**Music Data Analysis**

A leading music-catering company is planning to analyse large amount of data received from varieties of sources, namely mobile app and website to track the behaviour 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.

**Fields present in the data files**

Data files contain below fields.

Column Name/Field Name Column Description/Field Description

User\_id Unique identifier of every user

Song\_id Unique identifier of every song

Artist\_id Unique identifier of the lead artist of the song

Timestamp Timestamp when the record was generated

Start\_ts Start timestamp when the song started to play

End\_ts End timestamp when the song was stopped

Geo\_cd Can be 'A' for USA region,

'AP' for asia pacific region,

'J' for Japan region,

'E' for europe and

'AU' for australia region

Station\_id Unique identifier of the station from where the song was played Song\_end\_type How the song was terminated.

0 means completed successfully

1 means song was skipped

2 means song was paused

3 means other type of failure like device issue, network error etc.

Like 0 means song was not liked

1 means song was liked

Dislike 0 means song was not disliked

1 means song was disliked

**LookUp Tables**

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

**Table Name Description**

Station\_Geo\_Map Contains mapping of a geo\_cd with station\_id

Subscribed\_Users Contains user\_id, subscription\_start\_date and subscription\_end\_date.

Contains details only for subscribed users

Song\_Artist\_Map Contains mapping of song\_id with artist\_id alongwith royalty associated with each play of the song.

User\_Artist\_Map Contains an array of artist\_id(s) followed by a user\_id

You tube video for xml data loading:

<https://www.youtube.com/watch?v=IH-wsq5aF8M>

**DATASET:**

1. Data coming from web applications reside in /data/web and has xml format.

2. Data coming from mobile applications reside in /data/mob and has csv format.

3. Data present in lookup directory should be used in HBase.

Below is the link for same.

<https://drive.google.com/drive/folders/0B_P3pWagdIrrMjJGVlNsSUEtbG8?usp=sharing>

**Data Enrichment**

Rules for data enrichment

1. If any of like or dislike is NULL or absent, consider it as 0.

2. 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.

3. If corresponding lookup entry is not found, consider that record to be invalid.

NULL or absent field Look up field Look up table (Table from which record can be updated)

Geo\_cd Station\_id Station\_Geo\_Map

Artist\_id Song\_id Song\_Artist\_Map

**Data Analysis (SHOULD BE IMPLEMETED IN SPARK)**

It is not only the data which is important, rather it is the insight it can be used to generate important. Once we have made the data ready for analysis, we have to perform below analysis on a daily basis.

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

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.

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

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.

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

# **Solution of Music Data Analysis:**

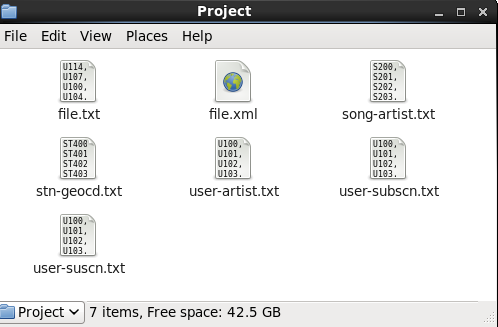
**In-scope :**

Consideration while doing data enrichment:

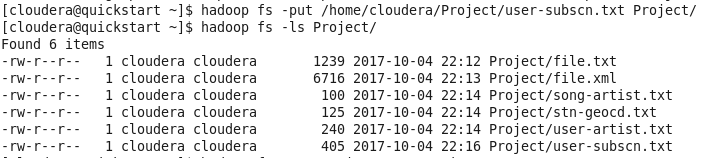
* All timestamps must have the format of a long integer for analysis..
* If both *like* and *dislike* are 1, consider that record to be invalid.
* If any of the fields from User\_id, Song\_id, Timestamp, Start\_ts, End\_ts, Geo\_cd is NULL or absent, consider that record to be invalid.
* If Song\_end\_type is NULL or absent, treat it to be 3
* 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.

