**Project 1 – Music Data Analysis**

**Student Name:** Sri Harini Kamaraju

**Course:** Big Data Engineering with Hadoop & Spark

**Contents**

1. Project Description…………………………………………………………………………………………………………… 2
2. Design of the Project………………………………………………………………………………………………………... 2

2.1 Flow of Operations …………………………………………………………………………………………………….. 3

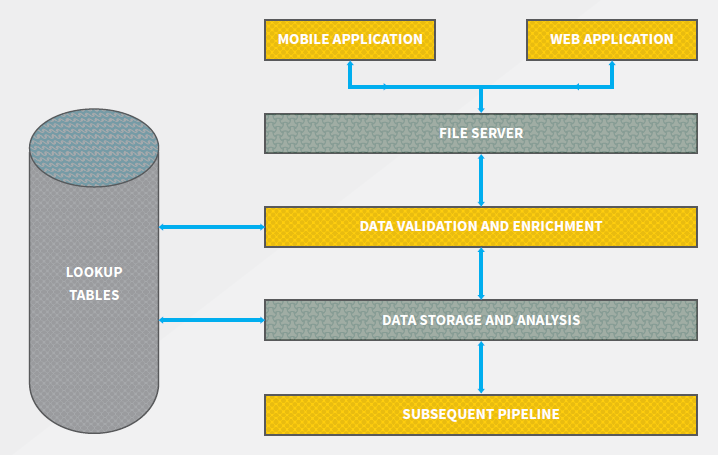
1. Data Files …………………………………………………………………………………………………………………………. 4
   1. Project Data Description ………………………………………………………………………………………. 4
   2. Lookup Tables ………………………………………………………………………………………………………. 5
2. Data Ingestion and Initial Validation ………………………………………………………………………………… 5
   1. Rules for data ingestion and data filtering ……………………………………………………………. 5
   2. Data Creation ……………………………………………………………………………………………………….. 6
   3. Lookup Tables and Initial Validation ……………………………………………………………………. 12
   4. Creating and populating Lookup tables in HBase ………………………………………………… 14
   5. Initial data validation and formatting …………………………………………………………………. 20
3. Data Enrichment ……………………………………………………………………………………………………………. 26
   1. Rules for Data Enrichment …………………………………………………………………………………. 26
   2. Rules for Post Enrichment ………………………………………………………………………………….. 26
   3. Data Filtering and Enrichment ……………………………………………………………………………. 26
4. Data Analysis …………………………………………………………………………………………………………………. 35
   1. Challenges and Optimizations ……………………………………………………………………………. 35
   2. Data Analysis (Using Hive) …………………………………………………………………………………. 35
   3. Data Analysis (Using Spark Submit) ……………………………………………………………………. 43
5. Post Analysis …………………………………………………………………………………………………………………. 48
   1. Data export to MySQL using Sqoop …………………………………………………………………… 48
   2. Job Scheduling ………………………………………………………………………………………………….. 53
6. Conclusion ……………………………………………………………………………………………………………………. 56
7. **Project Description**

A leading music-catering company is planning to analyze large amount of data received from varieties of sources, namely mobile app and website to track the behavior of users, classify users, calculate royalties associated with the song and make appropriate business strategies. The file server receives data files periodically after every 3 hours.

1. **Design of the Project**

The following flowchart shows the High level design of this project

**2.1 Flow of Operations**

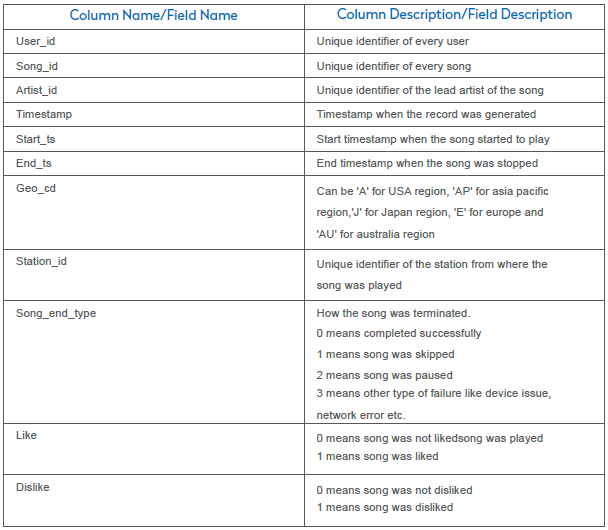


1. **Data Files**

Data set consists of user information, song details like song\_id , Artist\_id, and number of likes and dislikes received for each song.

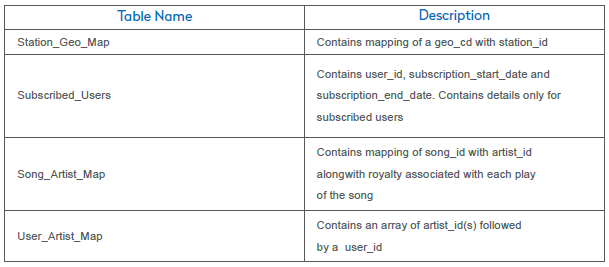
**3.1 Project Data Description**

Data files contain the below fields.



**3.2 LookUp Tables**

There are some existing look up tables present in NoSQL databases. They play an important role in data enrichment and analysis.



1. **Data Ingestion and Initial Validation**

**4.1 Rules for data ingestion and data filtering**

* Data coming from web applications reside in /data/web and has xml format.
* Data coming from mobile applications reside in /data/mob and has csv format.
* Data files come every 3 hours.
* All the timestamp fields in data coming from web application is of the format YYYY-MM-DD HH:MM:SS.
* All the timestamp fields in data coming from mobile application is a long integer interpreted as UNIX timestamps.
* Finally, all timestamps must have the format of a long integer to be interpreted as UNIX timestamps.
* If both like and dislike are 1, consider that record to be invalid.
* If any of the fields from User\_id, Song\_id, Timestamp, Start\_ts, End\_ts, Geo\_cd is NULL or absent, consider that record to be invalid.
* If Song\_end\_type is NULL or absent, treat it to be 3
* Create a temporary identfier for all the data files received in the last 3 hours (may be an integer batch\_id which is auto incremented or a string obtained a\_er combining current date and current hour, to keep track of valid and invalid records per batch).

**4.2 Data Creation**

As specified in the rules for data ingestion, the data coming from web should be in xml format and data coming from mobile must be in csv formats.

Both these rules are taken into account while we create data for each of the sources.

Initially, we have to give proper permission to all the scripts present under scripts sub directory. This can be done by issuing command

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

There are 2 files for creation.

1. Web data
2. Mob data

Since we are simulating this project in a pseudo distributed mode, we cannot connect to a server and pull in data. Hence we are generating the data using python scripts.

Getting into generate\_web\_data.py:

Importing python classes into the code.

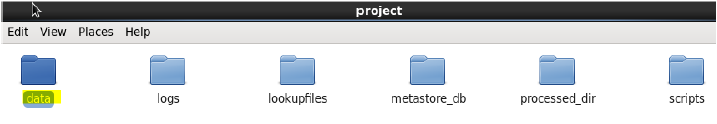
*from random import randint*

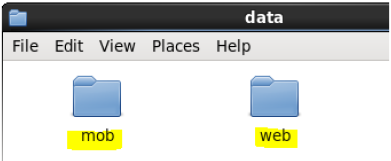
*from random import choice*

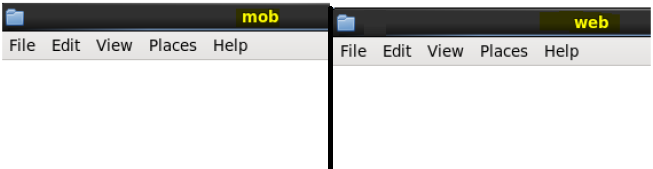
Open is a function in python that will open a file present in the given location. If file is not present, it will create a file in that location. The second parameter is the mode which specifies that the file will be opened/created in write mode.

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

Before running the scripts, no file is present in both the folders mob and web.







Setting the counter to 20 so that 20 distinct records will be generated.

*count = 20*

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

Manually creating lists for all the specified lists like geo\_cd,song\_end\_type, time\_stamp,start\_ts,end\_ts.

*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"]*

This condition is checked for the creation of few invalid records in middle.

Also, the format of the user\_id is U101(a random number between 100 and 120), song\_id is S101(a random number between 200 and 210),station\_id is ST401(a random number between 400 and 415)

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

*user\_id = ""*

*else:*

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

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

This condition is checked here to create some invalid records in the artist\_id column.

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

*artist\_id = ""*

*else:*

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

The timestamp,start\_ts and end \_ts are selected at random from the above lists.

*timestamp = choice(timestamp\_list)*

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

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

This condition is checked here to create some invalid records in the geo\_cd column. Geo\_cd is assigned by selecting a random number from the above list.

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

*geo\_cd = ""*

*else:*

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

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

song\_end\_type, like and dislike values are selected at random for each record.

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

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

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

All these values are written onto a file in the xml format.

*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")*

For the invalid record to be created, the count value is modified after each record.

*count = count-1*

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

*file.close()*

The code is similar for the mob data too, except that

1. The file name is file.txt
2. The timestamp format for mobile data is different (specified in the lists).
3. the data written to the file is in txt format instead of an xml.

*from random import randint*

*from random import choice*

File name specified is file.txt. As seen in the screenshot above, the file is not present before the program is run.

*file = open("/home/acadgild/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"]*

The timestamp format specified here is different from web data.

*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))*

Data is written in a comma separated format into the file.

*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()*

After both the files are run, the data is generated and the files are created in the respective folders.

Command to run:

generate\_web\_data.py -- Generates some random data coming from web application

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

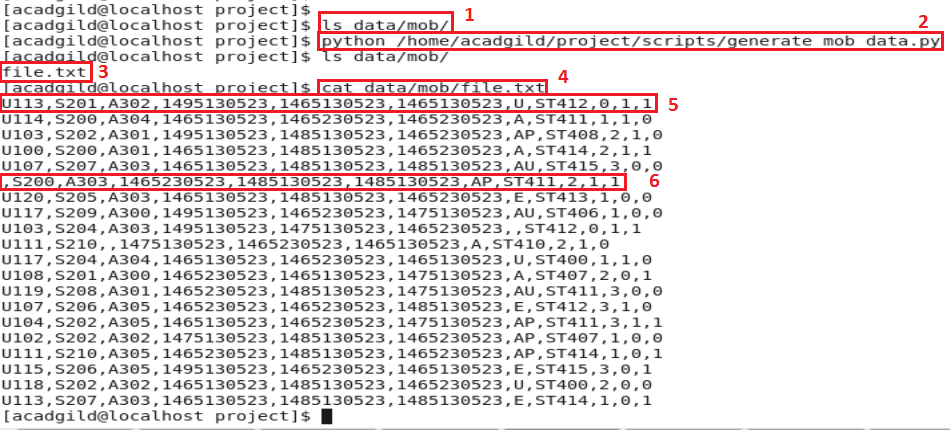


With reference to the screenshot above,

1. Listing data in web folder before running the script and it can be seen that there is no data in the folder.
2. Running the python script.
3. After the script is run, when an ls is performed on web folder, we can see the **file.xml** is created.
4. Displaying the contents of the file using cat command.
5. It can be seen that the data inside the file is in xml format.

generate\_mob\_data.py -- Generates some random data coming from mobile application

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



With reference to the screenshot above,

1. Listing data in mob folder before running the script and it can be seen that there is no data in the folder.
2. Running the python script.
3. After the script is run, when an ls is performed on mob folder, we can see the **file.txt** is created.
4. Displaying the contents of the file using cat command.
5. It can be seen that the data inside the file is in comma separated format.
6. Since the data should also contain invalid records, we wrote conditions for the same and hence some records contain null values. These records will be enriched in the further phases.
   1. **Lookup Tables and Initial Validation**

Location of the file: Start-daemons.sh -> /home/acadgild/project/scripts/start-daemons.sh

This shell script will start all the required hadoop daemons.

