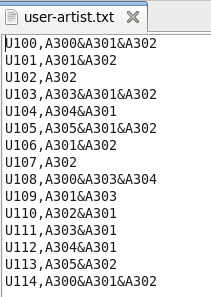
**song-artist**Columns: **song\_id**, **artist\_id**

****

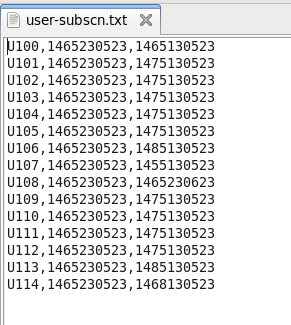
**stn-geocd**Columns: **station\_id, geo\_cd**

****

**user-artist**Columns: **user\_id, artists\_array**

****

**user-subscn**Columns: **user\_id**, **subscn\_start\_dt, subscn\_end\_dt**

****

**Steps to perform data analysis on the Music Data:**

**Step 1:** Launch all necessary daemons

**Step 2:** Start Job Scheduling (using Crontab)

**Step 3:** Populate Look-Up tables (i.e. Load all data to HBase)

**Step 4:** Perform Data Formatting (using Pig and Hive)

**Step 5:** Perform Data Enrichment and Cleaning (using Hive)

**Step 6:** Perform Data Analysis (using Spark)

**Step 1:**

Launch all necessary daemons

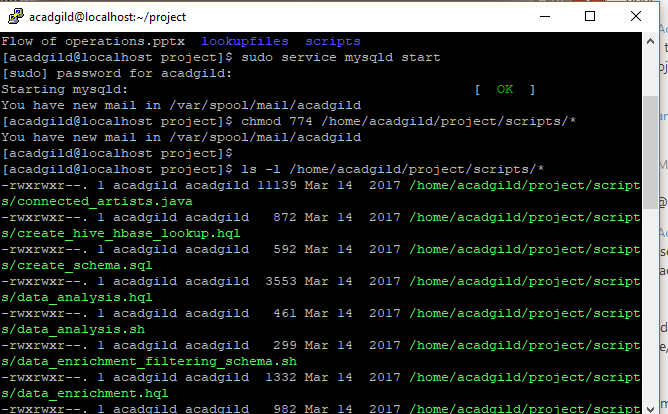
* Launch the Mysql Service (needed for Hive)

sudo service mysqld start

* Give permissions to scripts folder in project, so we are able to run scripts from the bash shell.

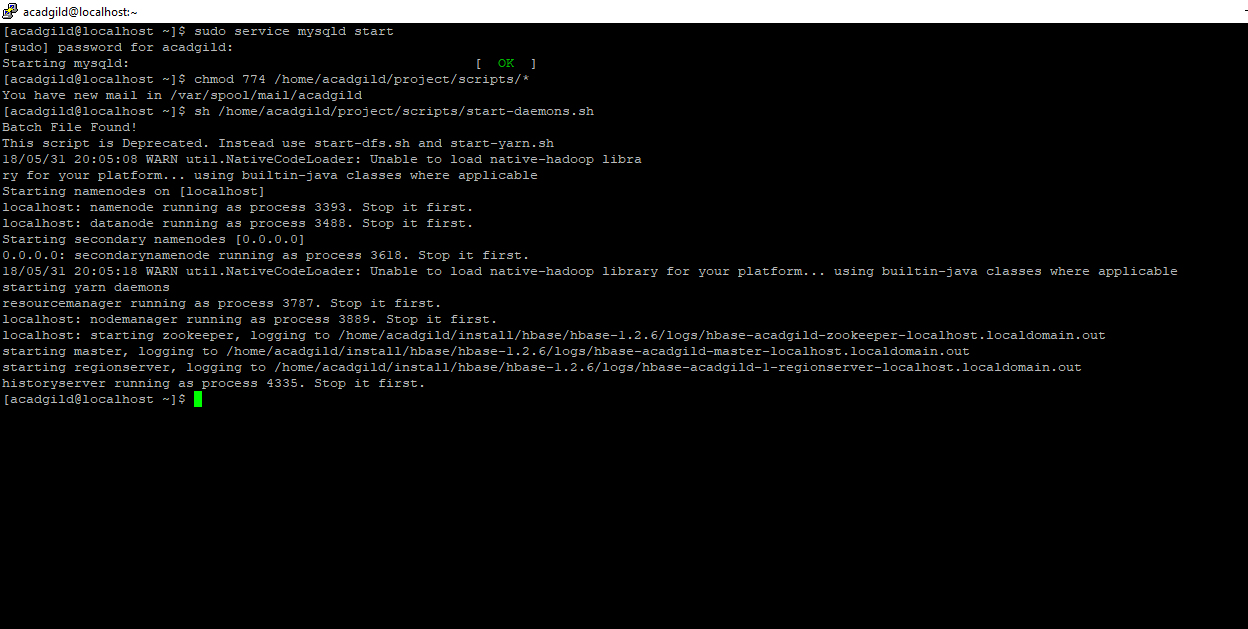
chmod 774 /home/acadgild/project/scripts/\*

ls -l /home/acadgild/project/scripts/\*

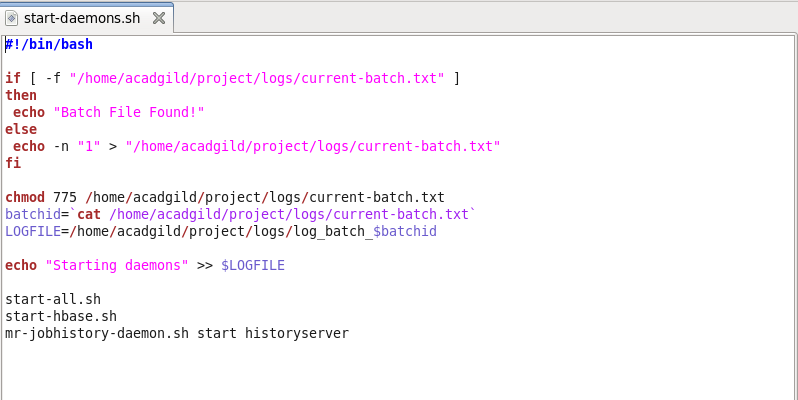


* Run the shell script start-daemons.sh

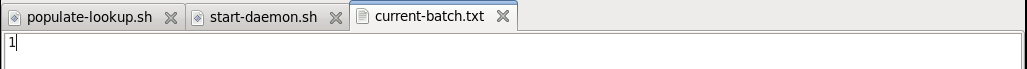
sh /home/acadgild/project/scripts/start-daemons.sh



**In the shell script start-daemons.sh used above, we perform the following operations:**



* Check if a file **current-batch.txt** has been created or not,  
  If already created, print **Batch File Found!** else create the file and add 1 to it to signify batch 1.



* Give permissions to the file, so that we are able to modify it on the run.
* Get the batch id number from the batch file created above and create a Log File for the batch using the batch id. This will be log\_batch\_1.

Through out the course of the analysis process this log file will document the tasks that are performed for the Music Data Analysis.

* Add a log to the Log File signifying that the all necessary daemons have been started



* Start the dfs, yarn, hbase and jobhistory daemons.

**Step 2:**Start Job Scheduling

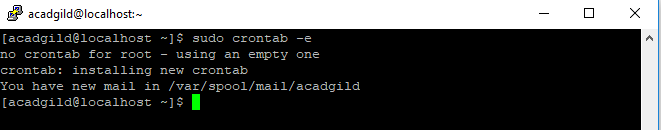
* Open the crontab file and insert the statement:

\* \*/3 \* \* \* /home/acadgild/project/scripts/wrapper.sh

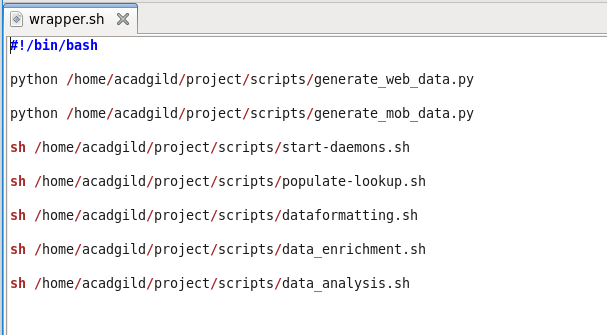
Crontab is used for Job Scheduling. In the -e mode, Crontab schedules execution of commands by a regular user.  
The statement above runs the wrapper.sh shell script every 3 hours.