# **Data Loading:**

**Step 1: Download all file from specified path to Project folder**



**Step 2: Transfer all file to HDFS under Project folder for further processing.**



**Loading file.xml file from HDFS**

spark-shell --packages com.databricks:spark-xml\_2.10:0.4.1

import org.apache.spark.sql.SQLContext

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

**Create a data frame from above RDD**

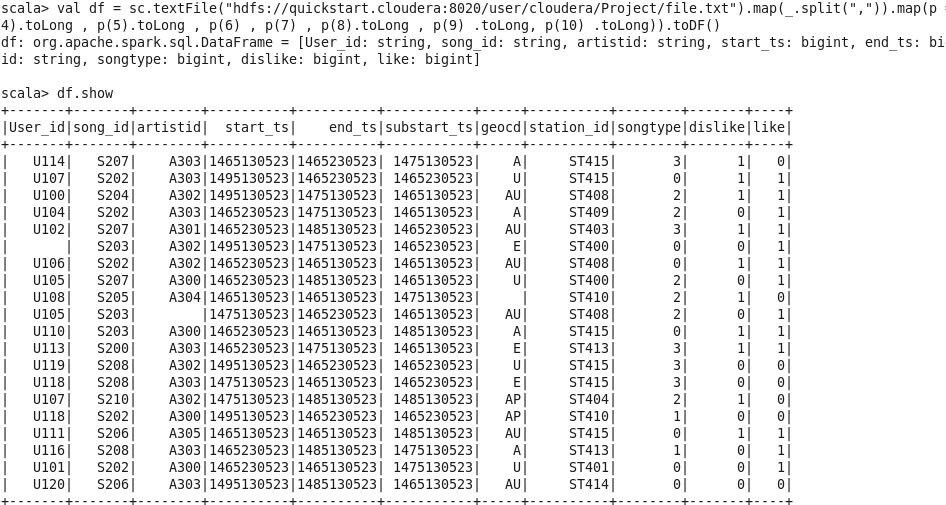
XML Loader

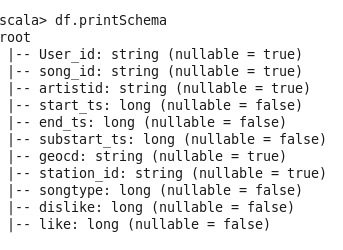
val df = sqlContext.read.format("com.databricks.spark.xml").option("rowTag","record").load("hdfs://quickstart.cloudera:8020/user/cloudera/Project/file.xml")

or

case class musicplay(User\_id: String , song\_id: String, artistid: String, start\_ts: Long, end\_ts: Long , substart\_ts: Long, geocd: String, station\_id: String , songtype: Long, dislike: Long, like: Long)

val df = sc.textFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/file.txt").map(\_.split(",")).map(p => musicplay(p(0) , p(1) , p(2) , p(3).toLong , p(4).toLong , p(5).toLong , p(6) , p(7) , p(8).toLong , p(9) .toLong, p(10) .toLong)).toDF()





**Create a temp table from data frame**

df.registerTempTable("musicrecords");

val main = sqlContext.sql("SELECT \* FROM musicrecords")

**Execute below four import command**

**import sqlContext.implicits.\_**

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

**import org.apache.spark.sql.DataFrame;**

**import org.apache.spark.sql.SQLContext;**

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

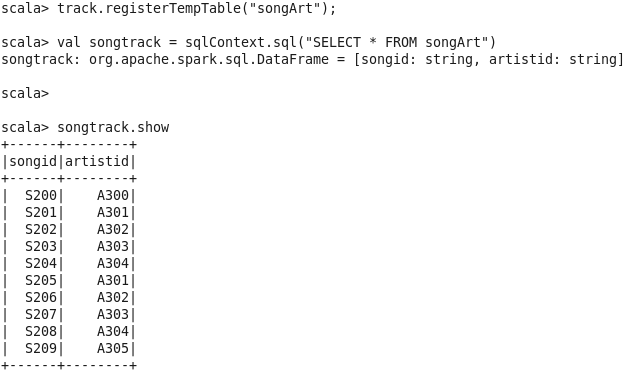
**Load table song-artist.txt**

case class SongArt (songid: String , artistid: String)

val track = sc.textFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/song-artist.txt").map(\_.split(",")).map(p => SongArt(p(0), p(1))).toDF()

track.registerTempTable("songArt");

val songtrack = sqlContext.sql("SELECT \* FROM songArt")