*#!/bin/bash*

If the file ‘current-batch.txt’ is present in the specified location, then it will print ‘*Batch File Found!*’ If not, then the file is created and the “1” will be inserted as contents of the file.

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

*then*

*echo "Batch File Found!"*

*else*

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

*fi*

Changing the permission of the created file. A batchid is taken from the content of the created batch file. This batchid is used again to perform the task in order of batches.

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

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

A file named log file concatenated with the batch id is created and this file wil be used to enter all the logs for that particular batch.

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

Logs being entered into the log file.

*echo "Starting daemons" >> $LOGFILE*

Starting all the required services.

*# To Start Hadoop Daemons:*

*start-all.sh*

*# To start the HMASTER service:*

*start-hbase.sh*

*# To Start the JobHistory server Services:*

*mr-jobhistory-daemon.sh start historyserver*

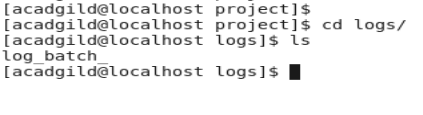
*# To Start the mysql service*

*sudo service mysqld start*

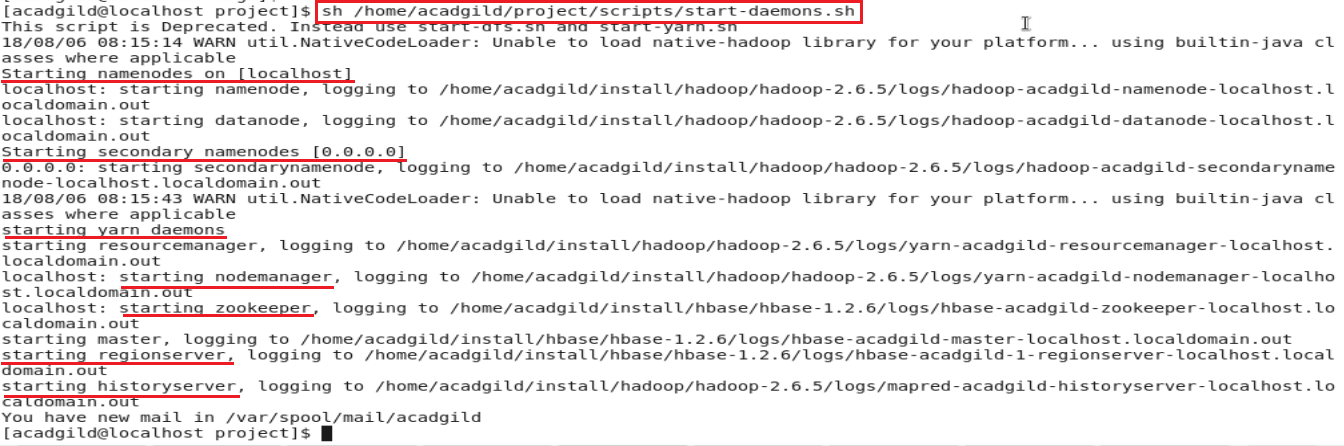
*# To Start HIVE metastore:*

*hive --service metastore*

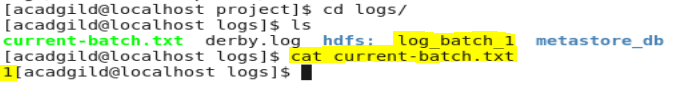
Before this script is run, there is no file in the logs directory.



After the script is run, all the required services are up and running.



Now as the script is successfully executed, a log file should have been created in the logs directory.



As seen in the above screenshot, the current-batch.txt file is generated along with the log file log\_batch\_1.

The file current-batch.txt contains the number of the bath that is currently being executed which is presently 1.

**4.4 Creating and populating Lookup tables in HBase**

*#!/bin/bash*

Fetching batch\_id and opening log file corresponding to the current job from the logs folder.

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

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

Entering logs into log file.

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

Creating 3 lookup tables in hbase via hbase shell.

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

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

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

Entering logs into log file.

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

Reading contents from a file stn-geocd.txt and inserting data into the respective table station-geo-map in hbase using put command.

*file="/home/acadgild/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"*

Reading contents from a file song-artist.txt and inserting data into the respective table song-artist-map in hbase using put command.

*file="/home/acadgild/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"*

Reading contents from a file user-subscn.txt and inserting data into the respective table subscribed-users in hbase using put command.

*file="/home/acadgild/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"*

Once all the tables are created in hbase, a hive script is run.

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

The hql file is as below :

Creating a database named project if it doesn’t already exist. And also using the project.

*CREATE DATABASE IF NOT EXISTS project;*

*USE project;*

A table named users\_artists is created with 2 fields.

*CREATE TABLE users\_artists*

*(*

*user\_id STRING,*

*artists\_array ARRAY<STRING>*

*)*

*ROW FORMAT DELIMITED*

*FIELDS TERMINATED BY ','*

*COLLECTION ITEMS TERMINATED BY '&';*

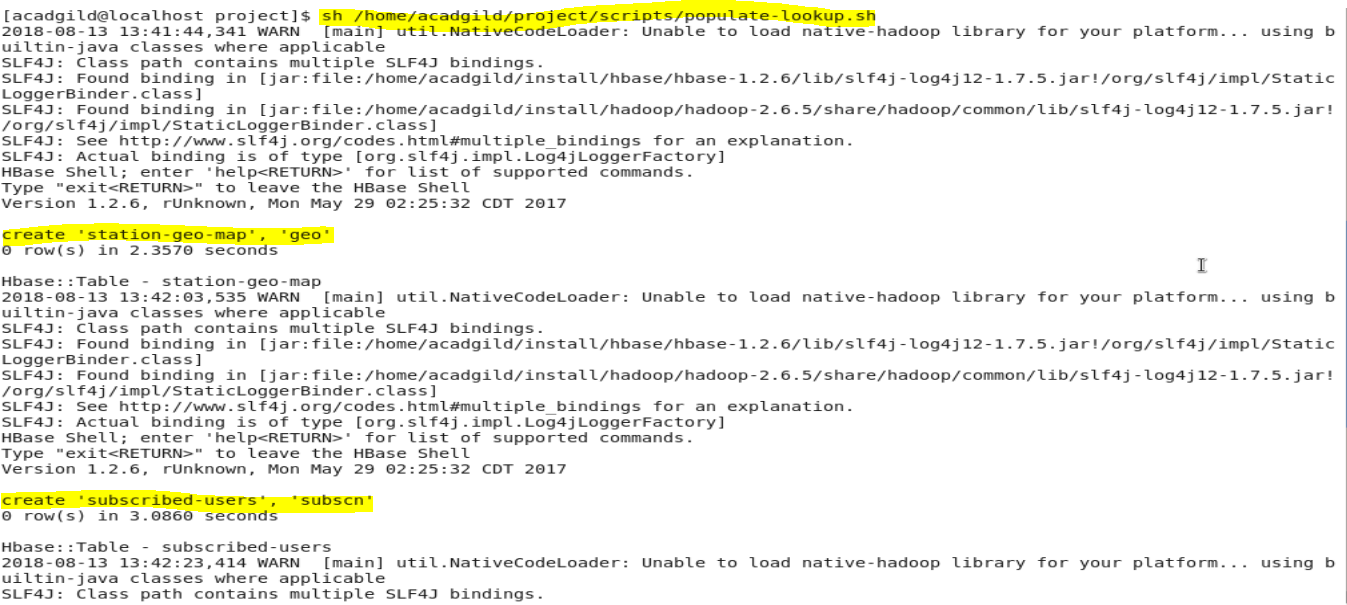
This table is populated from the file user-artist.txt and the existing content will be overrriden.

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

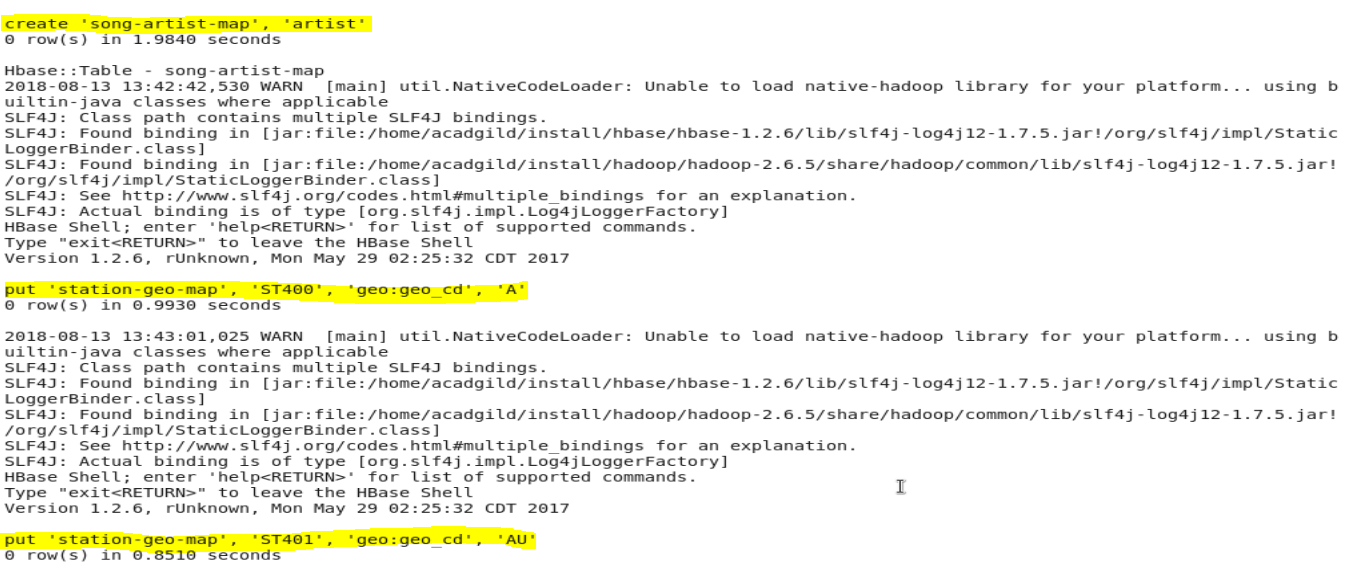
*OVERWRITE INTO TABLE users\_artists;*

Once this script is run, we can see the following tables is created and data is being inserted into the tables in hbase and one table in hive.