In the shell script wrapper.sh used above, all the processes needed to perform analysis on the Music Data is called once every 3 hours thereby creating a new batch. This is the job scheduling



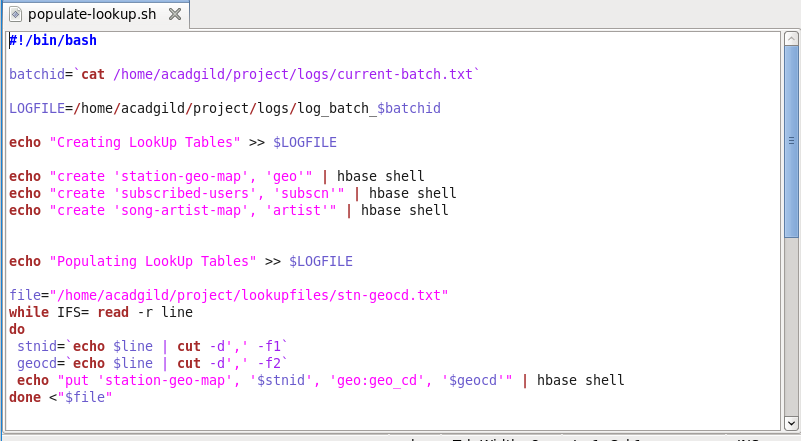
**Step 3:**

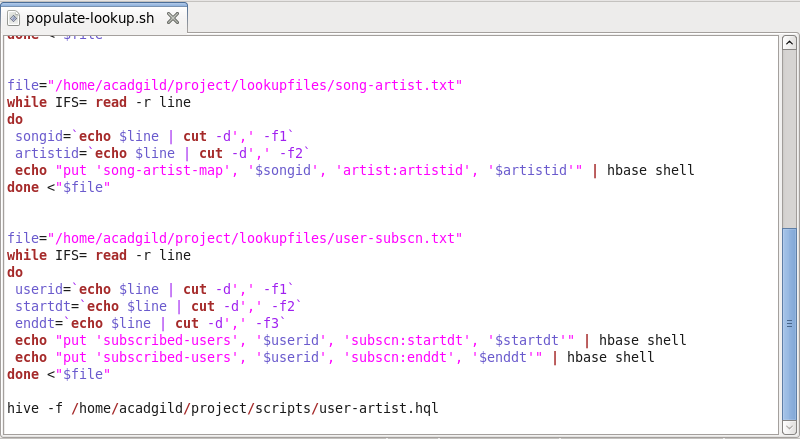
Populate Look-Up tables

Below is the shell script populate-lookup.sh that is used to load the data for the lookup tables into HBase tables.

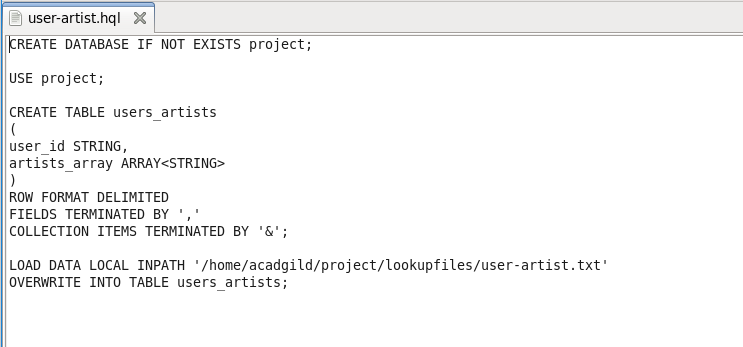
The following operations are performed:

* Get the batch id number from the batch file and get the Log File for the batch using the batch id. This will be log\_batch\_1
* Add logs to the Log File signifying that the lookup tables are being created and populated
* Create the HBase tables for the lookup data files: song-artist, stn-geocd and user-subscn with their column families
* For every lookup data file, read each line, extract the columns (comma separated) and add he data as rows to the corresponding HBase tables created above
* Run the hive script user-artist.hql. This will populate a hive table with the data in the lookup data file user-artist. This is because this file has an array column that is difficult to populate in HBase.

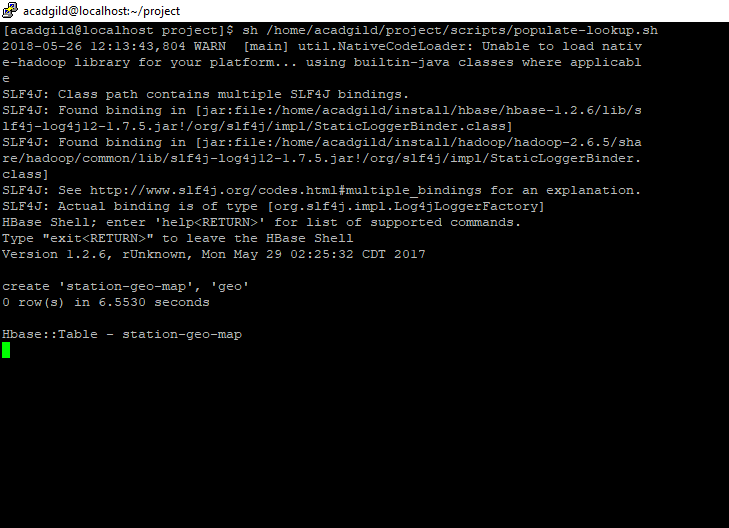


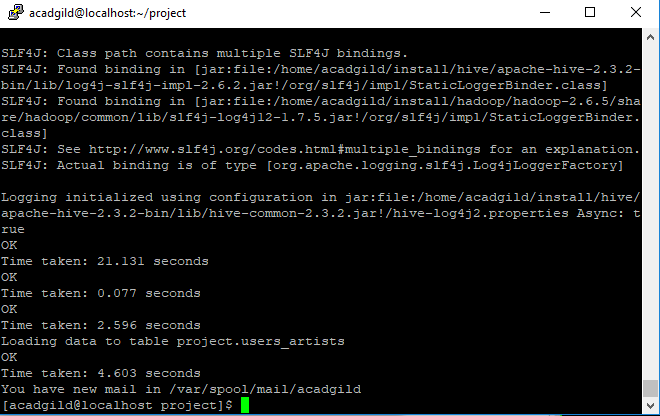
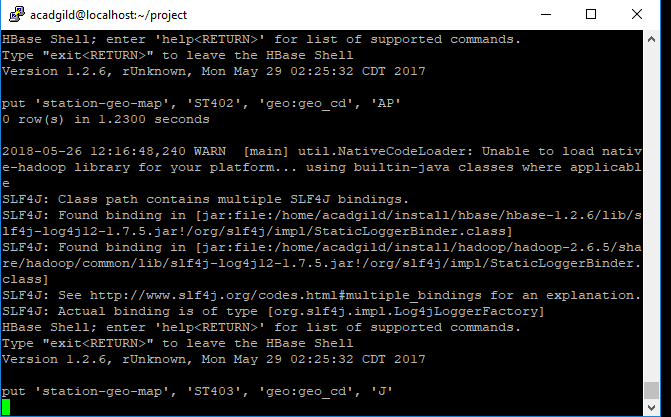


After the data in user-artist is loaded in the Hive Table users-artists, it is then saved as a text file as below (for data analysis using spark)

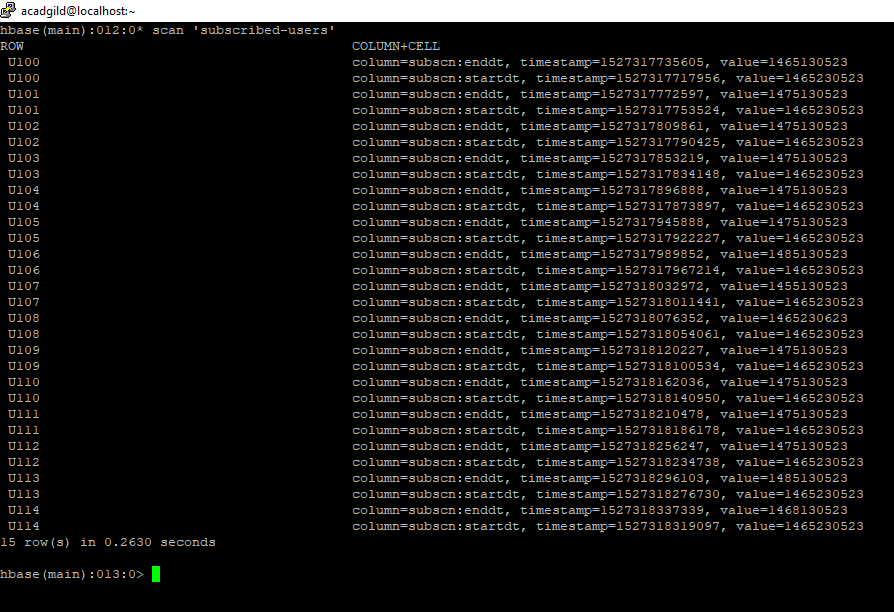
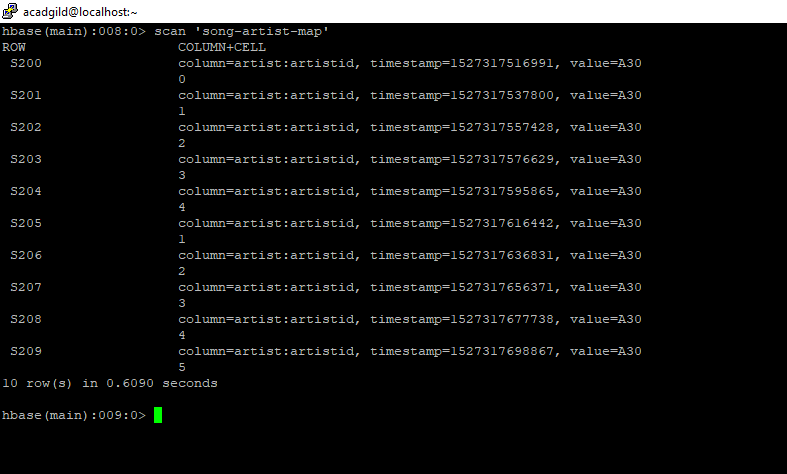
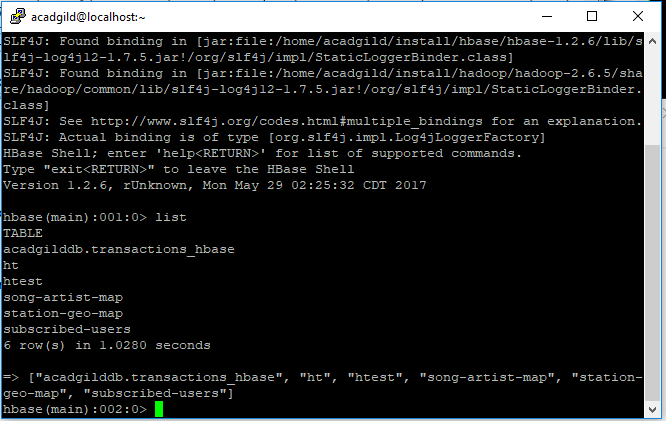
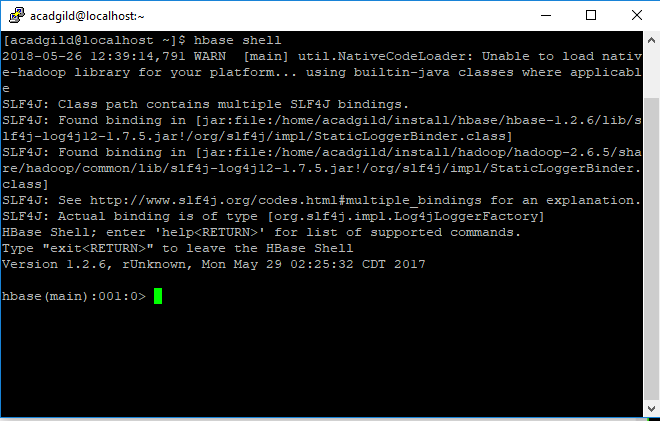