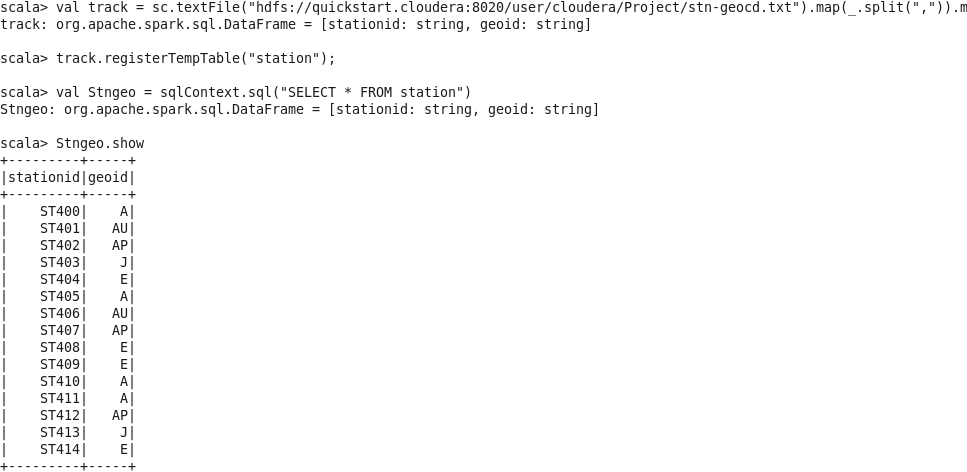
**Load table stn-geocd.txt**

case class Station (stationid: String , geoid: String)

val track = sc.textFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/**stn-geocd**.txt").map(\_.split(",")).map(p => Station(p(0), p(1))).toDF()

track.registerTempTable("station");

val Stngeo = sqlContext.sql("SELECT \* FROM station")



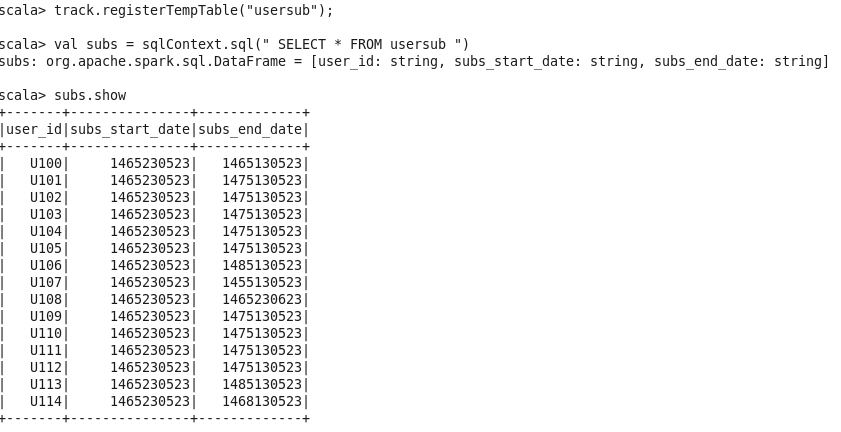
**Load table user-subcn.txt**

case class Subscription(user\_id: String , subs\_start\_date: String, subs\_end\_date: String)

val track = sc.textFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/**user-subscn**.txt").map(\_.split(",")).map(p => Subscription(p(0), p(1),p(2))).toDF()

track.registerTempTable("usersub");

val subs = sqlContext.sql(" SELECT \* FROM usersub ")



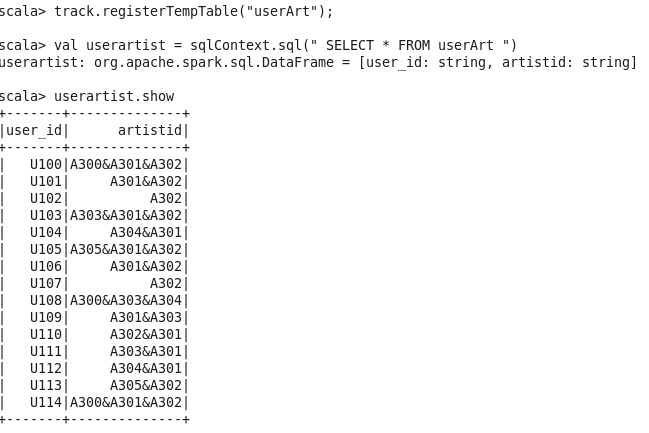
**Load table user-artist.txt**

case class userartist(user\_id: String , artistid: ARRAY<STRING>)

val track = sc.textFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/**user-artist**.txt").map(\_.split(",")).map(p => userartist(p(0), p(1))).toDF()

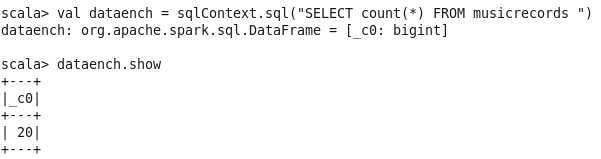
track.registerTempTable("userArt");

val userartist = sqlContext.sql(" SELECT \* FROM userArt ")