Command: *sh /home/acadgild/project/scripts/populate-lookup.sh*

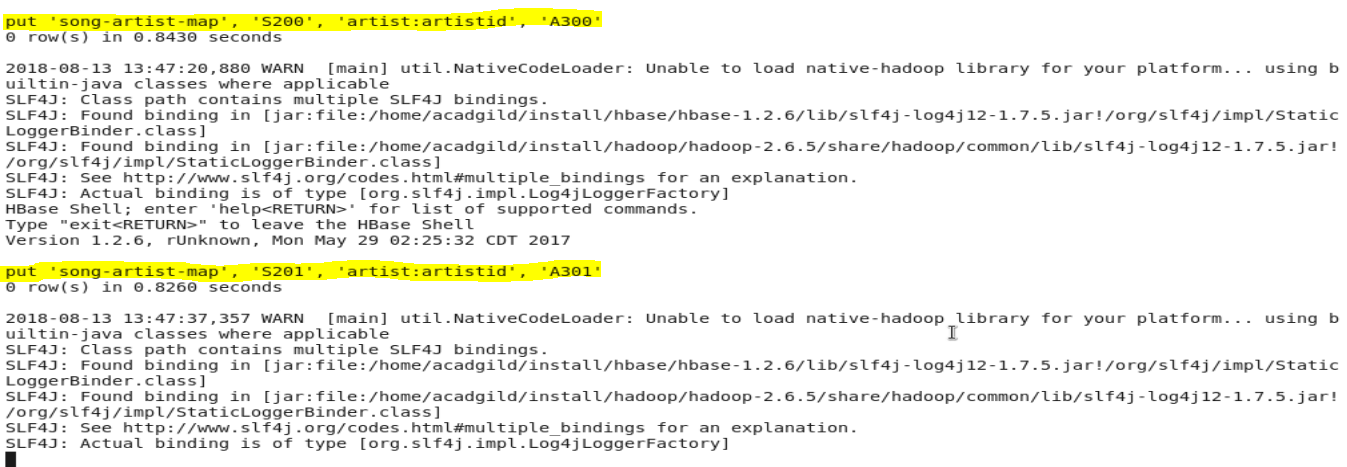


Creation of tables ‘station-geo-map’ and ‘subscribed-users’ can be seen in the above screenshot.

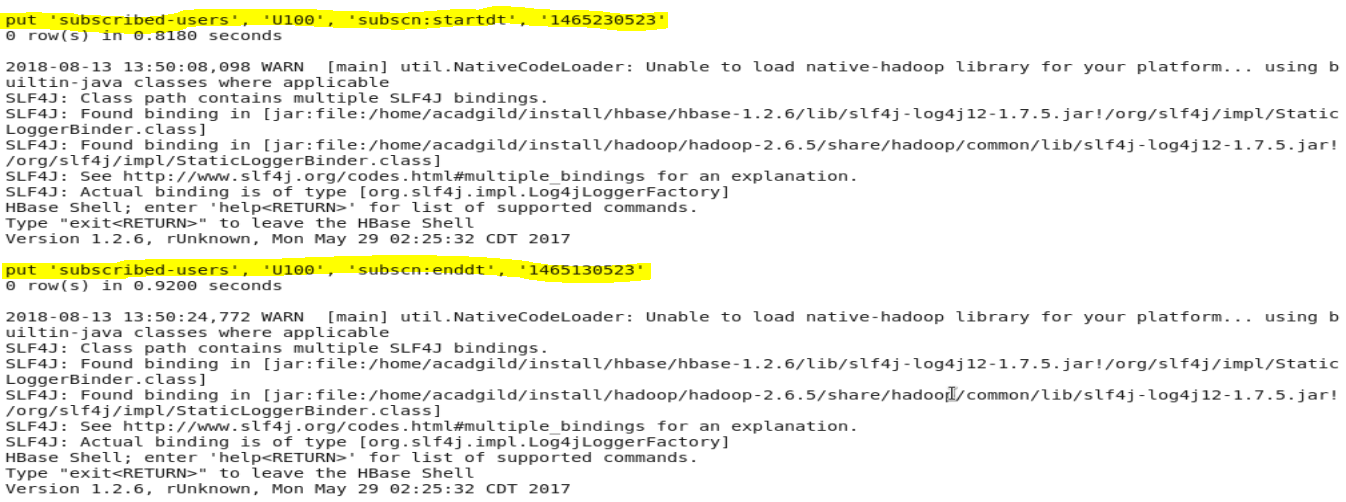


Creation of table ‘song-artist-map’ is shown in the above screenshot.

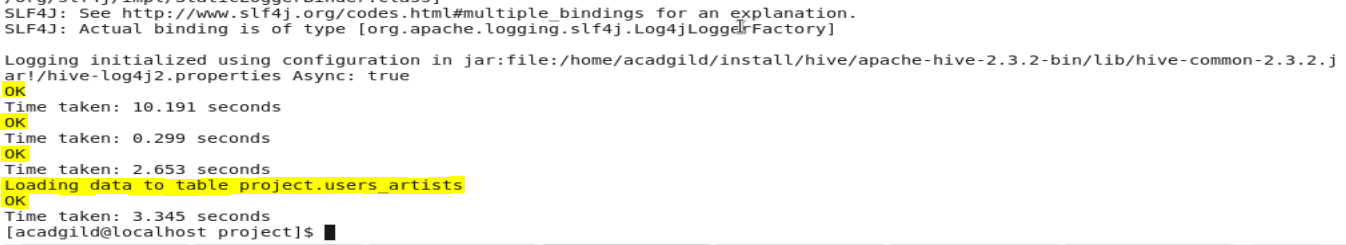
Inserting data into table station-geo-map using put command is seen in the above screenshot.



Inserting data into table song-artist-map using put command is seen in the above screenshot.



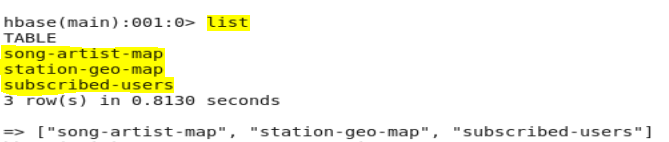
Inserting data into table subscribed-users using put command is seen in the above screenshot.



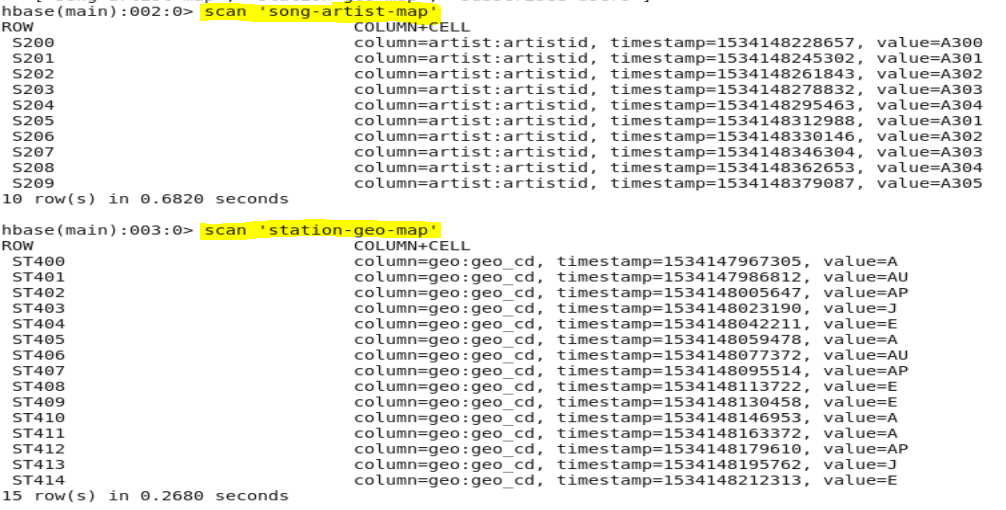
The hql file is run after all the tables are populated in hbase. The table is created and data is loaded into the table users\_artists.

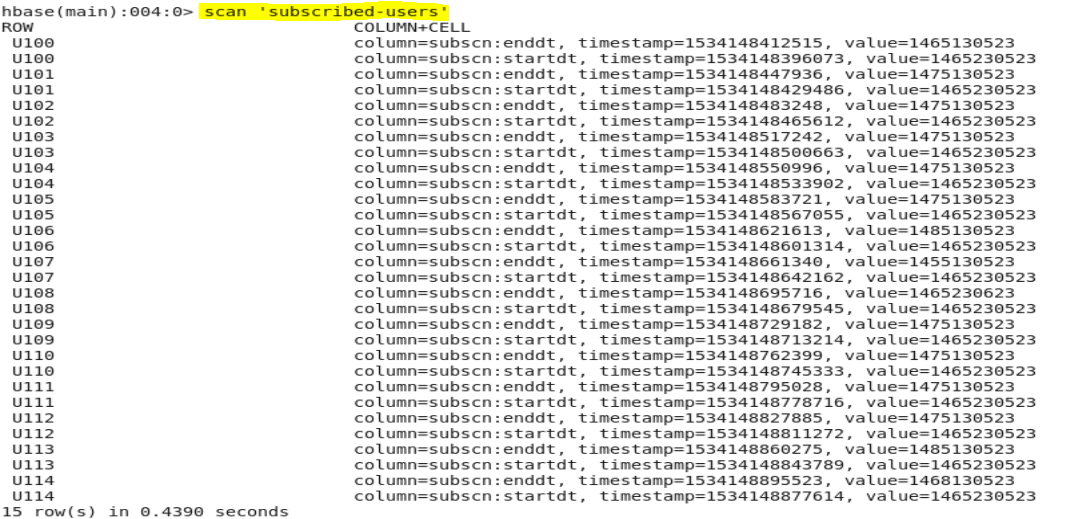
After the script is successfully run, we can verify the tables in both hbase and hive.

The list of tables in hbase in seen in the screenshot below.

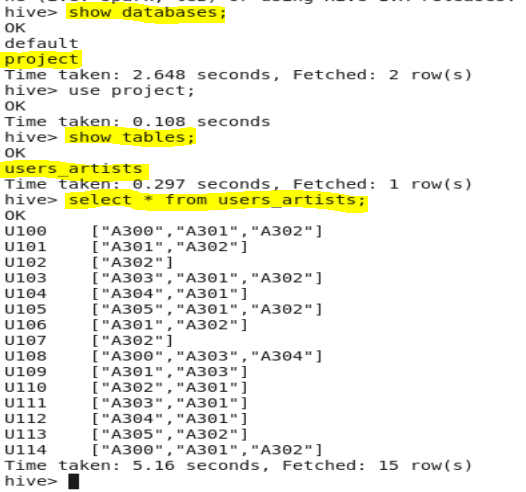


The records of these tables can be seen in the screenshots below.





The database ‘project’ is created in hive. In the database ‘Project’, the table ‘users\_artists’ is seen. The table is also populated.



**4.5 Initial data validation and formatting**

According to the Initial data validation rules,

* All the timestamp fields in data coming from web application is of the format YYYY-MM-DD HH:MM:SS.
* All the timestamp fields in data coming from mobile application is a long integer interpreted as UNIX timestamps.
* Finally, all timestamps must have the format of a long integer to be interpreted as UNIX timestamps.

So, the generated data has to be validated and has to be formatted accordingly. This is done using the following shell scripts: dataformatting.sh

*#!/bin/bash*

Fetching batch\_id and opening log file corresponding to the current job from the logs folder.

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

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

Entering logs into log file.

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

Recursively removing folders with the below names if present using hadoop command rm -r.

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

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

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

Creation of directories mob and web at the particular locations using hadoop command mkdir.

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

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

Copying the generated data onto HDFS using command put.

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

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

Entering logs into log file.

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

Running the following pig file to perform data formatting.

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

Entering logs into log file.

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

The formatted data is now entered into a hive table. This is done using the following hql file.