Below is a view of the execution of the above:

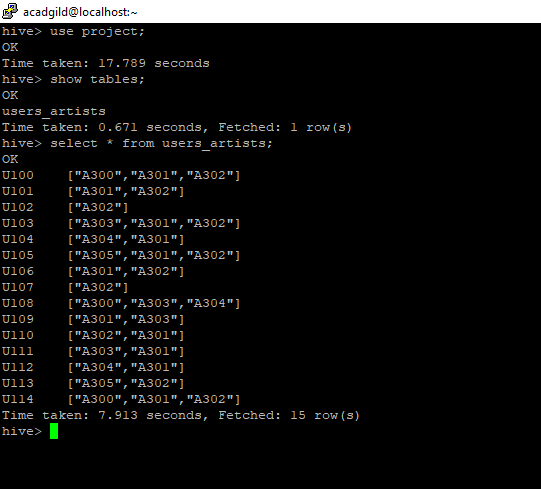
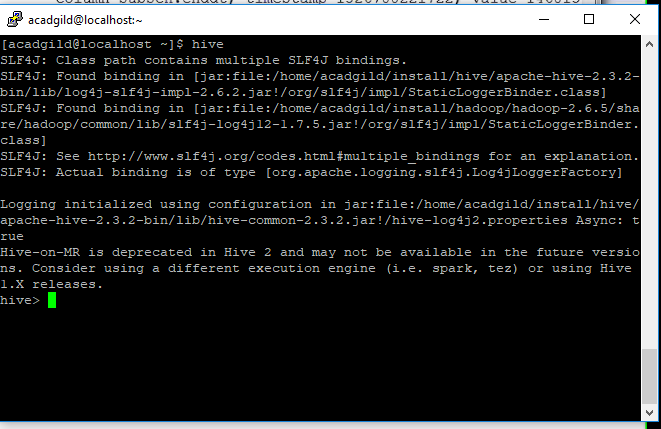




Output of the above (HBase):



Output of the above (Hive)



**Step 4:**

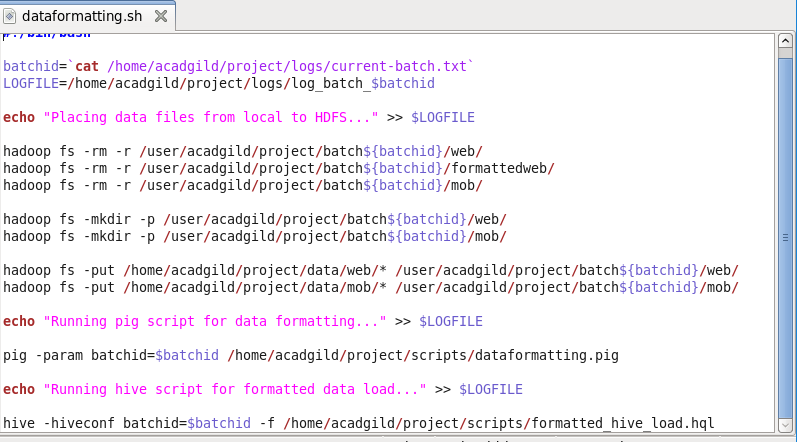
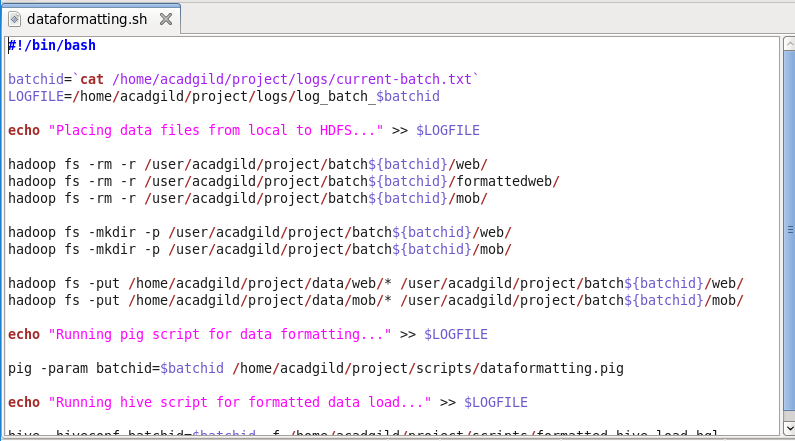
Perform Data Formatting

Below is the shell script dataformatting.sh that is used to:

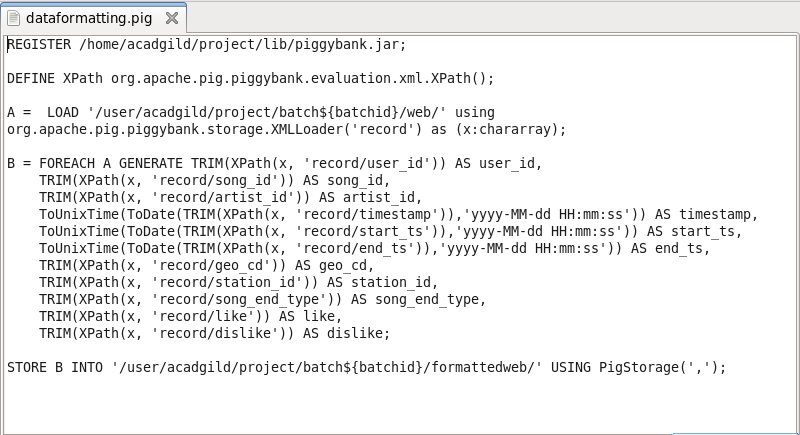
* Format the web xml data using Pig to a csv fomat and
* Load the 2 data files, mob and web (formatted by Pig), to a Hive Table for data enrichment

The following operations are performed:

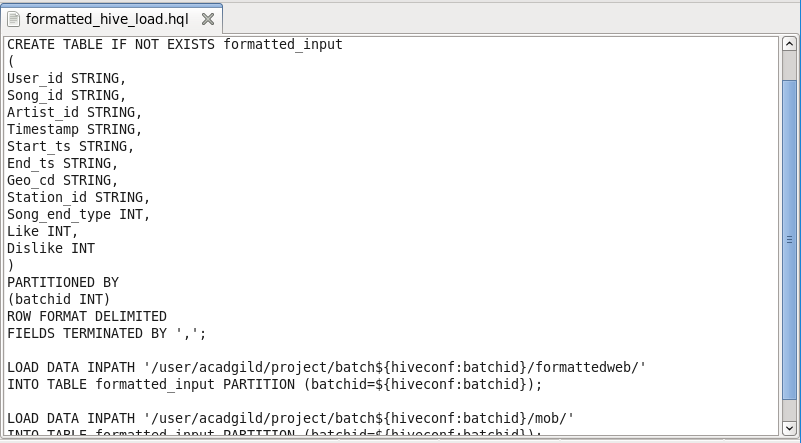
* Get the batch id number from the batch file and get the Log File for the batch using the batch id. This will be log\_batch\_1
* Add logs to the Log File signifying that the data is placed in the HDFS and the running of the Pig and Hive scripts for data formatting and loading respectively.
* Delete, if they exist, folders for the mob, web and formattedweb. This is done in-case any old data remains because of execution failure.
* Create the above folders web and mob that were deleted above and move the data from the Local FS to the HDFS. The formattedweb folder is created in the Pig Script.
* Run the pig script dataformatting.pig. This will format the web data (stored in the web folder in the HDFS) in xml format to csv format and store it in the HDFS in the folder formattedweb.
* Run the hive script formatted\_hive\_load.hql. This will load the data in the mob
* folder and formattedweb folder in the HDFS to a table formatted\_input in Hive which will be used for data enrichment later.



