1. **Design**

To solve the Music Data Analysis Requirement the following modules are used

i. **LoadHBaseTables –** This is used for loading the lookup data from csv files to the relevant tables in Hbase. Hbase tables used are as below:
   - **StationIdGeoCd –** This HBase table is used for storing mapping between stationId and GeoCd
   - **SongArtist -** This HBase table is used for storing mapping beween Song and Artist
   - **UserArtist –** This HBase table is used for storing mapping beween userId and List of artists he follows
   - **UserSubscription-** This HBase table is used for storing mapping beween userId and subscription start timestamp and end timestamp

ii. **MusicDataProcessorApp -** This is the main application for processing music data. It in turn calls multiple submodules as below

iii. **WebMusicDataProcessor –** This is the class for processing music data stored in /data/web/file-1.xml folder and use processData method to return dataframes

iv. **MobileMusicDataProcessor -** This is the class for processing music data stored in /data/mob/file.txt folder and use processData method to return dataframes

v. **MusicDataEnricher -** This is the used for enriching and validating the datasets. New columns are added to dataframe for the processed data. The new columns added to temporary table MusicDataDetailed are as below:

a. modified_Geo_cd **–** This field is used to populate Null/blank values in Geo_cd column. If Geo_cd column is not blank, the value will copied as it is to modified_Geo_cd. If Geo_cd column is blank then consults the look table StationIdGeoCd based on stationId, If found populate it, else put value Invalid

b. modified_Artist_id **–** This field is used to populate Null/blank values in Artist_id column. If Artist_id column is not blank, the value will copied as it is to modified_Artist_id. If Artist_id column is blank then consults the look table SongArtist table based on songId, If found populate it, else put value Invalid

c. follower – This field is used when the user follows the artist. For each record, for userId consult UserArtist table to find the list of artists user is following. If it is blank, then follower will be 0. If it is not blank and the corresponding artistId for the record is present in artist List from the map, then set follower to 1

d. subscribed – This filed is used to know if the user was a subscriber when he played the music. This is done by getting the userId for the record and consulting **UserSubscription** table to get the tuple (start_ts,end_ts). If it does not exist then subscribed field will be 0. If I exists, If the start_ts of record is in between (start_ts,end_ts) of UserSubscritpion then subscribed field will be 1 else it will be 0

e. isValid- This field is used to know if the record is valid or not. Isvalid if valid will have value 1 else 0. The following validations are done:

If User_id is blank then (0)

If Song_id is blank then 0

If modified_Geo_cd is Invalid then 0

If modified_Arist_id is Invalid then 0

If timestamp is 0 then 0

If Start_ts is 0 then 0

If Start_ts > End_ts then0

vi. **MusicDataPopulateMapFromLookupTables -** This is the used for populating maps from HBase tables used for lookup

vii. **MusicDataAnalyzer - -** This is used for analyzing the data in dataframes and store the results in HDFS file:

Top10Stations_<Timestamp> - Top 10 stations where maximum number of songs played which is liked by unique user

MusicDurationByUserType_<Timestmp> - Music Duration by user category of user: Subscribed, Unsubscribed

Top10SongsHavignMaximumRevenue_<Timestamp> - Top 10 songs having maximum revenue

Top10ConnectedArtists_<Timestamp> - Top 10 connected artists

Top10UnsubscribedUsers_<Timestamp> - Top 10 unsubscribed users

2. Implementation

MODULE1: LoadHBaseTables: Loading HBase Tables with Lookup Data

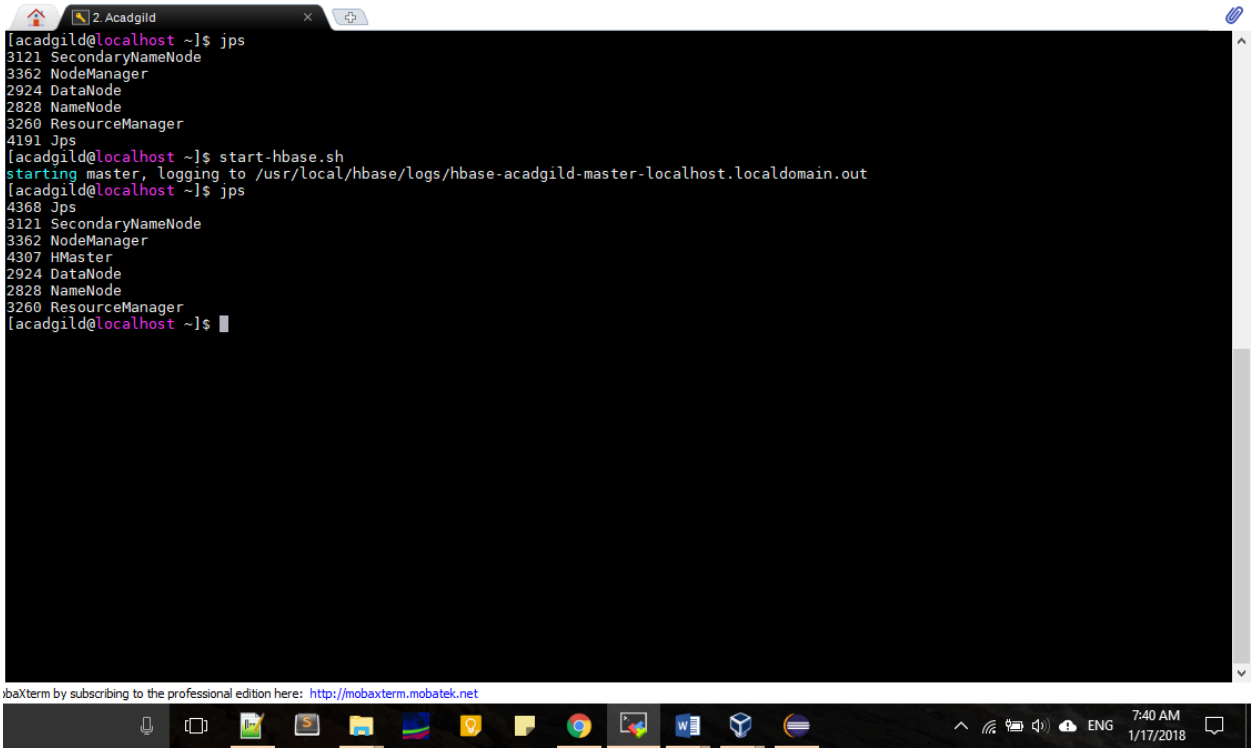
Step1: Start all the services

Start the services using start-all.sh

Start HBase using:

start-hbase.sh

Confirm that services are up using command: jps



Step2: Create class for loading Hbase tables

Create a project load-hbase-tables and create a scala class LoadHBaseTables which is explained below:

- Import all the dependent packages and create the package

package final_project

```
import org.apache.log4j.{Level, LogManager, PropertyConfigurator}
import org.apache.spark._
import org.apache.spark.sql.SQLContext
import org.apache.hadoop.hbase.util.Bytes
import org.apache.hadoop.hbase.client.{ Put, HTable }
import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.HTableDescriptor
import org.apache.hadoop.hbase.HColumnDescriptor
import org.apache.hadoop.hbase.client.HBaseAdmin
import org.apache.hadoop.hbase.client.HTable
import org.apache.spark.rdd.RDD
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.log4j.{Level, LogManager, PropertyConfigurator}
```

- Create object LoadHBaseTables and define main method and logs
-

```
object LoadHBaseTables {
  def main(args: Array[String]) {
    val log = LogManager.getRootLogger
    log.setLevel(Level.INFO)
```

- Get all the arguments, argument 1 is for filePath of lookup file, argument 2 is table name, argument 3 is composite fields of separated fields of Column Family and Column, which can be multiple

```
    val filePath = args(0)
    val tablename = args(1)
    val columnFamilyField = args(2)
```

```
val columnFamilyFieldList = columnFamilyField.split(",")
val columnFamilyFieldListLength = columnFamilyFieldList.length
```

- Create configuration, Spark Context, SQL Context

```
val conf = new SparkConf().setAppName("LoadHbaseTable").setMaster("local[2]")
val sc = new SparkContext(conf)
sc.setLogLevel("ERROR")
val sqlContext = new SQLContext(sc)
```

- Create HBase configuration

```
val hconf = HBaseConfiguration.create
hconf.set(TableInputFormat.INPUT_TABLE, tablename)
val admin = new HBaseAdmin(hconf)
```

- Check if tables is already created or not . If not created, create table and its column family

```
if (!admin.isTableAvailable(tablename)) {

 log.info("Creating table " + tablename)
 val tableDescription = new HTableDescriptor(tablename)
 tableDescription.addFamily(new HColumnDescriptor(columnFamilyFieldList(0).getBytes()))
 admin.createTable(tableDescription)
 if (admin.isTableAvailable(tablename)) {
   log.info("Table " + tablename + " is created successfully")
 }
} else {
 log.warn("Table " + tablename + " already exists")
}
```

- Get the table and

```
val table = new HTable(hconf, tablename)
```

- Process the csv file which has lookup data and create tuples using map with key and value. If there are multiple fields given, value is taken as a tuple

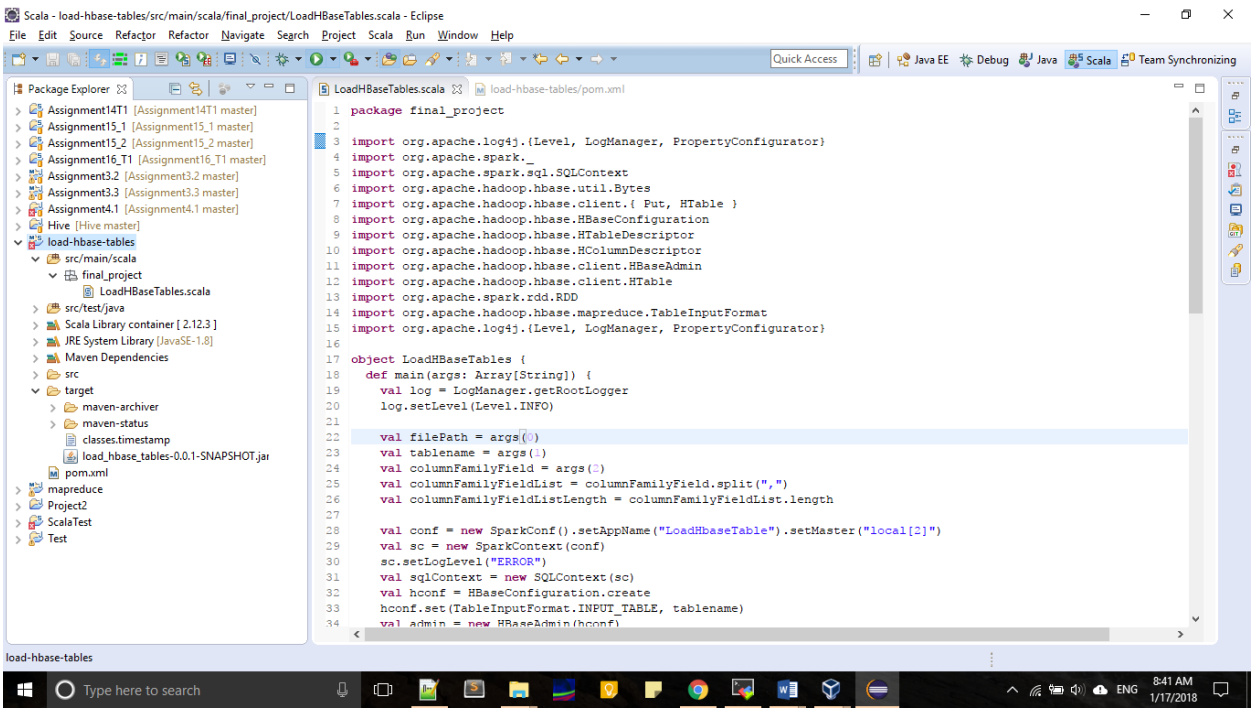
```
val file = sc.textFile("file://" + filePath)

var records1:Array[(String,String)] = null
var records2:Array[(String,String,String)] = null
if (columnFamilyFieldListLength == 2) {
   records1 = file.map(_.split(",")).map(x => (x(0), x(1))).collect
} else if (columnFamilyFieldListLength == 4) {
   records2 = file.map(_.split(",")).map(x => (x(0), x(1), x(2))).collect
}
```

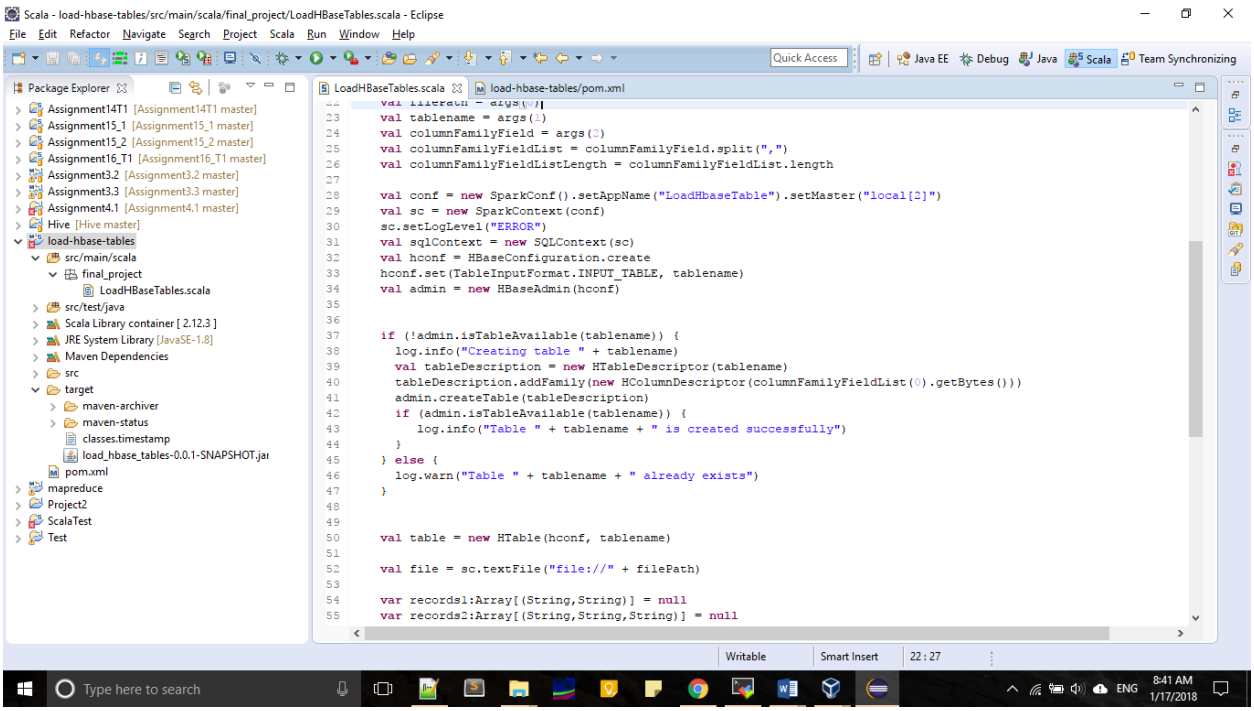
- Save the columns

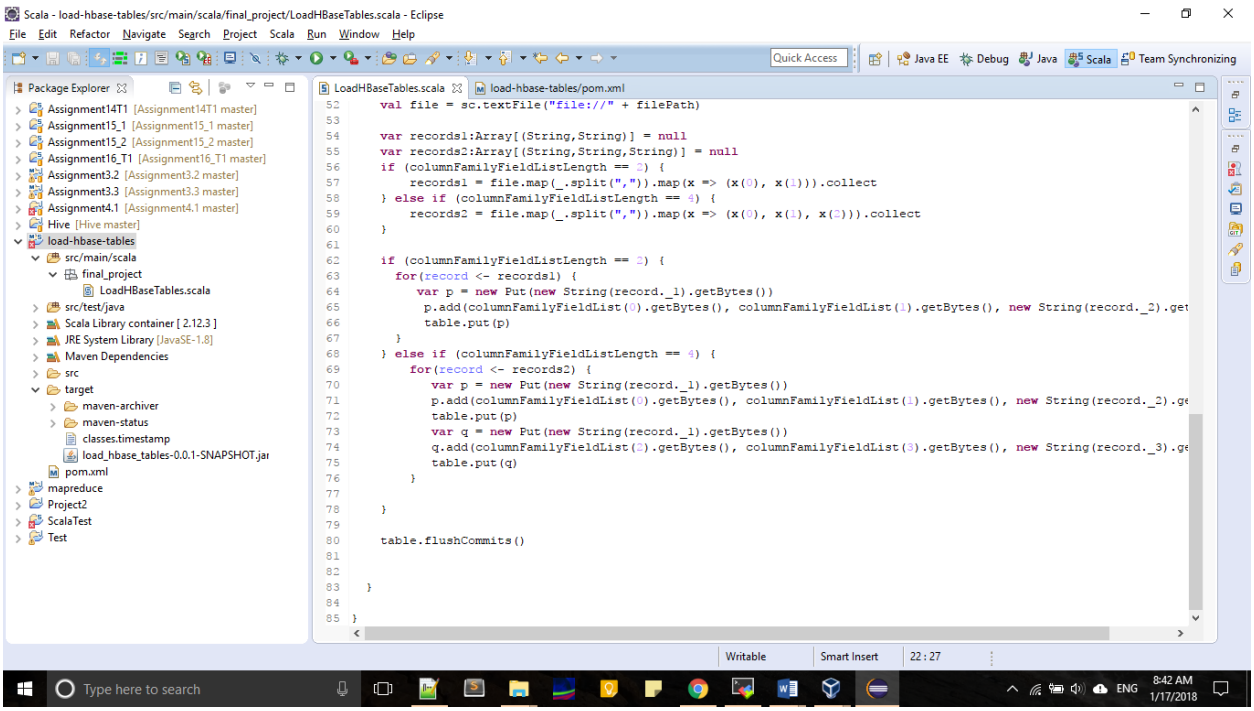
```
if (columnFamilyFieldListLength == 2) {
  for(record <- records1) {
    var p = new Put(new String(record._1).getBytes())
    p.add(columnFamilyFieldList(0).getBytes(), columnFamilyFieldList(1).getBytes(), new String(record._2).getBytes())
    table.put(p)
  }
} else if (columnFamilyFieldListLength == 4) {
    for(record <- records2) {
      var p = new Put(new String(record._1).getBytes())
      p.add(columnFamilyFieldList(0).getBytes(), columnFamilyFieldList(1).getBytes(), new String(record._2).getBytes())
      table.put(p)
      var q = new Put(new String(record._1).getBytes())
      q.add(columnFamilyFieldList(2).getBytes(), columnFamilyFieldList(3).getBytes(), new String(record._3).getBytes())
```

```
        table.put(q)
      }

  }

  table.flushCommits()


 }

}
```

Screenshots are as below:

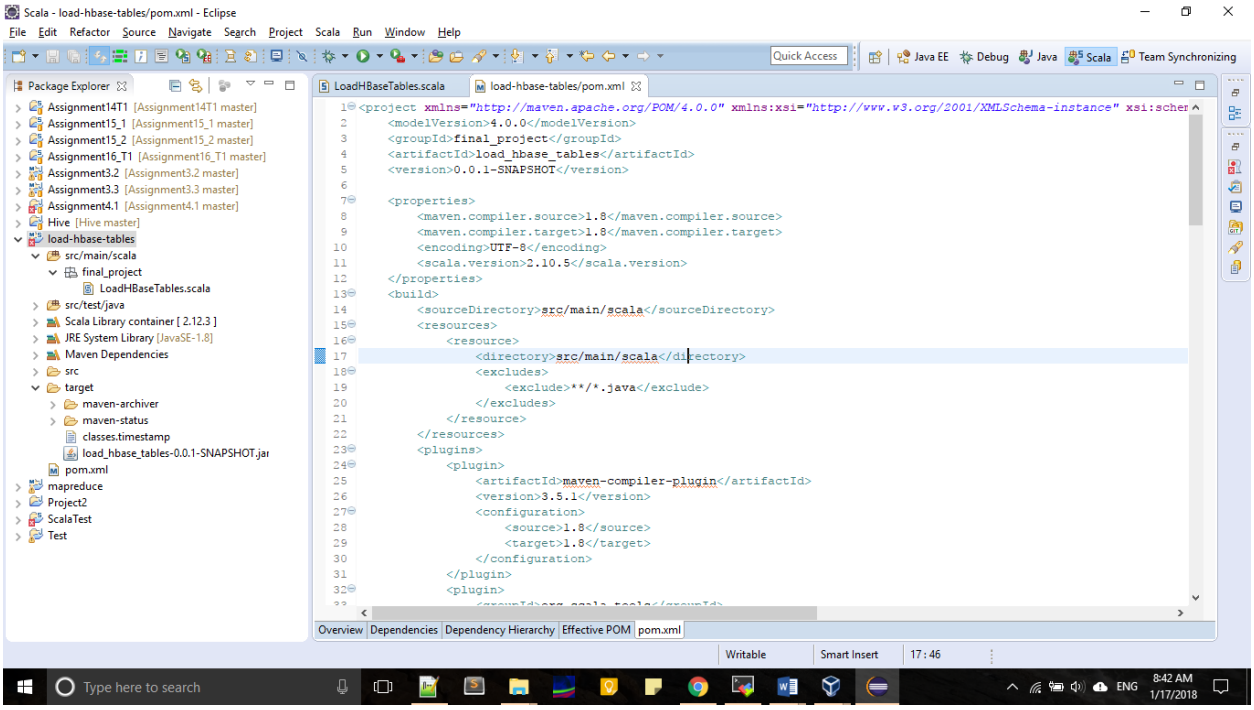




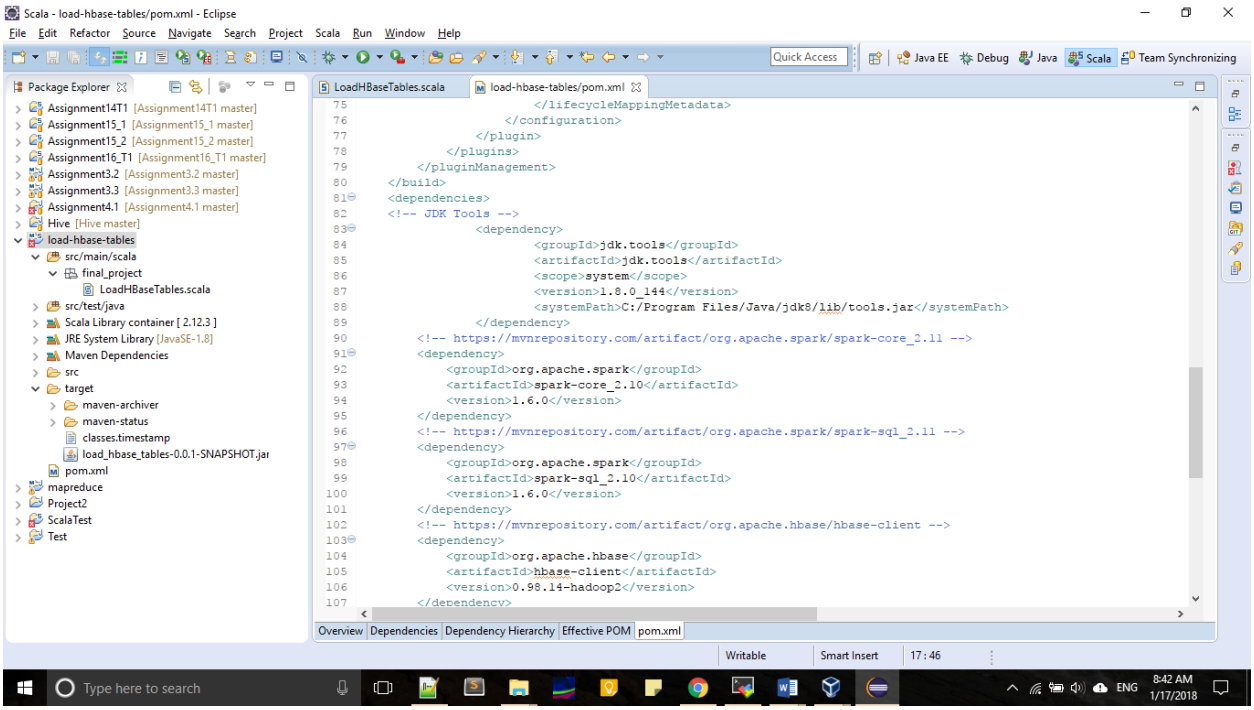


Step3: Compile the file using eclipse and run the code

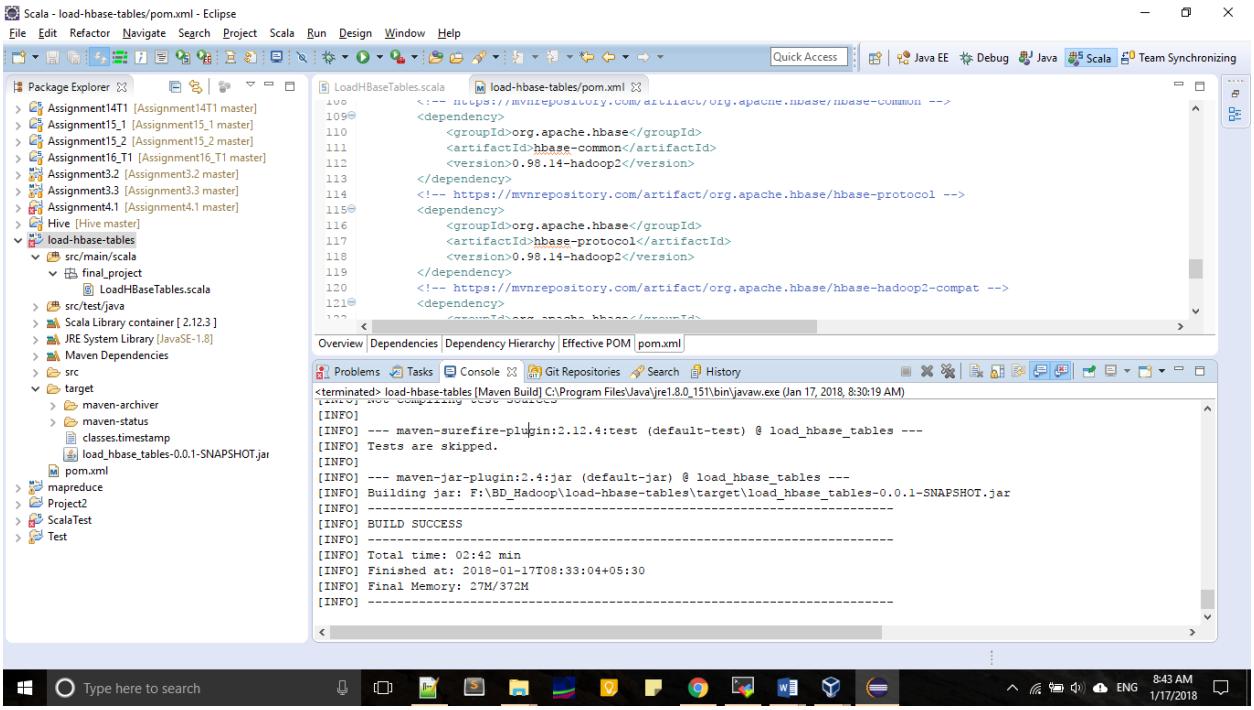
- Define the pom file



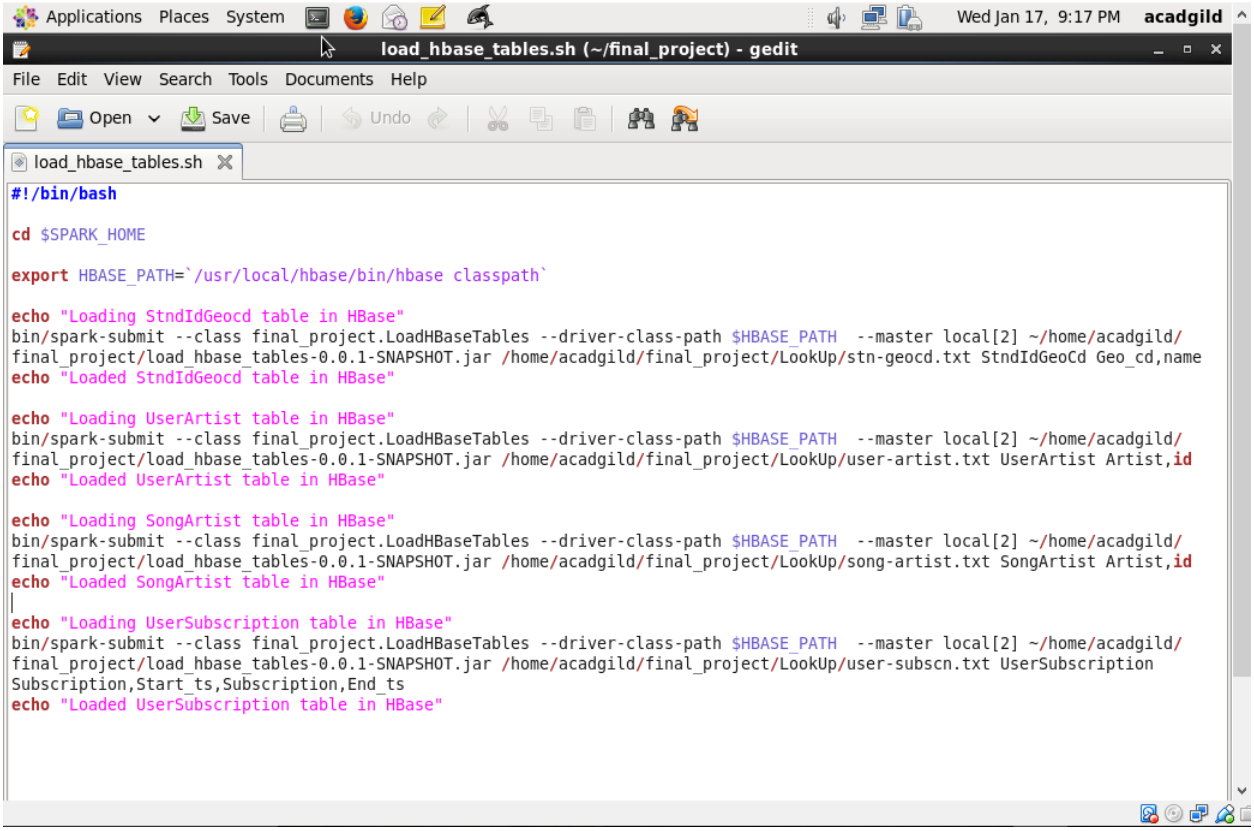
-



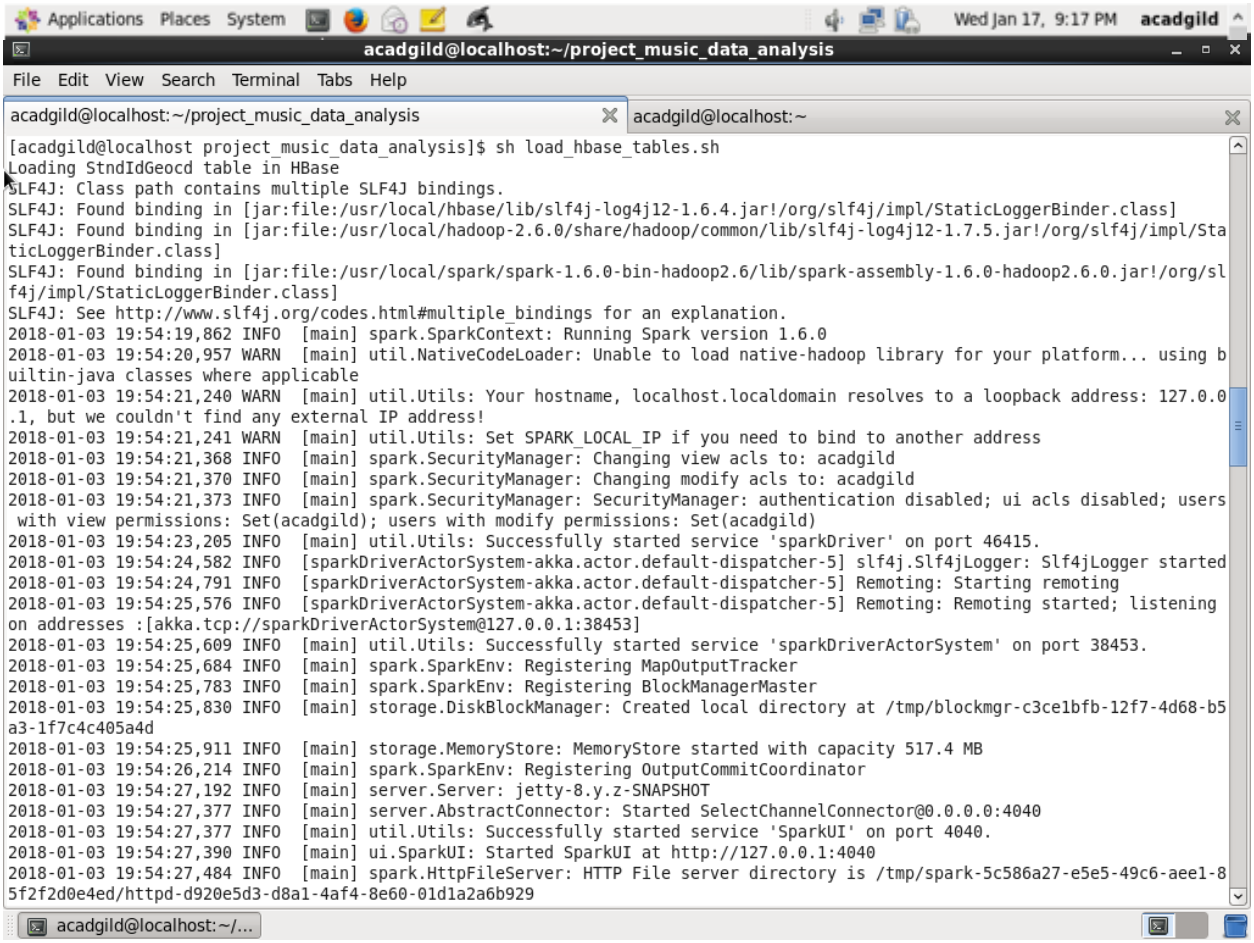
- Compile using maven install, screenshot is as below:

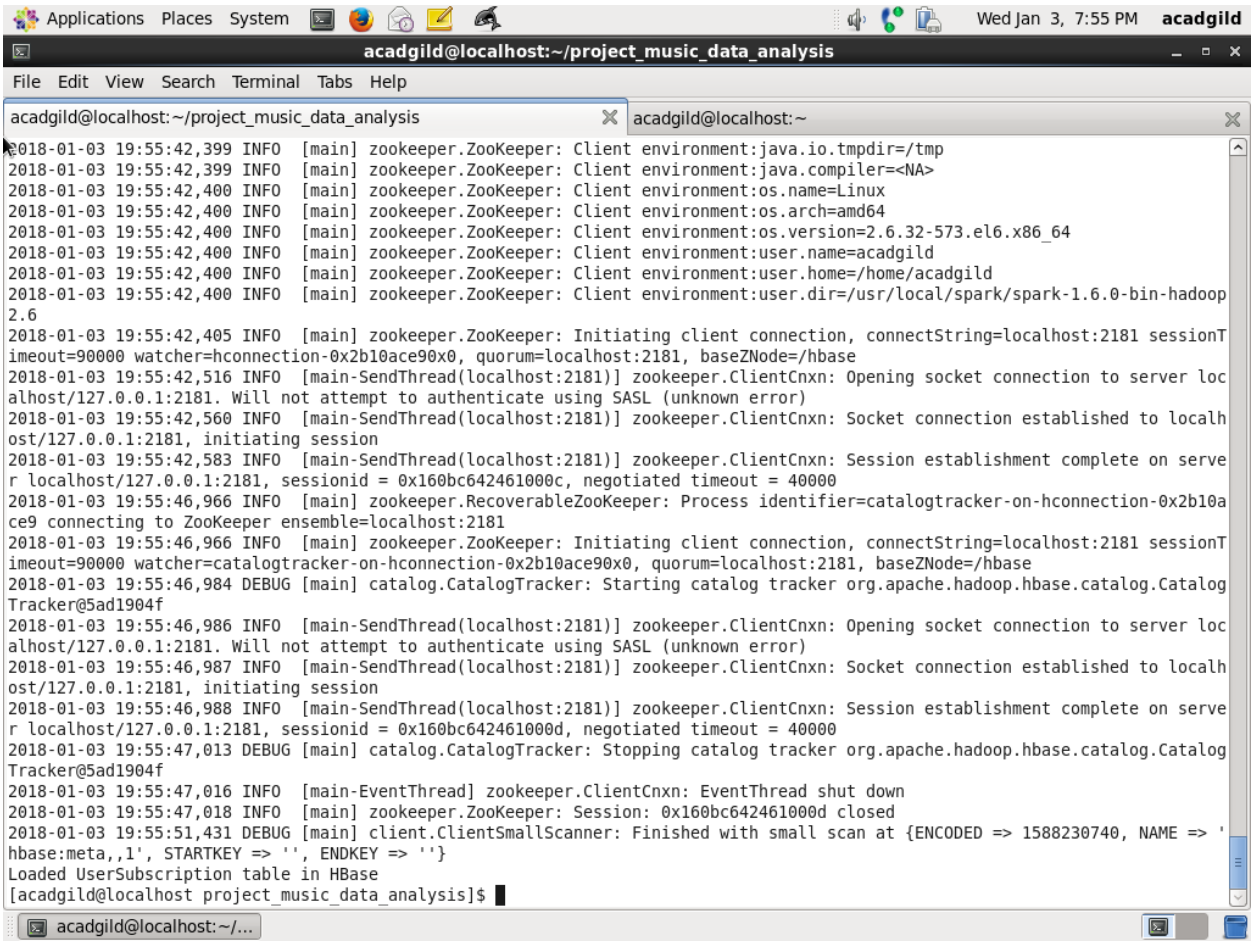


- Write a script to load all the tables:



- Run the script to load all the tables


Applications Places System
Wed Jan 17, 9:17 PM acadgild
acadgild@localhost:~/project_music_data_analysis
File Edit View Search Terminal Tabs Help
acadgild@localhost:~/project_music_data_analysis
acadgild@localhost:~
[acadgild@localhost project_music_data_analysis]$ sh load_hbase_tables.sh
Loading StndIdGeocd table in HBase
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/slf4j-log4j12-1.6.4.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-2.6.0/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/spark/spark-1.6.0-bin-hadoop2.6/lib/spark-assembly-1.6.0-hadoop2.6.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
2018-01-03 19:54:19,862 INFO [main] spark.SparkContext: Running Spark version 1.6.0
2018-01-03 19:54:20,957 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2018-01-03 19:54:21,240 WARN [main] util.Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1, but we couldn't find any external IP address!
2018-01-03 19:54:21,241 WARN [main] util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
2018-01-03 19:54:21,368 INFO [main] spark.SecurityManager: Changing view acls to: acadgild
2018-01-03 19:54:21,370 INFO [main] spark.SecurityManager: Changing modify acls to: acadgild
2018-01-03 19:54:21,373 INFO [main] spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(acadgild); users with modify permissions: Set(acadgild)
2018-01-03 19:54:23,205 INFO [main] util.Utils: Successfully started service 'sparkDriver' on port 46415.
2018-01-03 19:54:24,582 INFO [sparkDriverActorSystem-akka.actor.default-dispatcher-5] slf4j.Slf4jLogger: Slf4jLogger started
2018-01-03 19:54:24,791 INFO [sparkDriverActorSystem-akka.actor.default-dispatcher-5] Remoting: Starting remoting
2018-01-03 19:54:25,576 INFO [sparkDriverActorSystem-akka.actor.default-dispatcher-5] Remoting: Remoting started; listening on addresses :[akka.tcp://sparkDriverActorSystem@127.0.0.1:38453]
2018-01-03 19:54:25,609 INFO [main] util.Utils: Successfully started service 'sparkDriverActorSystem' on port 38453.
2018-01-03 19:54:25,684 INFO [main] spark.SparkEnv: Registering MapOutputTracker
2018-01-03 19:54:25,783 INFO [main] spark.SparkEnv: Registering BlockManagerMaster
2018-01-03 19:54:25,830 INFO [main] storage.DiskBlockManager: Created local directory at /tmp/blockmgr-c3ce1bfb-12f7-4d68-b5a3-1f7c4c405a4d
2018-01-03 19:54:25,911 INFO [main] storage.MemoryStore: MemoryStore started with capacity 517.4 MB
2018-01-03 19:54:26,214 INFO [main] spark.SparkEnv: Registering OutputCommitCoordinator
2018-01-03 19:54:27,192 INFO [main] server.Server: jetty-8.y.z-SNAPSHOT
2018-01-03 19:54:27,377 INFO [main] server.AbstractConnector: Started SelectChannelConnector@0.0.0.0:4040
2018-01-03 19:54:27,377 INFO [main] util.Utils: Successfully started service 'SparkUI' on port 4040.
2018-01-03 19:54:27,390 INFO [main] ui.SparkUI: Started SparkUI at http://127.0.0.1:4040
2018-01-03 19:54:27,484 INFO [main] spark.HttpFileServer: HTTP File server directory is /tmp/spark-5c586a27-e5e5-49c6-aee1-85f2f2d0e4ed/httpd-d920e5d3-d8a1-4af4-8e60-01d1a2a6b929
acadgild@localhost:~/...

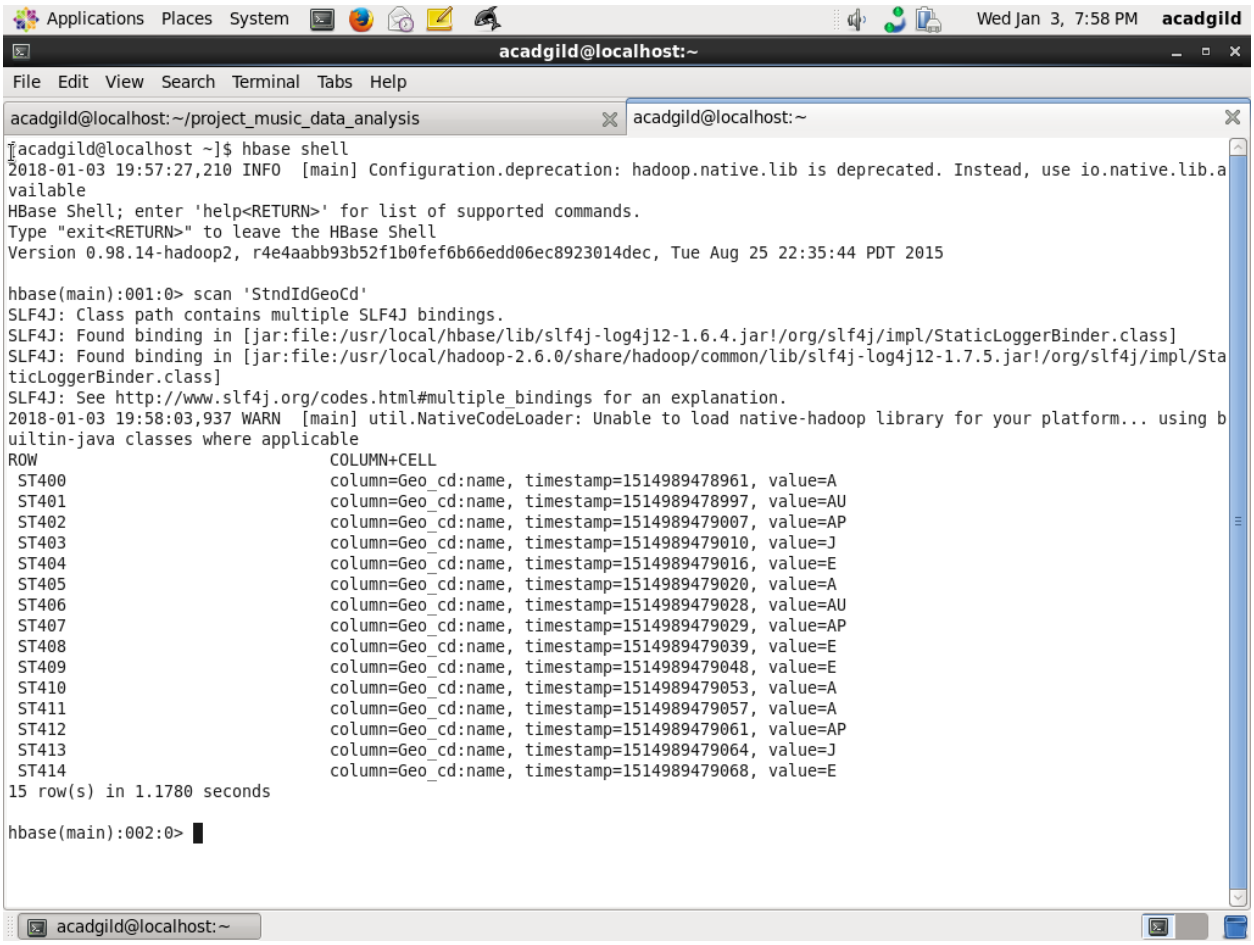


Step3: Verify that all the lookup tables are populated correctly

- Verify all the tables are populated correctly first login to HBase using hbase shell
- Table StnIdGeoCd is verified using

scan ‘StnIdGeoCd’

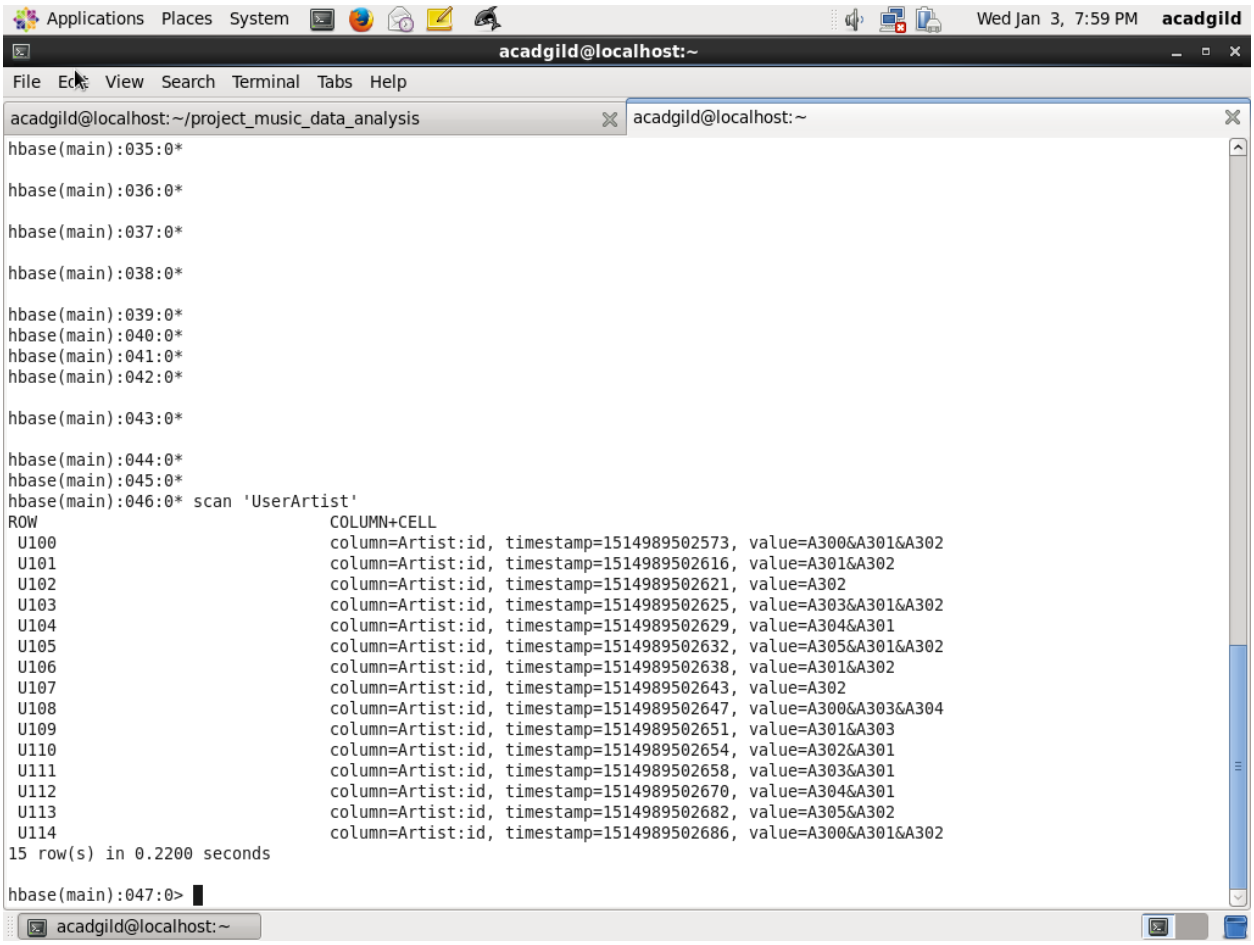
Screenshot is as below:

- 

- Table UserArtist is verified using

scan ‘UserArtist’