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

Let’s take a deeper look into both the pig file and the hql file.

dataformatting.pig:

Registering the piggybank jar file for using it’s XPath() class. Since the data coming from web is in xml format, we need XPath class to read the xmls data.

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

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

Loading the data from web folder into a relation A.

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

Performing some formatting by trimming the spaces from the fields.

Also, converting the 3 timestamp fields : timestamp, start\_ts and end\_ts into Unix timestamp format.

*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 timestp,*

*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;*

Storing the final validated and formatted relation into a folder called formattedweb on local file system.

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

Formatted\_hive\_load.hql:

Using the earlier created database named project.

*USE project;*

A table named formatted\_input is created with the following schema with all the columns present in the dataset. This table is partitioned on the batch\_id.

*CREATE TABLE IF NOT EXISTS formatted\_input*

*(*

*user\_id STRING,*

*song\_id STRING,*

*artist\_id STRING,*

*timestmp STRING,*

*start\_ts STRING,*

*end\_ts STRING,*

*geo\_cd STRING,*

*station\_id STRING,*

*song\_end\_type INT,*

*liked INT,*

*disliked INT*

*)*

*PARTITIONED BY*

*(batchid INT)*

*ROW FORMAT DELIMITED*

*FIELDS TERMINATED BY ',';*

The preprocessed data from pig that was loaded into formatted web folder is loaded into this table from local file system.

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

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

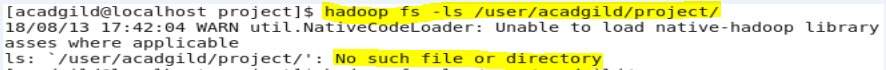
The data from mob folder is also loaded into this table.

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

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

Below are a few observations.

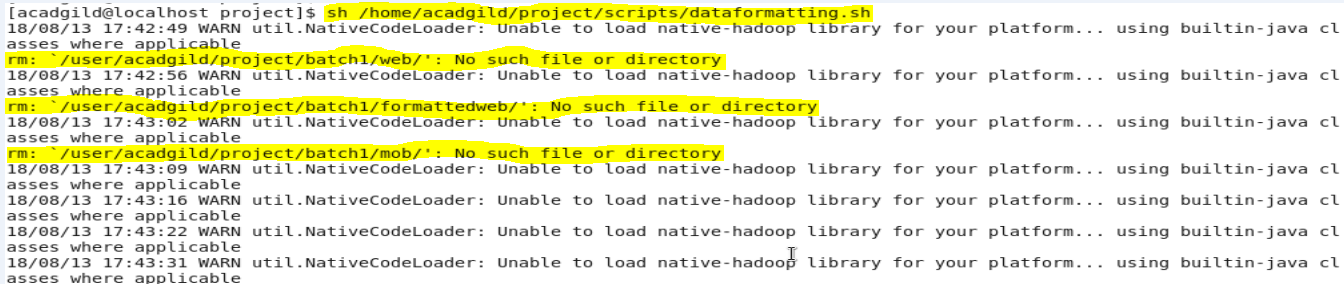
Before execution, there is no folder called ‘project’ on HDFS.



After the script is run,

It tries to delete 3 nonexistent folders, hence the message that no such file or directory is present.

Next, the creation of the folders mob and web and copying data from local to HDFS folders is done. But these operations do not put up any messages on the console. The next 4 lines in the screenshot below correspond to these steps.



After these steps, the pig script is run and generates the following log.

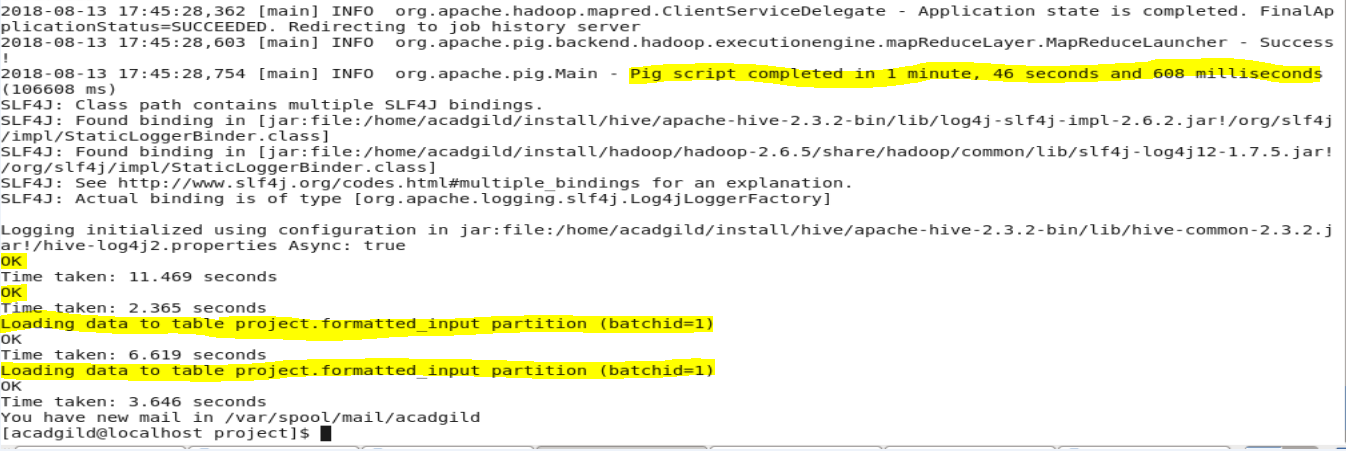
The script is successfully completed and 20 records corresponding to the web data are written into the formattedweb folder on local file system.



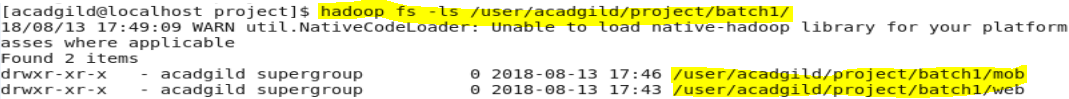
Once the pig is successfully complete, the hive script is picked up.

The table in hive is created and data is loaded into the hive table.

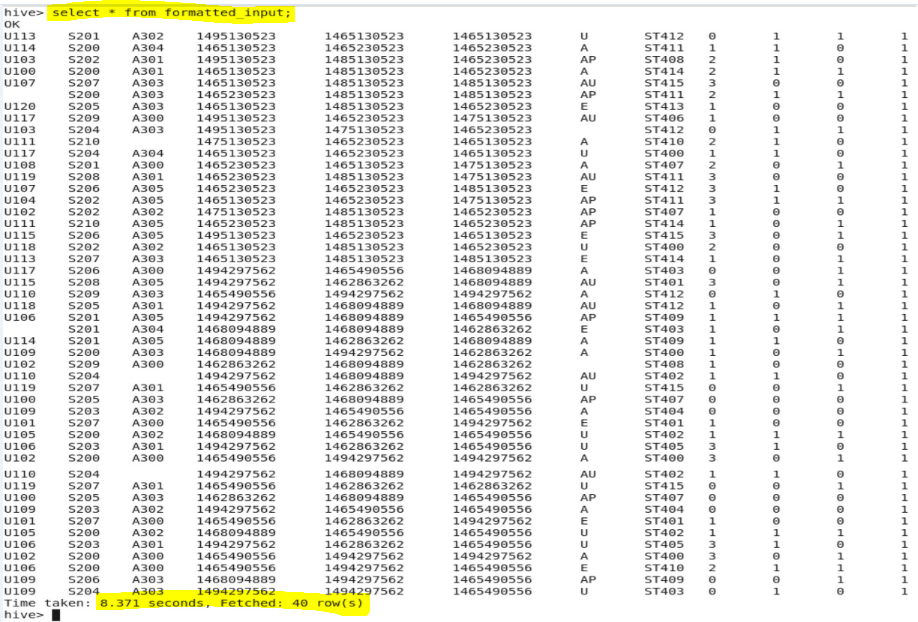
Also, the data from mob folder is also loaded into the same table.



And thus, the script is completely executed successfully. The folders mob and web are created in HDFS.



The table formatted\_input is created in hive and populated with 40 records. 20 from formattedweb and 20 from mob data.

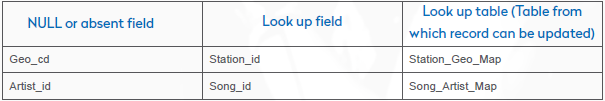


As we can see, the data is validated and formatted but still there are some invalid records in the data. This data will be further formatted in the data enrichment phase.

1. **Data Enrichment**

**5.1 Rules for data enrichment**

* If any of like or dislike is NULL or absent, consider it as 0.
* If fields like Geo\_cd and Artist\_id are NULL or absent, consult the lookup tables for fields Station\_id and Song\_id respectively to get the values of Geo\_cd and Artist\_id.
* If corresponding lookup entry is not found, consider that record to be invalid.



**5.2 Rules for Post Enrichment**

* Move all valid records in /hadoop/processing\_dir in HDFS and invalid records in Local File System at /usr/invalid directory.
* Maintain a copy of valid records in /usr/validated in Local File System. Run a cleaner everyday to clean validated files which are more than 7 days old.

**5.3 Data Filtering and Enrichment**

In the previous phase, the data is validated and formatted but still there are some invalid records in the data. This data will be further formatted in the data enrichment phase.

Data is present in hive in a table called *formatted\_input* and the lookup tables: *station\_geo\_map, subscribed\_users* and *song\_artist\_map* are present in HBase. For removing the null entries and transforming invalid records to valid ones, we have to perform some kind of join operations with the lookup tables and replace the null values accordingly.

For performing this, a hive table is created that will contain the table data from HBase. This can be done using the following hql file: *create\_hive\_hbase\_lookup.hql*

Using the earlier created database ‘project’.

*USE project;*

Creating 3 external tables for the three lookup tables present in HBase. These tables are creates using the HBaseStorageHandler. The corresponding table name and the column family are specified as table properties in the create table command.

*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'*

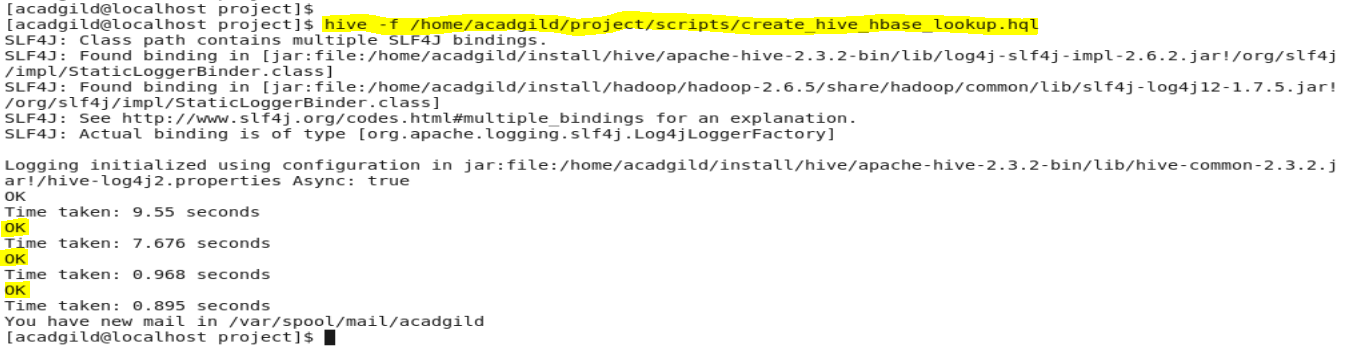
*with serdeproperties*

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

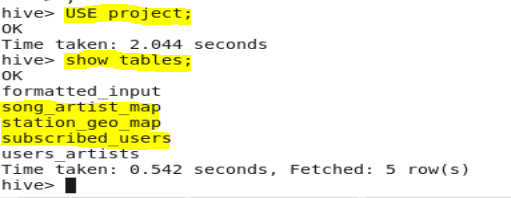
*tblproperties("hbase.table.name"="song-artist-map");*

After running this script, the tables get created in hive.

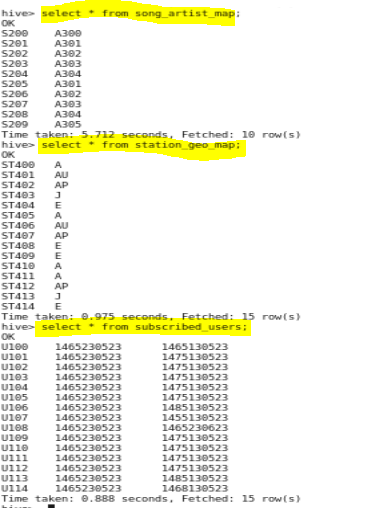
Command to run: *hive -f /home/acadgild/project/scripts/create\_hive\_hbase\_lookup.hql*



The tables can be seen in hive when a list operation is performed on the project database.



Also, in the above script, the table is only created and there is no mention of loading the data into the table. But since these are external tables, the data will be pointed to the content of the parent table in HBase. Hence we see populated tables when we query them.



Now that the lookup tables are created in hive as well, we can go ahead and perform data enrichment based on the data and the corresponding lookup.

This enrichment is done in the shell script: data\_enrichment.sh

Command to run: *sh /home/acadgild/project/scripts/data\_enrichment.sh*

*#!/bin/bash*

Extracting batch id and opening the log file from the designated logs folder.

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

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

Both the valid as well as invalid data has to be stored in local fole system according to the requirement. Hence 2 folders for valid and invalid data are created with the batch\_ids.

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

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

Entering logs into log file.

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

Running the hql file to do the required joins and remove the null entries using lookups.

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

Creating the vald and invalid directories if they are not already present.

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

*then*

*mkdir -p "$VALIDDIR"*

*fi*

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

*then*

*mkdir -p "$INVALIDDIR"*

*fi*

Entering logs into log file.

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

Extracting the records whose status is ‘pass’ and putting them into the valid dir. And those corresponding to the status ‘fail’ are put into invalid dir.

*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*

Entering logs into log file.

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

As part of Post Enrichment steps, data older than 7 days has to be cleaned up, so this is done using the below command.

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

Now, let’s have a closer look at the hql file which does the actual enrichment.

data\_enrichment.hql

Setting some properties to hive.

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

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

Using the already created database project;

*USE project;*

Creation of the final table with enriched data on which the analyses will be performed.

*CREATE TABLE IF NOT EXISTS enriched\_data*

*(*

*user\_id STRING,*

*song\_id STRING,*

*artist\_id STRING,*

*timestmp STRING,*

*start\_ts STRING,*

*end\_ts STRING,*

*geo\_cd STRING,*

*station\_id STRING,*

*song\_end\_type INT,*

*liked INT,*

*disliked INT*

*)*

*PARTITIONED BY*

*(batchid INT,*

*status STRING)*

*STORED AS ORC;*

Using multiple lookup tables performing joins and inserting the data both valid and invalid along with the status flag in the table just created.

*INSERT OVERWRITE TABLE enriched\_data*

*PARTITION (batchid,status)*

*SELECT*

*i.user\_id,*

*i.song\_i,,*

--According to data enrichment rules, if any artist\_id is null, then corresponding entry from lookup table is selected and will be populated.

*IF(i.artist\_id is NULL OR i.artist\_id='',sa.artist\_id,i.artist\_id) AS artist\_id,*

*i.timestmp,*

*i.start\_ts,*

*i.end\_ts,*

--According to data enrichment rules, if any geo\_cd is null, then corresponding entry from lookup table is selected and will be populated.

*IF(i.geo\_cd is NULL OR i.geo\_cd='',sg.geo\_cd,i.geo\_cd) AS geo\_cd,*

*i.station\_id,*

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

--According to data enrichment rules, if either liked or disliked is null, then it should be considered as 0.

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

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

*i.batchid,*

-- For deciding whether the record is valid or not, we have to validate the following conditions and if either of them is true, then the record status is ‘fail’

* If both like and dislike are 1, consider that record to be invalid.
* If any of the fields from User\_id, Song\_id, Timestamp, Start\_ts, End\_ts, Geo\_cd is NULL or absent, consider that record to be invalid.
* If corresponding lookup entry is not found, consider that record to be invalid.

*IF((i.liked=1 AND i.disliked=1)*

*OR i.user\_id IS NULL*

*OR i.song\_id IS NULL*

*OR i.timestmp IS NULL*

*OR i.start\_ts IS NULL*

*OR i.end\_ts IS NULL*

*OR i.user\_id=''*

*OR i.song\_id=''*

*OR i.timestmp=''*

*OR i.start\_ts=''*

*OR i.end\_ts=''*

*OR sg.geo\_cd=''*

*OR sg.geo\_cd IS NULL*

*OR sa.artist\_id IS NULL*

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

--Left Outer Join operation is performed on the lookup tables and the formatted\_input table.

*FROM formatted\_input i*

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

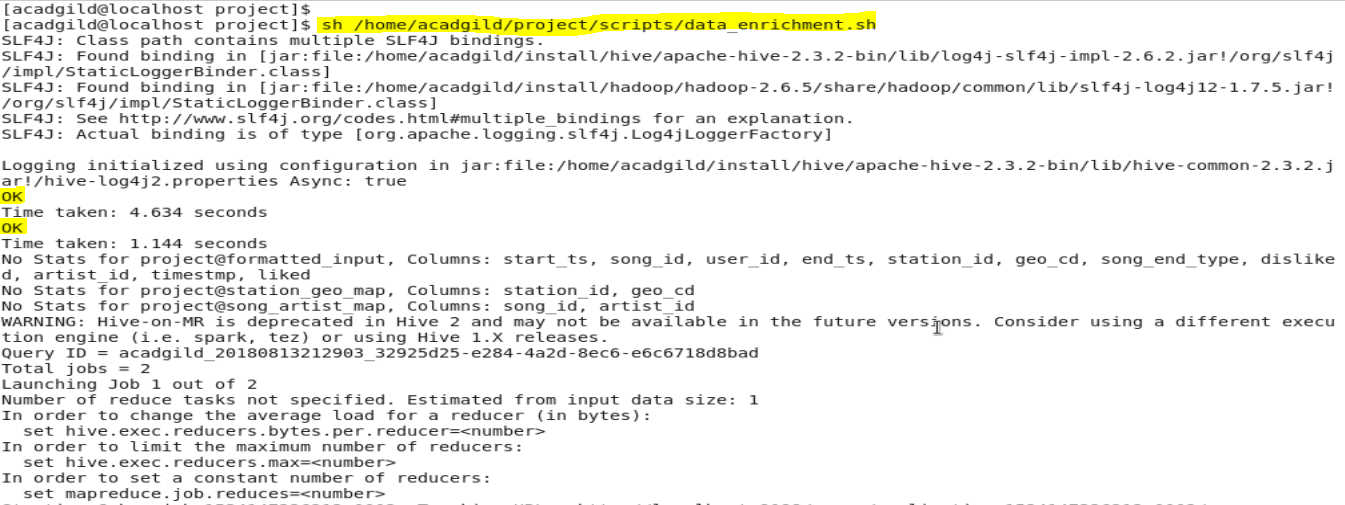
*LEFT OUTER JOIN song\_artist\_map sa ON i.song\_id = sa.song\_id*

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

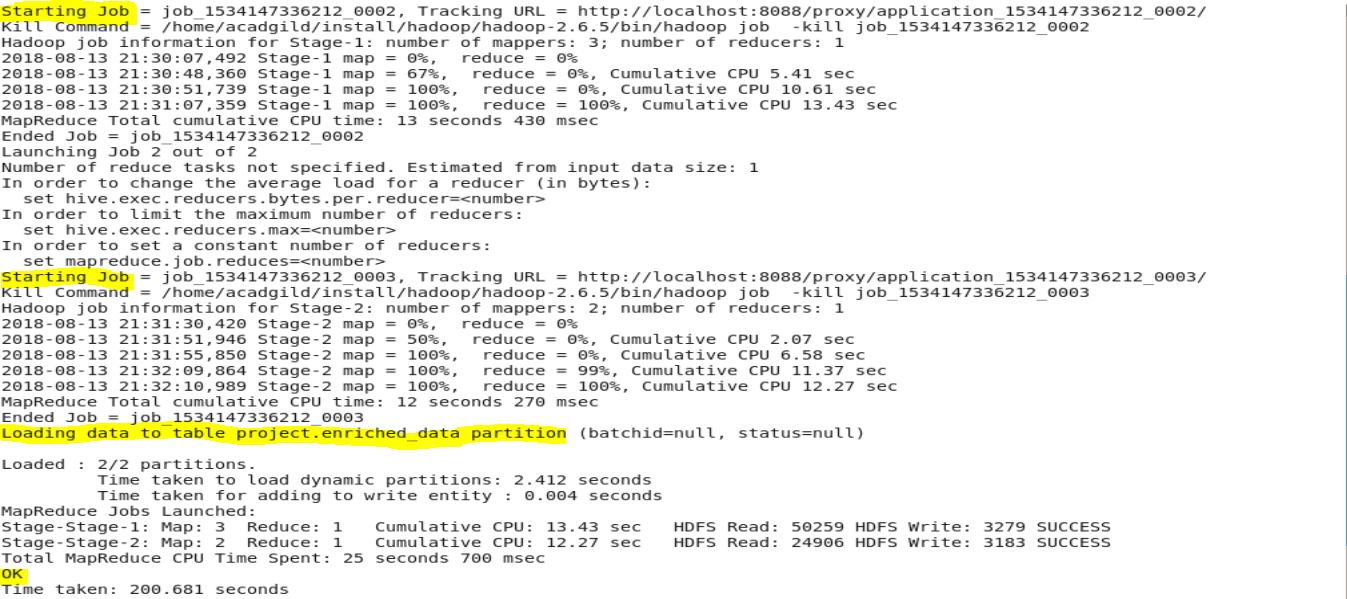
Since there are 2 left outer joins are to be performed, this is a heavy operation and It will take time and a good amount of RAM to finish this operation successfully.

On executing the dataenrichment shell script, below are the observations.

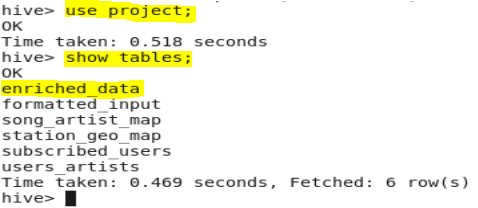
Command to run: *sh /home/acadgild/project/scripts/data\_enrichment.sh*



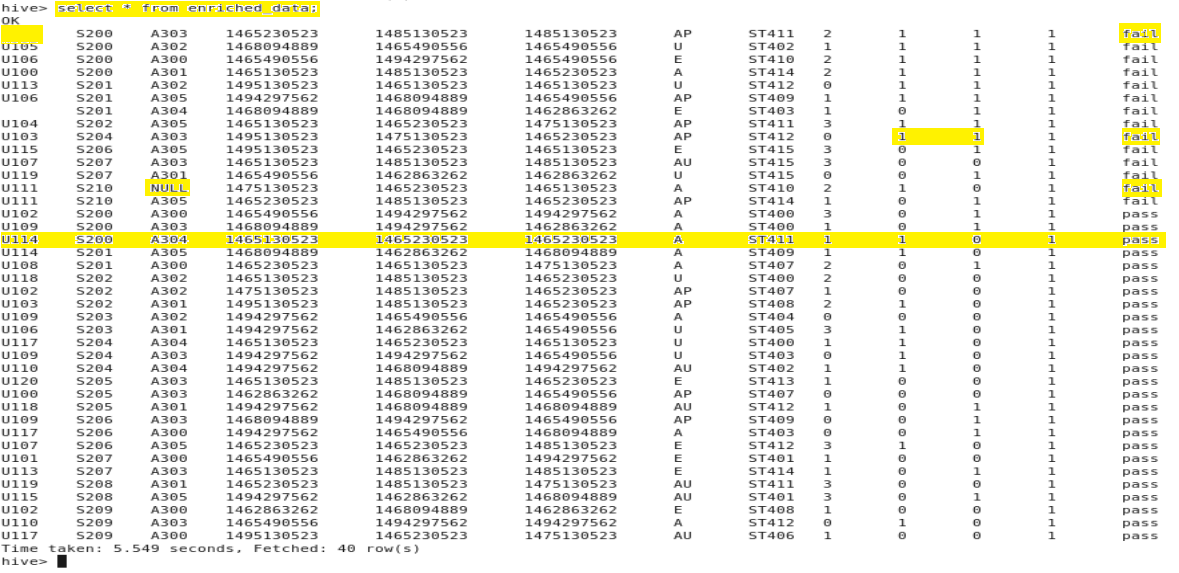
The shell script is run and the hql script is picked up. The database is changed to project and the table enriched\_data is created.



The map reduce job for the insert statement is successfully completed and data is loaded into the enriched\_data table.



In the above screenshot, it can be seen that the table enriched\_data is created in project database.

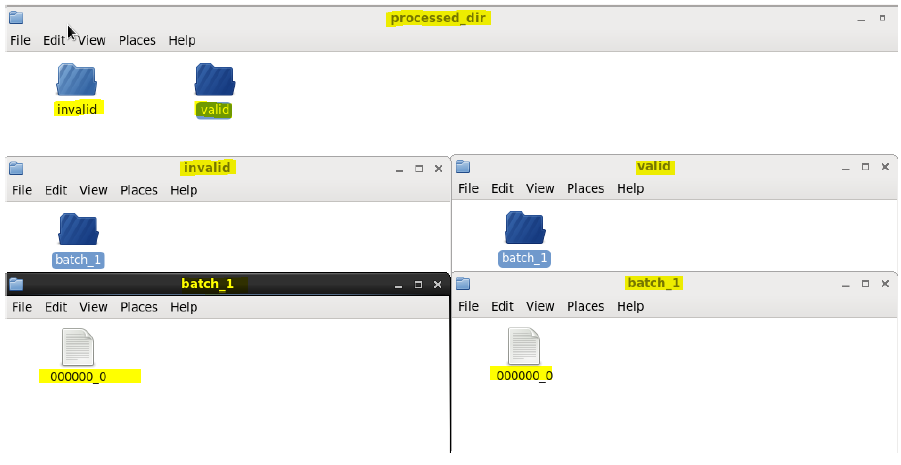


When the enriched\_data table is queried, we can see that the The status falg is added in the table and both valid as well as invalid records are present in the table.

For example,

1. The record for which user\_id is blank, is marked as ‘fail’ case.
2. The record for which artist\_id is null is marked as ‘fail’ case.
3. The record for which both liked and disliked are 1, is marked as ‘fail’ case.
4. The record where all the columns are present and valid, it is specified as ‘pass’ case.

As part of post Enrichment steps, the valid and invalid records are to be saved in a directory called processed\_dir.



As seen in the screenshot above, valid and invalid directories are created and files with the respective data are present in the folder with the batch id.

1. **Data Analysis**

**6.1 Challenges and Optimizations**

* LookUp tables are in NoSQL databases. Integrate them with the actual data flow.
* Try to make joins as less expensive as possible.
* Data Cleaning, Validation, Enrichment, Analysis and Post Analysis have to be automated. Try using schedulers.
* Appropriate logs have to be maintained to track the behavior and overcome failures in the pipeline.

**6.2 Data Analysis (Using Hive)**

The data analysis is done using the shell script. Data\_analysis.sh

Command to run: *sh /home/acadgild/project/scripts/data\_analysis.sh*

*#!/bin/bash*

Extracting the batchid and the location of the logfile to enter in the logs.

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

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

Entering logs into log file.

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

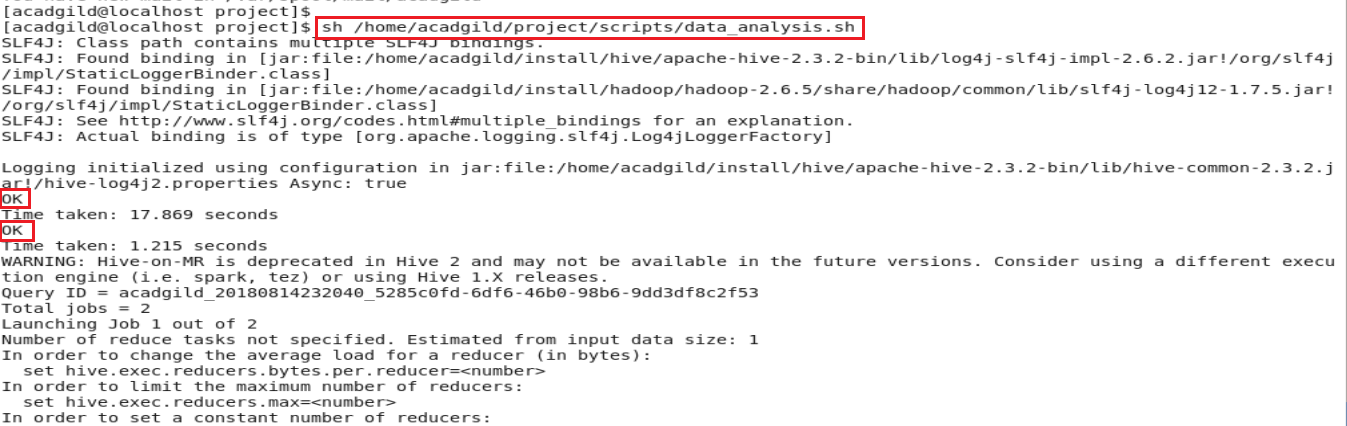
Command to execute the hql file in which the hive queries are written to solve the problem statements.

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

Once the hive script is run, the output data is transported onto MYSQL using the below script.

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

Below is the observation when the script is run.



Let’s have a closer look at the data\_analysis.hql file.

In this file, a table is created for every problem statement asked with the required columns and then the data is inserted using the appropriate query.

One condition that has to be considered in all cases is that the status of the record has to be ’pass’ meaning that only valid records are considered.

Using the existing database named project.

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

*USE project;*

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

A table names top\_10\_stations is created and is partitioned on batchid.

*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;*

The count of distinct user\_id and distinct song\_id are considered who have liked the song.

*INSERT OVERWRITE TABLE top\_10\_stations*

*PARTITION(batchid=${hiveconf:batchid})*

*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=${hiveconf:batchid}*

*AND liked=1*

*GROUP BY station\_id*

*ORDER BY total\_distinct\_songs\_played DESC*

*LIMIT 10;*

When the resultant table is queried, we can see the results. Below are the top 10 station\_ids where maximum number of songs were played, which were liked by unique users.



**6.2.2 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.**

A table named users\_behaviour is created with batch\_id as the partition\_id.

*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;*

*INSERT OVERWRITE TABLE users\_behaviour*

*PARTITION(batchid=${hiveconf:batchid})*

*SELECT*

Categorizing user\_ids as subscribed or unsubscribed based on its presence in subscribed\_users table(su) and whether or not their subscription\_end\_date is less than or greater that the timestamp of the song played by the user.

*CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestmp 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.timestmp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED'*

*END AS user\_type,*

Calculating the duration of the song using the start\_ts and end\_ts of that song.

*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*

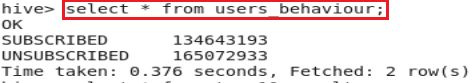
*WHERE ed.status='pass'*

*AND ed.batchid=${hiveconf:batchid}*

Grouping the data as subscribed and unsubscribed users.

*GROUP BY CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestmp 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.timestmp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN 'SUBSCRIBED' END;*



As seen in the screenshot above, on querying the resultant table, we see the duration of songs played is displayed grouped by subscribed and unsubscribed users.

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

A table named connected\_artists is created with batch\_id as the partition\_id.

*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;*

Determining the artist\_id and distinct user\_ids after performing inner join between 2 datasets explained below.

*INSERT OVERWRITE TABLE connected\_artists*

*PARTITION(batchid=${hiveconf:batchid})*

*SELECT*

*ua.artist\_id,*

*COUNT(DISTINCT ua.user\_id) AS user\_count*

*FROM*

Using explode to separate the elements of array into multiple rows. Lateral View first applies the explode function to each row of base table and then joins resulting output rows to the input rows to form a virtual table having the supplied table alias.

*(*

*SELECT user\_id, artist\_id FROM users\_artists*

*LATERAL VIEW explode(artists\_array) artists AS artist\_id*

*) ua*

*INNER JOIN*

Selecting the artist\_id, song\_id and user\_id from the valid data in enriched\_data table.

*(*

*SELECT artist\_id, song\_id, user\_id*

*FROM enriched\_data*

*WHERE status='pass'*

*AND batchid=${hiveconf:batchid}*

*) ed*

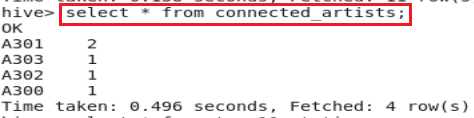
*ON ua.artist\_id=ed.artist\_id*

*AND ua.user\_id=ed.user\_id*

*GROUP BY ua.artist\_id*

*ORDER BY user\_count DESC*

*LIMIT 10;*



As seen in the screenshot above, on querying the resultant table, we see the connected artist\_ids whose songs are most listened by the unique users who follow them.

**6.2.4 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.**

A table named top\_10\_royalty\_songs is created with batch\_id as the partition\_id.

*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;*

Extracting the song\_id, its calculated duration for the songs for whom liked value is 1 or the song\_end\_type is 0 indicating the song was completely played.

*INSERT OVERWRITE TABLE top\_10\_royalty\_songs*

*PARTITION(batchid=${hiveconf:batchid})*

*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'*

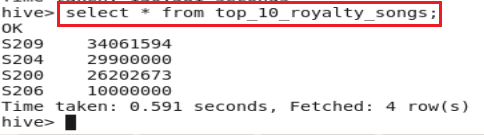
*AND batchid=${hiveconf:batchid}*

*AND (liked=1 OR song\_end\_type=0)*

*GROUP BY song\_id*

*ORDER BY duration DESC*

*LIMIT 10;*



As seen in the screenshot above, on querying the resultant table, the song\_ids that have generated the maximum revenue are displayed.

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

A table named top\_10\_unsubscribed\_users is created with batch\_id as the partition\_id.

*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;*

The user\_id who is unscubscribed based on it’s existence in subscribed\_users table and its subscription end date in by performing a left outer join on the subscribed\_users table.

*INSERT OVERWRITE TABLE top\_10\_unsubscribed\_users*

*PARTITION(batchid=${hiveconf:batchid})*

*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=${hiveconf:batchid}*

*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;*



As seen in the screenshot above, on querying the resultant table, the user\_ids and the duration of songs played by him are displayed. These are the top unsubscribed users in the data.

**6.3 Data Analysis (Using Spark submit)**

The same queries can be run as a spark job by integrating hive and spark. Below is the code for the same.

*package com.acadgild.sparkhive*

*import org.apache.spark.sql.SparkSession*

*object SparkHiveInt {*

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

*val sparkSession = SparkSession.builder.master("local")*

*.appName("spark session example")*

*.config("spark.sql.warehouse.dir", "/user/hive/warehouse")*

*.config("hive.metastore.uris", "thrift://localhost:9083")*

*.enableHiveSupport().getOrCreate()*

*val batchId = args(0)*

*val set\_properties = sparkSession.sqlContext.sql("set hive.auto.convert.join=false")*

*val use\_project\_database = sparkSession.sqlContext.sql("USE project")*

*val create\_hive\_table\_top\_10\_stations = sparkSession.sqlContext.sql("CREATE TABLE IF NOT EXISTS project.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")*

*val insert\_into\_top\_10\_stations = sparkSession.sqlContext.sql("INSERT OVERWRITE TABLE project.top\_10\_stations" +*

*s" PARTITION (batchid=$batchId)" +*

*" SELECT" +*

*" station\_id," +*

*" COUNT(DISTINCT song\_id) AS total\_distinct\_songs\_played," +*

*" COUNT(DISTINCT user\_id) AS distinct\_user\_count" +*

*" FROM project.enriched\_data" +*

*" WHERE status='pass'" +*

*s" AND (batchid=$batchId)" +*

*" AND liked=1" +*

*" GROUP BY station\_id" +*

*" ORDER BY total\_distinct\_songs\_played DESC" +*

*" LIMIT 10")*

*val create\_tale\_users\_behaviour = sparkSession.sqlContext.sql("CREATE TABLE IF NOT EXISTS project.users\_behaviour" +*

*" (user\_type STRING," +*

*" duration INT)" +*

*" PARTITIONED BY (batchid INT)" +*

*" ROW FORMAT DELIMITED" +*

*" FIELDS TERMINATED BY ','" +*

*" STORED AS TEXTFILE;")*

*val insert\_into\_users\_behaviour = sparkSession.sqlContext.sql("INSERT OVERWRITE TABLE project.users\_behaviour" +*

*s" PARTITION(batchid=$batchId)" +*

*" SELECT" +*

*" CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestmp 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.timestmp 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'" +*

*s" AND ed.batchid=$batchId" +*

*" GROUP BY CASE WHEN (su.user\_id IS NULL OR CAST(ed.timestmp 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.timestmp AS DECIMAL(20,0)) <= CAST(su.subscn\_end\_dt AS DECIMAL(20,0))) THEN" + " 'SUBSCRIBED' END;")*

*val create\_table\_connected\_artists = sparkSession.sqlContext.sql("CREATE TABLE IF NOT EXISTS project.connected\_artists" +*

*" (artist\_id STRING," +*

*" user\_count INT)" +*

*" PARTITIONED BY (batchid INT)" +*

*" ROW FORMAT DELIMITED" +*

*" FIELDS TERMINATED BY ','" +*

*" STORED AS TEXTFILE;")*

*val insert\_into\_connected\_artists = sparkSession.sqlContext.sql("INSERT OVERWRITE TABLE project.connected\_artists" +*

*s" PARTITION(batchid=$batchId)" +*

*" 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'" +*

*s" AND batchid=$batchId) 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;")*

*val create\_table\_royalty\_songs = sparkSession.sqlContext.sql("CREATE TABLE IF NOT EXISTS project.top\_10\_royalty\_songs" +*

*" (song\_id STRING," +*

*" duration INT)" +*

*" PARTITIONED BY (batchid INT)" +*

*" ROW FORMAT DELIMITED" +*

*" FIELDS TERMINATED BY ','" +*

*" STORED AS TEXTFILE;")*

*val insert\_into\_royalty\_songs = sparkSession.sqlContext.sql("INSERT OVERWRITE TABLE project.top\_10\_royalty\_songs" +*

*s" PARTITION(batchid=$batchId)" +*

*" 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'" +*

*s" AND batchid=$batchId" +*

*" AND (liked=1 OR song\_end\_type=0)" +*

*" GROUP BY song\_id" +*

*" ORDER BY duration DESC" +*

*" LIMIT 10;")*

*val unsubscribed\_users = sparkSession.sqlContext.sql("CREATE TABLE IF NOT EXISTS project.top\_10\_unsubscribed\_users" +*

*" (user\_id STRING," +*

*" duration INT)" +*

*" PARTITIONED BY (batchid INT)" +*

*" ROW FORMAT DELIMITED" +*

*" FIELDS TERMINATED BY ','" +*

*" STORED AS TEXTFILE;")*

*val insert\_into\_unsubscribed\_users = sparkSession.sqlContext.sql("INSERT OVERWRITE TABLE project.top\_10\_unsubscribed\_users" +*

*s" PARTITION(batchid=$batchId)" +*

*" 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'" +*

*s" AND ed.batchid=$batchId" +*

*" AND (su.user\_id IS NULL OR (CAST(ed.timestmp AS DECIMAL(20,0)) > CAST(su.subscn\_end\_dt AS DECIMAL(20,0))))" +*

*" GROUP BY ed.user\_id" +*

*" ORDER BY duration DESC" +*

*" LIMIT 10;")*

*val show\_tables = sparkSession.sqlContext.sql("show tables")*

*show\_tables.show(10, false)*

*sparkSession.sqlContext.sql("SELECT \* from project.top\_10\_stations").show()*

*sparkSession.sqlContext.sql("SELECT \* from project.users\_behaviour").show()*

*sparkSession.sqlContext.sql("SELECT \* from project.connected\_artists").show()*

*sparkSession.sqlContext.sql("SELECT \* from project.top\_10\_royalty\_songs").show()*

*sparkSession.sqlContext.sql("SELECT \* from project.top\_10\_unsubscribed\_users").show()*

*}*

*}*

The jar file is attached along with the project folder.

Command to run the jar : *spark-submit –class com.acadgild.sparkhive.SparkHiveInt –master local –deploy-mode client SparkHive.jar 1*

1. **Post Analysis**

Once the analysis is complete, multiple actions can be taken place later on. It includes:

1. Moving result of analysis to the RDMS for data storage and quick retrieval.

2. Form visualizations on the top of analyzed data.

3. Send data to data science or machine learning pipelines for further forecast.

**7.1 Data export to MySQL using Sqoop**

Using the bash file shown below, data\_export.sh we are going to export the data from the hive tables into mysql using Sqoop export.

*#!/bin/bash*

Extracting batch id and opening the log file from the designated logs folder.

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

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

Entering logs into log file.

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

Running the mysql script to create the schema for all the 5 resultant tables.

*mysql –username root –password Root@123 < /home/acadgild/project/scripts/create\_schema.sql*

Entering logs into log file.

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

Sqoop export commands to connect to both hive and mysql and export one table data from hive into mysql.

*sqoop export --connect jdbc:mysql://localhost/project --username root --password Root@123 --table top\_10\_stations –columns station\_id,total\_distinct\_songs\_played,distinct\_user\_count --export-dir /user/hive/warehouse/project.db/top\_10\_stations/batchid=$batchid --input-fields-terminated-by ',' –lines-terminated-by ‘\n’;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password Root@123 --table users\_behaviour –columns user\_type,duration --export-dir /user/hive/warehouse/project.db/users\_behaviour /batchid=$batchid --input-fields-terminated-by ',' –lines-terminated-by ‘\n’;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password Root@123 --table connected\_artists –columns artist\_id,user\_count --export-dir /user/hive/warehouse/project.db/connected\_artists /batchid=$batchid --input-fields-terminated-by ',' –lines-terminated-by ‘\n’;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password Root@123 --table top\_10\_royalty\_songs –columns song\_id,duration --export-dir /user/hive/warehouse/project.db/top\_10\_royalty\_songs /batchid=$batchid --input-fields-terminated-by ',' –lines-terminated-by ‘\n’;*

*sqoop export --connect jdbc:mysql://localhost/project --username root --password Root@123 --table top\_10\_unsubscribed\_users –columns user\_id,duration --export-dir /user/hive/warehouse/project.db/ top\_10\_unsubscribed\_users /batchid=$batchid --input-fields-terminated-by ',' –lines-terminated-by ‘\n’;*

Lets have a closer look into the mysql schema creation file.

create\_schema.sql

Creating a database named ‘project’ and using the same so that all the tables will be created in that database.

*CREATE DATABASE IF NOT EXISTS project;*

*USE project;*

Creating one table for each of the problem statements.

*CREATE TABLE IF NOT EXISTS top\_10\_stations*

*( station\_id VARCHAR(50),*

*total\_distinct\_songs\_played INT,*

*distinct\_user\_count INT);*

*CREATE TABLE IF NOT EXISTS users\_behaviour*

*(user\_type VARCHAR(50),*

*duration BIGINT);*

*CREATE TABLE IF NOT EXISTS connected\_artists*

*(artist\_id VARCHAR(50),*

*user\_count INT);*

*CREATE TABLE IF NOT EXISTS top\_10\_royalty\_songs*

*(song\_id VARCHAR(50),*

*duration BIGINT);*

*CREATE TABLE IF NOT EXISTS top\_10\_unsubscribed\_users*

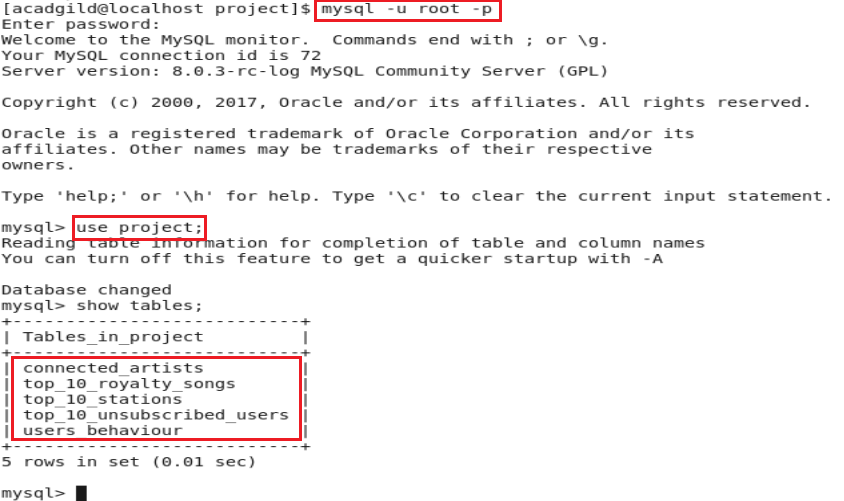
*(user\_id VARCHAR(50),*

*duration BIGINT);*

Committing all the table creations.

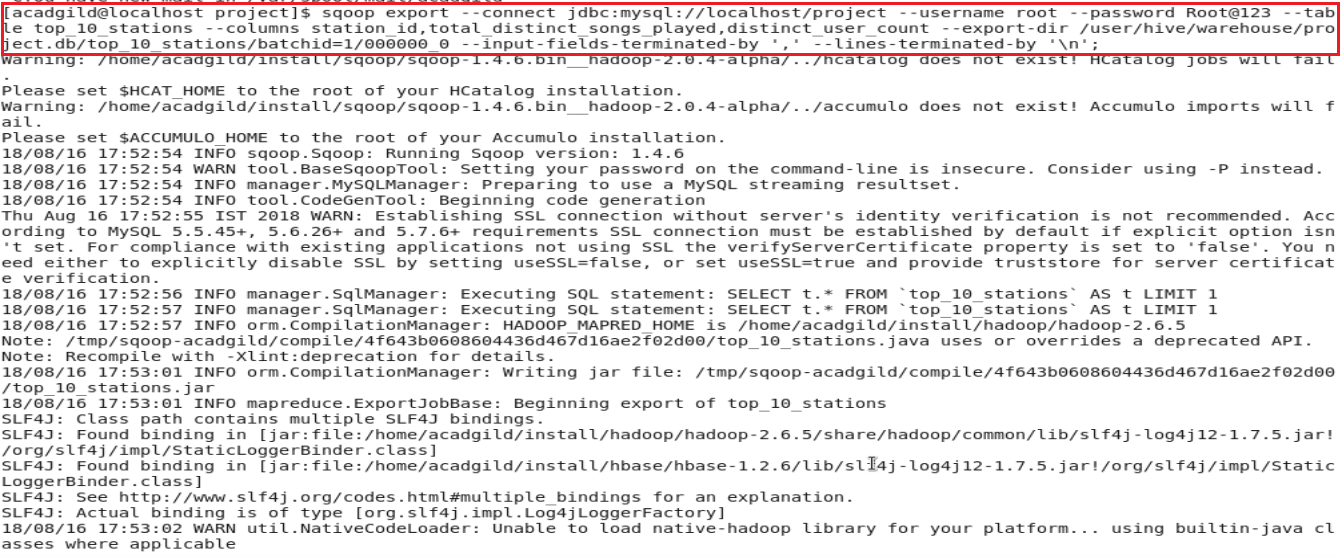
*commit;*

Once the shell script is run, the sql script is picked up and the tables are created in MySQL.

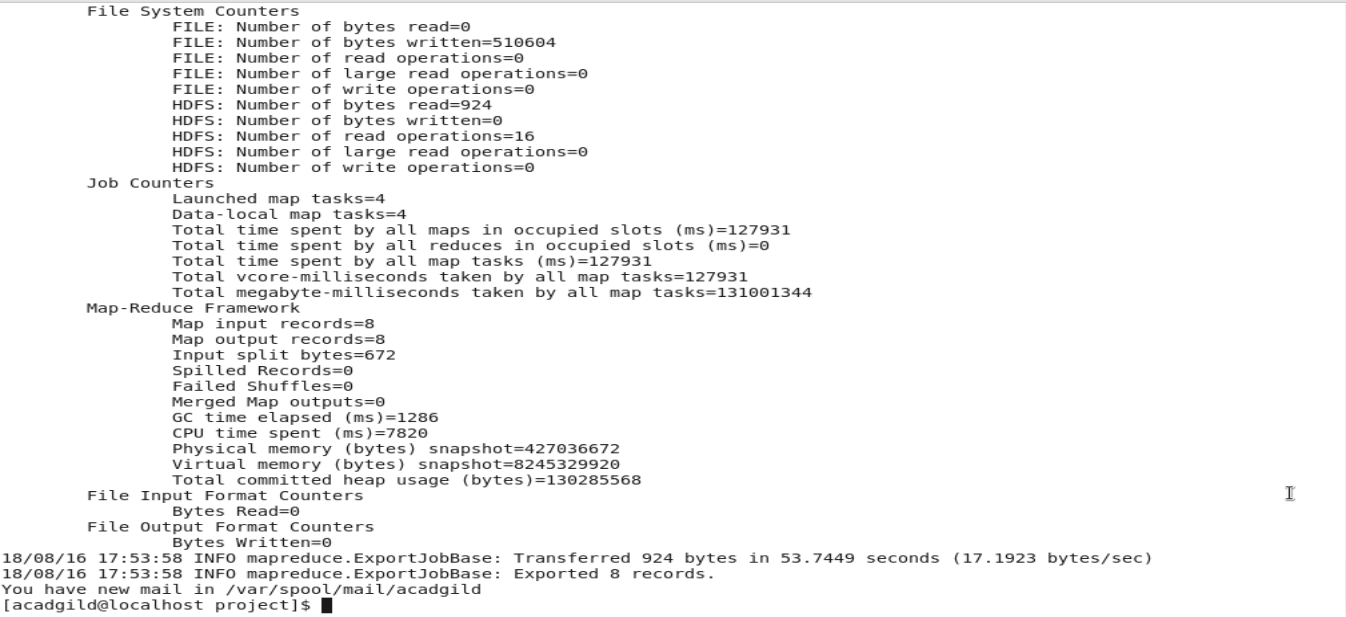


Upon logging into MySQL, we can see that the 5 resultant tables one for each problem statement are created in MySQL.

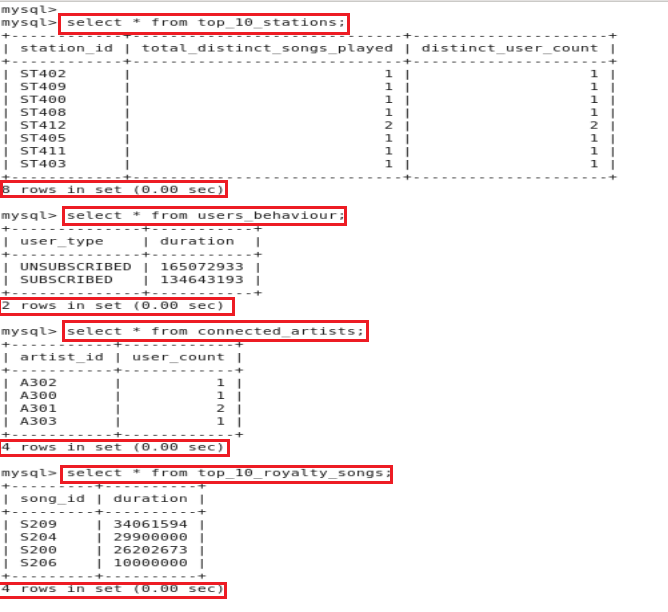
Now we can see the data exported successfully into the MYSQL Database for all the 5 queries.



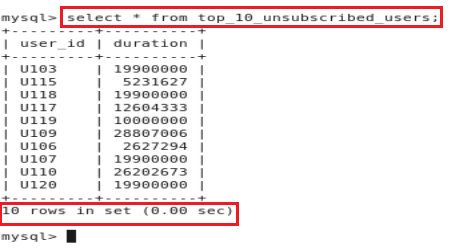
The sqoop export command exported the tables from the hive and it stored in the MySQL. The below screen shot show the successful Sqoop export from hive to MySQL. The data stored in the MySQL is shown in the successive screen shots.



The data base project had been exported from the hive and the below screen shot shows the data base presence, output from top\_10\_stations,users\_behaviour, connected\_artists and top\_10\_royalty\_songs shown below,



Output from top\_10\_unsubscribed\_users:



**7.2 Job Scheduling**

Now after exporting data into MySQL batchid will be incremented to additional 1 means one batch of data operations is successfully completed and new batch of data will be loaded for the analysis after every 3 hours.

As part of data\_analysis.sh once the analysis and the post analysis export is done, the batch\_id is incremented.

Entering logs into log file.

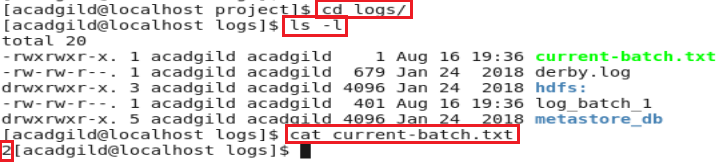
*echo "Incrementing batchid..." >> $LOGFILE*

Finally, since the processing and analysis of one batch is complete, the batchid is incremented for the next batch and it is placed onto the current-batch.txt file from which the next batch will pick up its id.

*batchid=`expr $batchid + 1`*

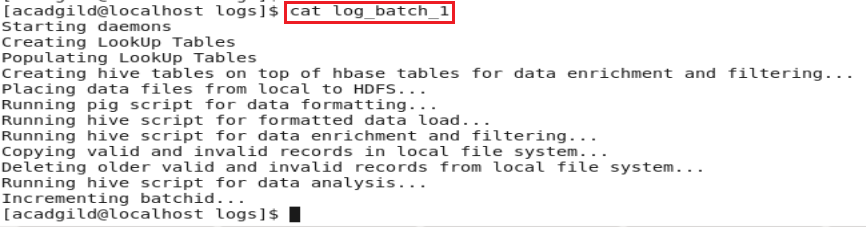
*echo -n $batchid > /home/acadgild/project/logs/current-batch.txt*

We can check logs to track the behavior of the operations we have done on the data and overcome failures in the pipeline and we can see the batchid incremented value in current-batch.txt.



Since the batchid is incremented, the value in current-batch.txt is 2.

The log file captured all the data and steps we performed so far,



Wrapping all the scripts inside the single script file and scheduling this file to run at the periodic interval of every 3 hours.

Wrapper.sh

*#!/bin/bash*

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

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

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

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

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

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

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

The wrapper.sh will be running for every 3 hours as per the job scheduling done below, as per the above order the wrapper.sh will run the scripts.

Creating Crontab to schedule the wrapper.sh script to run for every 3 hour interval.

Command to create a crontab : *crontab –e*

A file is opened where in the command to schedule the script can be given.



The crontab job scheduler will run the wrappr.sh every 3 hours and for every 3 hours we will get incremental batch ID’s. Hence, as per the request this job scheduling has been done.



**Problems faced during project installation and how it resolved**

1. The hive tables cannot be created when we run the data\_enrichment\_filtering\_schema.sh

Error: Exception in thread "main" java.lang.RuntimeException: Hive metastore database is not initialized.

Please use schematool (e.g. ./schematool -initSchema -dbType ...) to create the schema. If needed, don't forget to include the option to auto-create the underlying database in your JDBC connection string (e.g. ?createDatabaseIfNotExist=true for mysql)

Solution:

The metastore already existed, but not in complete form.

Before you run hive for the first time, run the below commands where the metastore\_db is located,

* schematool -initSchema -dbType derby
* mv metastore\_db metastore\_db.tmp
* schematool -initSchema -dbType derby

**Highlights of the Project**

* No join of query is used while analysis. Data is already enriched with new fields and using broadcast maps on Lookup tables so as to avoid any join.
* We used full automated bash scripts from start to end.

**Conclusion**

The requirement of the leading music catering company has successfully been implemented using spark and other hadoop technologies in an efficient way. The company can now easily track the behavior of users, classify users, calculate royalties associated with the song and make appropriate business strategies.