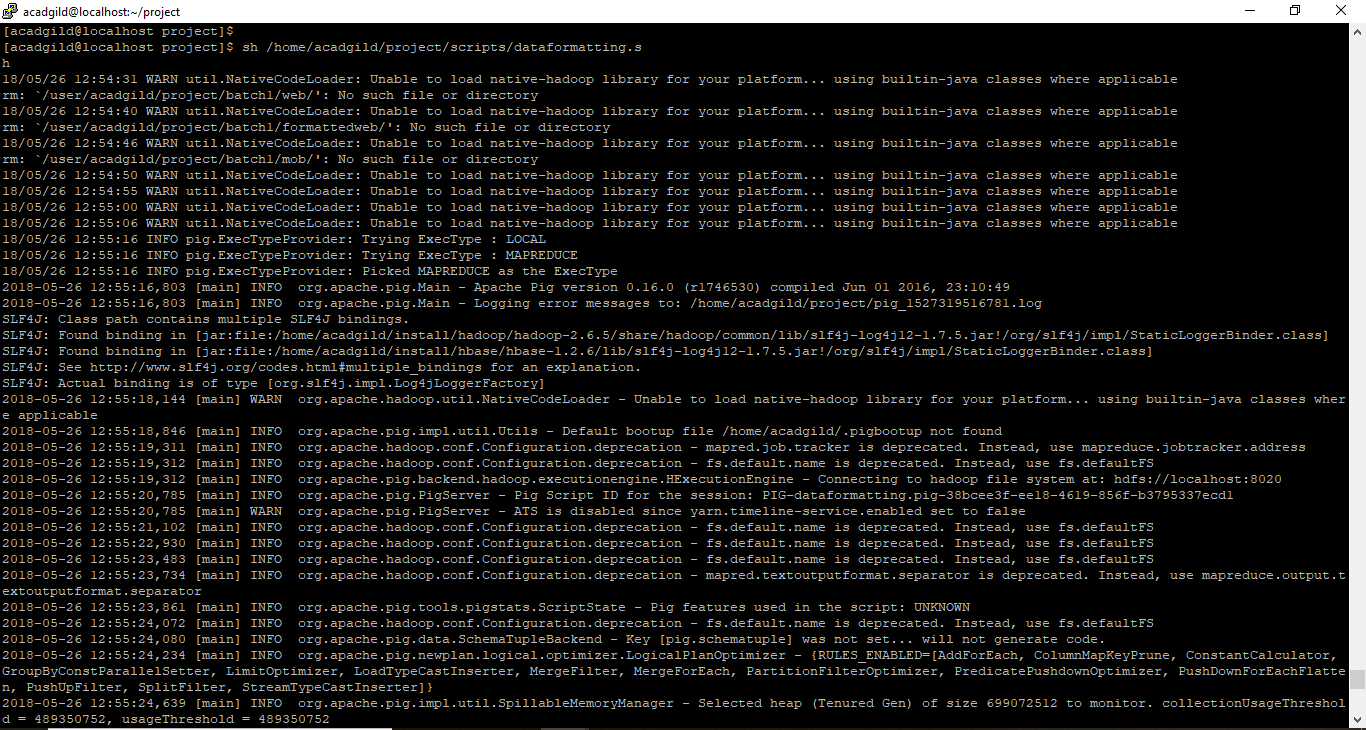
**dataformatting.pig**Stores the formatted data to a folder in the HDFS called formattedweb.



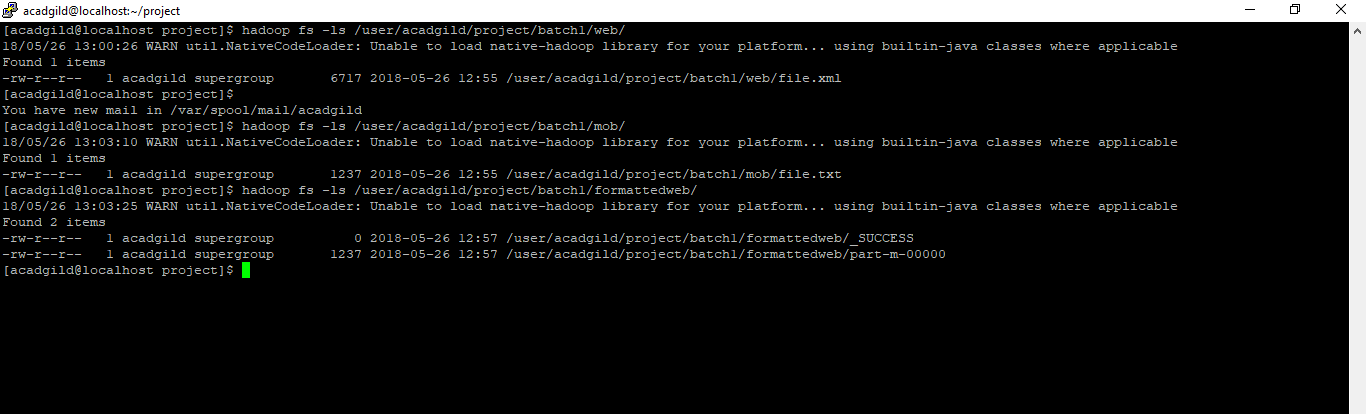
**formatted\_hive\_load.hql**Combines the data from mob and formattedweb to make one data-set and stores it partitioned by batchid.

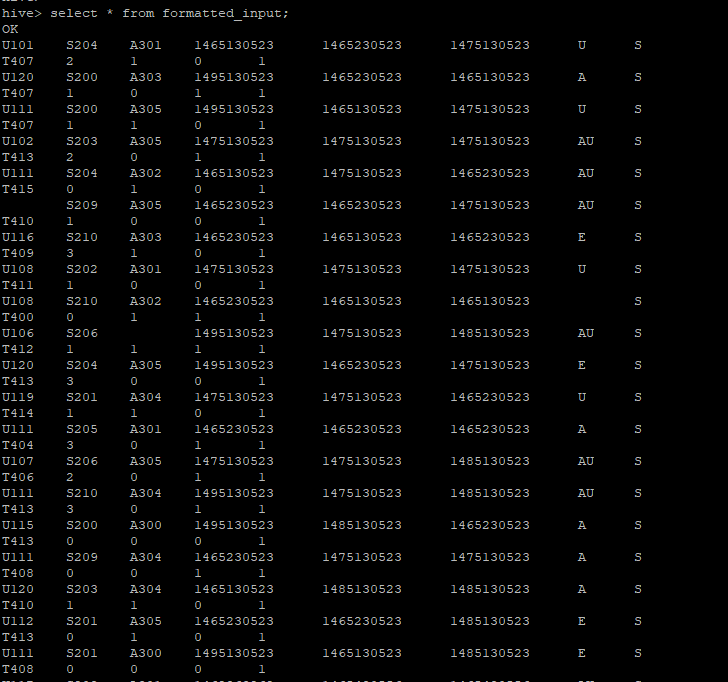


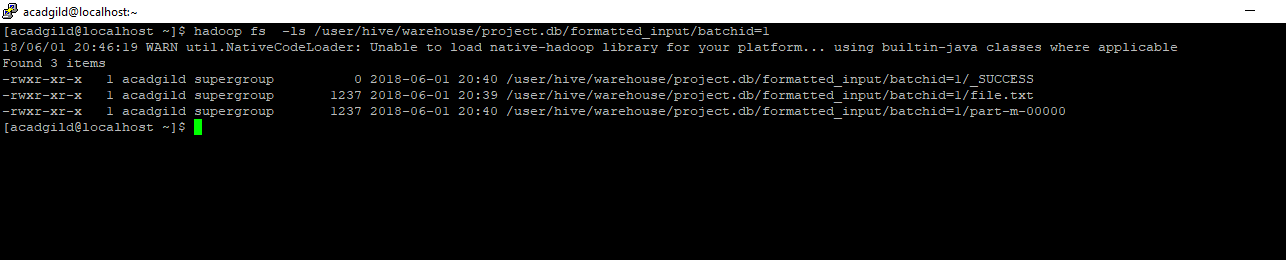
Below is a view of the execution of the above:



**Below is a look into the HDFS:**

****

****

****

**Step 5:**

Perform Data Enrichment and Cleaning

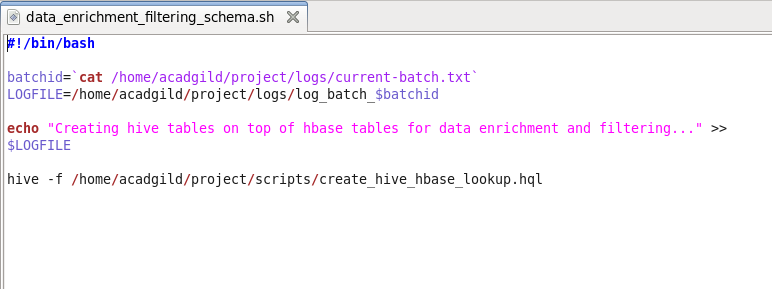
The data enrichment is carried out in two steps:

* Create lookup tables in Hive and import the data from the HBase lookup tables to them. This is done by shell script data\_enrichment\_filtering\_schema.sh
* Perform the data enrichment to the data in formatted\_input using the lookup tables. This is done by shell script data\_enrichment.sh