# Data Enrichment

val dataench = sqlContext.sql("SELECT count(\*) FROM musicrecords ")

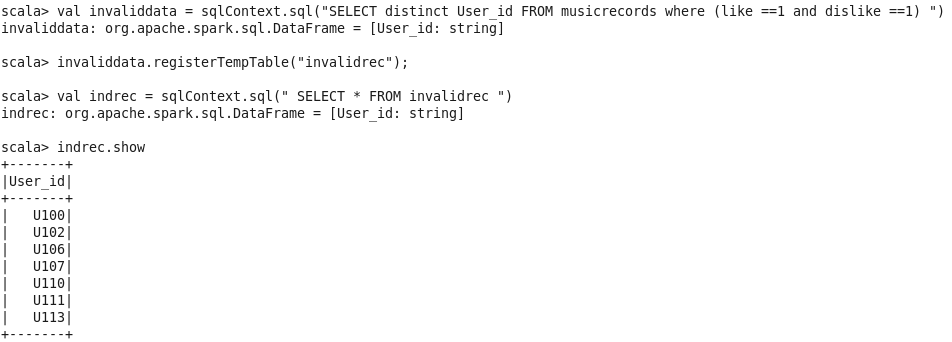


*If both like and dislike are 1, consider that record to be invalid.*

val invaliddata = sqlContext.sql("SELECT Distinct User\_id FROM musicrecords where like ==1 and dislike ==1 ")

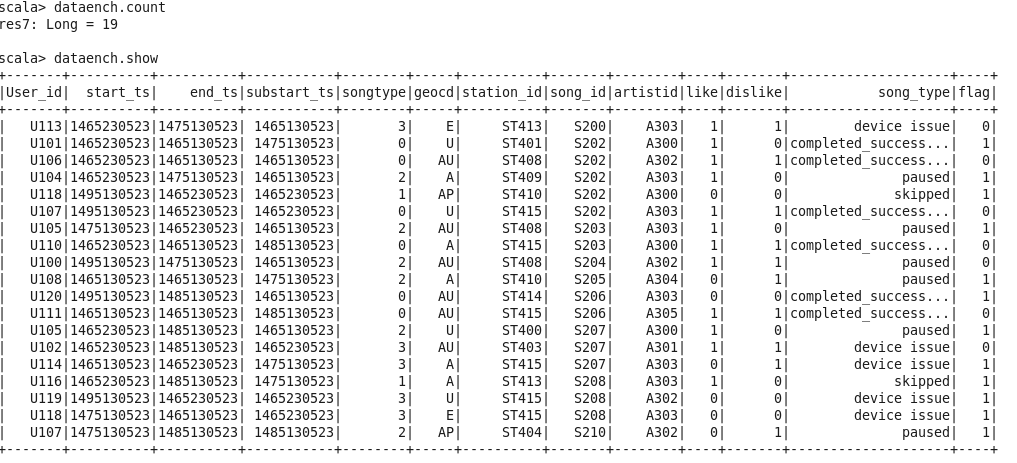
invaliddata.registerTempTable("invalidrec");

val indrec = sqlContext.sql(" SELECT \* FROM invalidrec ")



**Filter valid record based on data enrichment rules**

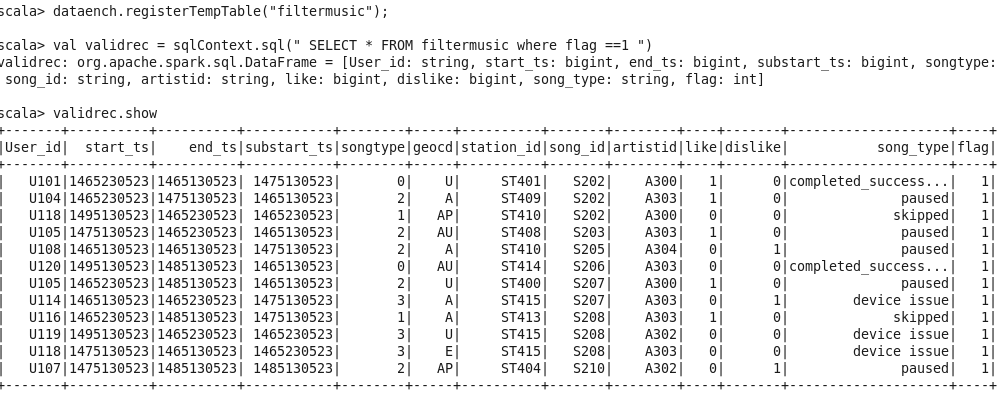
val dataench= sqlContext.sql("SELECT User\_id, start\_ts, end\_ts, substart\_ts, songtype, case when (M.geocd is null or M.geocd ='') then S.geoid else M.geocd end geocd, M.station\_id, M.song\_id, case when (M.artistid is null or M.artistid='') then A.artistid else M.artistid end artistid, case when like='' or like is null then 0 else like end like, case when dislike='' or dislike is null then 0 else dislike end dislike, case when songtype ==0 then 'completed\_successfully' when songtype ==1 then 'skipped' when songtype == 2 then 'paused' when songtype == 3 then 'device issue' end song\_type , case when like ==1 and dislike ==1 then 0 else 1 end flag FROM musicrecords M left join station S on S.stationid = M.station\_id left join songArt A on A.songid =M.song\_id where ((M.User\_id!='' or M.User\_id is null) and M.station\_id !='' )")