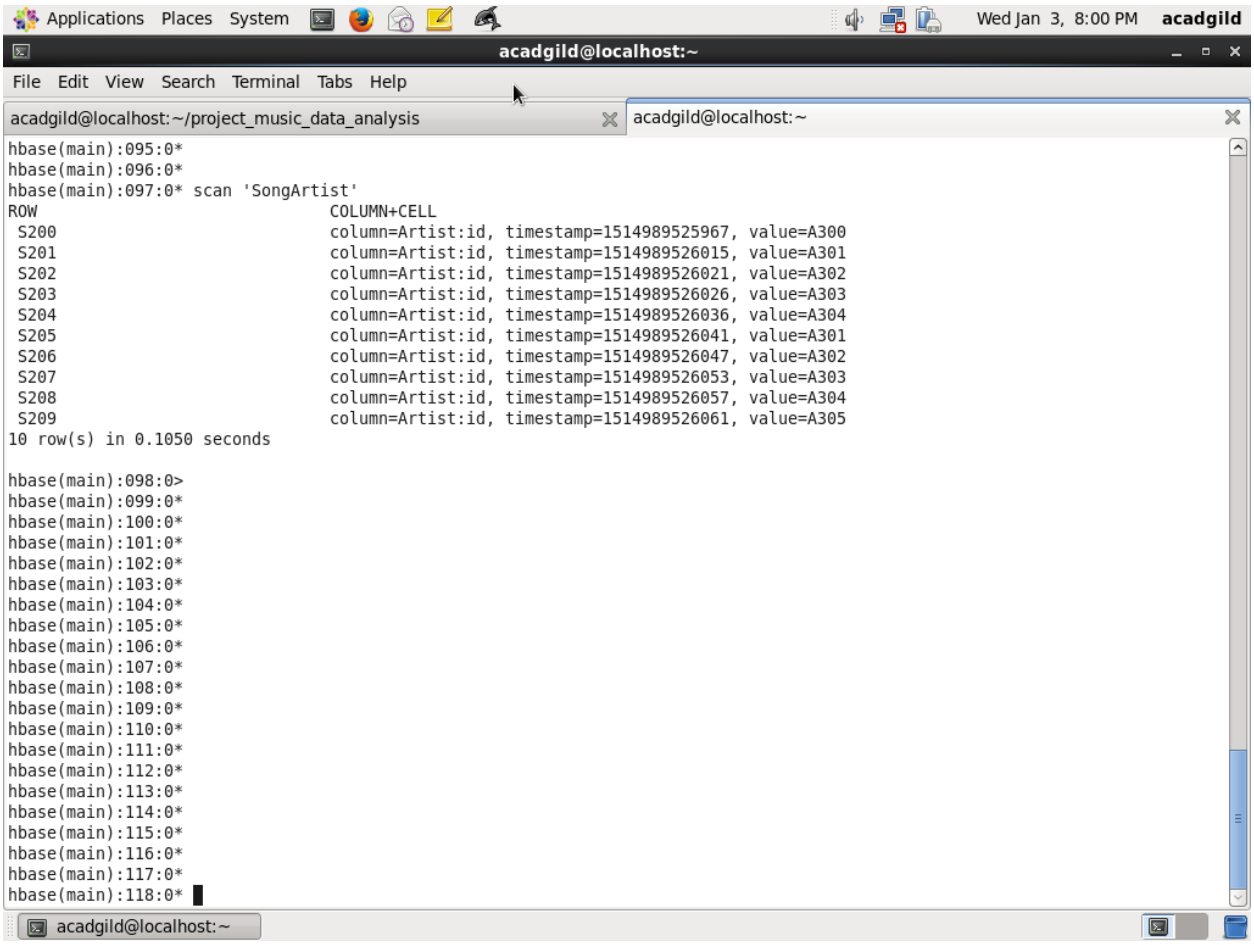
Screenshot is as below



- Table SongArtist is verified using

scan ‘SongArtist’

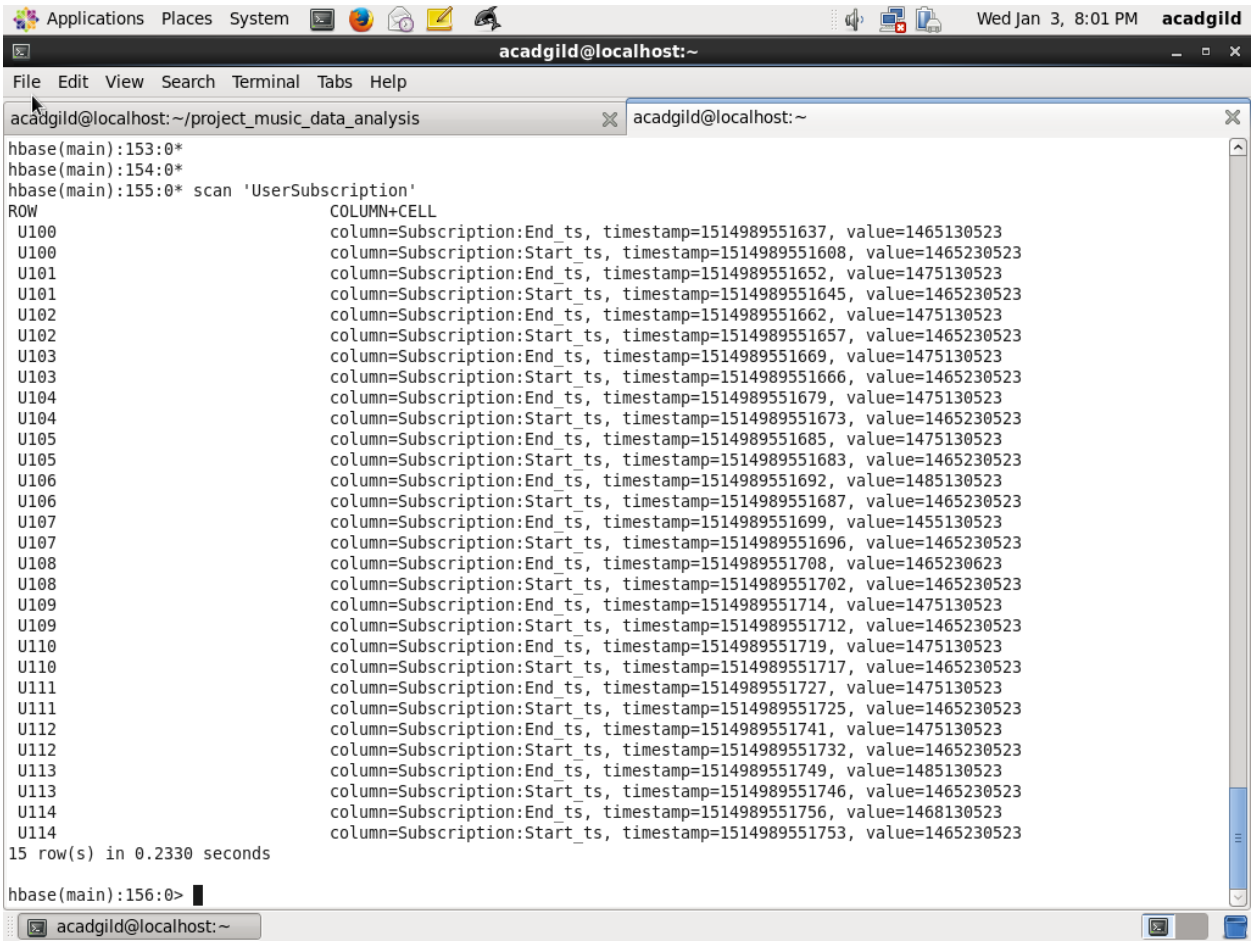
Screenshot is as below:



- Table UserSubscription is verified using

scan ‘UserSubscription’

Screenshot is as below:



MODULE2: Handling Web and Mobile Data

Step1: Write a scala class for handling Web data

- Write class WebMusicDataProcessor to process web music data stored in /data/web/file-1.xml and store it as dataframe

-Import dependent packages

```
package final_project

import org.apache.spark._

import scala.xml.XML
```

```
import org.apache.spark.sql.DataFrame
import scala.collection.mutable.ListBuffer
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.types._
import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }
import org.apache.spark.broadcast.Broadcast
import java.text.SimpleDateFormat
import scala.collection.mutable.HashMap

        // Define CustomException this is to solve the continue

case class CustomException(message:String) extends Exception(message)

        // Define the class with WebMusicDataProcessor with parametes
class WebMusicDataProcessor(param: String, context: SparkContext, sqc: SQLContext) extends
Serializable {
  val filePath: String = param
  val sc: SparkContext = context
  val sqlContext:SQLContext = sqc


// Define the method which does the processing of data  and return as dataframe


  def processData(): DataFrame = {
    val log = LogManager.getRootLogger
    log.setLevel(Level.INFO)
  //  val sqlContext = new SQLContext(sc)
    import sqlContext.implicits._
```

// Define dateFormat as yyyy-MM-dd HH:mm:ss and recordListBuffer which will act as buffer for storing data

```
val dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")

val recordListBuffer = new ListBuffer[(String, String, String, Long, Long, Long, String, String, String, String, String)]()
```

// Load data from XML path

```
val xml = XML.loadFile(filePath)
```

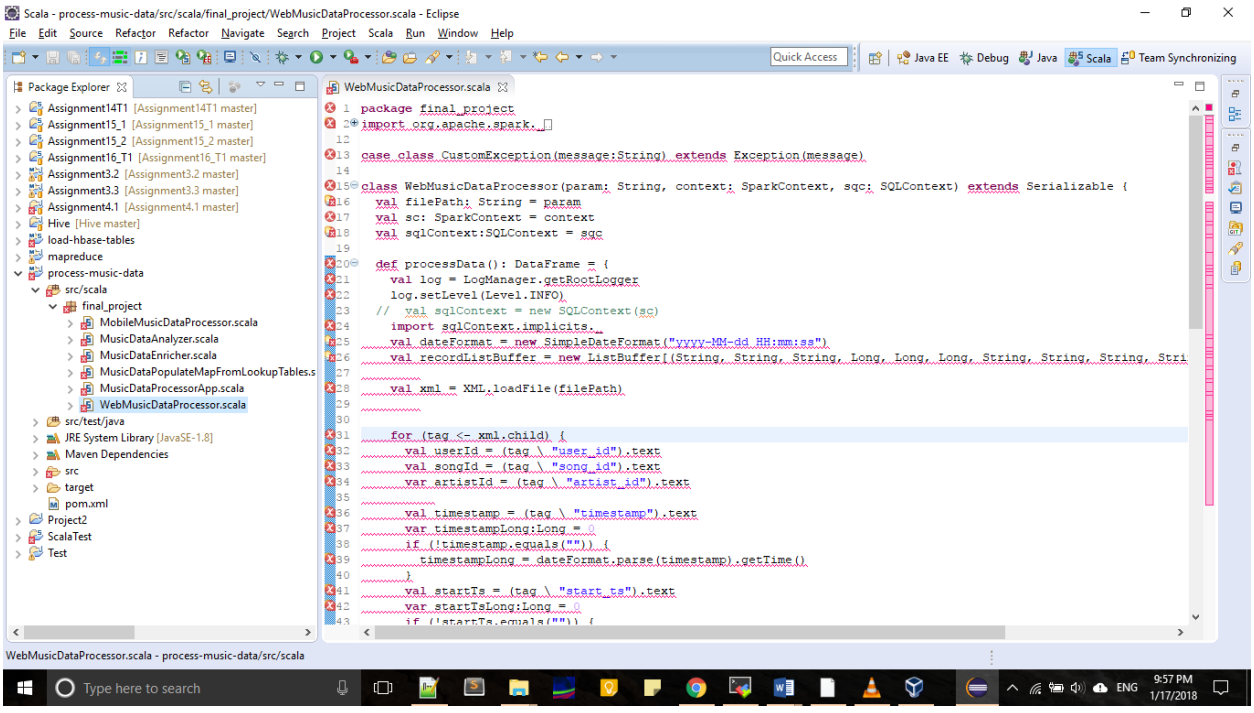
- Process all the fields user_id, song_id, artist_id, timestamp, start_ts,end_ts. Handle all null conditions

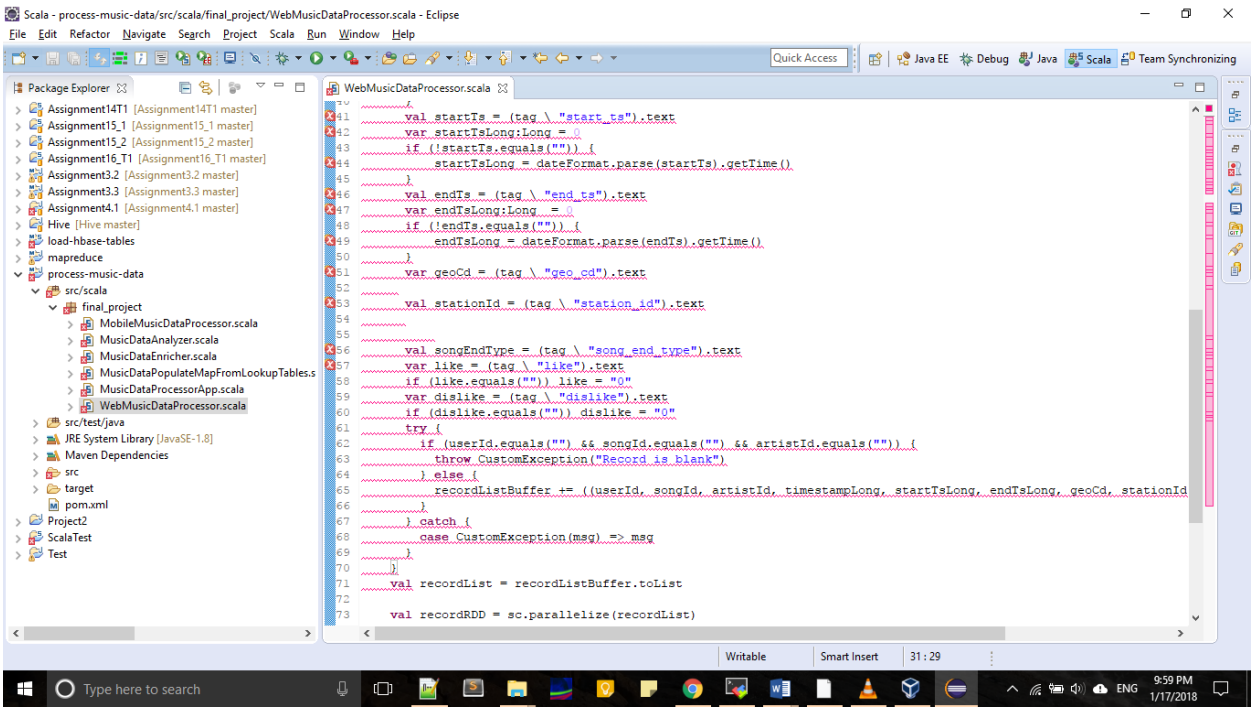
```
for (tag <- xml.child) {
  val userId = (tag \ "user_id").text
  val songId = (tag \ "song_id").text
  var artistId = (tag \ "artist_id").text

  val timestamp = (tag \ "timestamp").text
  var timestampLong:Long = 0
  if (!timestamp.equals("")) {
    timestampLong = dateFormat.parse(timestamp).getTime()
  }
  val startTs = (tag \ "start_ts").text
  var startTsLong:Long = 0
  if (!startTs.equals("")) {
     startTsLong = dateFormat.parse(startTs).getTime()
  }
  val endTs = (tag \ "end_ts").text
  var endTsLong:Long  = 0
  if (!endTs.equals("")) {
```

```
      endTsLong = dateFormat.parse(endTs).getTime()
    }
    var geoCd = (tag \ "geo_cd").text

    val stationId = (tag \ "station_id").text


    val songEndType = (tag \ "song_end_type").text
    var like = (tag \ "like").text
    if (like.equals("")) like = "0"
    var dislike = (tag \ "dislike").text
    if (dislike.equals("")) dislike = "0"
    try {

// Continue with the record in case fileds are blank
      if (userId.equals("") && songId.equals("") && artistId.equals("")) {
        throw CustomException("Record is blank")
      } else {
        recordListBuffer += ((userId, songId, artistId, timestampLong, startTsLong, endTsLong, geoCd,
stationId, songEndType, like, dislike))
      }
    } catch {
      case CustomException(msg) => msg
    }
  }

// From the buffer convert to dataFrame recordDF having fields User_id, Songs_id, Artist_id, Timestamp,
// Start_ts, End_ts, Geo_cd, Station_id, Song_end_type,      Likes, Dislikes
  val recordList = recordListBuffer.toList
```

```
  val recordRDD = sc.parallelize(recordList)
  val recordDF = recordRDD.toDF("User_id", "Songs_id", "Artist_id", "Timestamp",
    "Start_ts", "End_ts", "Geo_cd", "Station_id", "Song_end_type",
     "Likes", "Dislikes")

  log.info("Number of records =" + recordDF.count)
  log.info("Showing records for Web Music Data ")
  recordDF.show

  return recordDF

 }
```

Screenshot is as below:





Step2: define class for processing Mobile data

- Define a class MobileMusicDataProcessor for processing music data from /data/mob/file.txt

// define package as final_project and Import al the dependent packages

package final_project

import org.apache.spark._

import org.apache.spark.sql.DataFrame

import scala.collection.mutable.ListBuffer

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.types._

import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }

import org.apache.spark.broadcast.Broadcast

import scala.collection.mutable.HashMap

```
// Create a case class MusicData with all the fields

case class MusicData(User_id:String, Songs_id:String, Artist_id:String, Timestamp:Long,
    Start_ts:Long, End_ts:Long, Geo_cd:String, Station_id:String, Song_end_type:String,
    Likes:String, Dislikes:String)

// Define class MobileMusicDataProcessor with all the parameters

class MobileMusicDataProcessor(param: String, context: SparkContext, sqc:SQLContext) extends
Serializable {
  val filePath: String = param
  val sc: SparkContext = context
  val sqlContext:SQLContext = sqc

  // Define method processData
  def processData(): DataFrame = {
    val log = LogManager.getRootLogger
    log.setLevel(Level.INFO)

    import sqlContext.implicits._
// Load dataset from mobeile file path /data/mob/file.txt into RDD and slipt the fields
    val recordRDD = sc.textFile(filePath)
    val recordFieldsRDD = recordRDD.map(x => x.split(",")).filter(x=> x.length ==11)

    // Convert the RDD to dataframe with case class MusicData
    val recordDF = recordFieldsRDD.map(x => MusicData(x(0),
       x(1),
       x(2),
```

```
      if (x(3).equals("")) 0 else x(3).toLong,
      if (x(4).equals("")) 0 else x(4).toLong,
      if (x(5).equals("")) 0 else x(5).toLong,
      x(6),
      x(7),
      x(8),
      if (x(9).equals("")) "0" else x(9),
      if (x(10).equals("")) "0" else x(10))).toDF



   log.info("Number of records for Mobile Music Data =" + recordDF.count)
   log.info("Showing records for Mobile Music Data ")
   recordDF.show

   return recordDF

  }

}
```

MODULE3: Enrich and validate data

Step1: Define a class MusicDataEnricher which does the enrichment and validation of fields in dataframe using UDF

```
// Define package final_project and import all the dependent packages
package final_project

import org.apache.spark._
import org.apache.spark.sql.DataFrame
import scala.collection.mutable.ListBuffer
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.types._
import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }
import org.apache.spark.broadcast.Broadcast
import org.apache.spark.sql.functions.udf
import scala.collection.mutable.HashMap

// Define class MusicDataEnricher with the parmeters of all SparkContext, DataFrame and all the
 // broadcast maps which has lookup data
class MusicDataEnricher( allDataFrameParam: DataFrame,
broadcastStndIdGeoCdMapParam:Broadcast[Map[String, String]],
broadcastSongArtistMapParam:Broadcast[Map[String, String]], broadcastUserArtistMapParam:
Broadcast[Map[String, String]], broadcastUserSubscriptionParam: Broadcast[Map[String, (Long, Long)]])
extends Serializable {
        val allDataFrame: DataFrame =allDataFrameParam
 // val stndIdGeoCdMap:HashMap[String, String] = stndIdGeoCdMapParam
        val broadcastStndIdGeoCdMap:Broadcast[Map[String, String]] =
broadcastStndIdGeoCdMapParam
        val broadcastSongArtistMap:Broadcast[Map[String, String]] = broadcastSongArtistMapParam
        val broadcastUserArtistMap:Broadcast[Map[String, String]] = broadcastUserArtistMapParam
// The UDF method  fillNullValueGeoCd wlll take stationId and geoCd as parameter and if geoCd is not
 // blank then it will take as it is. If geoCd is blank it will use the lookup map  broadcastStndIdGeoCdMap
// using stationId, get the geoCd. If it is not there record will be marked as Invalid
```

```
// A new field modified_Geo_cd will be added to the dataframe
        def fillNullValueGeoCd = udf((stationId: String, geoCd: String) => {

                if (!geoCd.equals("")) geoCd
                else {
                  //  val geoCdVal = stndIdGeoCdMap.get(stationId).getOrElse("Invalid")
                  val geoCdVal = broadcastStndIdGeoCdMap.value.get(stationId).getOrElse("Invalid")
                        geoCdVal
                }
        })

// The UDF method  fillNullValueArtistId wlll take songId and artistId as parameter and if artistId is not
 // blank then it will take as it is. If artistId is blank it will use the lookup map  broadcastSongArtistMap
// using songId, get the artistId. If it is not there record will be marked as Invalid
// A new field modified_Artist_id will be added to the dataframe

def fillNullValueArtistId = udf((songId: String, artistId: String) => {
        if (!artistId.equals("")) artistId
        else {
                val artistIdVal = broadcastSongArtistMap.value.get(songId).getOrElse("Invalid")
   artistIdVal
        }
})

// The UDF method  findFollowers wlll take userId and artistId as parameter.
// Based on usedId from the map broadcastUserArtistMap, artistList is retrieved. If aritstId list is blank
// then 0 will be retuned. If artistList is not  blank then it will be split based on & and create a array
 // artistArray. If it contains  using artistId, 1 will be returned, else 0 will be returned
// A new field follower will be added to the dataframe
```

```
def findFollowers = udf((userId: String, artistId: String) => {
        val artistList = broadcastUserArtistMap.value.get(userId).getOrElse("")
if (artistList.equals("")) "0"
        else {
                var artistArray = artistList.split("&")
                                if (artistArray contains artistId) "1"
                                        else "0"

        }

})
// The UDF method  findSubscribers wlll take userId and starts as parameter. Basedon userId
// lokkup is done on broadcastUserSubscription.value.get(userId). If subscription does not exits, it
// returns 0. If it exists and starts is in between subscription start time and end time return 1 else
/ /retun 0. A new field subscribed is added to the dataframe

def findSubscribers = udf((userId: String, startTs: Long) => {

        val subscriptionTuple= broadcastUserSubscription.value.get(userId).getOrElse((0L, 0L))
 if (subscriptionTuple._1 ==0 && subscriptionTuple._2 == 0 ) "0"
        else if (startTs >= subscriptionTuple._1  && startTs <=  subscriptionTuple._2) "1"
        else "0"

})
```

```
// Using UDF validate records validation is done. If userId, songId are blank return 0. If modifiedArtistId
// or modifiedGeoCdi si Invalid return 0,. If timestamp or start_ts is 0 then 0. If end_ts is less than //
start_ts return 0. Else return 1. Add a new field isValid to the dataframe

def validateRecords = udf((userId: String, songId: String, modifiedArtistId: String, modifiedGeoCd: String,
timestamp:Long, start_ts:Long, end_ts:Long) => {

        if (userId.equals("")) "0"

                else if (songId.equals("")) "0"

                        else if (modifiedArtistId.equals("Invalid")) "0"

                         else if (modifiedGeoCd.equals("Invalid")) "0"

                                 else if (timestamp == 0) "0"

                                         else if(start_ts == 0) "0"

                                                 else if (end_ts < start_ts) "0"

                                                         else "1"

})




// The method enrichData will execute all the UDFs defined above and enrich dataframe with new fields

def enrichData():DataFrame = {


 var newDataFrame = allDataFrame.withColumn("modified_Geo_cd",
fillNullValueGeoCd(allDataFrame("Station_id"), allDataFrame("Geo_cd")))

 newDataFrame = newDataFrame.withColumn("modified_Artist_id",
fillNullValueArtistId(newDataFrame("Songs_id"), newDataFrame("Artist_id")))

 newDataFrame = newDataFrame.withColumn("follower",  findFollowers(newDataFrame("User_id"),
newDataFrame("modified_Artist_id")))

 newDataFrame = newDataFrame.withColumn("subscribed", findSubscribers(newDataFrame("User_id"),
newDataFrame("Start_ts")))

 newDataFrame = newDataFrame.withColumn("isValid", validateRecords (newDataFrame("User_id"),
newDataFrame("Songs_id"), newDataFrame("modified_Artist_id"),
```