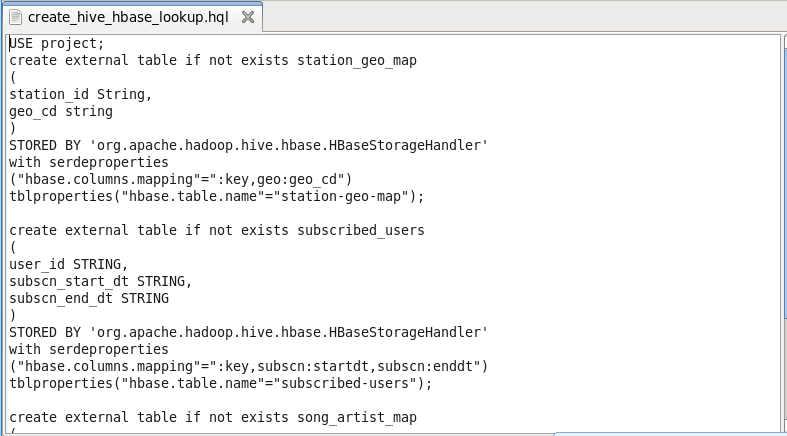
**1) data\_enrichment\_filtering\_schema.sh**

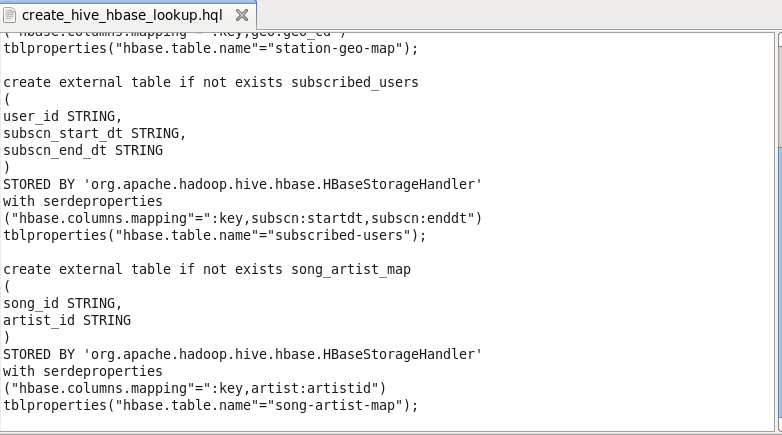
Below is the shell script data\_enrichment\_filtering\_schema.sh where the following operations are performed:

* Get the batch id number from the batch file and get the Log File for the batch using the batch id. This will be log\_batch\_1
* Add logs to the Log File signifying that the Hive lookup tables are created from the HBase lookup tables.
* Run the hive script create\_hive\_hbase\_lookup.hql. This will create the lookup tables in Hive and import the data from the HBase lookup tables to the Hive lookup tables.

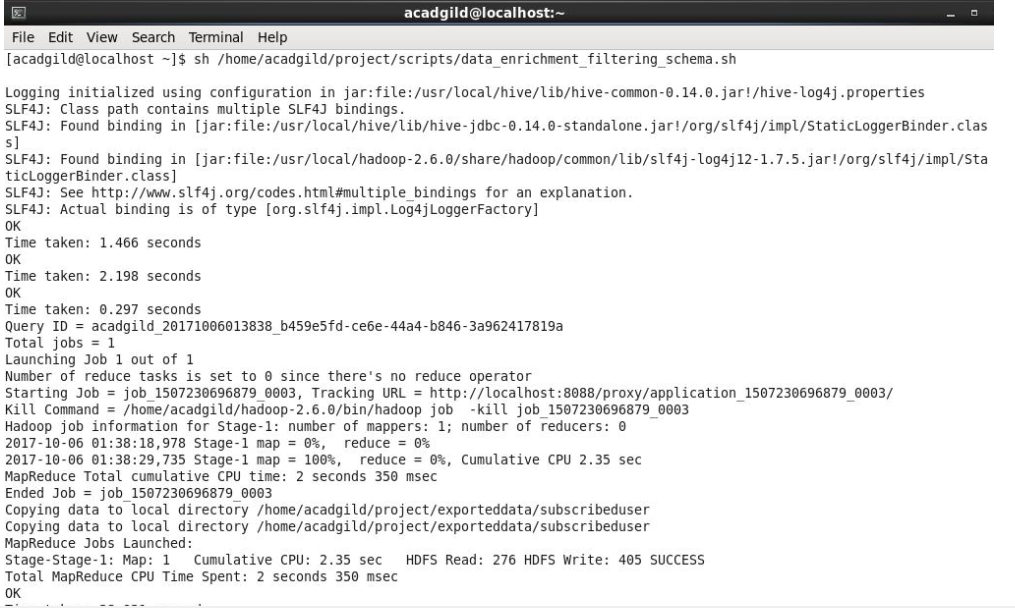
****

**create\_hive\_hbase\_lookup.hql**Create Hive lookup tables and save lookup table **subscribed\_users** to Local FS

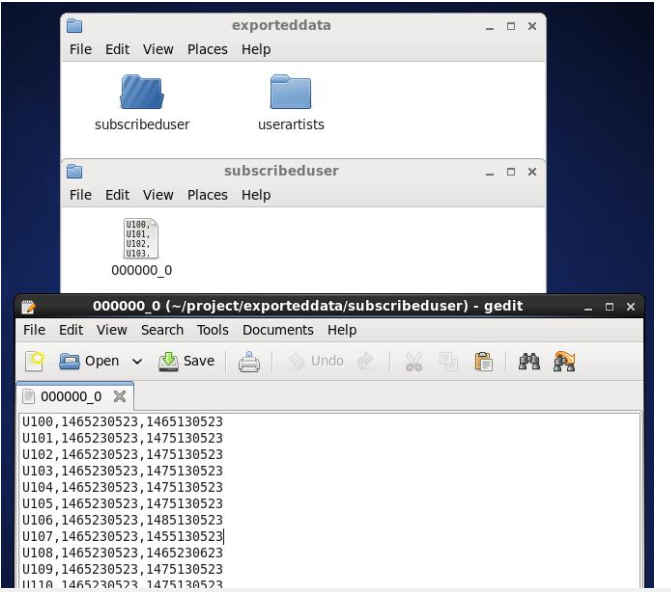


****

Below is a view of the execution of the above:

****

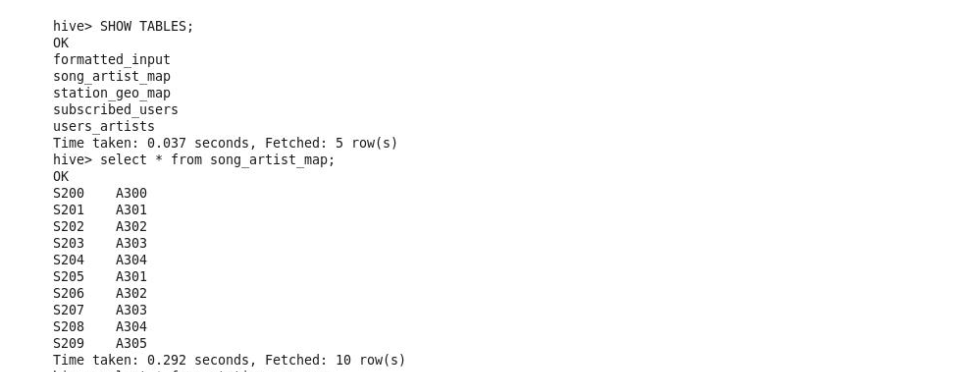
**Output of the saved table subscribed\_users in the Local FS**

****

**Output in Hive:**

The tables were created and populated as intended.

****

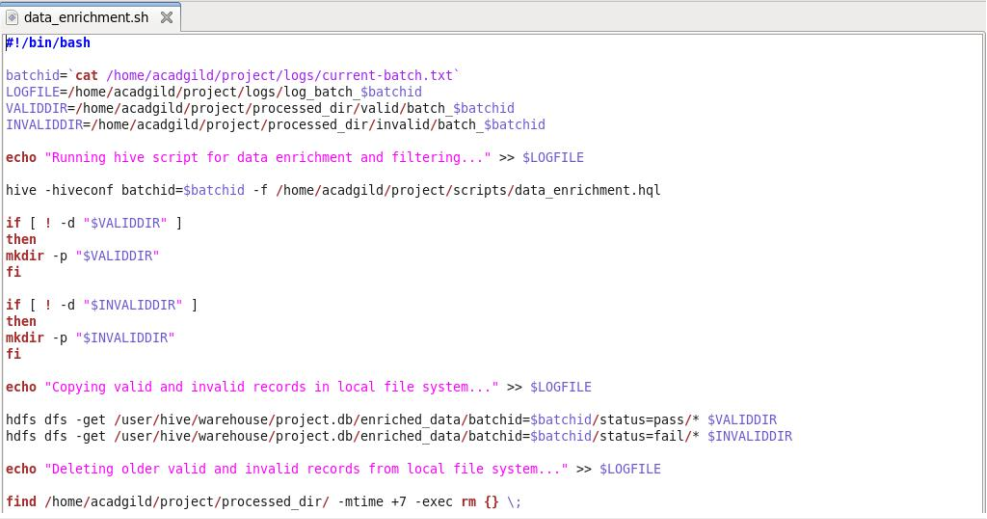
****

****

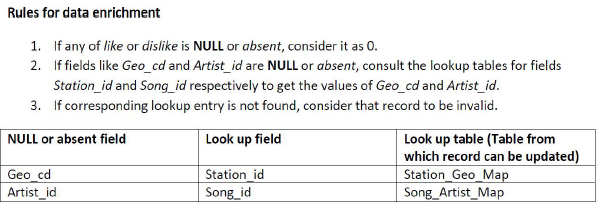
**2) data\_enrichment.sh**

Below is the shell script data\_enrichment.sh where the following operations are performed**:**

* Get the batch id number from the batch file and get the Log File for the batch using the batch id. This will be log\_batch\_1.
* Add logs to the Log File signifying that the data enrichment has begun.
* Run the hive script data\_enrichment.hql. This will create a Hive table
* enriched\_data that will hold the data that is enriched and partitioned based on given rules as pass or fail (status) and batchid.
* Add logs to the Log File signifying that the valid and invalid outputs are being recorded in their respective folders.
* Copy the data from the pass and fail folders (valid & invalid) in the Hive warehouse to the Local FS.