**Register valid records**

dataench.registerTempTable("filtermusic");

val validrec = sqlContext.sql(" SELECT \* FROM filtermusic where flag ==1 ")



**Save valid /invalid record at HDFS project folder**

Valid records

validrec.write.format("orc").save("hdfs://quickstart.cloudera:8020/user/cloudera/Project/validrec.orc")

Invalid records

invaliddata.write.format("orc").save("hdfs://quickstart.cloudera:8020/user/cloudera/Project/invalidrec.orc")

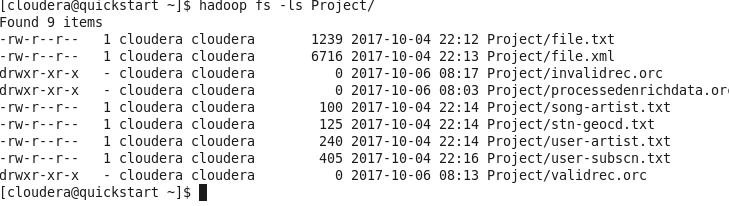


Table list which are available temporarily

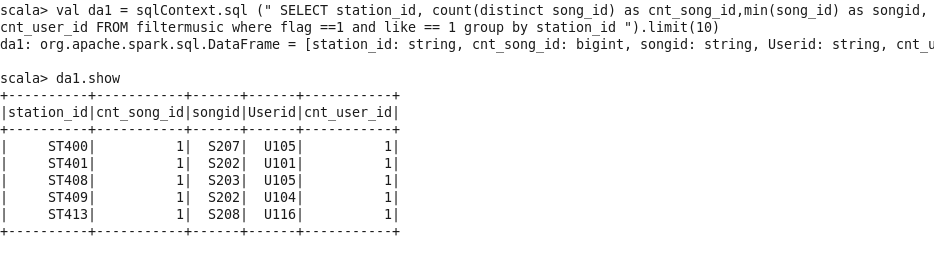
1. Filtermusic
2. userArt (userid, artistid)
3. usersub (user\_id , subs\_start\_date, subs\_end\_date)
4. station ((stationed , geoid)
5. songart (songid , artistid)

# **Data Analysis**

**Data Analysis 1.0**

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

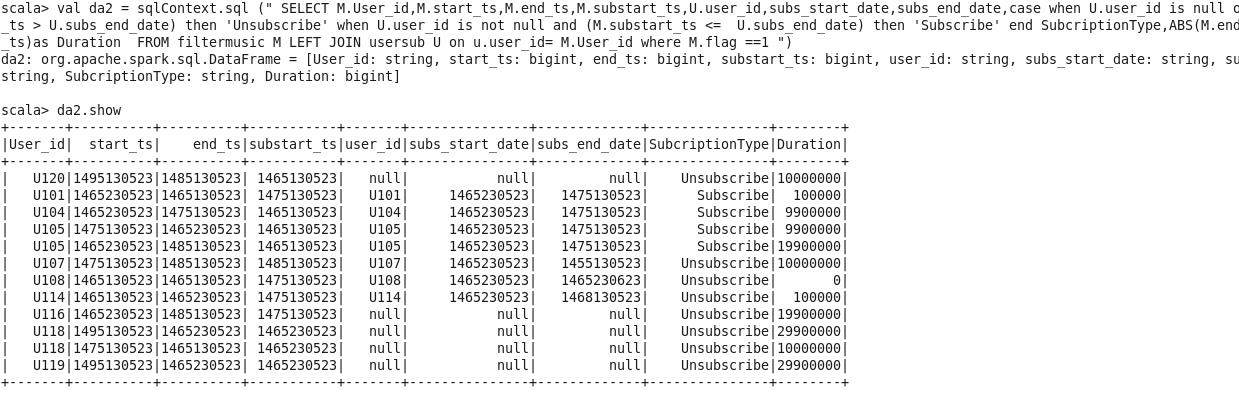
val da1 = sqlContext.sql (" SELECT station\_id, count(distinct song\_id) as cnt\_song\_id,min(song\_id) as songid, min(User\_id)as Userid,count(distinct (User\_id)) as cnt\_user\_id FROM filtermusic where flag ==1 and like == 1 group by station\_id ").limit(10)



**Data Analysis 2.0**

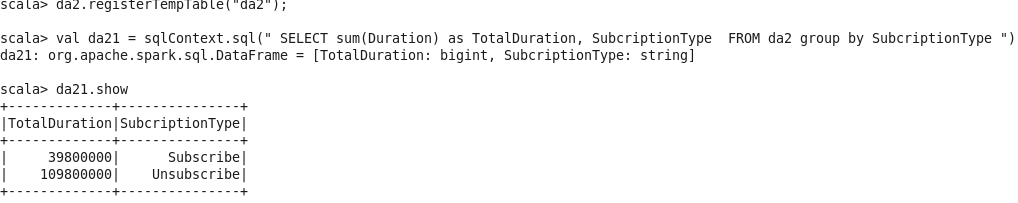
Determine total duration of songs played by each type of user, where type of user can be 'subscribed' or 'unsubscribed'. An unsubscribed user is the one whose record is either not present in Subscribed\_users lookup table or has subscription\_end\_date earlier than the timestamp of the song played by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

# val da2 = sqlContext.sql (" SELECT M.User\_id,M.start\_ts,M.end\_ts,M.substart\_ts,U.user\_id,subs\_start\_date,subs\_end\_date,case when U.user\_id is null or (M.substart\_ts > U.subs\_end\_date) then 'Unsubscribe' when U.user\_id is not null and (M.substart\_ts <= U.subs\_end\_date) then 'Subscribe' end SubcriptionType,ABS(M.end\_ts - M.start\_ts)as Duration FROM filtermusic M LEFT JOIN usersub U on u.user\_id= M.User\_id where M.flag ==1 ")



da2.registerTempTable("da2");

val da21 = sqlContext.sql(" SELECT sum(Duration) as TotalDuration, SubcriptionType FROM da2 group by SubcriptionType ")



**Data Analysis 3.0**

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

**@Hive execute below command**

**Create a Hive table**

CREATE TABLE users\_artists (user\_id STRING, artists\_array ARRAY<STRING>)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY '&';

**Load data in Hive managed table**

load data local inpath '/home/cloudera/Project/user-artist.txt' into table users\_artists;

**Create a output file in local dir**

INSERT OVERWRITE LOCAL DIRECTORY '/home/cloudera/Artist

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT user\_id, artist\_id FROM users\_artists LATERAL VIEW explode(artists\_array) artists AS artist\_id;

**Rename output folder**

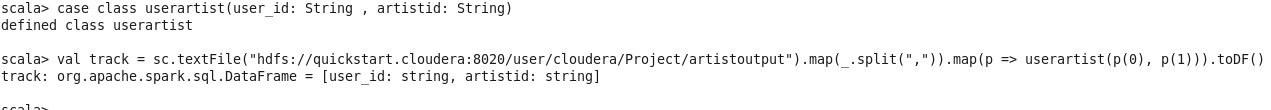
mv Artist/000000\_0 Artist/artistoutput

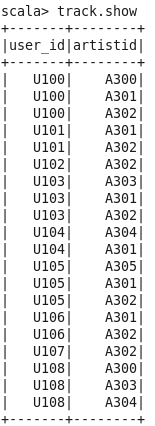
**copy artistoutput file to HDFS Project folder**

hadoop fs -put Artist/artistoutput Project/

case class userartist(user\_id: String , artistid: String)

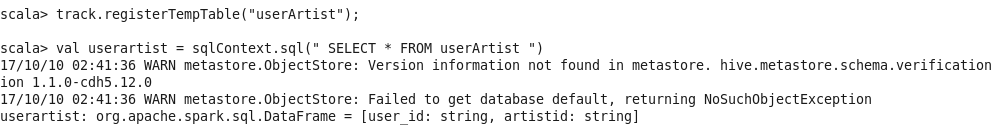
val track = sc.textFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/artistoutput").map(\_.split(",")).map(p => userartist(p(0), p(1))).toDF()



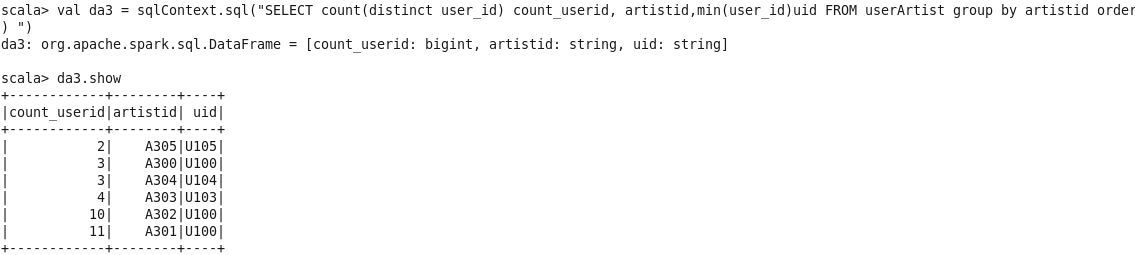


track.registerTempTable("userArtist");

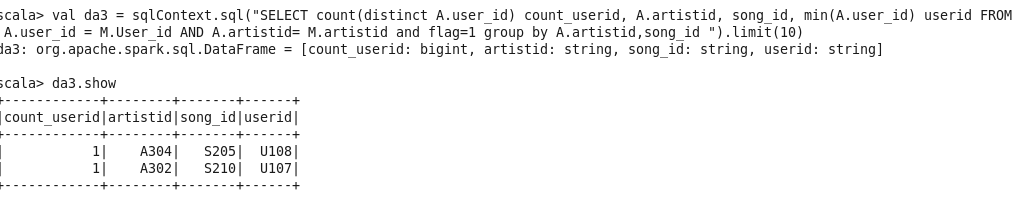
val userartist = sqlContext.sql(" SELECT \* FROM userArtist ")



**UserArtist grouping on Artist**



val da3 = sqlContext.sql("SELECT count(distinct A.user\_id) count\_userid, A.artistid, song\_id, min(A.user\_id) userid FROM userArtist A INNER JOIN filtermusic M on A.user\_id = M.User\_id AND A.artistid= M.artistid and flag=1 group by A.artistid,song\_id ").limit(10)

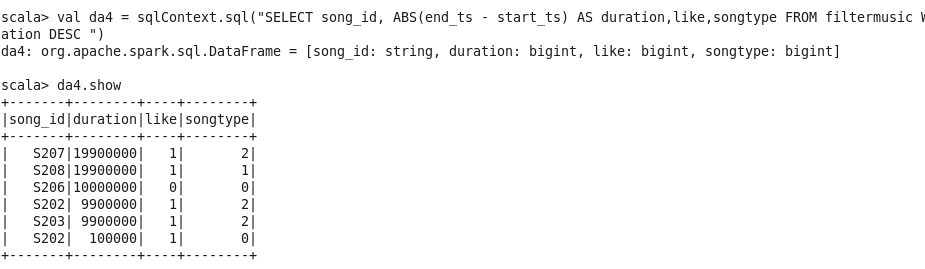


**Data Analysis 4.0**

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.

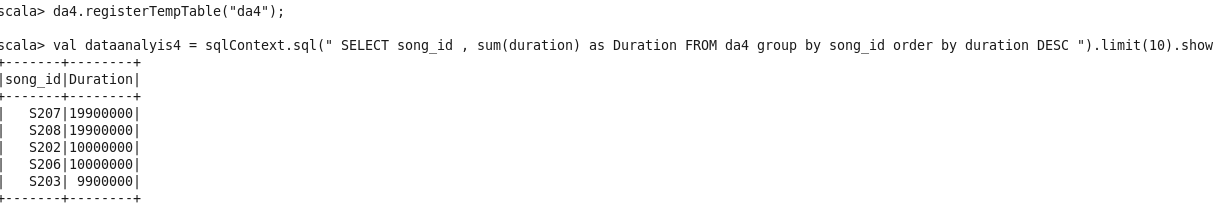
***NOTE: song\_end\_type = 0 means completed successfully***

val da4 = sqlContext.sql("SELECT song\_id, ABS(end\_ts - start\_ts) AS duration,like,songtype FROM filtermusic WHERE flag==1 and (like=1 OR songtype=0) ORDER BY duration DESC ")



da4.registerTempTable("da4");

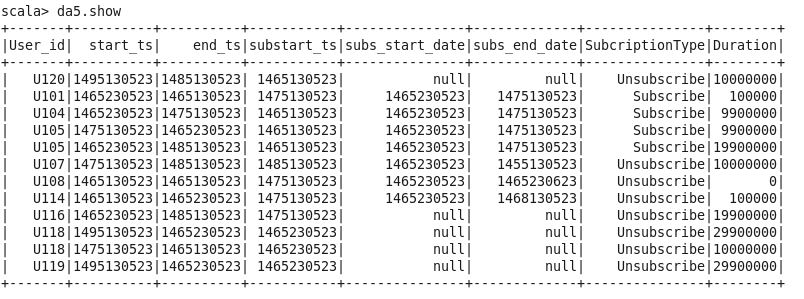
val dataanalyis4 = sqlContext.sql(" SELECT song\_id , sum(duration) as Duration FROM da4 group by song\_id order by duration DESC ").limit(10).show



**Data Analysis 5.0**

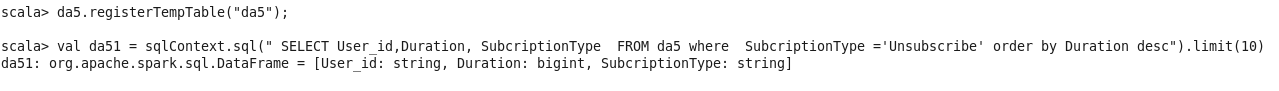
Determine top 10 unsubscribed users who listened to the songs for the longest duration.

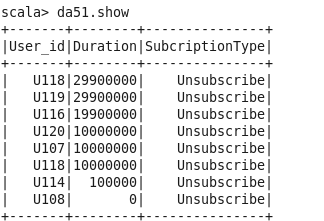
# val da5 = sqlContext.sql (" SELECT M.User\_id,M.start\_ts,M.end\_ts,M.substart\_ts,subs\_start\_date,subs\_end\_date, case when U.user\_id is null or (M.substart\_ts > U.subs\_end\_date) then 'Unsubscribe' when U.user\_id is not null and (M.substart\_ts <= U.subs\_end\_date) then 'Subscribe' end SubcriptionType, ABS(M.end\_ts - M.start\_ts) as Duration FROM filtermusic M LEFT JOIN usersub U on u.user\_id= M.User\_id where M.flag ==1 ")



da5.registerTempTable("da5");

val da51 = sqlContext.sql(" SELECT User\_id,Duration, SubcriptionType FROM da5 where SubcriptionType ='Unsubscribe' order by Duration desc").limit(10)





**Create an output folder and store all analysis results in HDFS**

da1.rdd.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da1")

da21.rdd.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da21")

da3.rdd.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da3")

da4.rdd.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da4")

da51.rdd.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da5")

**Export all analysis data to physical table @ mysql for further analysis or visualization**

**Create a table in mysql:**

1.

CREATE TABLE da1 (

Station\_id varchar(10) Not NULL,

Cnt\_song\_id Int,

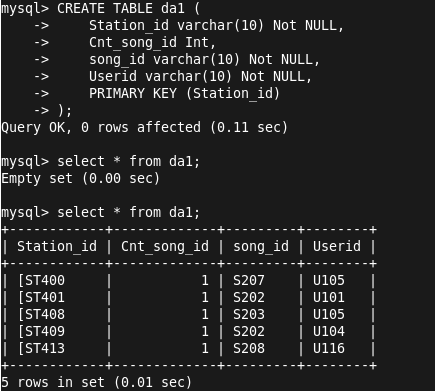
song\_id varchar(10) Not NULL,

Userid varchar(10) Not NULL,

PRIMARY KEY (Station\_id)

);

**Similarly created da2, da3, da4, da51 for all five analysis.**



sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table da1 --export-dir hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da1/part-00000

sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table da2 --export-dir hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da211/part-00193

sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table da3 --export-dir hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da3/part-00000

sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table da4 --export-dir hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da4/part-00000

sqoop export --connect jdbc:mysql://localhost/project --username root --password cloudera --table da5 --export-dir hdfs://quickstart.cloudera:8020/user/cloudera/Project/Result/da5/part-00000