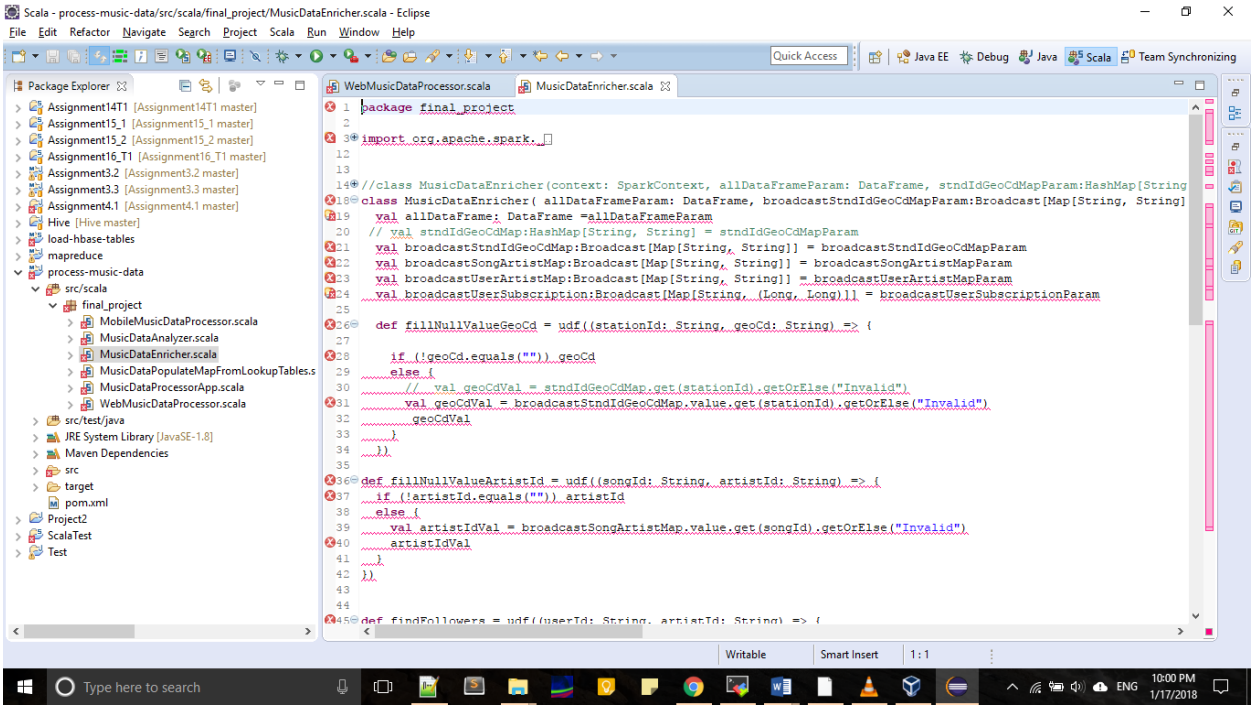
newDataFrame("modified_Geo_cd"), newDataFrame("Timestamp"), newDataFrame("Start_ts"), newDataFrame("End_ts")))

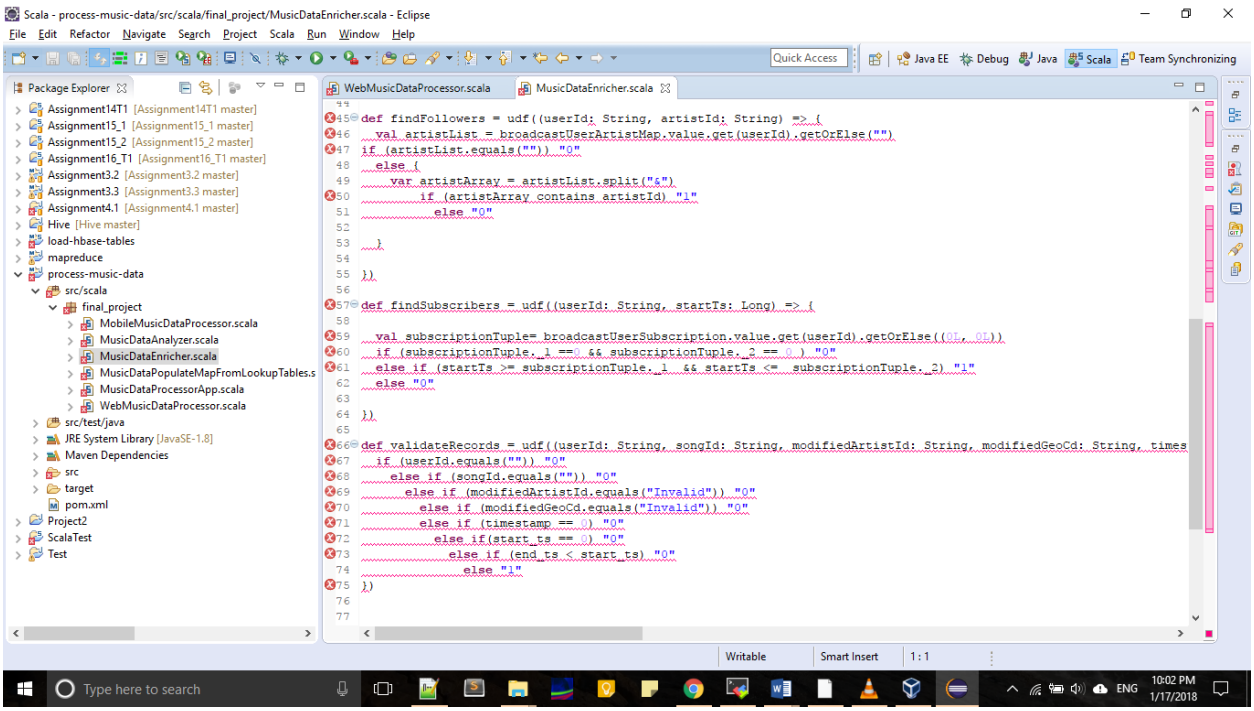
return newDataFrame

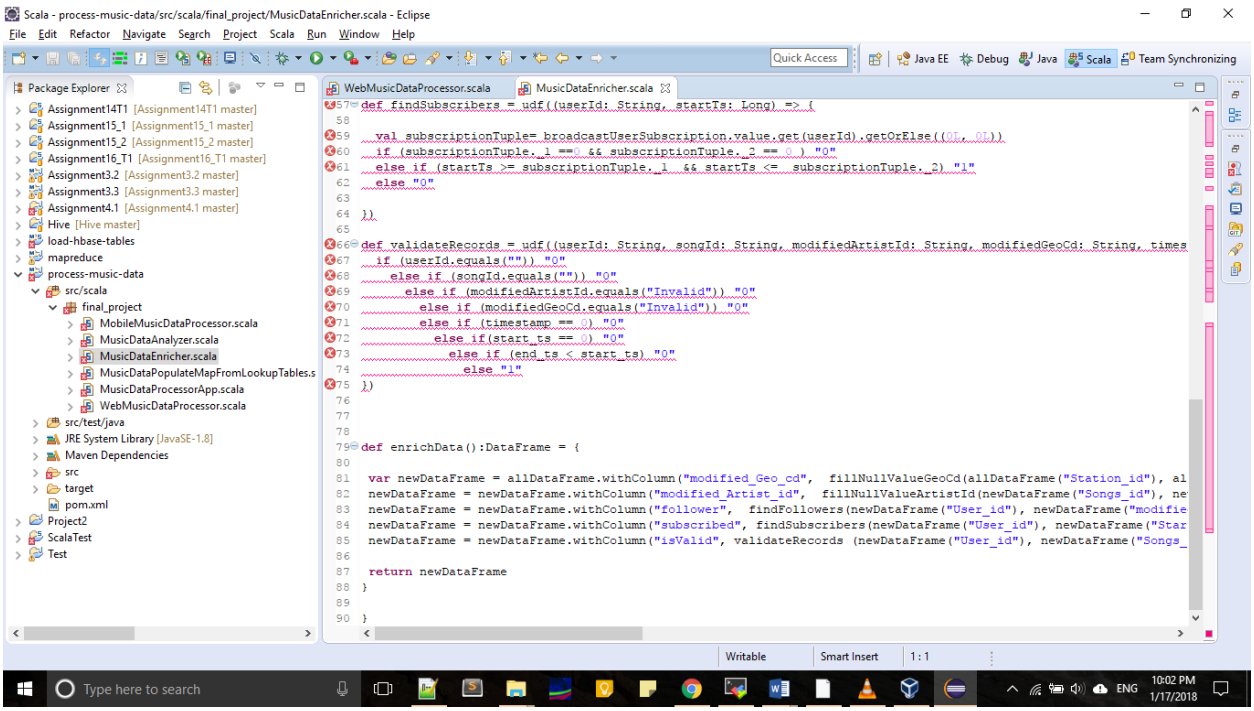
}

}

Screenshot is as below:







MODULE4: Analyze the data

- Analyze the data using defining MusicDataAnalyzer class, which will execute sql queries and the result will be stored as reports in HDFS

```
// Define the package final_project and import all the dependent packages


package final_project

import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }

import org.apache.spark.sql.AnalysisException

import org.apache.spark._

import org.apache.spark.sql.SQLContext

import org.apache.spark.sql.DataFrame

import scala.collection.mutable.HashMap


// Define the class MusicDataAnalyzer with parameters SparkContext, SQLContext and DataFrame


class MusicDataAnalyzer(context: SparkContext, sqc: SQLContext, musicDataDFParam: DataFrame)
extends Serializable {


  val musicDataDF: DataFrame = musicDataDFParam

  val sc = context

  val sqlContext = sqc


// Define logger and reportBasePath and currentTimestamp

  val log = LogManager.getRootLogger

  log.setLevel(Level.INFO)

  val reportBasePath = "/user/acadgild/project_music_data_analysis/reports/"

  val currentTimestamp = System.currentTimeMillis().toString


// Define method analyze which will in run call all the methods for analysis
```

```
// While calling method, each one is put in try catch block so that failing one does not impact others

  def analyze() = {
    import sqlContext.implicits._

// Define temporary table MusicDataDetailed on the dataframe
    musicDataDF.registerTempTable("MusicDataDetailed")
    log.info("Before calling getAllRecords()")

    try {
      getAllRecords()
    } catch {
      case e: Exception => log.error("Exception got while calling getAllRecords: " + e)
    }
    log.info("Before calling getTop10Stations()")

    try {
      getTop10Stations()
    } catch {
      case e: Exception => log.error("Exception got while calling getTop10Stations: " + e)
    }
    log.info("Before calling getMusicDurtionByUser()")

    try {
      getMusicDurtionByUserType()
    } catch {
      case e: Exception => log.error("Exception got while calling getTop10Stations: " + e)
    }
    log.info("Before calling getTo10ConnectedArtists()")
```

```
    try {
      getTop10ConnectedArtists()
    } catch {
      case e: Exception => log.error("Exception got while calling getTop10ConnectedArtists: " + e)
    }

    log.info("Before calling  getTop10UnsubscribedUsers()")


    try {
      getTop10UnsubscribedUsers()
    } catch {
      case e: Exception => log.error("Exception got while calling getTop10UnsubscribedUsers(): " + e)
    }

    log.info("Before calling  getTo10SongsHavignMaximumRevenue()")
    try {
      getTop10SongsHavingMaximumRevenue()
    } catch {
      case e: Exception => log.error("Exception got while calling
getTop10SongsHavingMaximumRevenue(): " + e)
    }

  }

// Method getAllRecords will select all the records from MusicDataDetailed

    def getAllRecords() {
```

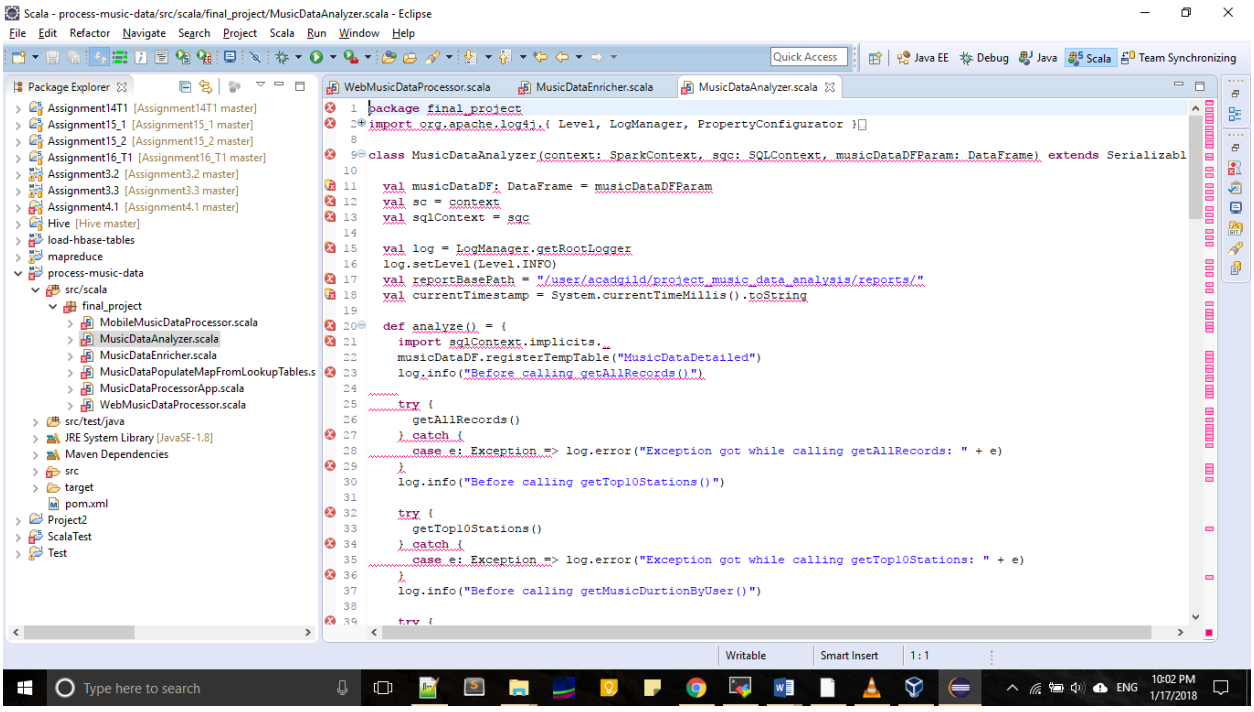
```
    log.info(" Get All records")
// Run the query to get the result
    val df = sqlContext.sql("SELECT * FROM MusicDataDetailed ")

   df.show()

// Store the result as a single file in HDFS with reportPath and report name MusicDataAllRecords concatenated with current timestamp
   val df1 = df.repartition(1)
   df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath + "/MusicDataAllRecords_" + currentTimestamp)



 }

// The method getTop10Stations will return top 10 stations were maximum songs are played which are
// liked by unique user
  def getTop10Stations() {
   log.info(" Top 10 Music Stations where maximum numbers of songs played which are liked by unique users")

// Execute query to get Station_id User_id and count of music played Group by Station_id, User_id and
// Likes is 1 and isValid is 1
   sqlContext.sql("SELECT Station_id, User_id, count(*) AS music_count FROM MusicDataDetailed "
     + " WHERE Likes='1' AND isValid='1' GROUP BY Station_id, User_id")
     .registerTempTable("MusicCountByStation")

// Execute query to get unique unique count for user liked, so music_count is greater than 1, it is
// considered 1
```

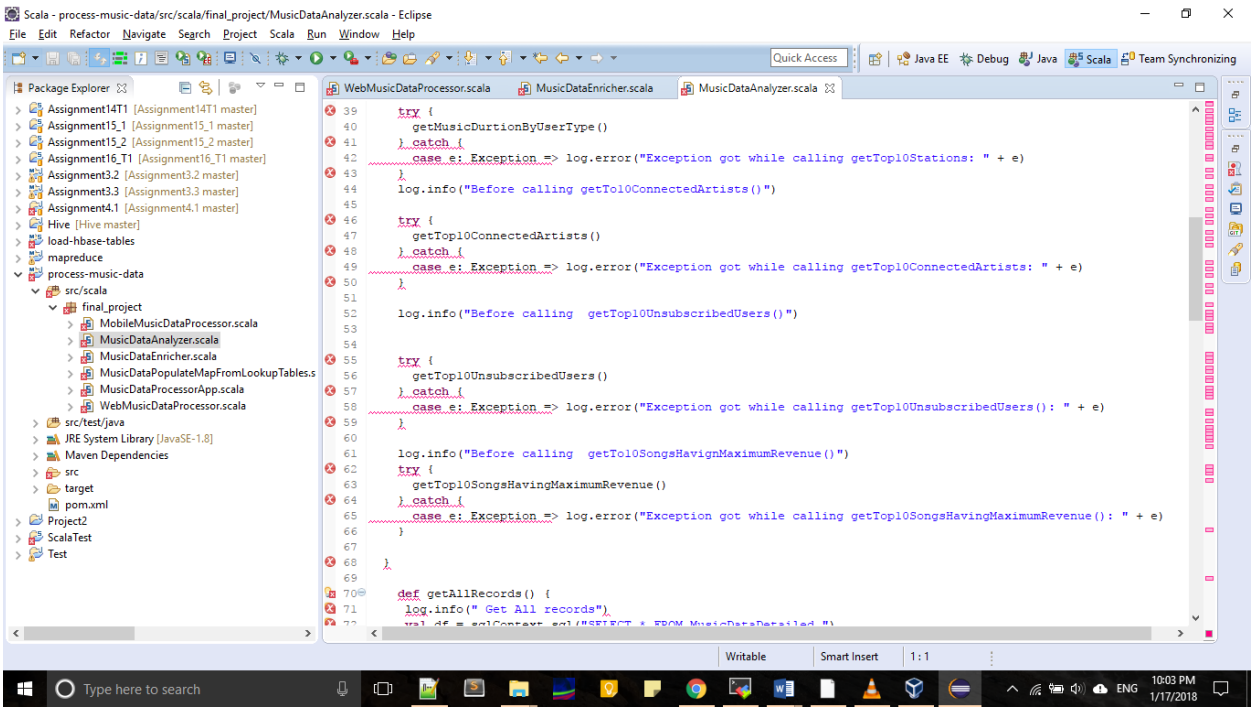
```
   sqlContext.sql("SELECT Station_id, User_id, CASE WHEN music_count> 1 THEN 1 ELSE music_count
END "
     + " AS unique_music_count FROM MusicCountByStation")
     .registerTempTable("UniqueMusicCountByStation")
// Using sum method of SQL, aggregate the total music count group by Station_id and order
// total_music_count desceding. Take first 10 records

   val df = sqlContext.sql("SELECT Station_id, sum(unique_music_count) AS total_music_count "
     + " FROM UniqueMusicCountByStation GROUP BY Station_id ORDER BY total_music_count "
     + " DESC LIMIT 10 ")
   df.show()

// Store the query output to HDFS as a csv report Top10Stations concatenated with timestamp in
 // reportBasePath
   val df1 = df.repartition(1)
   df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath +
"/Top10Stations_" + currentTimestamp)

  }

// The method getMusicDurtionByUserType returns total lke music by each category of user Subscribed
// or Unsubscribed
  def getMusicDurtionByUserType() {
   log.info(" Total duration of Songs played by Subscibed and Unsubsribed Users")

   sqlContext.sql("SELECT CASE WHEN subscribed='1' THEN 'Subscribed' ELSE 'Unsubscribed' END AS
User_type, (End_ts -Start_ts) AS duration  "
     + " FROM MusicDataDetailed WHERE isValid='1'")
     .registerTempTable("UserTypeDuration")
```

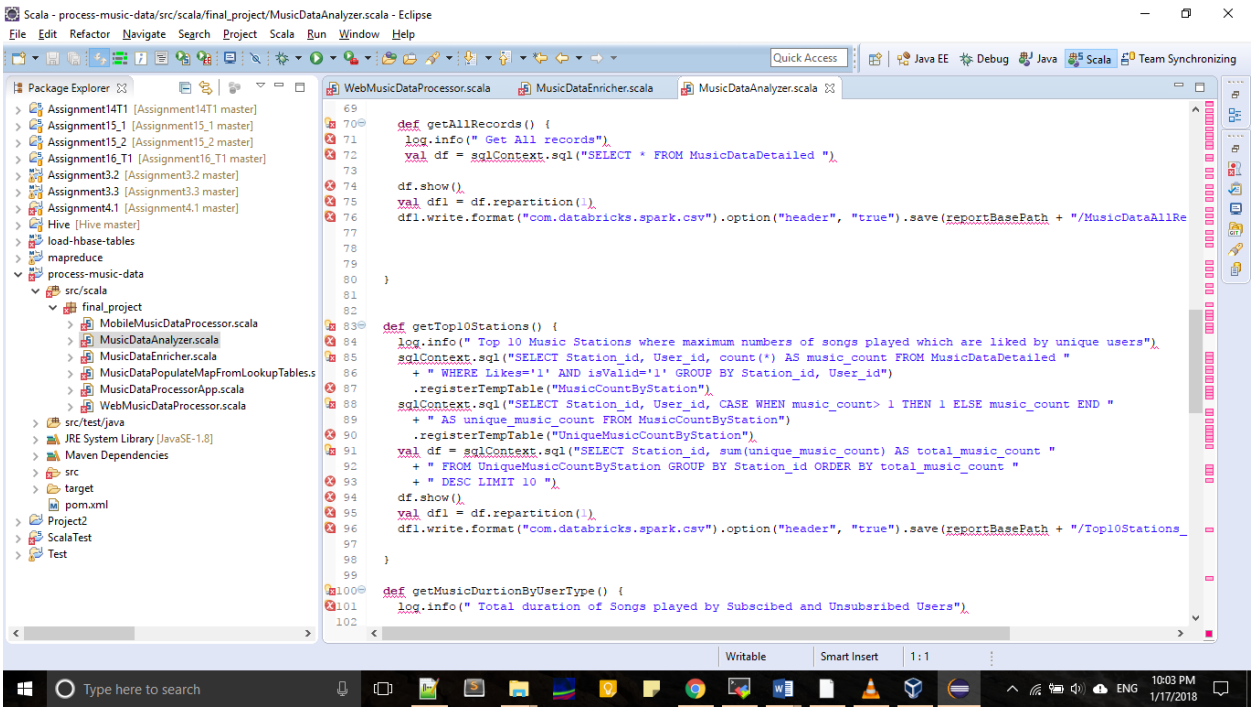
```
   val df = sqlContext.sql("SELECT User_type, SUM(duration) AS total_duration_milliseconds FROM
UserTypeDuration "
     + " GROUP BY User_type ORDER BY total_duration_milliseconds DESC")
   df.show()
   val df1 = df.repartition(1)


   df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath +
"/MusicDurationByUserType_" + currentTimestamp)




  }
// The method getTop10ConnectedArtists return top 10 connected artists who are followed by user
  def getTop10ConnectedArtists() {
   log.info(" Top 10 Connected Artists")
   sqlContext.sql("SELECT Artist_id, User_id, count(*) AS music_count FROM MusicDataDetailed "
     + " WHERE follower='1' AND isValid='1' GROUP BY Artist_id, User_id")
     .registerTempTable("MusicCountByArtist")
   sqlContext.sql("SELECT Artist_id, User_id, CASE WHEN music_count> 1 THEN 1 ELSE music_count END
"
     + " AS unique_music_count FROM MusicCountByArtist")
     .registerTempTable("UniqueMusicCountByUser")
   val df = sqlContext.sql("SELECT Artist_id, sum(unique_music_count) AS total_music_count "
     + " FROM UniqueMusicCountByUser GROUP BY Artist_id ORDER BY total_music_count "
     + " DESC LIMIT 10 ")
   df.show()
   val df1 = df.repartition(1)
   df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath +
"/Top10ConnectedArtists_" + currentTimestamp)
```

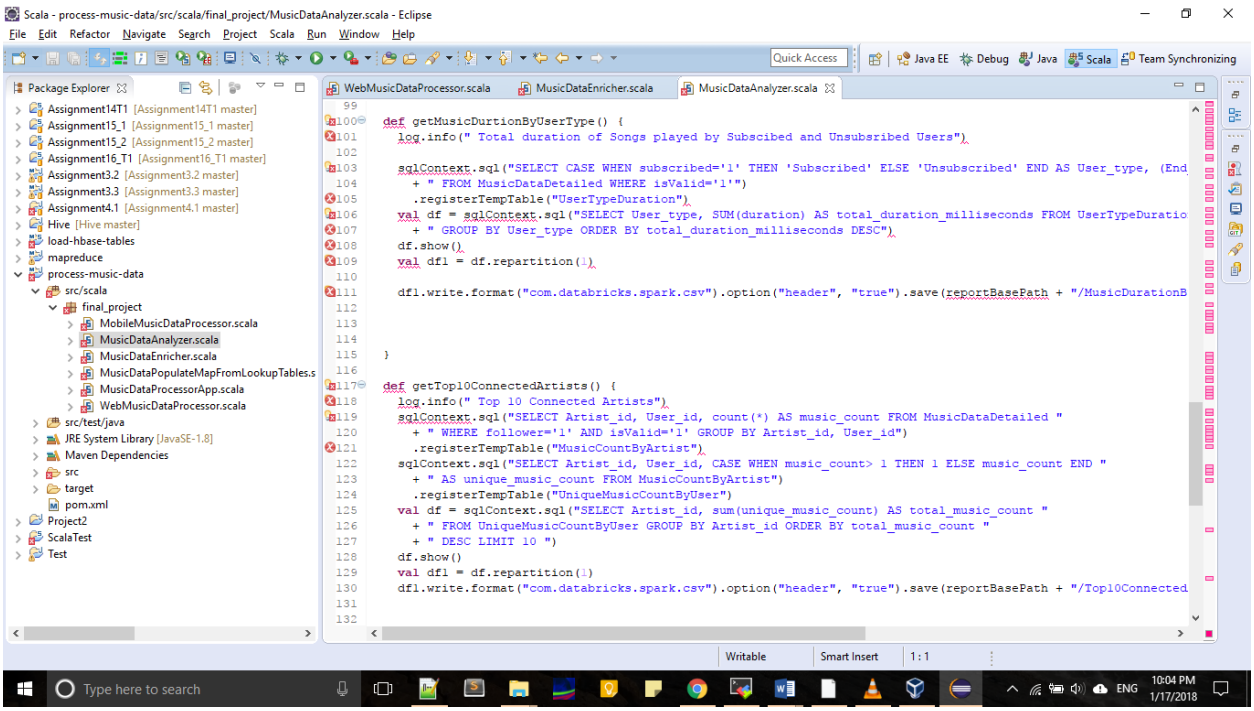
```
  }
// The method getTop10SongsHavingMaximumRevenue returns top 10 songs having maximum royalty r
//revenue
  def getTop10SongsHavingMaximumRevenue() {
    log.info(" Top 10 Songs Having maximum revenue")


    sqlContext.sql("SELECT Songs_id, CASE WHEN End_ts is NOT NULL AND Start_ts is NOT NULL and
End_ts > Start_ts THEN End_ts - Start_ts ELSE 0 END AS duration FROM MusicDataDetailed "
      + " WHERE (Likes='1' OR Song_end_type = '0') AND isValid='1' ")
      .registerTempTable("SongDuration")
    val df = sqlContext.sql("SELECT Songs_id, SUM(duration) AS total_duration_milliseconds FROM
SongDuration "
      + " GROUP BY Songs_id ORDER BY total_duration_milliseconds DESC LIMIT 10")
    df.show()
    val df1 = df.repartition(1)
    df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath +
"/Top10SongsHavignMaximumRevenue_" + currentTimestamp)



  }


// The method getTop10UnsubscribedUsers will retrun top 10 unsubscribed users
  def getTop10UnsubscribedUsers() {
    log.info(" Top 10 Unsubscribed Users")


    sqlContext.sql("SELECT User_id,CASE WHEN End_ts is NOT NULL AND Start_ts is NOT NULL and End_ts
> Start_ts THEN End_ts - Start_ts ELSE 0 END AS duration FROM MusicDataDetailed "
      + " WHERE subscribed='0' AND isValid='1'")
      .registerTempTable("SongDuration")
```

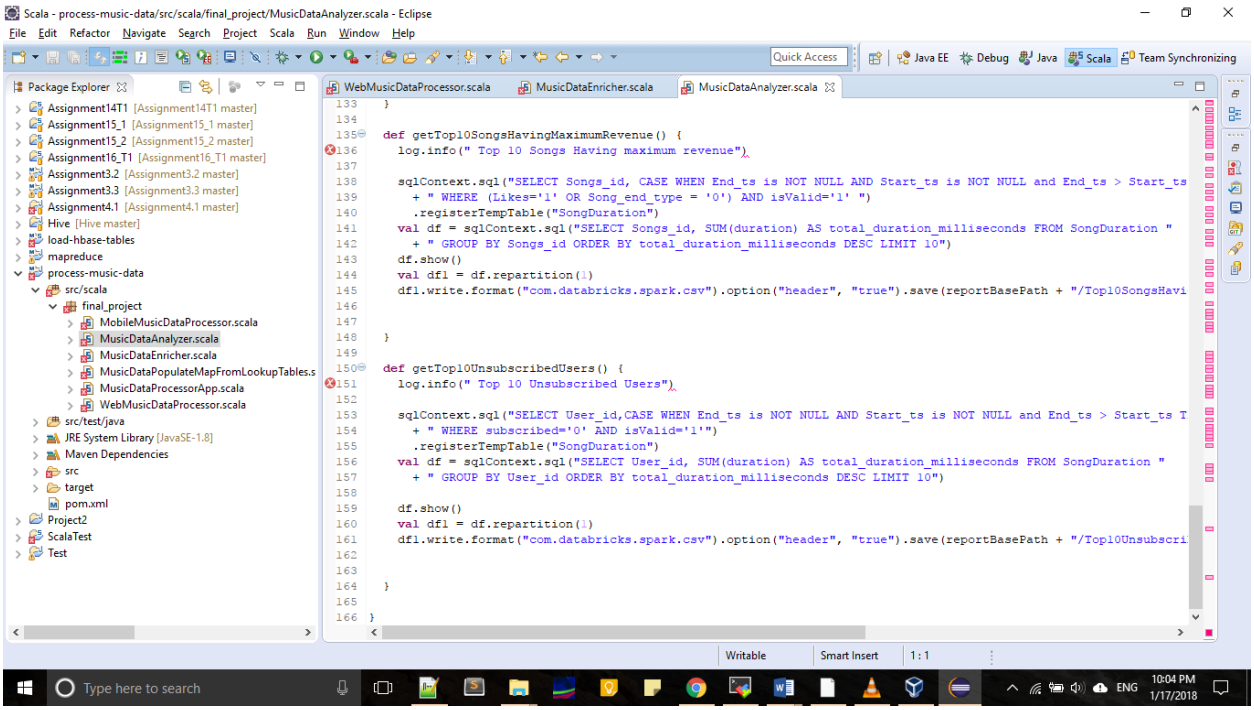
```
    val df = sqlContext.sql("SELECT User_id, SUM(duration) AS total_duration_milliseconds FROM SongDuration "
      + " GROUP BY User_id ORDER BY total_duration_milliseconds DESC LIMIT 10")

    df.show()

    val df1 = df.repartition(1)

    df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath + "/Top10UnsubscribedUsers_" + currentTimestamp)


  }

}
```




Scala - process-music-data/src/scala/final_project/MusicDataAnalyzer.scala - Eclipse
File Edit Refactor Navigate Search Project Scala Run Window Help
Quick Access
Java EE Debug Java Scala Team Synchronizing
Package Explorer
Assignment14T1 [Assignment14T1 master]
Assignment15_1 [Assignment15_1 master]
Assignment15_2 [Assignment15_2 master]
Assignment16_T1 [Assignment16_T1 master]
Assignment3.2 [Assignment3.2 master]
Assignment3.3 [Assignment3.3 master]
Assignment4.1 [Assignment4.1 master]
Hive [Hive master]
load-hbase-tables
mapreduce
process-music-data
src/scala
final_project
MobileMusicDataProcessor.scala
MusicDataAnalyzer.scala
MusicDataEnricher.scala
MusicDataPopulateMapFromLookupTables.s
MusicDataProcessorApp.scala
WebMusicDataProcessor.scala
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
pom.xml
Project2
ScalaTest
Test
WebMusicDataProcessor.scala
MusicDataEnricher.scala
MusicDataAnalyzer.scala
try {
getMusicDurtionByUserType()
} catch {
case e: Exception => log.error("Exception got while calling getTop10Stations: " + e)
}
log.info("Before calling getTo10ConnectedArtists()")
try {
getTop10ConnectedArtists()
} catch {
case e: Exception => log.error("Exception got while calling getTop10ConnectedArtists: " + e)
}
log.info("Before calling getTop10UnsubscribedUsers()")
try {
getTop10UnsubscribedUsers()
} catch {
case e: Exception => log.error("Exception got while calling getTop10UnsubscribedUsers(): " + e)
}
log.info("Before calling getTo10SongsHavignMaximumRevenue()")
try {
getTop10SongsHavingMaximumRevenue()
} catch {
case e: Exception => log.error("Exception got while calling getTop10SongsHavingMaximumRevenue(): " + e)
}
}
def getAllRecords() {
log.info(" Get All records")
Writable
Smart Insert
1 : 1
Type here to search
ENG
10:03 PM
1/17/2018


Scala - process-music-data/src/scala/final_project/MusicDataAnalyzer.scala - Eclipse
File Edit Refactor Navigate Search Project Scala Run Window Help
Quick Access
Java EE Debug Java Scala Team Synchronizing
Package Explorer
Assignment14T1 [Assignment14T1 master]
Assignment15_1 [Assignment15_1 master]
Assignment15_2 [Assignment15_2 master]
Assignment16_T1 [Assignment16_T1 master]
Assignment3.2 [Assignment3.2 master]
Assignment3.3 [Assignment3.3 master]
Assignment4.1 [Assignment4.1 master]
Hive [Hive master]
load-hbase-tables
mapreduce
process-music-data
src/scala
final_project
MobileMusicDataProcessor.scala
MusicDataAnalyzer.scala
MusicDataEnricher.scala
MusicDataPopulateMapFromLookupTables.s
MusicDataProcessorApp.scala
WebMusicDataProcessor.scala
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
pom.xml
Project2
ScalaTest
Test
WebMusicDataProcessor.scala
MusicDataEnricher.scala
MusicDataAnalyzer.scala
def getAllRecords() {
log.info(" Get All records")
val df = sqlContext.sql("SELECT * FROM MusicDataDetailed ")
df.show()
val df1 = df.repartition(1)
df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath + "/MusicDataAllRe
}
def getTop10Stations() {
log.info(" Top 10 Music Stations where maximum numbers of songs played which are liked by unique users")
sqlContext.sql("SELECT Station_id, User_id, count(*) AS music_count FROM MusicDataDetailed "
+ " WHERE Likes='1' AND isValid='1' GROUP BY Station_id, User_id")
.registerTempTable("MusicCountByStation")
sqlContext.sql("SELECT Station_id, User_id, CASE WHEN music_count> 1 THEN 1 ELSE music_count END "
+ " AS unique_music_count FROM MusicCountByStation")
.registerTempTable("UniqueMusicCountByStation")
val df = sqlContext.sql("SELECT Station_id, sum(unique_music_count) AS total_music_count "
+ " FROM UniqueMusicCountByStation GROUP BY Station_id ORDER BY total_music_count "
+ " DESC LIMIT 10 ")
df.show()
val df1 = df.repartition(1)
df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath + "/Top10Stations_
}
def getMusicDurtionByUserType() {
log.info(" Total duration of Songs played by Subscibed and Unsubsribed Users")
Writable
Smart Insert
1 : 1
Type here to search
ENG
10:03 PM
1/17/2018





Module5: MusicDataPopulateMapFromLookupTables : Load the maps from HBase tables

To load the HBase table, I used the class MusicDataPopulateMapFromLookupTables

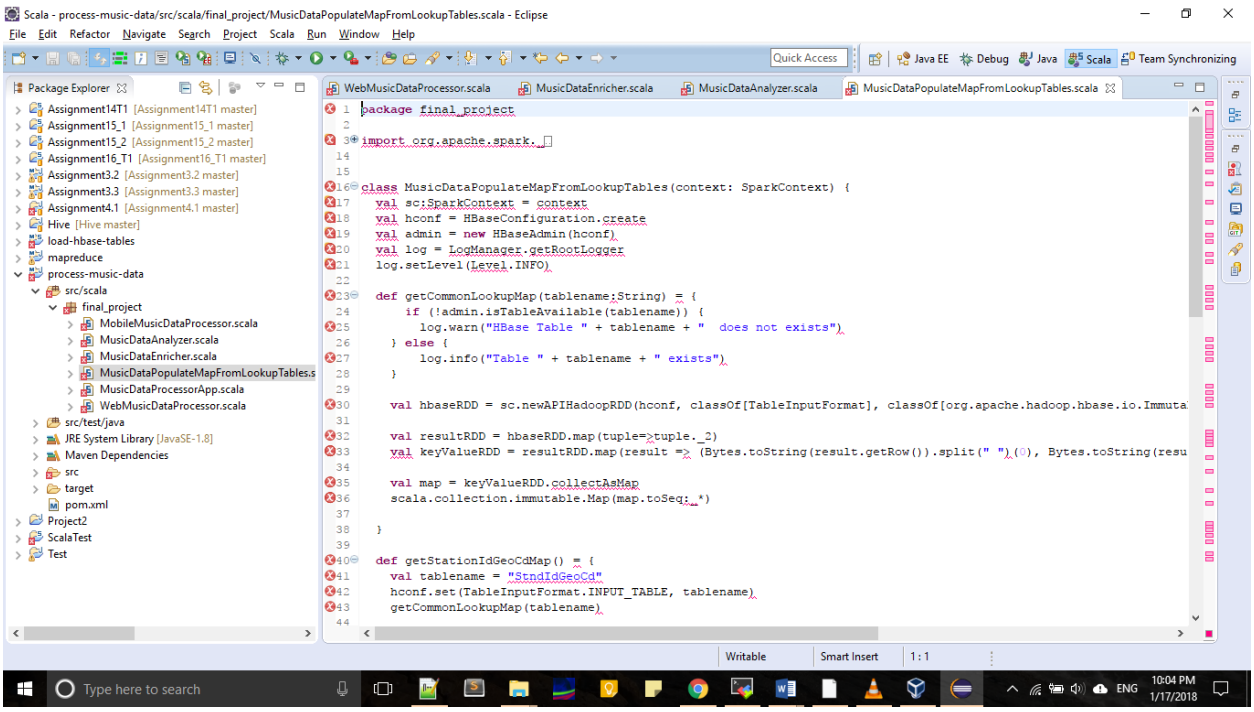
```
// Import the dependent packages
package final_project
import org.apache.spark._
import org.apache.spark.sql.SQLContext
import org.apache.hadoop.hbase.util.Bytes
import org.apache.hadoop.hbase.client.{Put,HTable}
import org.apache.hadoop.hbase.HBaseConfiguration
import org.apache.hadoop.hbase.HTableDescriptor
import org.apache.hadoop.hbase.HColumnDescriptor
import org.apache.hadoop.hbase.client.HBaseAdmin
import org.apache.hadoop.hbase.client.HTable
import org.apache.hadoop.hbase.mapreduce.TableInputFormat
import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }

// Create class MusicDataPopulateMapFromLookupTables with parameters
class MusicDataPopulateMapFromLookupTables(context: SparkContext) {
 val sc:SparkContext = context
 val hconf = HBaseConfiguration.create
 val admin = new HBaseAdmin(hconf)
 val log = LogManager.getRootLogger
 log.setLevel(Level.INFO)
  // Create map from HBase table with key and value as column value
 def getCommonLookupMap(tablename:String) = {
    if (!admin.isTableAvailable(tablename)) {
      log.warn("HBase Table " + tablename + "  does not exists")
  } else {
      log.info("Table " + tablename + " exists")
  }
```

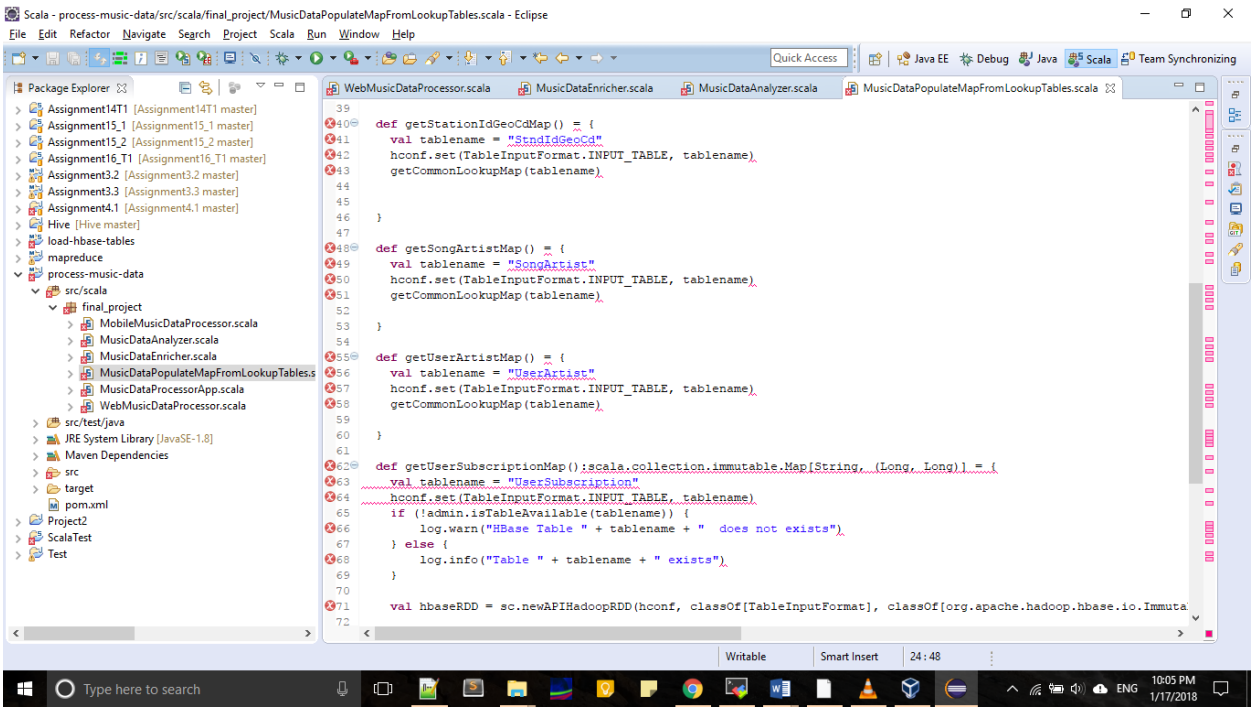
```
   val hbaseRDD = sc.newAPIHadoopRDD(hconf, classOf[TableInputFormat],
classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
classOf[org.apache.hadoop.hbase.client.Result])


   val resultRDD = hbaseRDD.map(tuple=>tuple._2)

   val keyValueRDD = resultRDD.map(result => (Bytes.toString(result.getRow()).split(" ")(0),
Bytes.toString(result.value)))


   val map = keyValueRDD.collectAsMap

   scala.collection.immutable.Map(map.toSeq:_*)


  }

  // Populate Map with key StationId and GeoCd from table StndIdGeoCd

  def getStationIdGeoCdMap() = {

   val tablename = "StndIdGeoCd"

   hconf.set(TableInputFormat.INPUT_TABLE, tablename)

   getCommonLookupMap(tablename)



  }

// Populate Map with key songId and Artist from table SongArtist

  def getSongArtistMap() = {

   val tablename = "SongArtist"

   hconf.set(TableInputFormat.INPUT_TABLE, tablename)

   getCommonLookupMap(tablename)


  }

// Populate Map with key userId and Artist List from table UserArtist
```

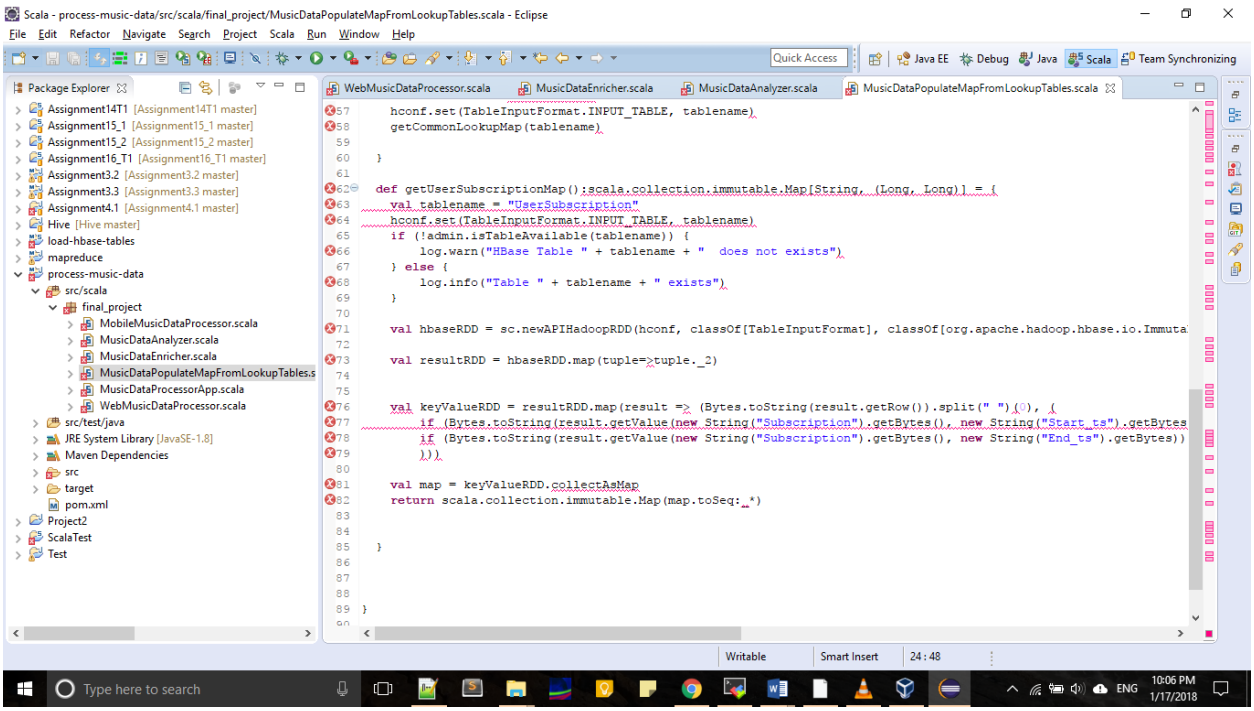
```
 def getUserArtistMap() = {
  val tablename = "UserArtist"
  hconf.set(TableInputFormat.INPUT_TABLE, tablename)
  getCommonLookupMap(tablename)

 }
 // Populate map with Key as UserId and Value as tuple (start_ts, end_ts)
 def getUserSubscriptionMap():scala.collection.immutable.Map[String, (Long, Long)] = {
  val tablename = "UserSubscription"
  hconf.set(TableInputFormat.INPUT_TABLE, tablename)
  if (!admin.isTableAvailable(tablename)) {
     log.warn("HBase Table " + tablename + "  does not exists")
  } else {
     log.info("Table " + tablename + " exists")
  }

  val hbaseRDD = sc.newAPIHadoopRDD(hconf, classOf[TableInputFormat],
classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
classOf[org.apache.hadoop.hbase.client.Result])

  val resultRDD = hbaseRDD.map(tuple=>tuple._2)

  val keyValueRDD = resultRDD.map(result => (Bytes.toString(result.getRow()).split(" ")(0), (
     if (Bytes.toString(result.getValue(new String("Subscription").getBytes(), new
String("Start_ts").getBytes)).equals("")) 0 else Bytes.toString(result.getValue(new
String("Subscription").getBytes(), new String("Start_ts").getBytes)).toLong,
     if (Bytes.toString(result.getValue(new String("Subscription").getBytes(), new
String("End_ts").getBytes)).equals("")) 0 else Bytes.toString(result.getValue(new
String("Subscription").getBytes(), new String("End_ts").getBytes)).toLong
```

```
    )))

  val map = keyValueRDD.collectAsMap
  return scala.collection.immutable.Map(map.toSeq:_*)


 }


}
```

Screenshot is as below:







MODULE6: Main object MusicDataProcessorApp

```
// Import all the dependent packages
package final_project
import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }
import org.apache.spark._
import org.apache.spark.sql.SQLContext
import org.apache.spark.sql.DataFrame
import scala.collection.mutable.HashMap

// define object MusicDataProcessorApp
object MusicDataProcessorApp {

// define method main
  def main(args: Array[String]) {



    val web_file_path = "/data/web/file-1.xml"
    val mobile_file_path = "file:///data/mob/file.txt"

    val log = LogManager.getRootLogger
    log.setLevel(Level.INFO)

// Define all the contexts
    val conf = new SparkConf().setAppName("MusicDataProcessApp").setMaster("local[2]")
    val sc = new SparkContext(conf)
    val sqlContext = new SQLContext(sc)
    import sqlContext.implicits._

    sc.setLogLevel("ERROR")
    // Populate the map from StationId GeoCd and create a broadcast object
```

```
    val populateMusicDataMap = new MusicDataPopulateMapFromLookupTables(sc)
    val stndIdGeoCdMap:Map[String, String] = populateMusicDataMap.getStationIdGeoCdMap()
    val broadcastStndIdGeoCdMap = sc.broadcast(stndIdGeoCdMap)

    // Populate the map songArtistMap  from songId Aritist and create a broadcast object

    val songArtistMap =  populateMusicDataMap.getSongArtistMap()
    val broadcastSongArtistMap = sc.broadcast(songArtistMap)
      // Populate the map userArtistMap  from userId  and Aritist and create a broadcast object

    val userArtistMap = populateMusicDataMap.getUserArtistMap()
    val broadcastUserArtistMap = sc.broadcast(userArtistMap)

      // Populate the map userSubscription from userId  and subscription tuple (start_ts, end_ts) and
// create a broadcast object
    val userSubscription:Map[String, (Long, Long)] = populateMusicDataMap.getUserSubscriptionMap()
    val broadcastuserSubscription = sc.broadcast(userSubscription)

// Call the processData method on WebMusicDataProcessor and get the dataframe webDataFrame
    val webMusicDataProcessor = new WebMusicDataProcessor(web_file_path, sc, sqlContext )

    log.info("Before calling webMusicDataProcessor.processData()")
    val webDataFrame:DataFrame = webMusicDataProcessor.processData()
    log.info("After calling webMusicDataProcessor.processData()")

// Call the processData method on MobileMusicDataProcessor and get the dataframe mobileDataFrame
     val mobileMusicDataProcessor = new MobileMusicDataProcessor(mobile_file_path, sc, sqlContext)

    log.info("Before calling mobileMusicDataProcessor.processData()")
```

```
    val mobileDataFrame:DataFrame = mobileMusicDataProcessor.processData()
    log.info("After calling webMusicDataProcessor.processData()")

// Combine webDataFrame and mobileDataFrame into one allDataFrame
    val allDataFrame:DataFrame = webDataFrame.unionAll(mobileDataFrame)
    log.info("allDataFrame count =" + allDataFrame.count)

// Call the enrichData method on MusicDataEnricher and enhance the data
     val musicDataEnricher = new MusicDataEnricher(allDataFrame, broadcastStndIdGeoCdMap,
broadcastSongArtistMap, broadcastUserArtistMap, broadcastuserSubscription)

    val enrichedAllDataFrame:DataFrame = musicDataEnricher.enrichData()
     log.info("Showing dataframe after enrichment")
     enrichedAllDataFrame.show(100)
      enrichedAllDataFrame.registerTempTable("DetailedMusicData")
     // Call the analyze method on musicDataAnalyzer
     val musicDataAnalyzer = new MusicDataAnalyzer(sc, sqlContext, enrichedAllDataFrame)
     musicDataAnalyzer.analyze()
  }



}
```

COMPILATION: pom.xml file is as below:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

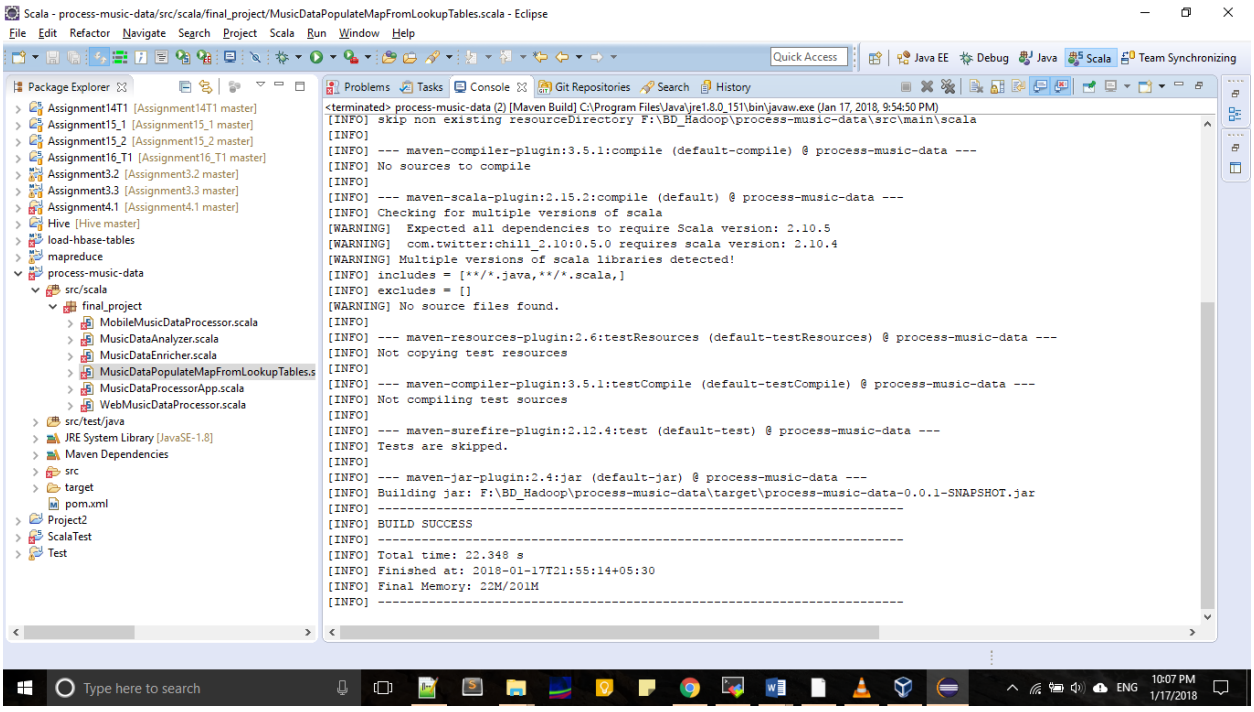
```
<groupId>final_project</groupId>
<artifactId>process_music_data</artifactId>
<version>0.0.1-SNAPSHOT</version>
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <encoding>UTF-8</encoding>
  <scala.version>2.10.5</scala.version>
</properties>
<build>
  <sourceDirectory>src/main/scala</sourceDirectory>
  <resources>
    <resource>
      <directory>src/main/scala</directory>
      <excludes>
        <exclude>**/*.java</exclude>
      </excludes>
    </resource>
  </resources>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.5.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
    <plugin>
```

```
        <groupId>org.scala-tools</groupId>
        <artifactId>maven-scala-plugin</artifactId>
        <version>2.15.2</version>
        <executions>
          <execution>
            <goals>
              <goal>compile</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-core_2.11 -->
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.10</artifactId>
      <version>1.6.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql_2.11 -->
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.10</artifactId>
      <version>1.6.0</version>
    </dependency>
     <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-hive_2.10</artifactId>
```

```
    <version>1.6.0</version>
    <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-client -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>0.98.14-hadoop2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-common -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-common</artifactId>
    <version>0.98.14-hadoop2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-protocol -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-protocol</artifactId>
    <version>0.98.14-hadoop2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-hadoop2-compat -->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-hadoop2-compat</artifactId>
    <version>0.98.14-hadoop2</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-annotations -->
```

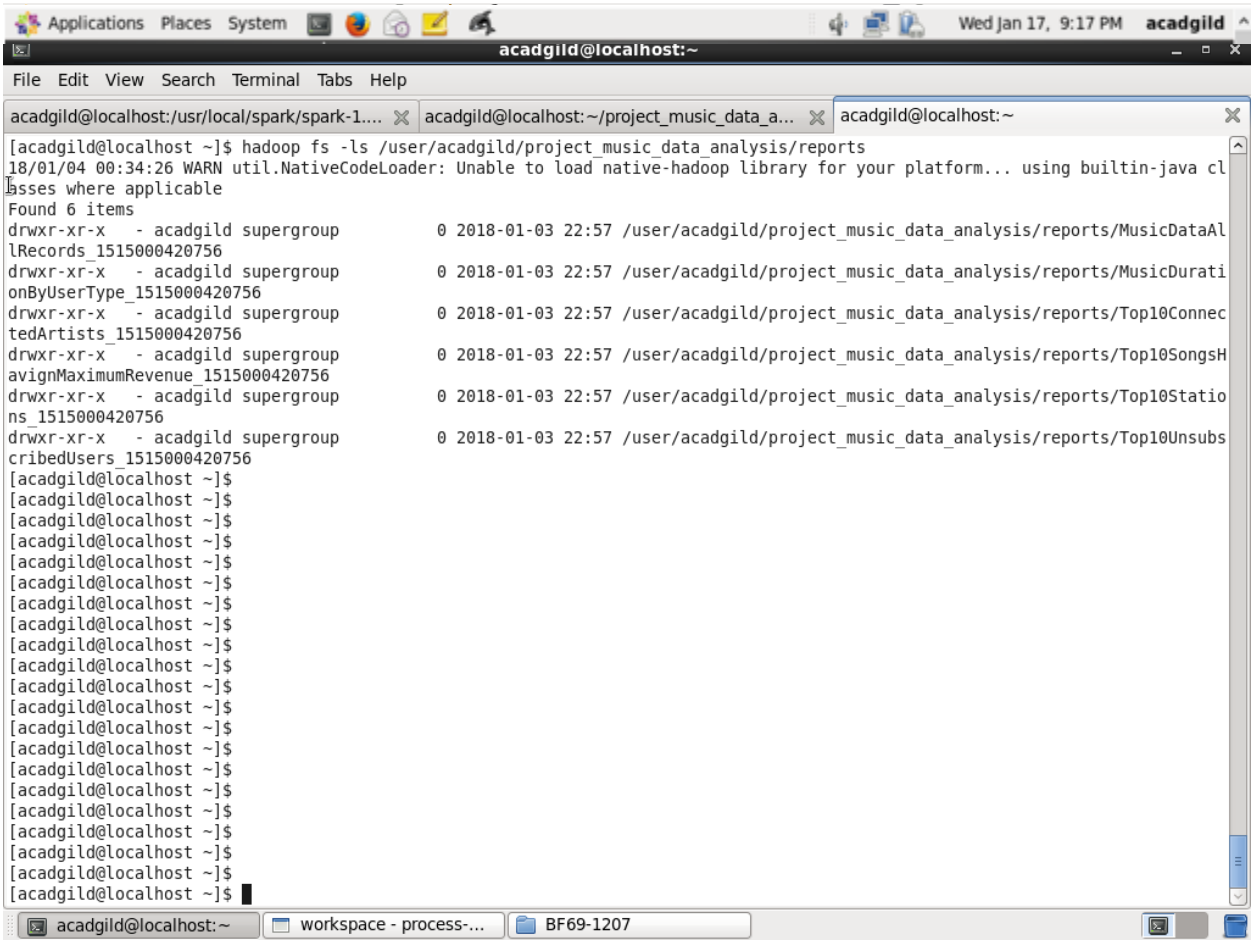
```
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-annotations</artifactId>
            <version>0.98.14-hadoop2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-server -->
        <dependency>
            <groupId>org.apache.hbase</groupId>
            <artifactId>hbase-server</artifactId>
            <version>0.98.14-hadoop2</version>
        </dependency>
</dependencies>

</project>
```

Compilation using Run -> Maven Install, screenshot is as below:

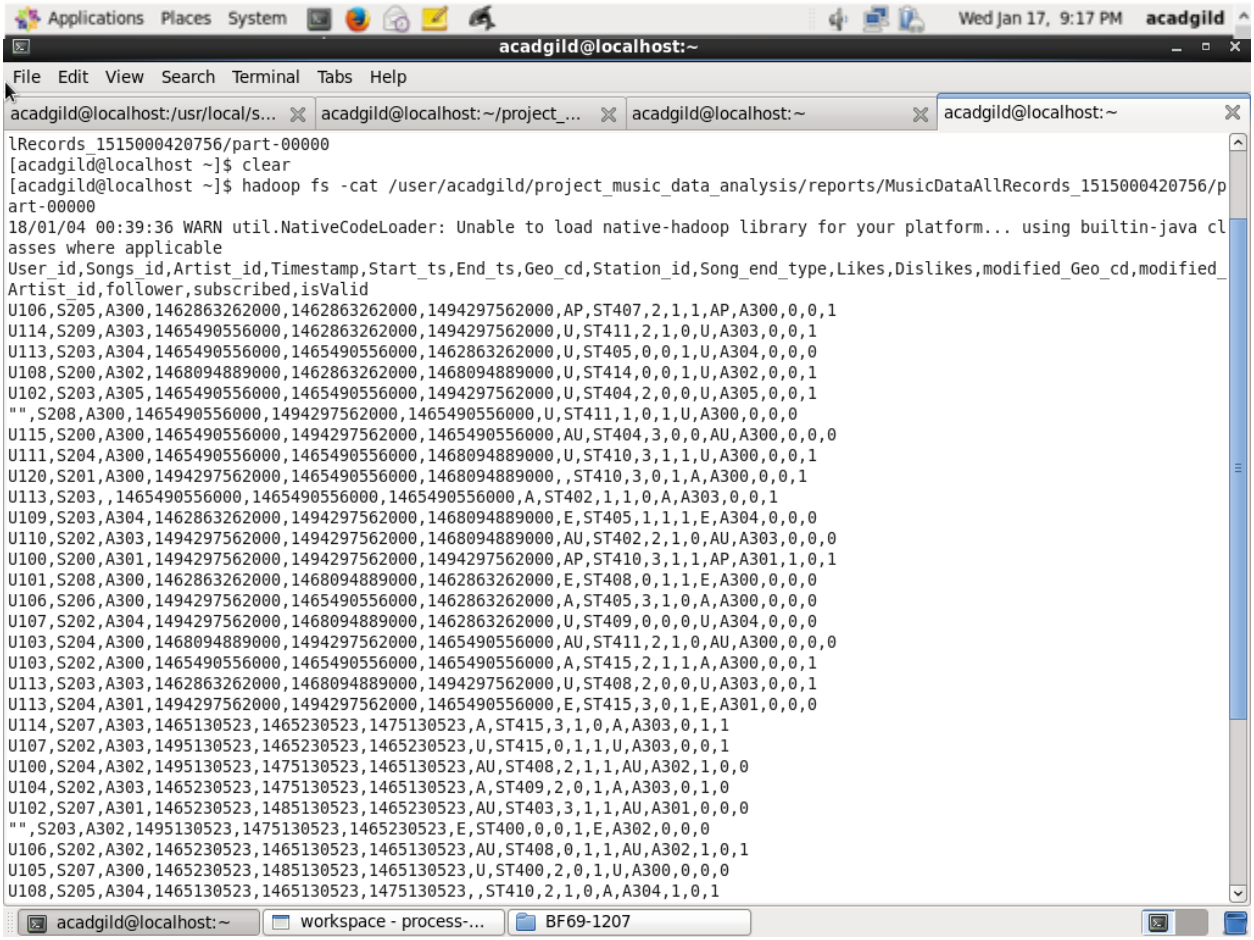


OUTPUT:

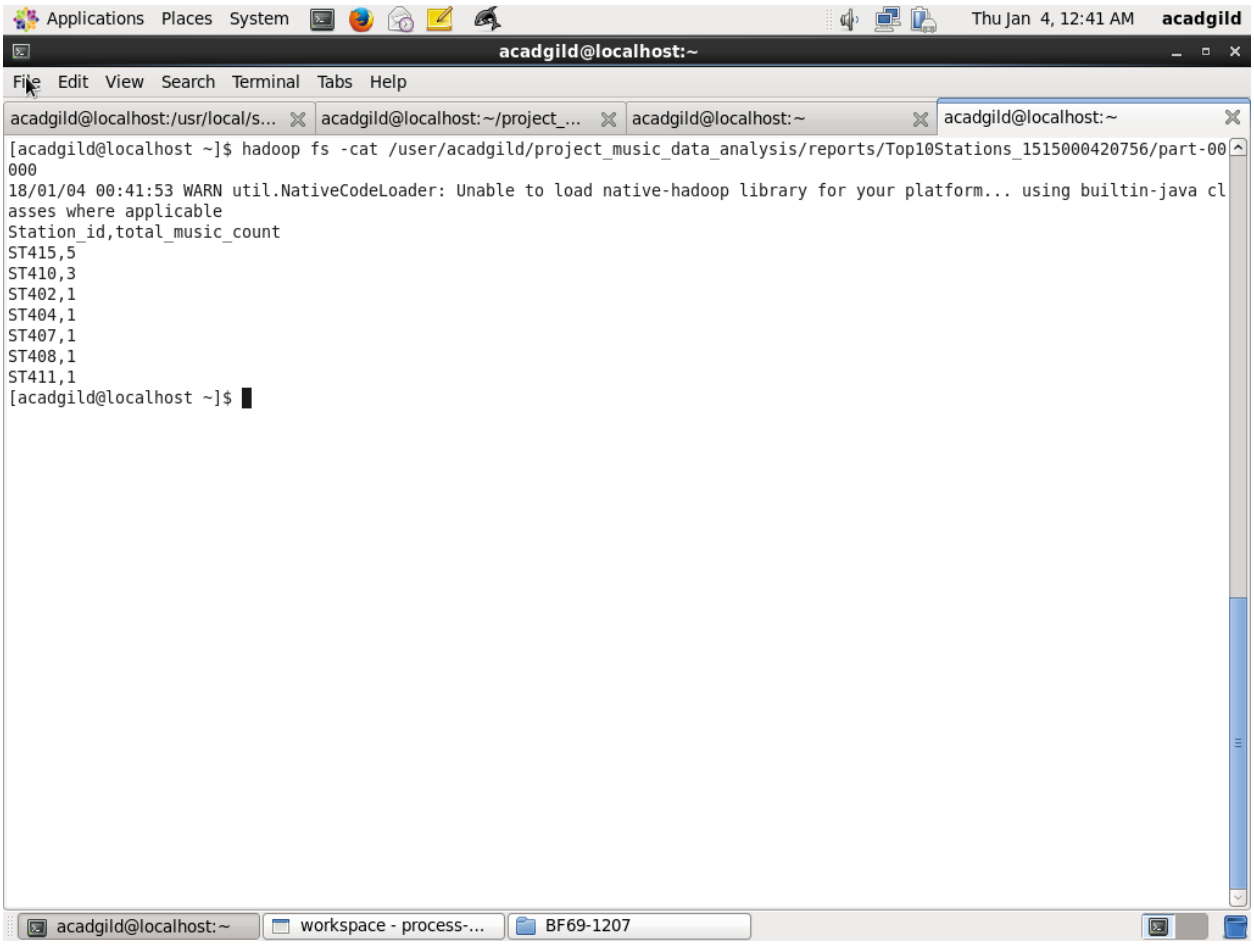
All the folders are in HDFS are as below:



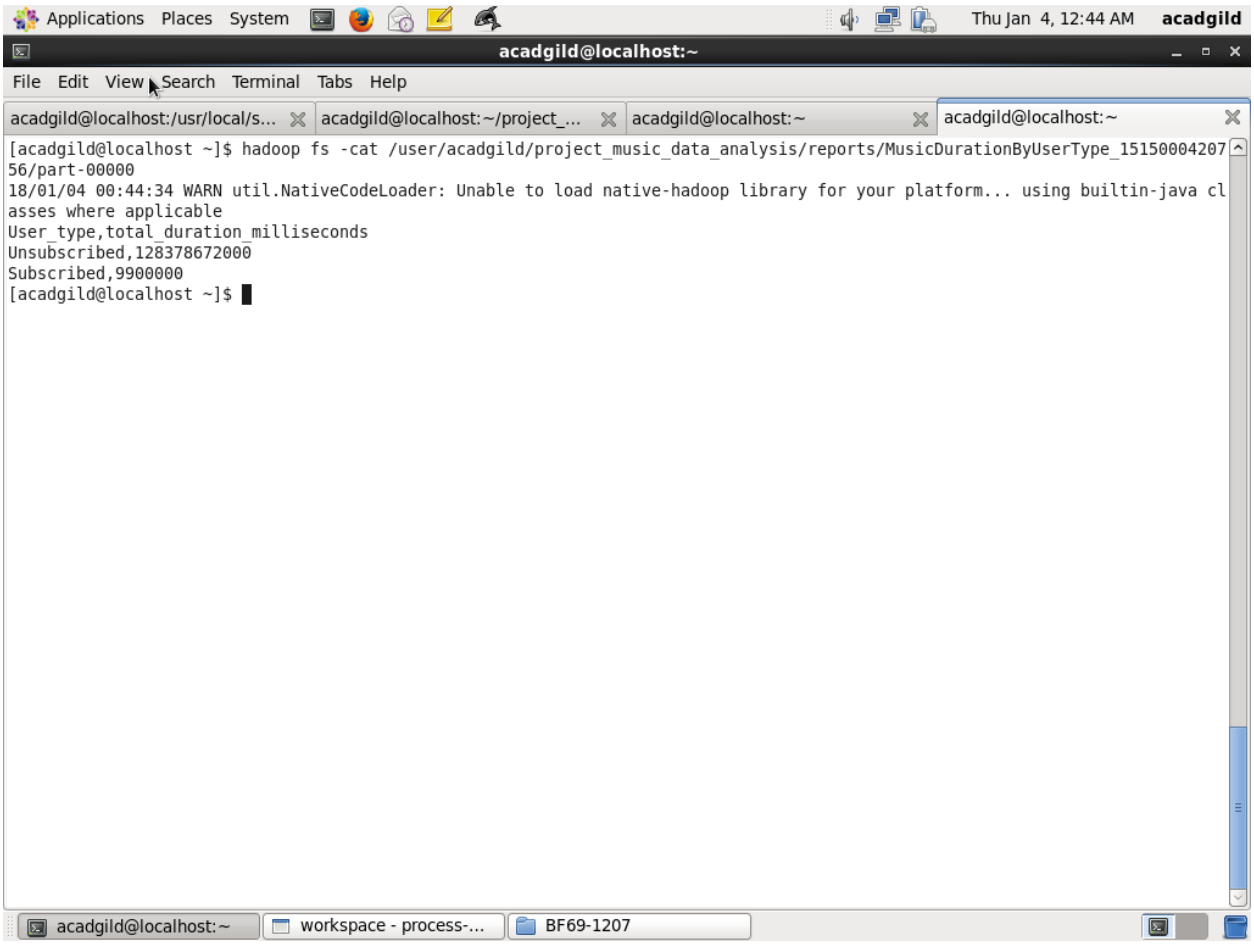
All Records – This will contain all the records in MusicDataDetailed



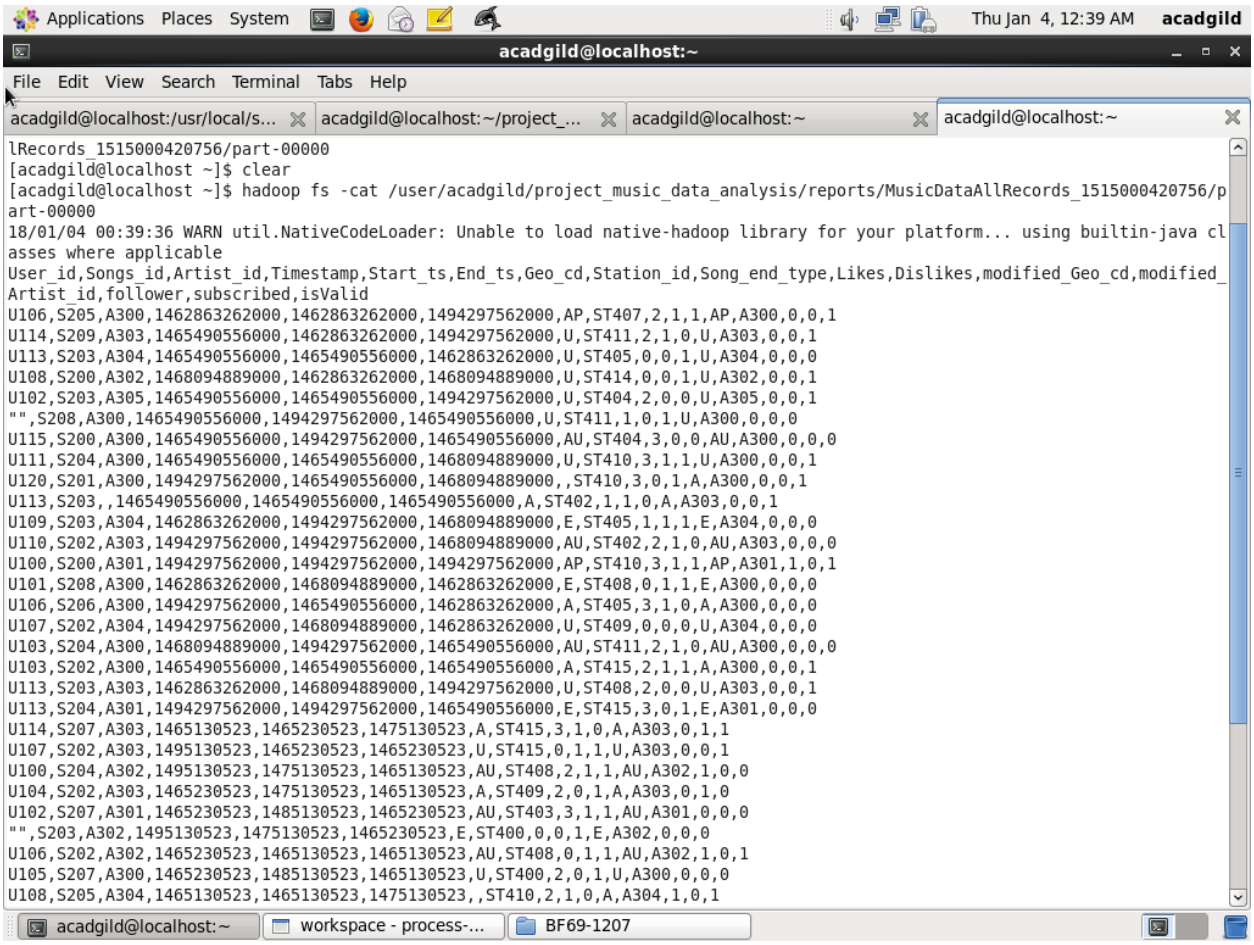
Top10Stations – This will top 10 stations having maximum music played which are liked by unique users



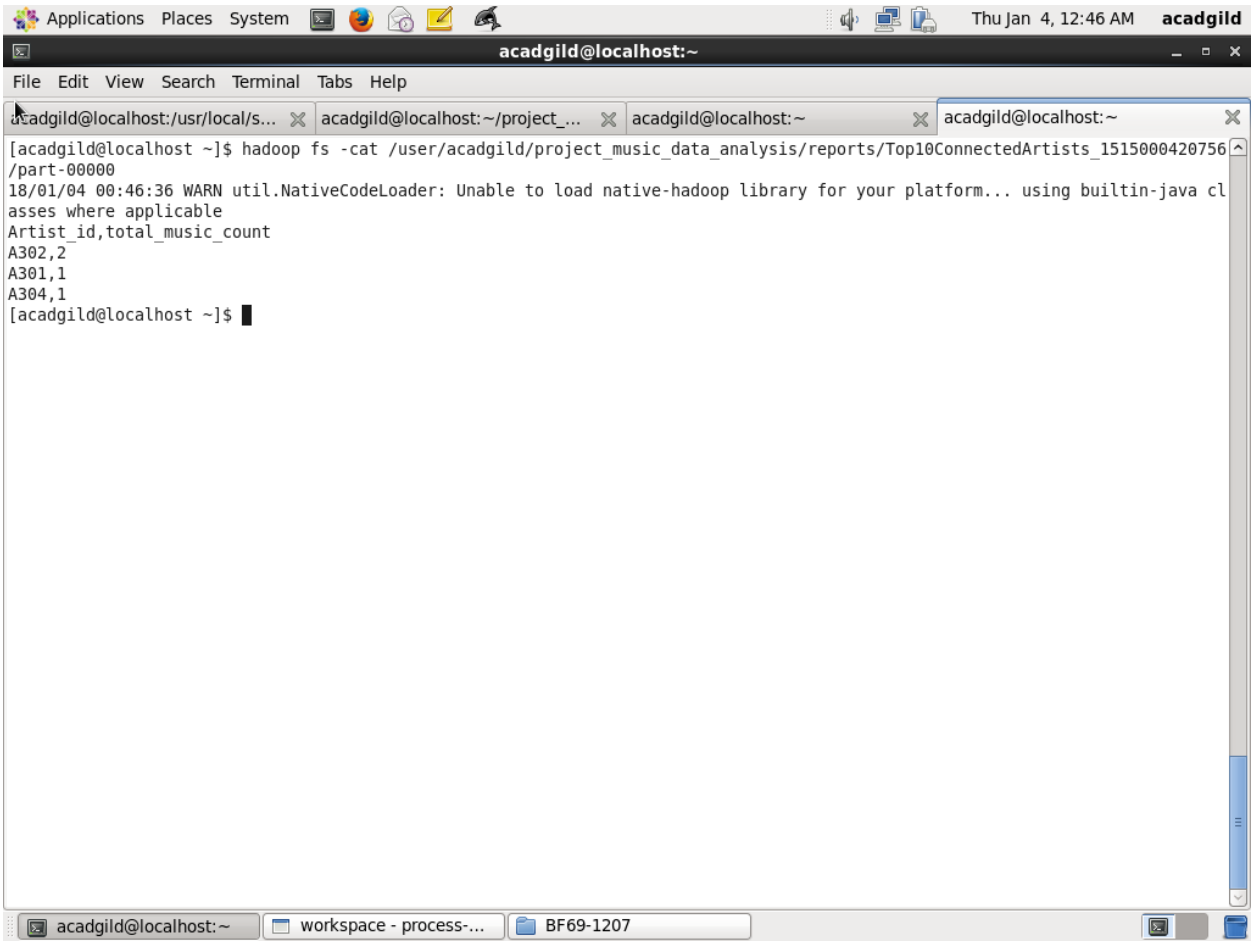
DurationByCategory – This will report total duration of music played by each time of user (unsubscribed/unsubscribed)



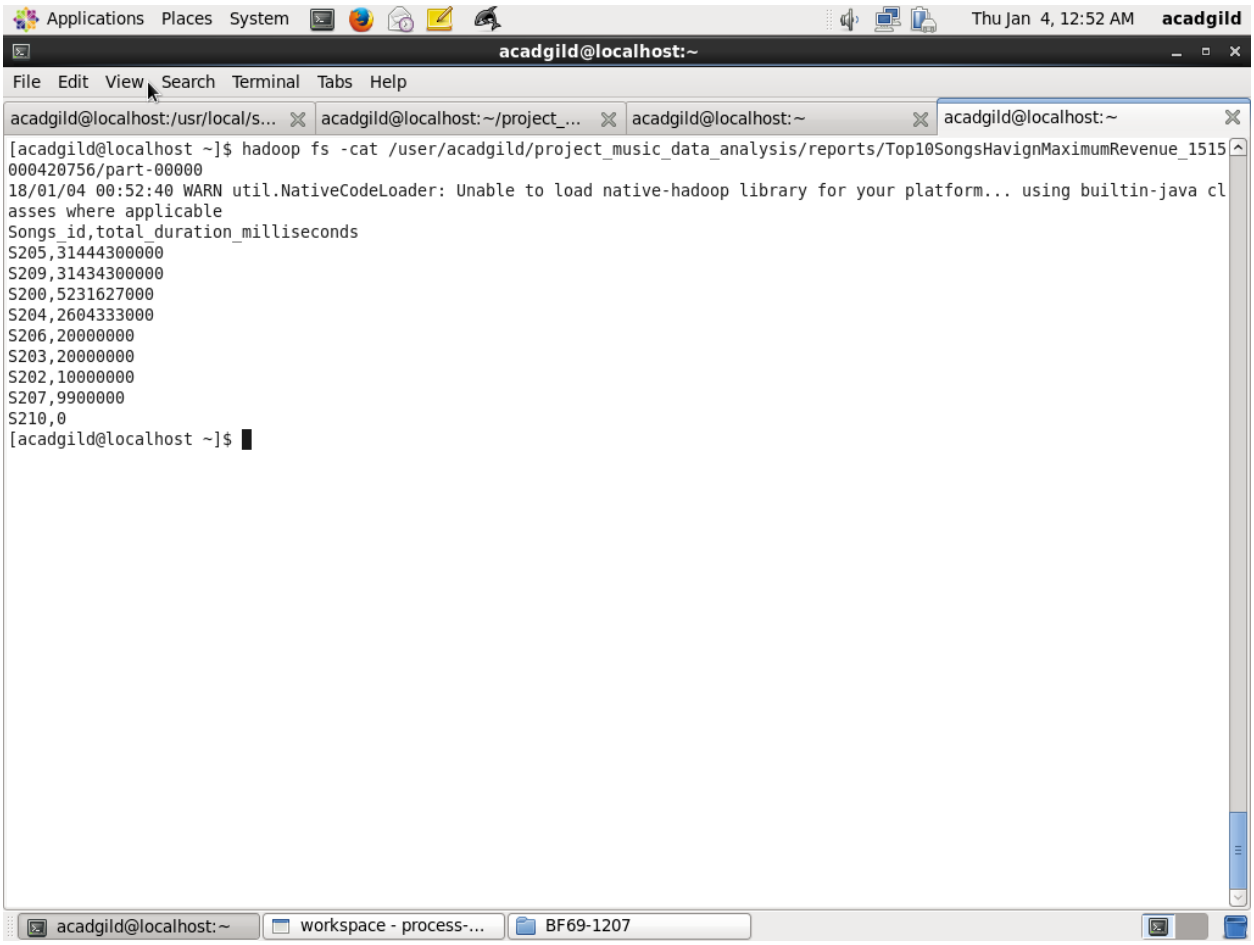
DurationByCategory – This will report total duration of music played by each time of user (unsubscribed/unsubscribed)

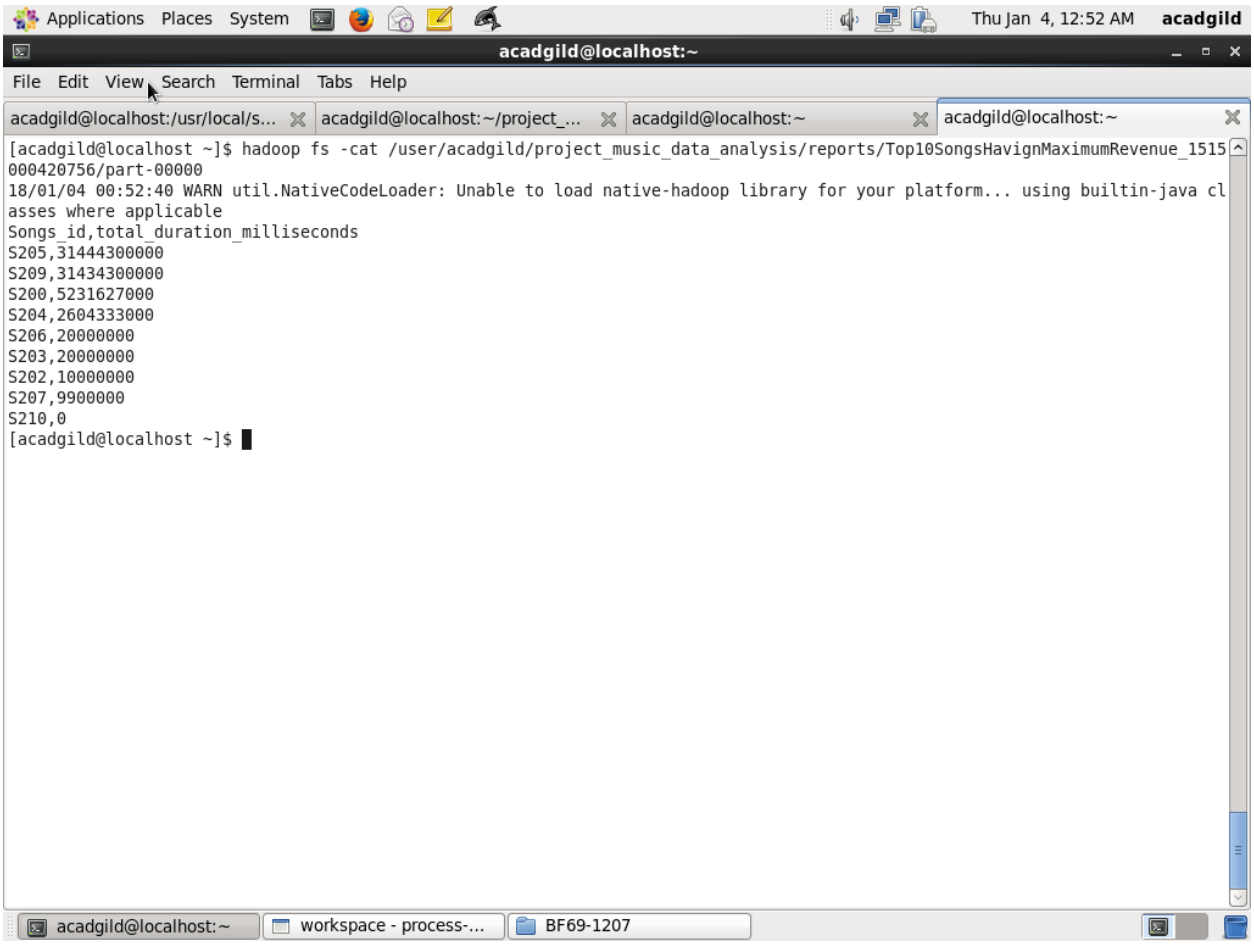


Top10ConnectedArtists : Top 10 Connected Artists

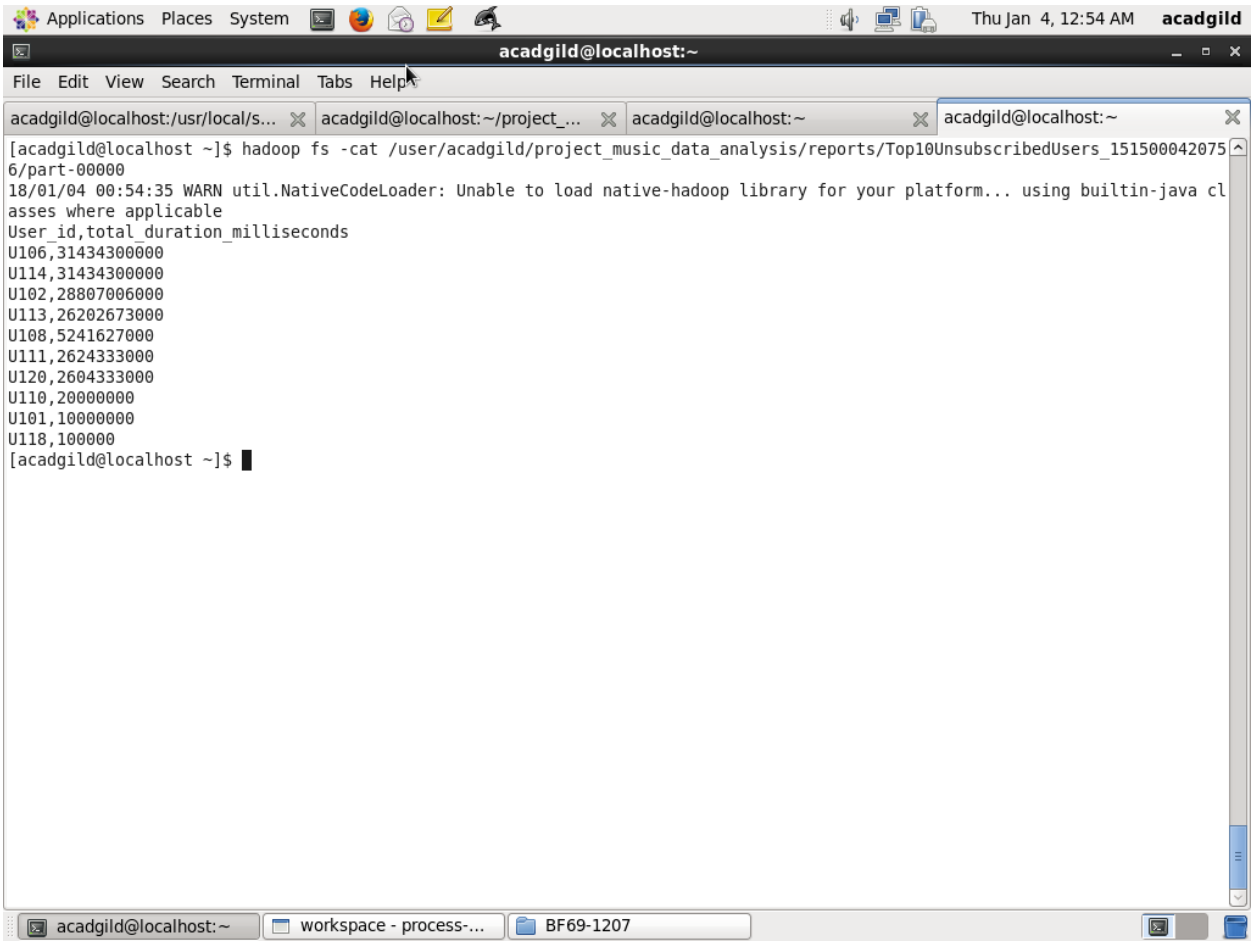


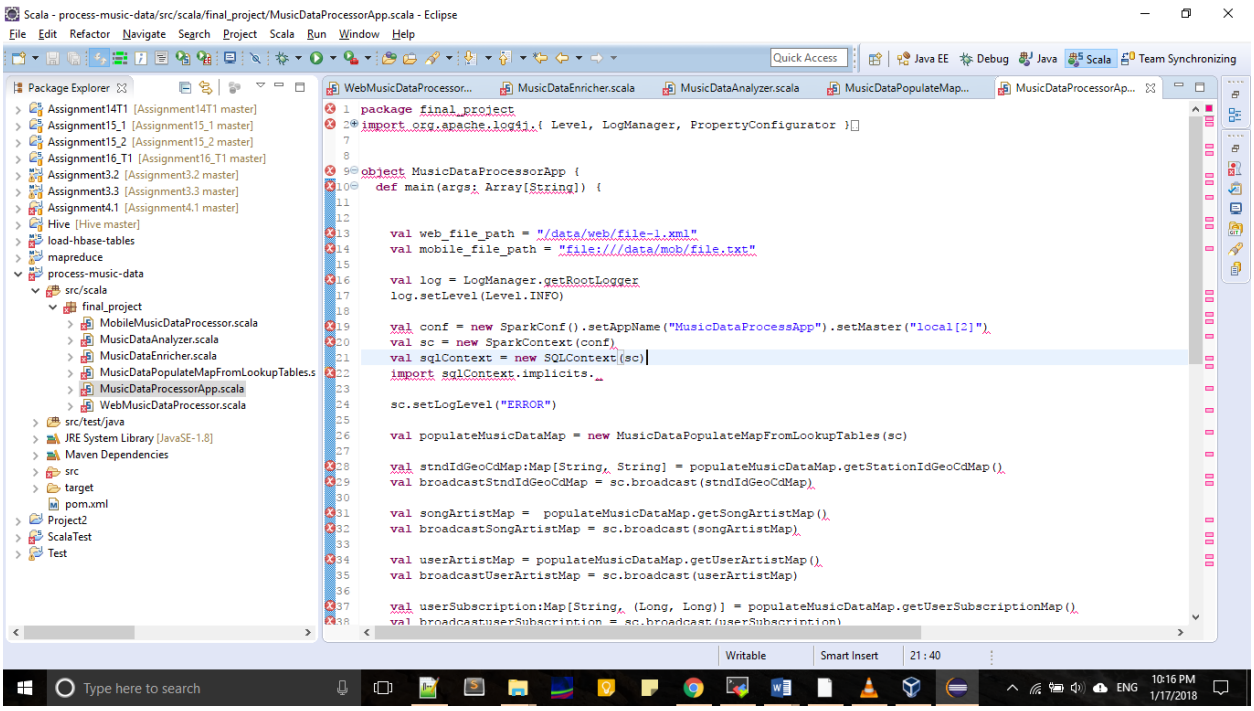
Top10SongsHavingMaximumRevenue: Top 10 songs have maximum revenue

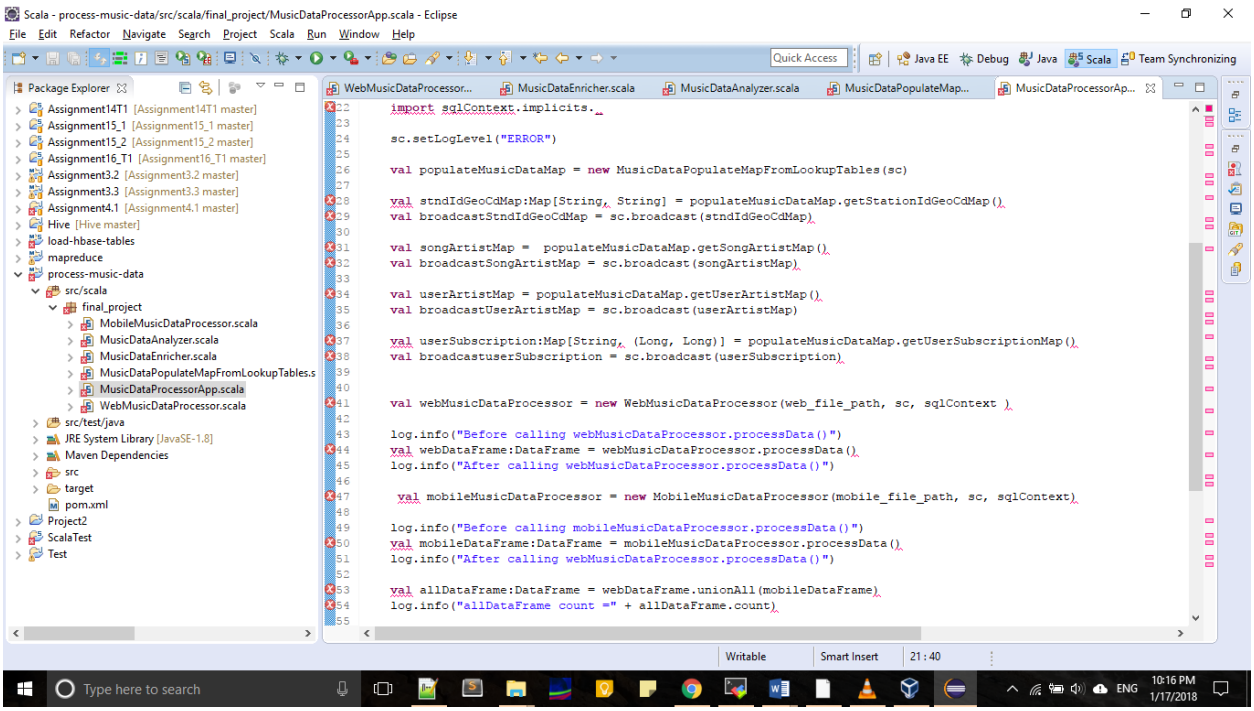

Applications Places System
Thu Jan 4, 12:52 AM acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:/usr/local/s...
acadgild@localhost:~/project_...
acadgild@localhost:~
acadgild@localhost:~
[acadgild@localhost ~]$ hadoop fs -cat /user/acadgild/project_music_data_analysis/reports/Top10SongsHavignMaximumRevenue_1515
000420756/part-00000
18/01/04 00:52:40 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Songs_id,total_duration_milliseconds
S205,31444300000
S209,31434300000
S200,5231627000
S204,2604333000
S206,20000000
S203,20000000
S202,10000000
S207,9900000
S210,0
[acadgild@localhost ~]$
acadgild@localhost:~
workspace - process-...
BF69-1207

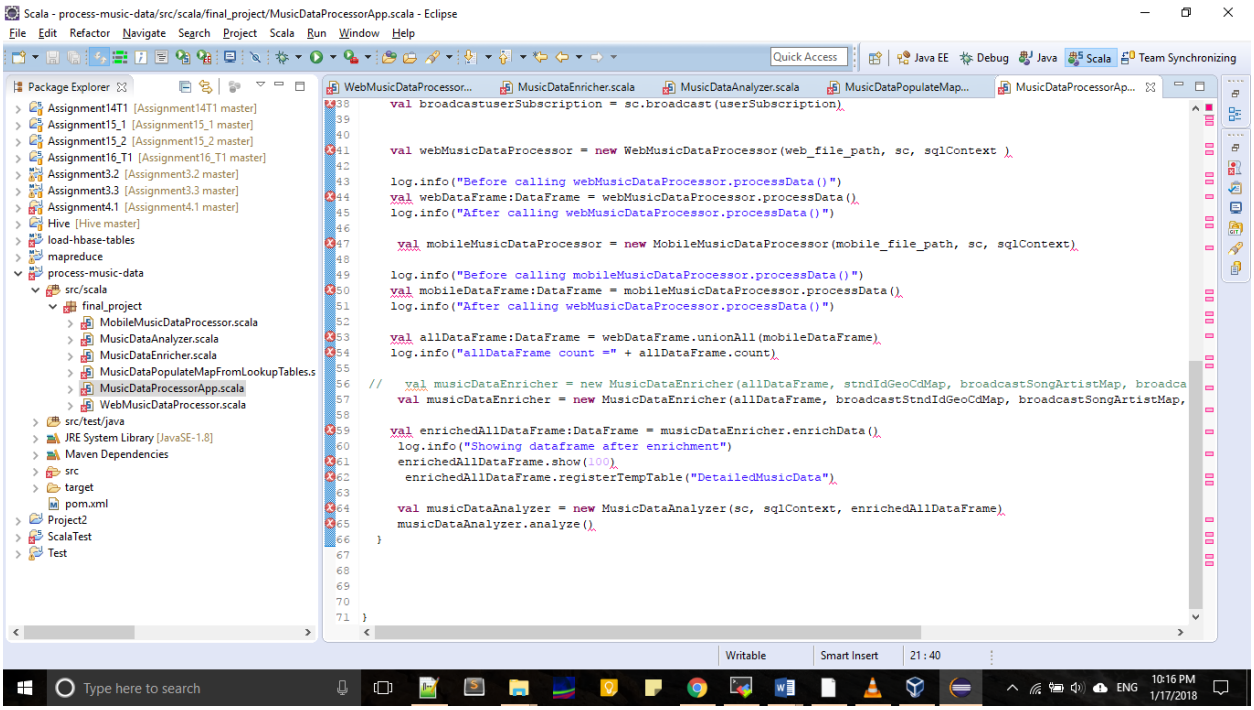


Top10UnsubscribedUsers – Top 10 unsubscribed users who listened to the music for maximum duration


Applications Places System
Thu Jan 4, 12:54 AM acadgild
acadgild@localhost:~
File Edit View Search Terminal Tabs Help
acadgild@localhost:/usr/local/s...
acadgild@localhost:~/project_...
acadgild@localhost:~
acadgild@localhost:~
[acadgild@localhost ~]$ hadoop fs -cat /user/acadgild/project_music_data_analysis/reports/Top10UnsubscribedUsers_151500042075
6/part-00000
18/01/04 00:54:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
User_id,total_duration_milliseconds
U106,31434300000
U114,31434300000
U102,28807006000
U113,26202673000
U108,5241627000
U111,2624333000
U120,2604333000
U110,20000000
U101,10000000
U118,100000
[acadgild@localhost ~]$
acadgild@localhost:~
workspace - process-...
BF69-1207


Scala - process-music-data/src/scala/final_project/MusicDataProcessorApp.scala - Eclipse
File Edit Refactor Navigate Search Project Scala Run Window Help
Quick Access
Java EE Debug Java Scala Team Synchronizing
Package Explorer
Assignment14T1 [Assignment14T1 master]
Assignment15_1 [Assignment15_1 master]
Assignment15_2 [Assignment15_2 master]
Assignment16_T1 [Assignment16_T1 master]
Assignment3.2 [Assignment3.2 master]
Assignment3.3 [Assignment3.3 master]
Assignment4.1 [Assignment4.1 master]
Hive [Hive master]
load-hbase-tables
mapreduce
process-music-data
src/scala
final_project
MobileMusicDataProcessor.scala
MusicDataAnalyzer.scala
MusicDataEnricher.scala
MusicDataPopulateMapFromLookupTables.s
MusicDataProcessorApp.scala
WebMusicDataProcessor.scala
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
pom.xml
Project2
ScalaTest
Test
WebMusicDataProcessor...
MusicDataEnricher.scala
MusicDataAnalyzer.scala
MusicDataPopulateMap...
MusicDataProcessorAp...
1 package final_project
2 import org.apache.log4j.{ Level, LogManager, PropertyConfigurator }
7
8
9 object MusicDataProcessorApp {
10 def main(args: Array[String]) {
11
12
13 val web_file_path = "/data/web/file-1.xml"
14 val mobile_file_path = "file:///data/mob/file.txt"
15
16 val log = LogManager.getRootLogger
17 log.setLevel(Level.INFO)
18
19 val conf = new SparkConf().setAppName("MusicDataProcessApp").setMaster("local[2]")
20 val sc = new SparkContext(conf)
21 val sqlContext = new SQLContext(sc)
22 import sqlContext.implicits._
23
24 sc.setLogLevel("ERROR")
25
26 val populateMusicDataMap = new MusicDataPopulateMapFromLookupTables(sc)
27
28 val stndIdGeoCdMap:Map[String, String] = populateMusicDataMap.getStationIdGeoCdMap()
29 val broadcastStndIdGeoCdMap = sc.broadcast(stndIdGeoCdMap)
30
31 val songArtistMap = populateMusicDataMap.getSongArtistMap()
32 val broadcastSongArtistMap = sc.broadcast(songArtistMap)
33
34 val userArtistMap = populateMusicDataMap.getUserArtistMap()
35 val broadcastUserArtistMap = sc.broadcast(userArtistMap)
36
37 val userSubscription:Map[String, (Long, Long)] = populateMusicDataMap.getUserSubscriptionMap()
38 val broadcastUserSubscription = sc.broadcast(userSubscription)
Writable
Smart Insert
21 : 40
Type here to search
ENG
10:16 PM
1/17/2018



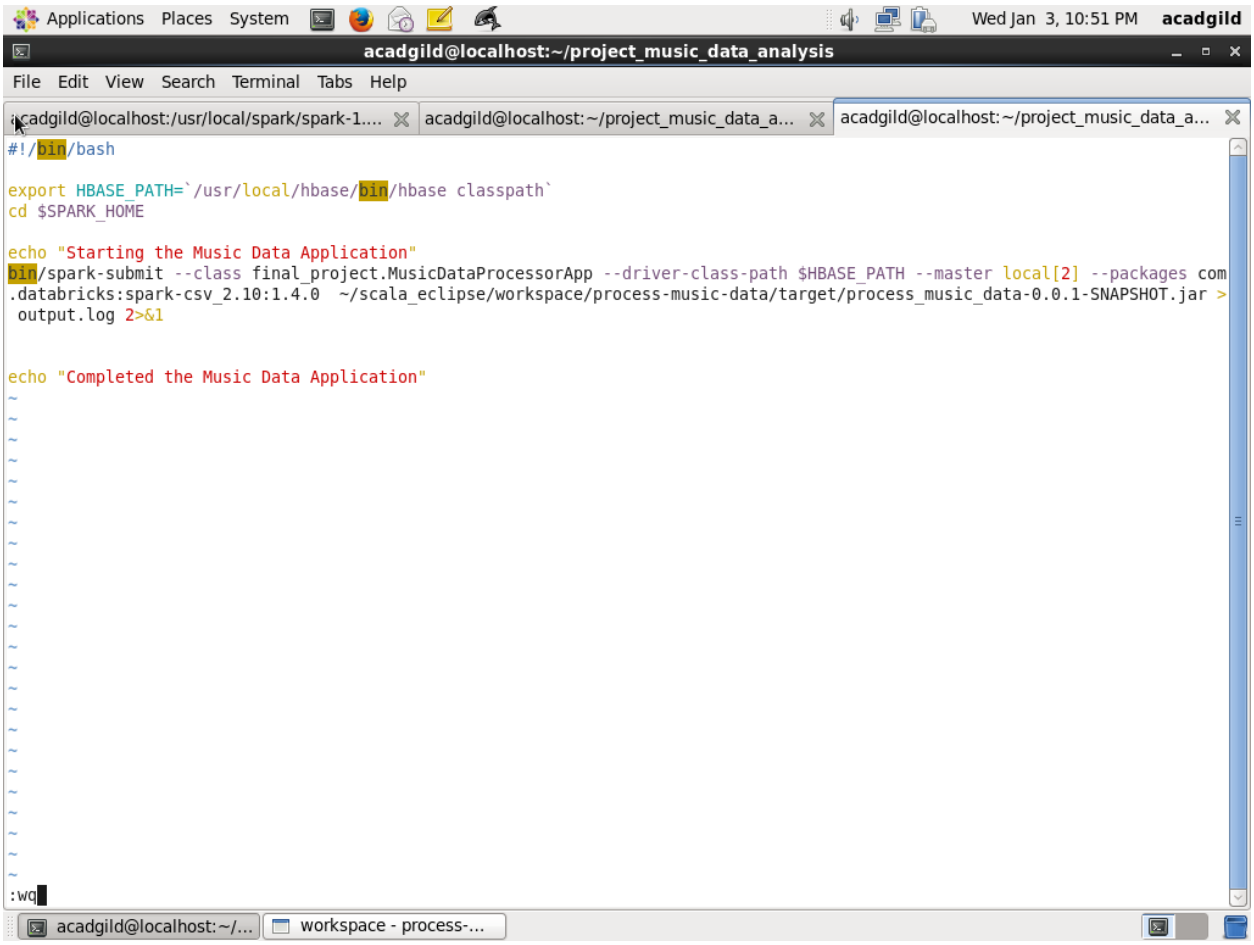


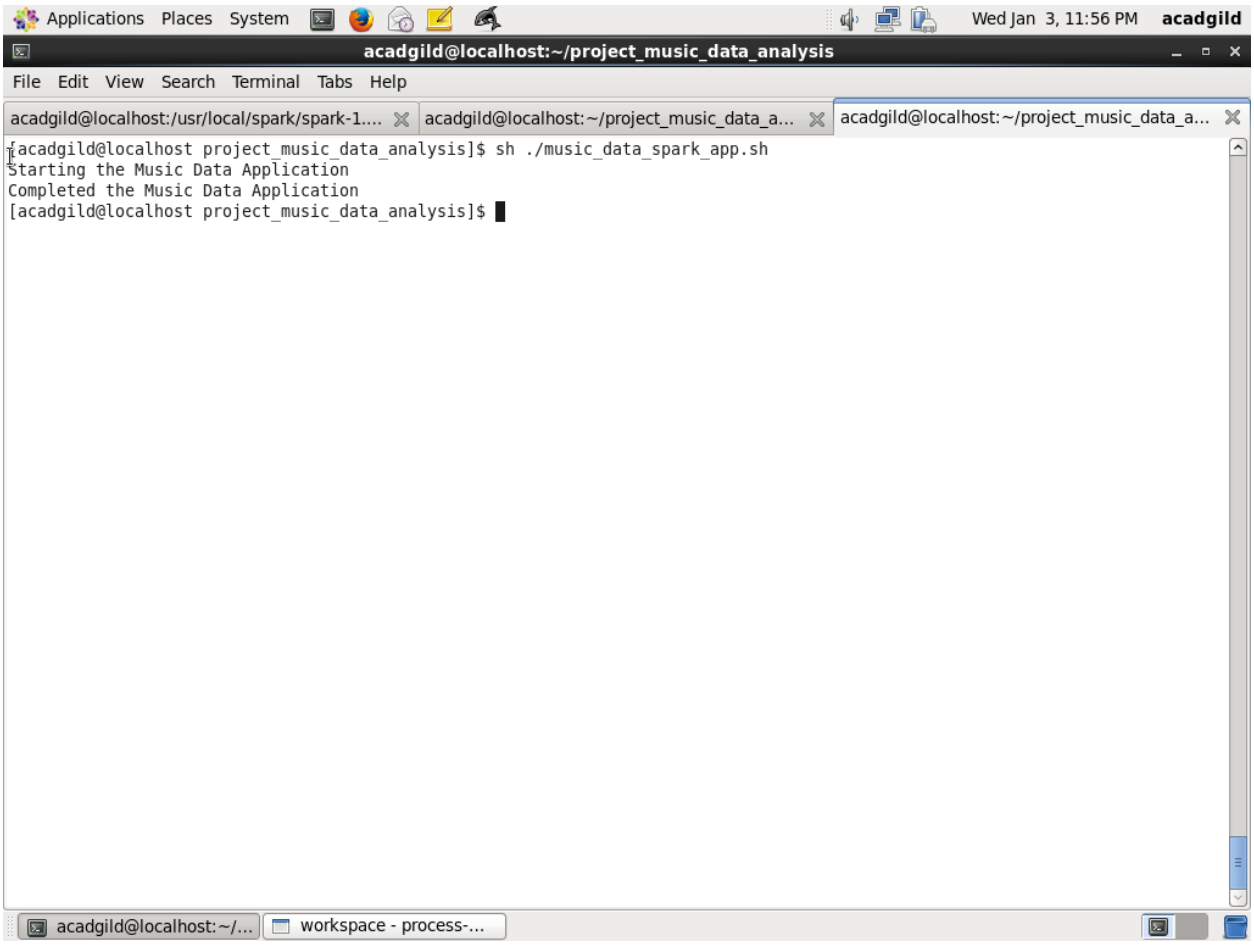
Module7: Script for running the Music Data App

A bash script music_data_spark_app.sh is created and it content is as below:

```
#!/bin/bash

export HBASE_PATH=`/usr/local/hbase/bin/hbase classpath`

cd $SPARK_HOME

echo "Starting the Music Data Application"

bin/spark-submit --class final_project.MusicDataProcessorApp --driver-class-path $HBASE_PATH --master local[2] --packages com.databricks:spark-csv_2.10:1.4.0  ~/scala_eclipse/workspace/process-music-data/target/process_music_data-0.0.1-SNAPSHOT.jar > output.log 2>&1

echo "Completed the Music Data Application"
```

Screenshot is as below:


Applications Places System
Wed Jan 3, 10:51 PM acadgild
acadgild@localhost:~/project_music_data_analysis
File Edit View Search Terminal Tabs Help
acadgild@localhost:/usr/local/spark/spark-1....
acadgild@localhost:~/project_music_data_a...
acadgild@localhost:~/project_music_data_a...
#!/bin/bash
export HBASE_PATH=`/usr/local/hbase/bin/hbase classpath`
cd $SPARK_HOME
echo "Starting the Music Data Application"
bin/spark-submit --class final_project.MusicDataProcessorApp --driver-class-path $HBASE_PATH --master local[2] --packages com
.databricks:spark-csv_2.10:1.4.0 ~/scala_eclipse/workspace/process-music-data/target/process_music_data-0.0.1-SNAPSHOT.jar >
output.log 2>&1
echo "Completed the Music Data Application"
:wq
acadgild@localhost:~/...
workspace - process-...

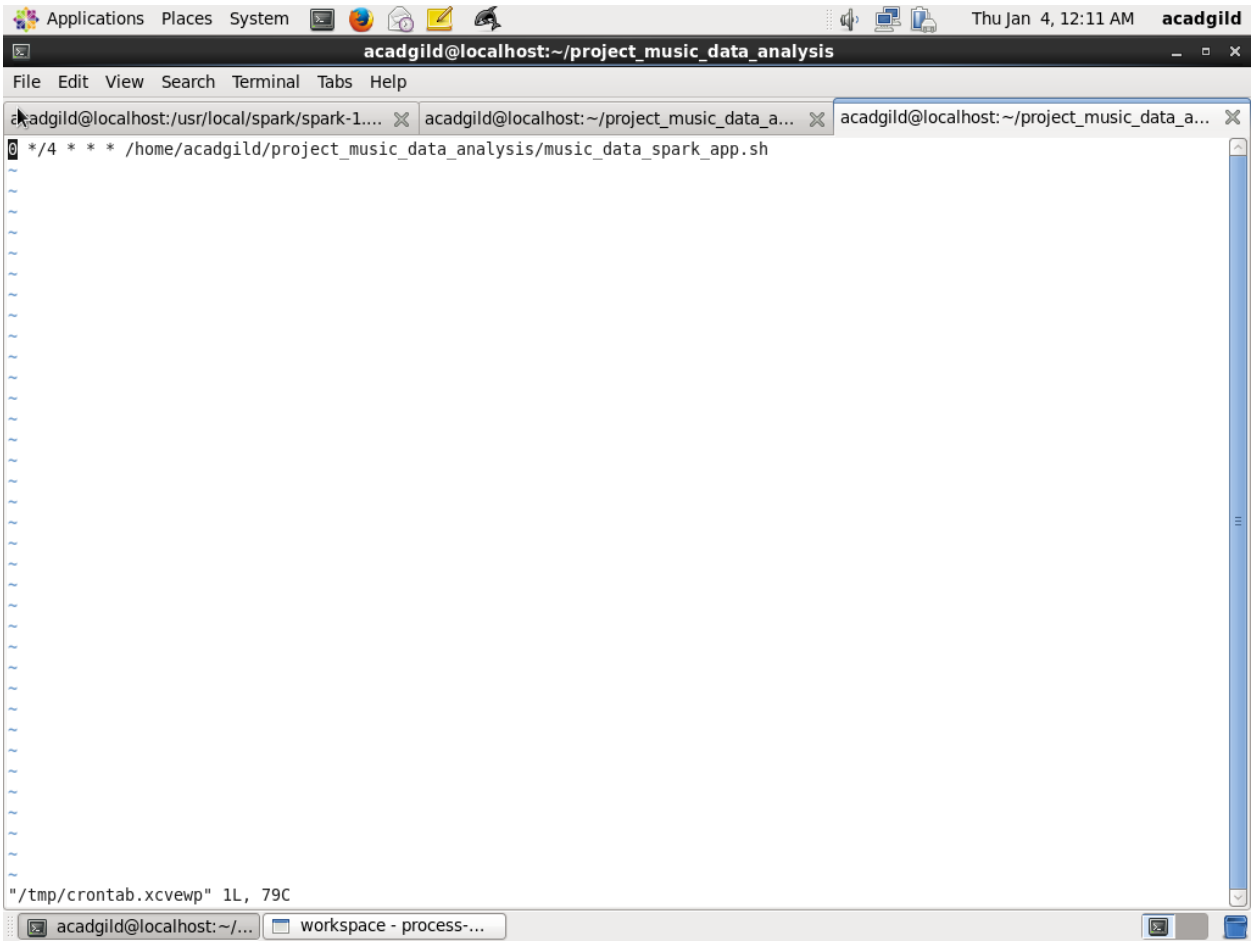


Next, a cron job is created using “crontab –e” so that the script is run every 3 hours

Content is as below

0 */4 * * * /home/acadgild/project_music_data_analysis/music_data_spark_app.sh

The screenshot is as below:



3. Highlights of the project

i. No join of query is used while analysiss. Data is already enriched with new fields and using broadcast maps on Lookup tables so as to avoid any join

ii. Logger has been used in all over the code to allow

4. Issues Faced and how to resolve it

While doing project, I face following issues and how I resolved it

i. XML processing – I found the databricks provides he XML processing APIs in spark. But it was not working. The main issue I face

I was trying the process the xml dataset (file-1.xml) to covert to spark dataframe as given for the final project. I have created a scala Object ReadXMLMusicData.scala based on the steps given in

https://github.com/databricks/spark-xml

Created maven project with pom.xml attached. Using mvn install, I am able to generate jar file read_xml_music_data-0.0.1-SNAPSHOT.jar . However when I try to deploy to spark using spark-submit on Acadgild VM

bin/spark-submit --class ReadXMLMusicData --master local[2] --driver-class-path /home/acadgild/.m2/repository/com/databricks/spark-xml_2.10/0.4.1/spark-xml_2.11-0.4.1.jar ~/scala_eclipse/workspace/read_xml_music_data/target/read_xml_music_data-0.0.1-SNAPSHOT.jar

```
Exception in thread "main" java.lang.NoSuchMethodError:
org.apache.spark.sql.DataFrameReader.load(Ljava/lang/String;)Lorg/apache/spark/sql/Dataset;
    at ReadXMLMusicData$.main(ReadXMLMusicData.scala:12)
    at ReadXMLMusicData.main(ReadXMLMusicData.scala)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:497)
    at
org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:731)
    at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:181)
    at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:206)
    at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:121)
    at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

I tried all the options

1. --packages com.databricks:spark-xml_2.10:0.4.1
2. --packages com.databricks:spark-xml_2.11:0.4.1
3. --driver-class-path /home/acadgild/.m2/repository/com/databricks/spark-xml_2.10/0.4.1/spark-xml_2.10-0.4.1.jar
4. --driver-class-path /home/acadgild/.m2/repository/com/databricks/spark-xml_2.10/0.4.1/spark-xml_2.11-0.4.1.jar

Acadgild gave the following solution:

DataFrame() was supplanted in Spark 2 by Dataset(). You'll need to import org.apache.spark.sql.Dataset and use that if you're running a Spark 1.6 client with a Spark 2.1 server-side. it would be a lot better off using at least Spark 2.1.+ dependencies.

Follow the below link to download newer version of spark.

Spark_Download

But I solved this I used a xml APIs from scala.xml.XML. Relevant links are as below:

https://alvinalexander.com/scala/how-to-extract-data-from-xml-nodes-in-scala

https://hadoopist.wordpress.com/2016/01/08/parsing-a-basic-xml-using-hadoop-and-spark-core-apis/

http://www.scala-lang.org/api/2.11.1/scala-xml

ii. Task Not Serializable Exception:

I am encoutering "org.apache.spark.SparkException: Task Not Serializable" In the code below when Artist_id is blank I try to get it using Song_Id from broadcastSongArtistMap. The dataset used is the mobile dataset. Basically exception is thrown in the line below:

if (x(2).equals("")) broadcastSongArtistMap.value.get(x(1)).getOrElse("Invalid") else x(2),

The job is submitted using spark-submit. I would very much appreciate, if you can give me a solution on this (I tied lot of things, but could not make it work)

Code snippet is as below:

```
val songArtistMap = Map("S200" -> "A300",
                        "S201" -> "A301",
                        "S202" -> "A302",
                        "S203" -> "A303",
                        "S204" -> "A304",
                        "S205" -> "A301",
                        "S206" -> "A302",
                        "S207" -> "A303",
                        "S208" -> "A304",
                        "S209" -> "A305"
                      )
val broadcastSongArtistMap = sc.broadcast(songArtistMap)

case class MusicData(User_id:String, Song_id:String, Artist_id:String, Timestamp:Long,
    Start_ts:Long, End_ts:Long, Geo_cd:String, Station_id:String, Song_end_type:String,
    Like:String, Dislike:String)

val recordRDD = sc.textFile("/home/acadgild/final_project/file.txt")
val recordFieldsRDD = recordRDD.map(x => x.split(",")).filter(x=> x.length ==11)

val recordDF = recordFieldsRDD.map(x => MusicData(x(0),
      x(1),
      if (x(2).equals("")) broadcastSongArtistMap.value.get(x(1)).getOrElse("Invalid") else x(2),
```

```
            x(3).toLong,
            x(4).toLong,
            x(5).toLong,
            x(6),
            x(7),
            x(8),
            if (x(9).equals("")) "0" else x(9),
            if (x(10).equals("")) "0" else x(10))).toDF

        recordDF.show
```

Solution: The problem was that SparkContext as parameter to class MobileMusicDataProcessor was not Serializable. When I removed the SparkContext, it worked

iii. Continue does not work for scala:

There is no support for continue in scala. To use it I created a case class CustomException(message:String) extends Exception(message)

Then used

iv. HBase support with Spark

I used the following link:

https://acadgild.com/blog/spark-on-hbase/

https://acadgild.com/blog/apache-hbase-beginners-guide/

Even using that the multiple fields were not working. To solve this, I used the following code:

```
val hbaseRDD = sc.newAPIHadoopRDD(hconf, classOf[TableInputFormat],
classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
classOf[org.apache.hadoop.hbase.client.Result])

    val resultRDD = hbaseRDD.map(tuple=>tuple._2)


    val keyValueRDD = resultRDD.map(result => (Bytes.toString(result.getRow()).split(" ")(0), (
        if (Bytes.toString(result.getValue(new String("Subscription").getBytes(), new
String("Start_ts").getBytes)).equals("")) 0 else Bytes.toString(result.getValue(new
String("Subscription").getBytes(), new String("Start_ts").getBytes)).toLong,
        if (Bytes.toString(result.getValue(new String("Subscription").getBytes(), new
String("End_ts").getBytes)).equals("")) 0 else Bytes.toString(result.getValue(new
String("Subscription").getBytes(), new String("End_ts").getBytes)).toLong
        )))
```

v. Map was giving compilation error

There are two types of Map Mutable and Immutable

I needed to convert to Immutable map to make the compilation work using the following:

```
  val map = keyValueRDD.collectAsMap
   return scala.collection.immutable.Map(map.toSeq:_*)
```

vi. toDF does not work and giving compilation error

To solve this I needed o add:

```
import sqlContext.implicits._
```

To every class

vii. The following command was giving exception:

```
sqlContext.resisterTempTable(“MusicDataDetailed”)
```

The problem was not I was creating sqlContext in every class and dataframe was created tow different classes. To solve this I had to use one sqlContext created at MusicDataProcessorApp

```
  try {
        if (userId.equals("") && songId.equals("") && artistId.equals("")) {
          throw CustomException("Record is blank")
        } else {
          recordListBuffer += ((userId, songId, artistId, timestampLong, startTsLong, endTsLong, geoCd, stationId,
songEndType, like, dislike))
        }
      } catch {
        case CustomException(msg) => msg
      }
    }
```

viii. Not able to store query result to file..

To use this I used databricks csv API:

```
val df = sqlContext.sql("SELECT Songs_id, SUM(duration) AS total_duration_milliseconds FROM
SongDuration "
      + " GROUP BY Songs_id ORDER BY total_duration_milliseconds DESC LIMIT 10")
    df.show()
    val df1 = df.repartition(1)
    df1.write.format("com.databricks.spark.csv").option("header", "true").save(reportBasePath +
"/Top10SongsHavignMaximumRevenue_" + currentTimestamp)
```

```
bin/spark-submit --class final_project.MusicDataProcessorApp --master local[2] --packages
com.databricks:spark-csv_2.10:1.4.0 --driver-java-options "-Dlog4j.configuration=file:/usr/local/spark/spark-
1.6.0-bin-hadoop2.6/conf/log4j.properties" ~/scala_eclipse/workspace/process-music-
data/target/process_music_data-0.0.1-SNAPSHOT.jar > output.log 2>&1
```

ix. AnalysisException whenever I used a field which does not exist because of case. For example Song_Id, but the schma it is Song_id. Even if I restart or whatever I does not work. I needed to change the schema to Songs_id to solve this

x.