****

**data\_enrichment.hql**



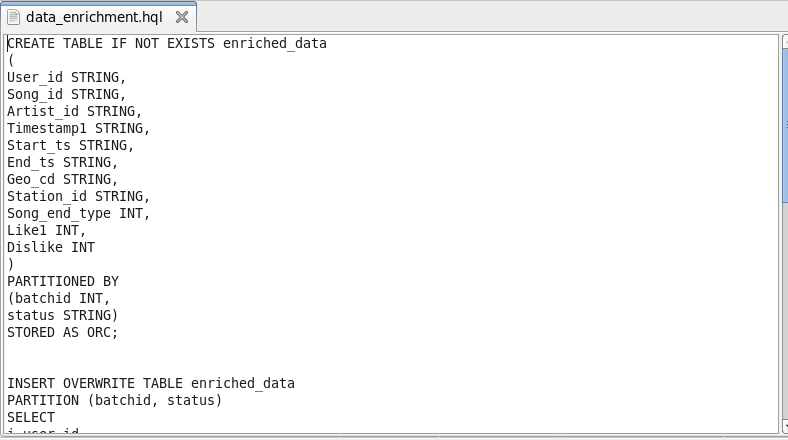
For the data enrichment, a table **enriched\_data** is created and the table is overwritten with the result of the below operations:

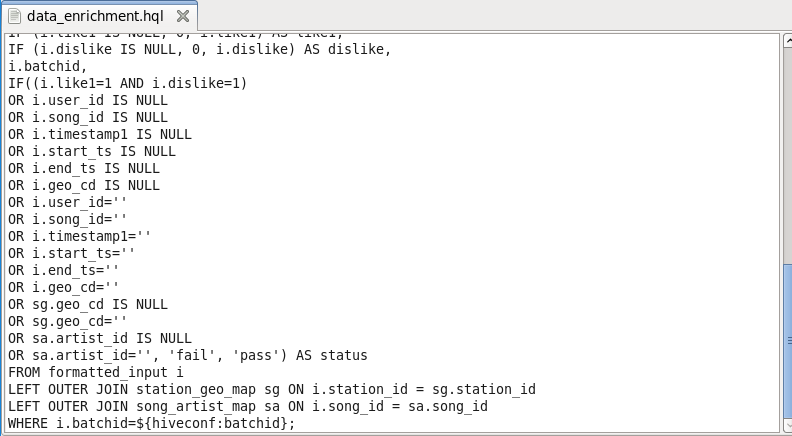
* The data in the **formatted\_input** table is joined with the lookup tables **station\_geo\_map** and **song\_artist\_map** to fill in the data gaps that can be obtained by said tables.

*and*

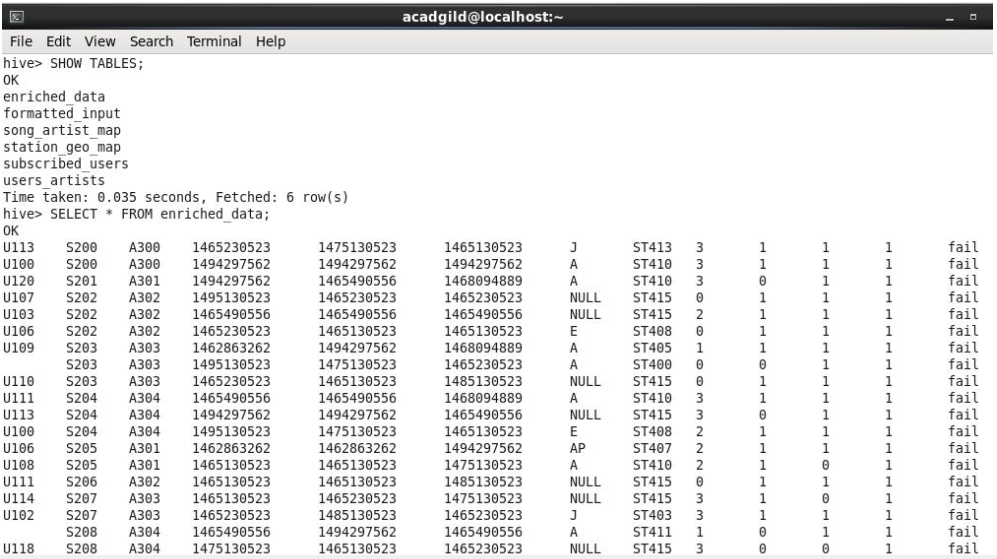
* The same data is then filtered by the rules given above and partitioned by status (pass or fail) & batchid.

The data of the enriched\_data table is then stored in a folder in the Local FS.

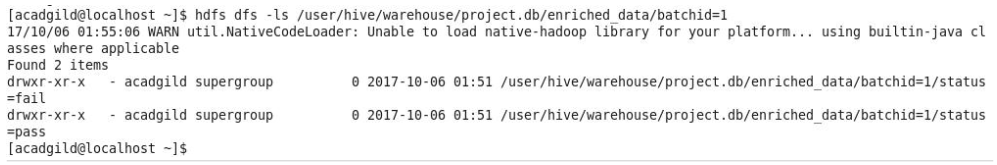




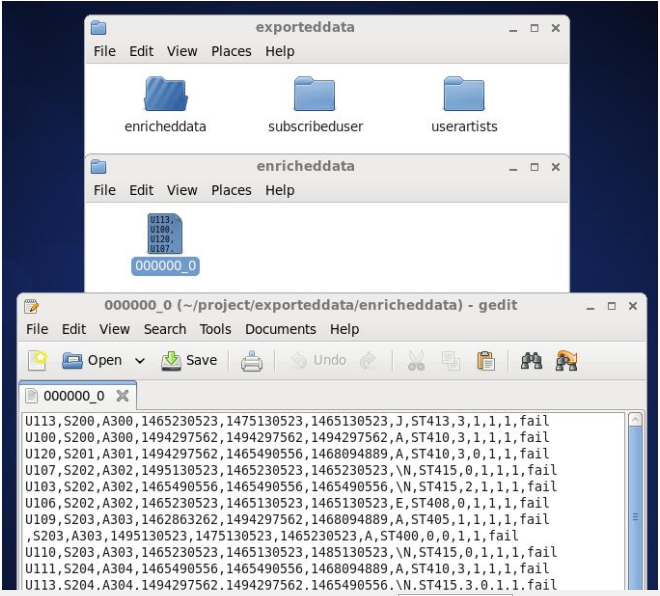
**Output in Hive :**



**Output in Hive Warehouse:**Below is a view of the enriched\_data in the Hive warehouse, partitioned by status and batched



**Output saved in the Local FS:**



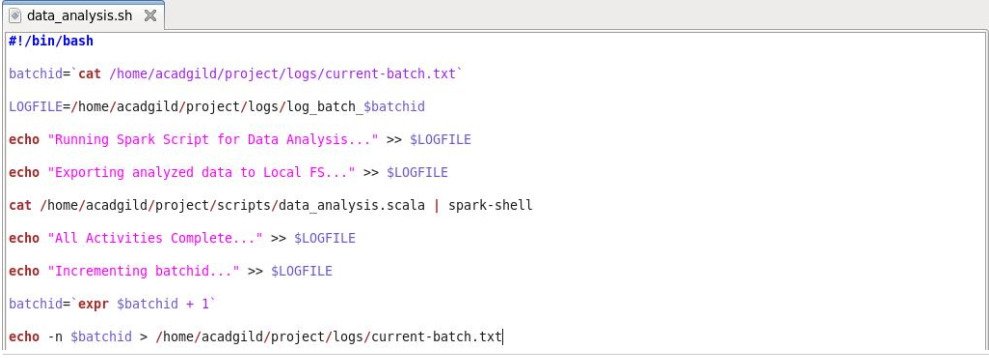
**Step 6:**

**Perform Data Analysis**

**Below is the shell script data\_analysis.sh where the following operations are**

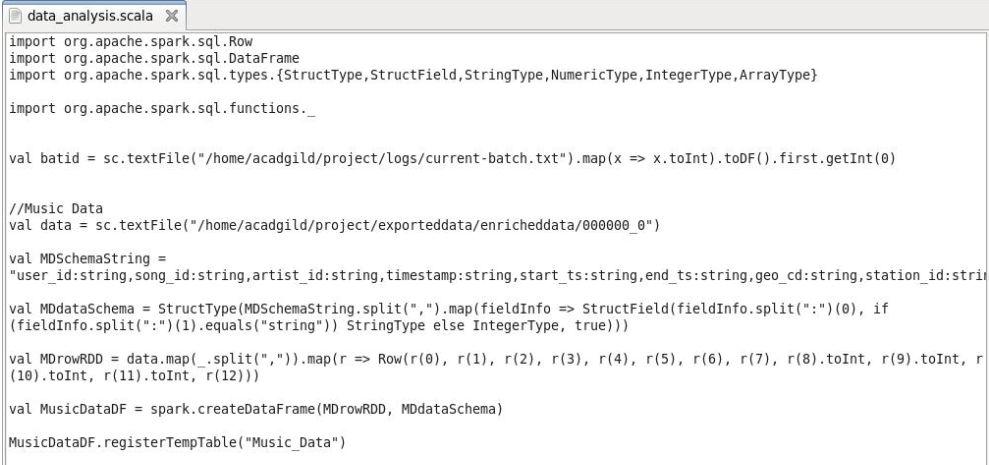
**performed:**

* Get the batch id number from the batch file and get the Log File for the batch using the batch id. This will be log\_batch\_1
* Add logs to the Log File signifying that the data analysis is being performed using Spark and that the result is being exported to the Local FS.
* Run the spark script data\_analysis.scala. This will perform the data analysis required in the problem statement given and save the result to the Local FS.
* Add logs to the Log File signifying that the data analysis has completed and that the batch is being incremented. Here from 1 to 2
* Get batchid number from batch file and increment the batchid by 1.

****

**Initialization:**

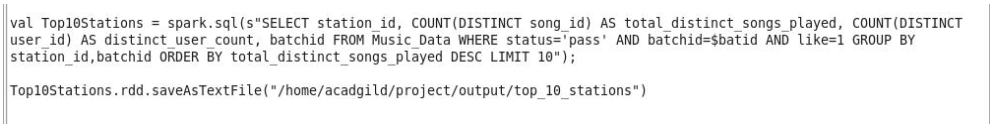
* Import Row, DataFrame, Structure type and function dependencies needed to perform analysis.
* Get the batchid from the batch file and store it in the variable batid
* Get the data that was exported and saved in the Local FS from the steps above i.e. enriched\_data, subscribed\_user and user\_artists and perform the foll. on each of them
* Create the schema for the data
* Create a DataFrame from the schema and data
* Create a temporary table from the DataFrame created



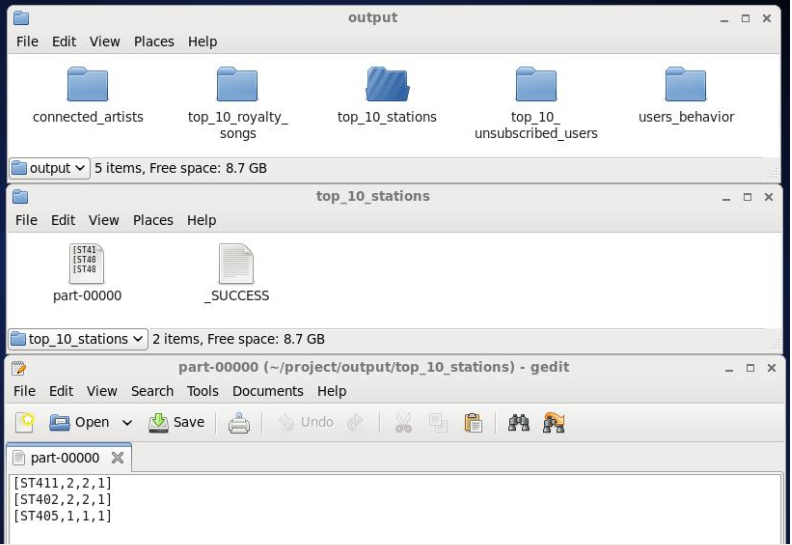
**Problem Statement 1:  
Determine top 10 station\_id(s) where maximum number of songs were played, which were liked by unique users.  
Code:**

val Top10Stations = spark.sql(SELECT station\_id,COUNT(DISTINCT song\_id) AS total\_distinct\_songs\_played, COUNT(DISTINCT user\_id) AS distinct\_user\_count, batchid FROM Music\_Data WHERE status='pass' AND batchid=$batid AND like=1 GROUP BY station\_id,batchid ORDER BY total\_distinct\_songs\_played DESC LIMIT 10");

Top10Stations.rdd.saveAsTextFile("/home/acadgild/project/output/top\_10\_stations")



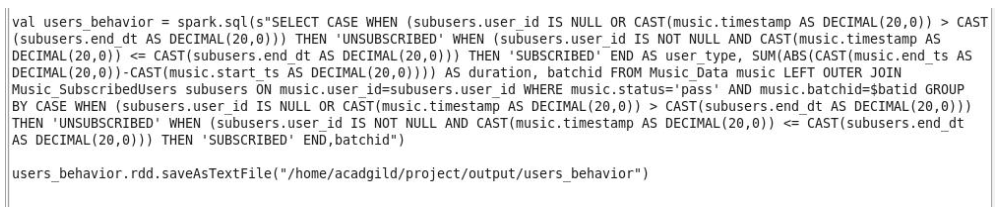
**Output**

****

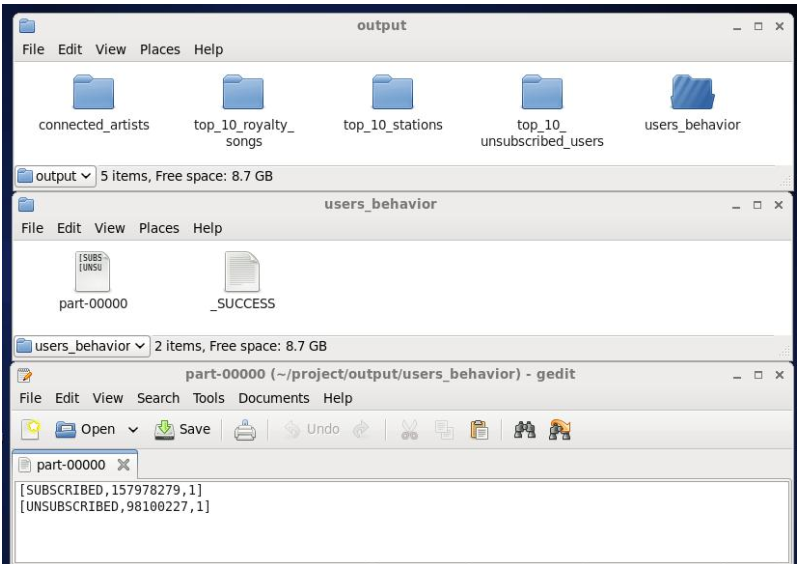
**Problem Statement 2:  
Determine total duration of songs played by each type of user, where type of user can be 'subscribed' or 'unsubscribed'. An unsubscribed user is the one whose record is either not present in Subscribed\_users  
lookup table or has *subscription\_end\_date* earlier than the *timestamp* of the song played by him.**  
**Code:**

val users\_behavior = spark.sql(s"SELECT CASE WHEN (subusers.user\_id IS NULL OR CAST(music.timestamp AS DECIMAL(20,0)) > CAST(subers.end\_dt AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED' WHEN (subusers.user\_id IS NOT NULL AND CAST(music.timestamp AS DECIMAL(20,0)) <= CAST(subusers.end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END AS user\_type, SUM(ABS(CAST(music.end\_ts AS DECIMAL(20,0))-CAST(music.start\_ts AS DECIMAL(20,0)))) AS duration, batchid FROM Music\_Data music LEFT OUTER JOIN Music\_SubscribedUsers subusers ON music.user\_id=subusers.user\_id WHERE music.status='pass' AND music.batchid=$batid GROUP BY CASE WHEN (subusers.user\_id IS NULL OR CAST(music.timestamp AS DECIMAL(20,0)) > CAST(subusers.end dt AS DECIMAL(20,0))) THEN 'UNSUBSCRIBED' WHEN (subusers.user\_id IS NOT NULL AND CAST(music.timestamp AS DECIMAL(20,0)) <= CAST(subusers.end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END,batchid")

users\_behavior.rdd.saveAsTextFile("/home/acadgild/project/output/user\_behavior")

****

**Output :**

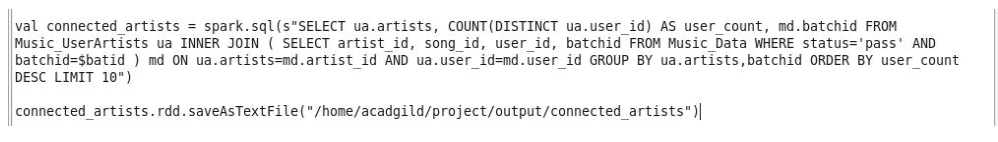
****

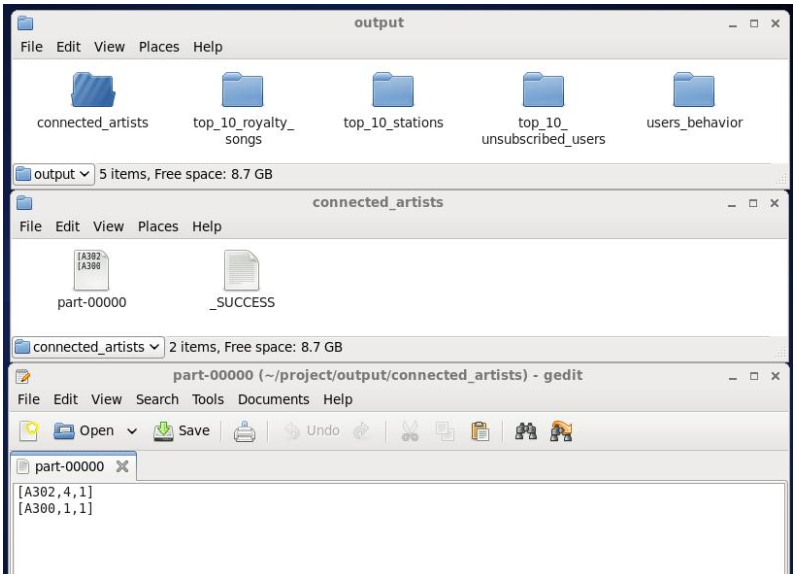
**Problem Statement 3:  
Determine top 10 connected artists. Connected artists are those whose songs are most listened by the unique users who follow them.**

**Code:**

val connected\_artists = spark.sql(s"SELECT ua.artists, COUNT(DISTINCT ua.user\_id) AS user\_count, md.batchid FRPM Music\_UserArtists ua INNER JOIN (SELECT artist\_id, song\_id,user\_id,batchid FROM Music\_Data WHERE status='pass' AND batchid=$batid ) md ON ua.artists=md.artist\_id AND ua.user\_id=md.user\_id GROUP BY ua.artists,batchid ORDER BY user\_count DESC LIMIT 10")

connected\_artists.rdd.saveAsTextFile("/home/acadgild/project/output/connected\_artists")

****

**Output: **

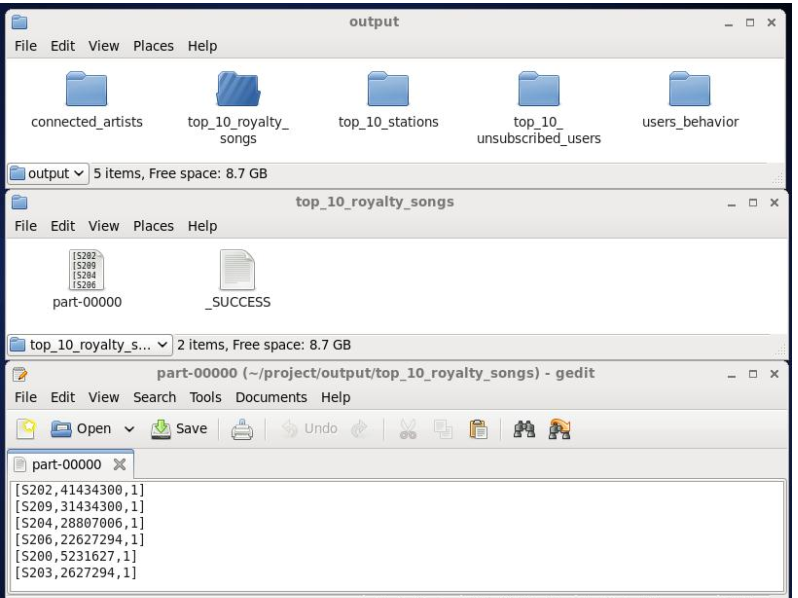
**Problem Statement 4:  
Determine top 10 songs who have generated the maximum revenue. Royalty applies to a song only if it was *liked* or was *completed successfully* or both.**  
**Code:**

val top\_10\_royality\_songs = spark.sql(s"SELECT song\_id, SUM(ABS(CAST(end\_ts AS DECIMAL(20,0))-CAST(start\_ts AS DECIMAL(20,0))) AS Duration, batchid FROM Music\_Data WHERE status='pass' AND batchid=$batid AND (like=1 OR song\_end\_type=0) GROUP BY song\_id,batchid ORDER BY duration DESC LIMIT 10")

top\_10\_royality\_songs.rdd.saveAsTextFile("/home/acadgild/project/output/top\_10\_royality\_songs")

****

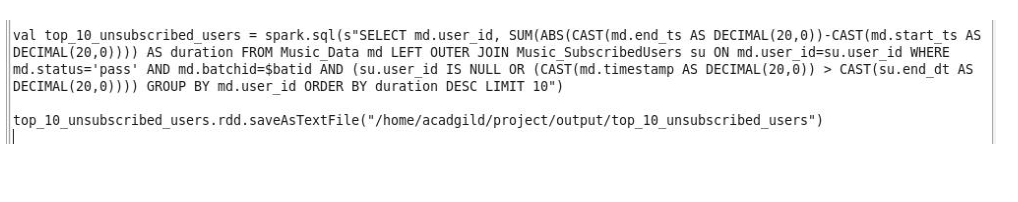
**Output:**

****

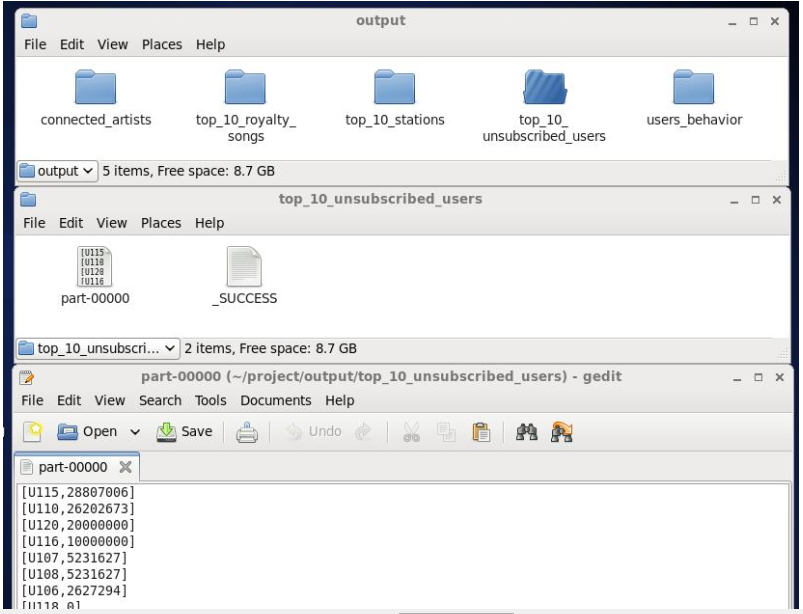
**Problem Statement 5:  
Determine top 10 unsubscribed users who listened to the songs for the longest duration.**  
**Code:**

val top\_10\_unsubscribed\_users = spark.sql(s"SELECT md.user\_id, SUM(ABS(CAST(ms.end\_ts AS DECIMAL(20,0))-CAST(md.start\_ts AS DECIMAL(20,0)))) AS duration FROM Music\_Data md LEFT OUTER JOIN Music\_SubscribedUsers su ON md.user\_id=su.user\_id WHERE md.status='pass' AND md.batchid=$batid AND (su.user\_id IS NULL OR (CAST(md.timestamp AS DECIMAL(20,0)) > CAST(su.end\_dt AS DECIMAL(20,0)))) GROUP BY md.user\_id ORDER BY duration DESC LIMIT 10")

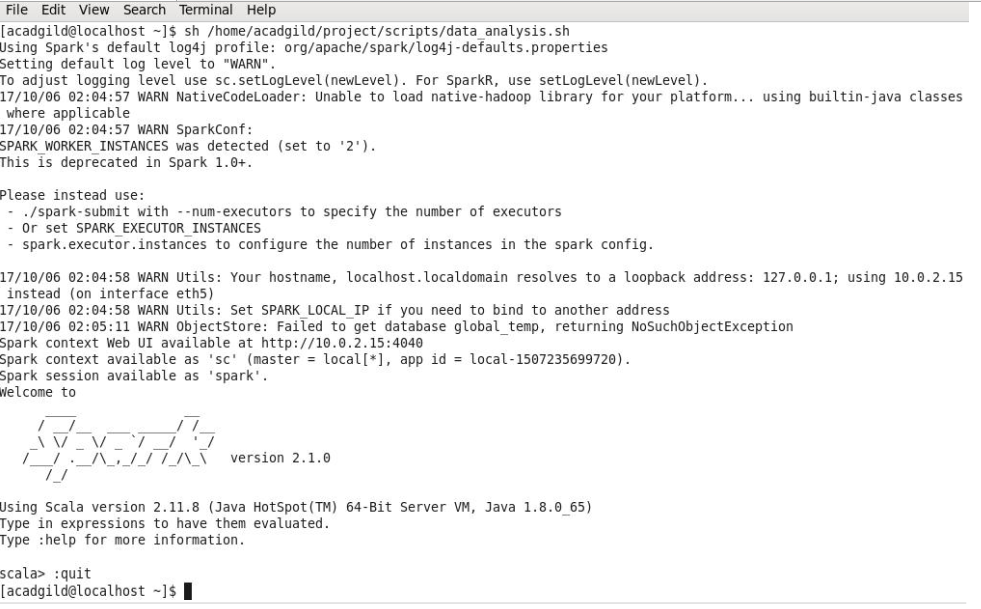
top\_10\_unsubscribed\_users.rdd.saveAsTextFile("/home/acadgild/project/output/top\_10\_royality\_songs")

****

**Output:**

****

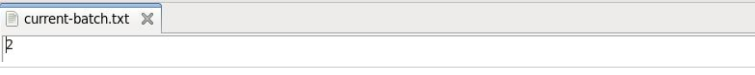
Below is a view of the execution of the above:

****

**Post Analysis:**A view of the log file post analysis.



The batchid is incremented from 1 to 2:



A view of the log folder:

