BIG DATA PROJECT DOCUMENTATION

29.05.2020

Rahul Anand

Setting Up Environment In HDFS

hdfs dfs -mkdir zomato_etl_rahul.anand
hdfs dfs -mkdir zomato_etl_rahul.anand/ zomato_ext
hdfs dfs -mkdir zomato_etl_rahul.anand/zomato_ext/zomato hdfs dfs -mkdir
zomato_etl_rahul.anand/zomato_ext/dim_country

My HDFS Folder Structure is as follows:

- zomato_etl_rahul.anand
  - zomato_ext
    - zomato
    - dim_country

Setting Up Environment In Local File System (UNIX)

mkdir zomato_etl mkdir
zomato_raw_files
mkdir zomato_etl/source mkdir
zomato_etl/source/json mkdir
zomato_etl/source/csv mkdir
zomato_etl/archive mkdir
zomato_etl/hive
mkdir zomato_etl/hive/ddl mkdir
zomato_etl/hive/dml mkdir
zomato_etl/spark mkdir
zomato_etl/spark/jars mkdir
zomato_etl/spark/scala mkdir
zomato_etl/script mkdir
zomato_etl/logs

My Local FileSystem Folder Structure is as follows:

- zomato_etl
  - source
    - json
    - csv
- archive
- hive
  - ddl
  - dml

- spark
  - jars
  - scala
- script (shell scripts and property files)
- logs (log_ddmmyyyy_hhmm.log)

Question 2

Copied file1.json, file2.json, file3.json into source/json folder for the spark application to access and operate on.

Spark Shell Module1 - Json To CSV Convertor

Coding

```
scala > val read_rdd = spark.read.json("file1.json")

scala > val order_rdd = read_rdd.select(explode($"restaurants.restaurant"))

scala > val final2_rdd = order_rdd.select($"col.R.res_id" as "Restaurant_Id", $"col.name" as "Restaurant Name", $"col.location.country_id" as "Country Code", $"col.location.city" as "City", $"col.location.address" as "Address", $"col.location.locality" as "Locality", $"col.location.locality_verbose" as "Locality Verbose", $"col.location.longitude" as "Longitude", $"col.location.latitude" as "Latitude", $"col.cuisines" as "Cuisines", $"col.average_cost_for_two" as "Average Cost for two", $"col.currency" as "currency", $"col.has_table_booking" as "Has table booking", $"col.has_online_delivery" as "Has Online delivery", $"col.is_delivering_now" as "Is delivering now", $"col.Switch_to_order_menu" as "Switch to order menu", $"col.price_range" as "Price range", $"col.user_rating.aggregate_rating" as "Aggregate rating",$"col.user_rating.rating_text" as "Rating text", $"col.user_rating.votes" as "Votes")

scala > final2_rdd.write.option("delimiter", "\t").csv("zomato_etl_rahul.anand/source/csv1")

hdfs dfs -mv zomato_etl_rahul.anand/source/csv1/part-*.csv
zomato_etl_rahul.anand/source/csv/zomato_20190609.csv

hdfs dfs -copyToLocal zomato_etl_rahul.anand/source/csv/zomato_20190609.csv
~/zomato_etl/source/csv
```

Conversion of file1.json is complete (Similarly, convert the other 2 json files)
Put all three converted files from local FileSystem to hdfs location

```
hdfs dfs -put zomato_etl/source/csv/zomato_20190609.csv
rahul.anand/zomato_etl_rahul.anand/zomato_ext/zomato
```

Question 3

Create External table named “zomato” partitioned by filedate and load zomato_<filedate>.csv into respective partition.

Create Managed table named “zomato_summary_log” table as per given schema to log the job id, step, spark-submit used, start-time, end-time and job status.

Create a Managed table “dim_country” using country_code.csv file given

Creation Of External zomato Table

Coding

```
hive > create external table rahul_database.zomato_rahul(
. . . . . .> `Restaurant ID` INT,
. . . . . .> `Restaurant Name` STRING,
. . . . . .> `Country Code` INT,
. . . . . .> `City` STRING,
. . . . . .> `Address` STRING,
. . . . . .> `Locality` STRING,
. . . . . .> `Locality Verbose` STRING,
. . . . . .> `Longitude` STRING,
. . . . . .> `Latitude` STRING,
. . . . . .> `Cuisines` STRING,
. . . . . .> `Average Cost for two` INT,
. . . . . .> `Currency` STRING,
. . . . . .> `Has Table booking` INT,
. . . . . .> `Has Online delivery` INT,
. . . . . .> `Is delivering now` INT,
. . . . . .> `Switch to order menu` INT,
. . . . . .> `Price range` INT,
. . . . . .> `Aggregate rating` STRING,
. . . . . .> `Rating text` STRING,
. . . . . .> `Votes` STRING)
. . . . . .> PARTITIONED BY ( filedate int )
. . . . . .> ROW FORMAT DELIMITED
. . . . . .> fields terminated by '\t'
```

```
. . . . . .> stored as textfile;

hive > alter table zomato_rahul
. . . . . .> set location
'hdfs://Theflopguyproductions/user/rahul.anand/zomato_etl_rahul.anand/zomato_ext/zomato'
```

Load Data Into zomato Table

Coding

```
hive > LOAD DATA INPATH
. . . . . .>
'hdfs://Theflopguyproductions/user/rahul.anand/zomato_etl_rahul.anand/zomato_etl/zomato/zomato_20190609.csv'
. . . . . .> OVERWRITE INTO TABLE zomato_rahul
. . . . . .> PARTITION (filedate='20190609');
```

Repeat this step for all the partitions changing the partition value.

Create Managed dim_country table and load country_code.csv into it

Coding

```
hive > create table rahul_database.dim_country_rahul(
. . . . . .> `Country Code` int,
. . . . . .> `Country` string)
. . . . . .> ROW FORMAT DELIMITED
. . . . . .> fields terminated by ','
. . . . . .> stored as textfile;

hive > alter table dim_country_rahul
. . . . . .>set location
'hdfs://Theflopguyproductions/user/rahul.anand/zomato_etl_rahul.anand/zomato_ext/dim_country';

hive > LOAD DATA INPATH
'hdfs://Theflopguyproductions/user/rahul.anand/zomato_etl_rahul.anand/zomato_ext/dim_country/country_code.cs
v'
. . . . . .>OVERWRITE INTO TABLE dim_country_rahul;
```

Create Managed zomato_summary_log table

Coding

```
hive > create table zomato_summary_log_rahul(
. . . . . .> `Job id` STRING,
. . . . . .> `Job step` STRING,
. . . . . .> `spark submit command` STRING,
. . . . . .> `Job Start time` STRING,
. . . . . .> `Job End time` STRING,
. . . . . .> `Job status` STRING);
```

Question 4

A spark application to load zomato_summary table from zomato table and apply given transformations:

All the columns from the zomato table, adding to it two partition columns “p_filedate” and “p_country_name”, two derived columns “m_rating_colour” and “m_cuisines”, two audit columns “user_id” and “create_datetime” with relevant constraints.

Load data in a historical or a filedate/country constraint

Creation Of Managed zomato_summary Table And Load Data From zomato Table

Coding

```
hive > create table zomato_summary_rahul(
. . . . . .> `Restaurant ID` INT,
. . . . . .> `Restaurant Name` STRING,
. . . . . .> `Country Code` INT,
. . . . . .> `City` STRING,
. . . . . .> `Address` STRING,
. . . . . .> `Locality` STRING,
. . . . . .> `Locality Verbose` STRING,
. . . . . .> `Longitude` STRING,
. . . . . .> `Latitude` STRING,
```

```
. . . . . .> `Cuisines` STRING,
. . . . . .> `Average Cost for two` INT,
. . . . . .> `Currency` STRING,
. . . . . .> `Has Table booking` INT,
. . . . . .> `Has Online delivery` INT,
. . . . . .> `Is delivering now` INT,
. . . . . .> `Switch to order menu` INT,
. . . . . .> `Price range` INT,
. . . . . .> `Aggregate rating` STRING,
. . . . . .> `Rating text` STRING,
. . . . . .> `Votes` STRING,
. . . . . .> `m_rating_colour`  STRING,
. . . . . .> `m_cuisines`  STRING,
. . . . . .> `create_datetime`  TIMESTAMP,
. . . . . .> `user_id` STRING)
. . . . . .> PARTITIONED BY ( p_filedate INT, p_country_name STRING )
. . . . . .> stored as ORC;
```

Historically Loading.

```
hive > insert into zomato_summary_rahul(
. . . . . .> `restaurant id`,
. . . . . .> `restaurant name`,
. . . . . .> `country code`,
. . . . . .> `city`,
. . . . . .> `address`,
. . . . . .> `locality`,
. . . . . .> `locality verbose`,
. . . . . .> `longitude`,
. . . . . .> `latitude`,
. . . . . .> `cuisines`,
. . . . . .> `average cost for two`,
. . . . . .> `currency`,
. . . . . .> `has table booking`,
. . . . . .> `has online delivery`,
. . . . . .> `is delivering now`,
. . . . . .> `switch to order menu`,
. . . . . .> `price range`,
. . . . . .> `aggregate rating`,
. . . . . .> `rating text`,
. . . . . .> `votes`,
. . . . . .> `p_filedate`,
. . . . . .> `p_country_name`,
```

```
. . . . . .> `m_cuisines`,
. . . . . .> `m_rating_colour`,
. . . . . .> `create_datetime`,
. . . . . .>  `user_id`)
. . . . . .> SELECT  s.`restaurant id`,
. . . . . .> nvl( s.`restaurant name`,'NA'),
. . . . . .> s.`country code`,
. . . . . .> nvl(s.`city`, 'NA'),
. . . . . .> nvl(s.`address`, 'NA'),
. . . . . .> nvl( s.`locality`,'NA'),
. . . . . .> nvl(s.`locality verbose`,'NA'),
. . . . . .> nvl( s.`longitude`, 'NA'),
. . . . . .> nvl( s.`latitude`, 'NA'),
. . . . . .> nvl(s.`cuisines`,'NA'),
. . . . . .> s.`average cost for two`,
. . . . . .> nvl(s.`currency`,'NA'),
. . . . . .> s.`has table booking`,
. . . . . .> s.`has online delivery`,
. . . . . .> s.`is delivering now`,
. . . . . .> s.`switch to order menu`,
. . . . . .> s.`price range`,
. . . . . .> nvl(s.`aggregate rating`,'NA'),
. . . . . .> nvl(s.`rating text`,'NA'),
. . . . . .> nvl(s.`votes`,'NA'),
. . . . . .> s.`filedate`,
. . . . . .> nvl(d.`country`,'NA') ,
. . . . . .> case when cuisines like any (
. . . . . .> '%Indian%', '%Andhra%', '%Hyderabadi%', '%Goan%', '%Bengali%',
. . . . . .> '%Bihari%', '%Chettinad%', '%Gujarati%','%Rajasthani%', '%Kerala%',
. . . . . .> '%Maharashtrian%','%Mangalorean%', '%Mithai%','%Mughlai%','%%')
. . . . . .> then 'Indian' else 'World Cuisines' end as m_cuisines,
. . . . . .> case when `aggregate rating` between 1.9 and 2.4 and
. . . . . .> `rating text` = 'Poor' then 'Red' when `aggregate rating`
. . . . . .> between 2.5 and 3.4 and `rating text`='Average' then 'Amber'
. . . . . .> when `aggregate rating` between 3.5 and 3.9 and `rating  text`='Good'
. . . . . .> then 'Light Green' when `aggregate rating` between 4.0 and 4.4 and
. . . . . .> `rating text`='Very Good' then 'Green' when `aggregate rating`
. . . . . .> between 4.5 and 5 and `rating text`='Excellent' then 'Gold' when
. . . . . .> `aggregate rating` = 0 and `rating text`='Not rated' then 'NA'
. . . . . .> end as m_rating_colour, current_timestamp(), logged_in_user()
. . . . . .> from zomato_rahul s, dim_country_rahul d where s.cuisines not like '%0%' and
. . . . . .> s.`country code` = d.`country code`;
```

FileDate and Country Name Constraint Loading.

```
hive > insert into zomato_summary_rahul(
. . . . . .> `restaurant id`,
. . . . . .> `restaurant name`,
. . . . . .> `country code`,
. . . . . .> `city`,
. . . . . .> `address`,
. . . . . .> `locality`,
. . . . . .> `locality verbose`,
. . . . . .> `longitude`,
. . . . . .> `latitude`,
. . . . . .> `cuisines`,
. . . . . .> `average cost for two`,
. . . . . .> `currency`,
. . . . . .> `has table booking`,
. . . . . .> `has online delivery`,
. . . . . .> `is delivering now`,
. . . . . .> `switch to order menu`,
. . . . . .> `price range`,
. . . . . .> `aggregate rating`,
. . . . . .> `rating text`,
. . . . . .> `votes`,
. . . . . .> `p_filedate`,
. . . . . .> `p_country_name`,
. . . . . .> `m_cuisines`,
. . . . . .> `m_rating_colour`,
. . . . . .> `create_datetime`,
. . . . . .> `user_id`)
. . . . . .> SELECT  s.`restaurant id`,
. . . . . .> nvl( s.`restaurant name`,'NA'),
. . . . . .> s.`country code`,
. . . . . .> nvl(s.`city`, 'NA'),
. . . . . .> nvl(s.`address`, 'NA'),
. . . . . .> nvl( s.`locality`,'NA'),
. . . . . .> nvl(s.`locality verbose`,'NA'),
. . . . . .> nvl( s.`longitude`, 'NA'),
. . . . . .> nvl( s.`latitude`, 'NA'),
. . . . . .> nvl(s.`cuisines`,'NA'),
. . . . . .> s.`average cost for two`,
. . . . . .> nvl(s.`currency`,'NA'),
. . . . . .> s.`has table booking`,
```

```
. . . . . .> s.`has online delivery`,
. . . . . .> s.`is delivering now`,
. . . . . .> s.`switch to order menu`,
. . . . . .> s.`price range`,
. . . . . .> nvl(s.`aggregate rating`,'NA'),
. . . . . .> nvl(s.`rating text`,'NA'),
. . . . . .> nvl(s.`votes`,'NA'),
. . . . . .> s.`filedate`,
. . . . . .> nvl(d.`country`,'NA') ,
. . . . . .> case when cuisines like any (
. . . . . .> '%Indian%', '%Andhra%', '%Hyderabadi%', '%Goan%', '%Bengali%',
. . . . . .> '%Bihari%', '%Chettinad%', '%Gujarati%','%Rajasthani%', '%Kerala%',
. . . . . .> '%Maharashtrian%','%Mangalorean%', '%Mithai%','%Mughlai%')
. . . . . .> then 'Indian' else 'World Cuisines' end as m_cuisines,
. . . . . .> case when `aggregate rating` between 1.9 and 2.4 and
. . . . . .> `rating text` = 'Poor' then 'Red' when `aggregate rating`
. . . . . .> between 2.5 and 3.4 and `rating text`='Average' then 'Amber'
. . . . . .> when `aggregate rating` between 3.5 and 3.9 and `rating  text`='Good'
. . . . . .> then 'Light Green' when `aggregate rating` between 4.0 and 4.4 and
. . . . . .> `rating text`='Very Good' then 'Green' when `aggregate rating`
. . . . . .> between 4.5 and 5 and `rating text`='Excellent' then 'Gold' when
. . . . . .> `aggregate rating` = 0 and `rating text`='Not rated' then 'NA'
. . . . . .> end as m_rating_colour, current_timestamp(), logged_in_user()
. . . . . .> from zomato_rahul s, dim_country_rahul d where s.cuisines not like '%0%' and
. . . . . .> s.`country code` = d.`country code`and s.filedate = '20190610'
. . . . . .> and d.country = 'India' ;
```

Removing Duplications.

```
hive > insert into zomato_summary_rahul
. . . . . .> select distinct * from
. . . . . .> zomato_summary_rahul;
```

Question 5

A spark application to load zomato_summary table from zomato table and apply given transformations:

All the columns from the zomato table, adding to it two partition columns “p_filedate” and “p_country_name”, two derived columns “m_rating_colour” and “m_cuisines”, two audit columns “user_id” and “create_datetime” with relevant constraints.

Load data in a historical or a filedate/country constraint

Module 1: JSON to CSV Convertor - Spark Application

A spark application that retrieves a .json file from local file system onto a designated hdfs location. Creates an RDD to read the nested json file by methods of ‘explode’ and ‘write.option()’. Giving an output of the resultant .csv file and creates an hdfs file system configuration to move the file to a designated zomato tables’ location and rename it to the required “zomato_*.csv” (* -> filedate). And also copies the converted/renamed csv file into local file system.

Coding

```
import java.lang.ArrayIndexOutOfBoundsException
import org.apache.hadoop.conf.Configuration
import org.apache.spark.sql.SparkSession
import org.apache.hadoop.fs._
import org.apache.hadoop.fs.{Path => HadoopPath}
import java.sql.Timestamp
import java.text.SimpleDateFormat
import org.apache.spark.sql._
import org.apache.spark.sql.functions._
import java.io._
import org.backuity.ansi.AnsiFormatter.FormattedHelper
import java.io.File
import java.nio.file.{Files => NioFiles, Paths => NioPaths, StandardCopyOption}

object Main {

  //Function to return list of files and extension in given  directory
  def getListOfFiles(dir: File, extensions: List[String]): List[File] = {
```

```
  dir.listFiles.filter(_.isFile).toList.filter { file =>
            extensions.exists(file.getName.endsWith(_))
  }
}

//Main Function begins here
def main(args: Array[String]) = {

  //Initialising timestamp variable to keep track of this running instance
  var end_timestamp = "NA"
  val start_timestamp_retrieve = new Timestamp(System.currentTimeMillis())
  val start_timestamp = timestamp_format.format(start_timestamp_retrieve).toString
  val job_step = "JSON-TO-CSV"
  val jsontocsv_status_location = "/home/rahul.anand/zomato_etl/logs" val
  timestamp_format = new SimpleDateFormat("yyyy/MM/dd-HH:mm:ss")
  //Building Spark Session
  val spark = SparkSession.builder().master("local").appName("Module1").getOrCreate()
  //Retrieving Spark Application ID
  val raw_id=spark.sparkContext.applicationId
  val app_id=raw_id.toString

  //Initiating Try Method to catch any run time errors
  try{
  //Opening a file module_1_status.log to write the current status of the application
  var file_writer = new PrintWriter(new File(jsontocsv_status_location+"/module_1_status.log"))
  var status = "RUNNING"
  file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
  file_writer.close()

  //Creating an HDFS configuration for access hdfs directories and performing actions
  val conf = new Configuration()
  conf.set("fs.default.name","hdfs://hdp1.infocepts.com")
  val hdfs = FileSystem.get(conf)

  //To Enable the usage of '$' symbol
  //By implicitly converting an RDD to a DataFrame
  import spark.implicits._

  //Prefixing Preset Location Values For Easier Access
  val infocepts_host = "hdfs://hdp1.infocepts.com/user/rahul.anand/"
```

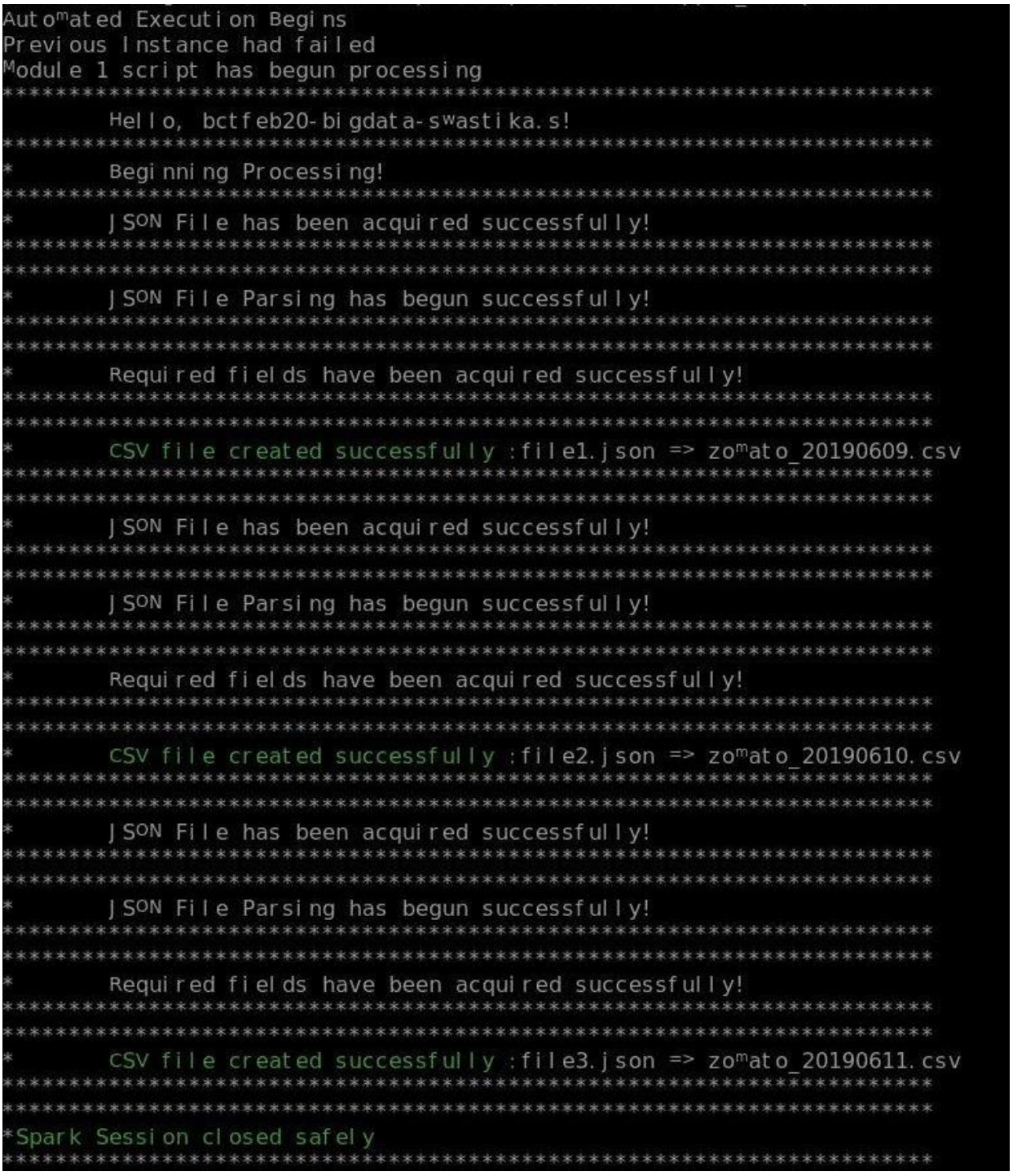
```
val swastika_home_hdfs = infocepts_host+"zomato_etl_rahul.anand" val json_source
= infocepts_host+"process_json/"
val  csv_dest  = infocepts_host+"process_json/csv/"
val file_path = "/home/rahul.anand/zomato_etl/source/json/" val
json_extension_check = List("json")
val file_path_loop = getListOfFiles(new File("/home/rahul.anand/zomato_etl/source/json"), json_extension_check)
        //Initialising Hadoop FileSystem Structure
if(!hdfs.exists(new Path(json_source))){hdfs.mkdirs(new Path(json_source))}
if(!hdfs.exists(new Path(csv_dest))){hdfs.mkdirs(new Path(csv_dest))}
if(!hdfs.exists(new Path(swastika_home_hdfs))){hdfs.mkdirs(new Path(swastika_home_hdfs))}
if(!hdfs.exists(new Path(swastika_home_hdfs+"/zomato_ext"))){hdfs.mkdirs(new
Path(swastika_home_hdfs+"/zomato_ext"))}
if(!hdfs.exists(new Path(swastika_home_hdfs+"/zomato_ext/zomato"))){hdfs.mkdirs(new
Path(swastika_home_hdfs+"/zomato_ext/zomato"))}
        //Getting current username
val user = System.getProperty("user.name")
println("**********************************************************************")
println("     Hello, "+user+"!")
println("**********************************************************************")
println("*     Beginning Processing!")

//Running a for loop to loop through files in the local filesystem (Eg. file1.json, file2.json...)
for (raw_file<-file_path_loop) {

//Splitting file path to retrieve the file name and extension
val process_file_string=raw_file.toString
val process_slash_file=process_file_string.split('/') //home/bctfeb20-bigdata-
swastika.s/zomato_etl/source/json/file1.json
val process_file=process_slash_file(6)
var file_date = args(0).toString //20190609

//Getting Required Arguments For Processing and Partitioning
if(process_file== "file1.json"){ file_date = "20190609"}
if(process_file=="file2.json"){    file_date = "20190610"}
if(process_file=="file3.json"){    file_date = "20190611"}
if(process_file=="file4.json"){    file_date = "20190612"}
if(process_file=="file5.json"){    file_date = "20190613"}

//Copying required file from local folder to hdfs temporary directory for processing
```

```
hdfs.copyFromLocalFile(new Path(file_path+process_file), new Path(json_source + process_file))

//Creating an RDD to read the json file
val read_file = spark.read.json(json_source + process_file)

println("**********************************************************************")
println("*     JSON File has been acquired successfully!")
println("**********************************************************************")

//Explode on the json unstructured data to retrieve required columns
val explode_rdd = read_file.select(explode($"restaurants.restaurant"))

println("**********************************************************************")
println("*     JSON File Parsing has begun successfully!")
println("**********************************************************************")

//Retrieving required columns from json
val final_rdd = explode_rdd.select($"col.R.res_id" as "Restaurant_Id", $"col.name" as "Restaurant Name",
$"col.location.country_id" as "Country Code", $"col.location.city" as "City", $"col.location.address" as "Address",
$"col.location.locality" as "Locality", $"col.location.locality_verbose" as "Locality Verbose",
$"col.location.longitude" as "Longitude", $"col.location.latitude" as "Latitude", $"col.cuisines" as "Cuisines",
$"col.average_cost_for_two" as "Average Cost for two", $"col.currency" as "currency", $"col.has_table_booking" as
"Has table booking", $"col.has_online_delivery" as "Has Online delivery", $"col.is_delivering_now" as "Is
delivering now", $"col.Switch_to_order_menu" as "Switch to order menu", $"col.price_range" as "Price range",
$"col.user_rating.aggregate_rating" as "Aggregate rating",$"col.user_rating.rating_text" as "Rating text",
$"col.user_rating.votes" as "Votes")

println("**********************************************************************")
println("*     Required fields have been acquired successfully!")
println("**********************************************************************")
//Writing the required columns into a temporary destination csv path
final_rdd.write.option("delimiter", "\t").csv(csv_dest+process_file)

println("**********************************************************************")
println(ansi"*    %green{CSV file created successfully :}")
print(process_file+" => "+"zomato_"+file_date+".csv")
println("**********************************************************************")
//Initialising Path variables to rename, move and copy to local execution of the csv file as well as deleting the
temporary directory
```

```
val part_file = hdfs.globStatus(new Path("hdfs://hdp1.infocepts.com/user/bctfeb20-bigdata-swastika.s/process_json/csv/"+process_file+"/part*"))(0).getPath().getName()
val csv_zomato_destination = new Path("hdfs://hdp1.infocepts.com/user/bctfeb20-bigdata-swastika.s/zomato_etl_rahul.anand/zomato_ext/zomato/zomato_"+file_date+".csv")
val part_file_source = new Path("hdfs://hdp1.infocepts.com/user/bctfeb20-bigdata-swastika.s/process_json/csv/"+process_file+"/"+part_file)
val csv_local=new HadoopPath("/home/bctfeb20-bigdata-swastika.s/zomato_etl/source/csv/zomato_"+file_date+".csv")
val json_local=NioPaths.get("/home/rahul.anand/zomato_etl/source/json/"+process_file) val
archive_local=NioPaths.get("/home/rahul.anand/zomato_etl/archive/"+process_file)
//renaming and moving the csv file to the zomato table location
hdfs.rename(part_file_source, csv_zomato_destination)
//copying the csv file to the local filesystem location
hdfs.copyToLocalFile(csv_zomato_destination, csv_local)
//moving the processed json files from source/json to archive folder in the local file system
NioFiles.move(json_local, archive_local, StandardCopyOption.REPLACE_EXISTING)
}
//Deleting the temporary hdfs directory
val temp_hdfs_dir = new Path(json_source)
hdfs.delete(temp_hdfs_dir, true)
spark.close()
println("**********************************************************************")
println(ansi"*%green{Spark Session closed safely}")
println("**********************************************************************")
//Writing the status log file to "SUCCESS" status after closing the spark session
file_writer=new PrintWriter(new File(jsontocsv_status_location+"/module_1_status.log"))
var end_time_retrieve = new Timestamp(System.currentTimeMillis())
end_timestamp = timestamp_format.format(end_time_retrieve).toString
status = "SUCCESS"
file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
file_writer.close()
}
catch{
case e: IOException => {
println("**********************************************************************")
println(ansi"%red{Had an IOException trying to read that file}")
println("**********************************************************************")
var file_writer=new PrintWriter(new File(jsontocsv_status_location+"/module_1_status.log"))
var end_time_retrieve = new Timestamp(System.currentTimeMillis())
end_timestamp = timestamp_format.format(end_time_retrieve).toString
```

```
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    case e: FileNotFoundException => {
    println("**********************************************************************")
    println(ansi"%red{Caught File Not Found Exception!}")
    println("**********************************************************************")
    var file_writer = new PrintWriter(new File(jsontocsv_status_location+"/jsontocsv_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    case e: ArrayIndexOutOfBoundsException => {
    println("**********************************************************************")
    println(ansi"%red{Caught Array Index Out Of Bounds Exception, Kindly Input Parameters On Invoking This Script}")
    println("**********************************************************************")
    var file_writer = new PrintWriter(new File(jsontocsv_status_location+"/jsontocsv_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    case _: Throwable => {
    println("**********************************************************************")
    println(ansi"%red{Caught an Error, Kindly Refer Logs. Failed Status Updated!}")
    println("**********************************************************************")
    var file_writer = new PrintWriter(new File(jsontocsv_status_location+"/module_1_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    }
  }}
```

Output Screenshots



Module 2: Load Data From CSV To zomato Table - Spark Application

A spark application that retrieves a .csv file from hdfs location into zomato table. Creating a loop to execute only .csv files and catch all exceptions. Connecting to hive using JDBC connect statement and loading data and creating partitions into zomato table using the filedate mentioned in the .csv filename

Coding

```
import org.apache.spark.sql.SparkSession
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.io.FileNotFoundException
import java.lang.ArrayIndexOutOfBoundsException
import java.sql.DriverManager;
import java.io.IOException
import org.apache.hadoop.fs._
import java.io._
import org.backuity.ansi.AnsiFormatter.FormattedHelper
import java.sql.Timestamp
import java.text.SimpleDateFormat

object Main{

  //Main  Function  begins   here
  def main(args: Array[String]) = {

    //Initialising timestamp variable to keep track of this running instance
    val timestamp_format = new SimpleDateFormat("yyyy/MM/dd-HH:mm:ss")
    val start_timestamp_retrieve = new Timestamp(System.currentTimeMillis())
    val start_timestamp = timestamp_format.format(start_timestamp_retrieve).toString
    val loadzomato_status_location = "/home/rahul.anand/zomato_etl/logs"
    //Building Spark Session
    val spark = SparkSession.builder().master("local").appName("Module2").getOrCreate()
    //Retrieving Spark Application ID
    val raw_id=spark.sparkContext.applicationId
    val app_id=raw_id.toString
    var end_timestamp ="NA"
    var job_step = "LOAD-CSV-TO-ZOMATO-TABLE"
    //Initiating Try Method to catch any run time errors
```

```
try{

var status = "RUNNING"
//Opening a file module_1_status.log to write the current status of the application
var file_writer = new PrintWriter(new File(loadzomato_status_location+"/module_2_status.log"))
file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
file_writer.close()

//Creating an HDFS configuration for access hdfs directories and performing actions
val hdfs_configuration = spark.sparkContext.hadoopConfiguration

//Prefixing Preset Location Values For Easier Access
val load_file_path = new Path("hdfs://hdp1.infocepts.com/user/bctfeb20-bigdata-
swastika.s/zomato_etl_rahul.anand/zomato_ext/zomato")
val file_system = load_file_path.getFileSystem(hdfs_configuration)
val user = System.getProperty("user.name")

println("**************************************************************************")
println("*Hello, "+user+"!")
println("**************************************************************************")
println("*Proceeding with loading data!")
println("**************************************************************************")

//obtaining HiveDriver package to set Class for JDBC connection
val driver_name = "org.apache.hive.jdbc.HiveDriver";

//Initialise Class with driver_name
Class.forName(driver_name);

println("**************************************************************************")
println("*Establishing JDBC Connnection")
println("**************************************************************************")

//Establishing JDBC Connection
val jdbc_connect =
DriverManager.getConnection("jdbc:hive2://hdp1.infocepts.com:2181,hdp2.infocepts.com:2181,hdp3.infocepts.com
:2181/default;password=rahul.anand;serviceDiscoveryMode=zooKeeper;user=bctfeb20-bigdata-
swastika.s;zooKeeperNamespace=hiveserver2", "rahul.anand", "rahul.anand123")

//Initialising a jdbc statement to enable execution of queries, etc
```

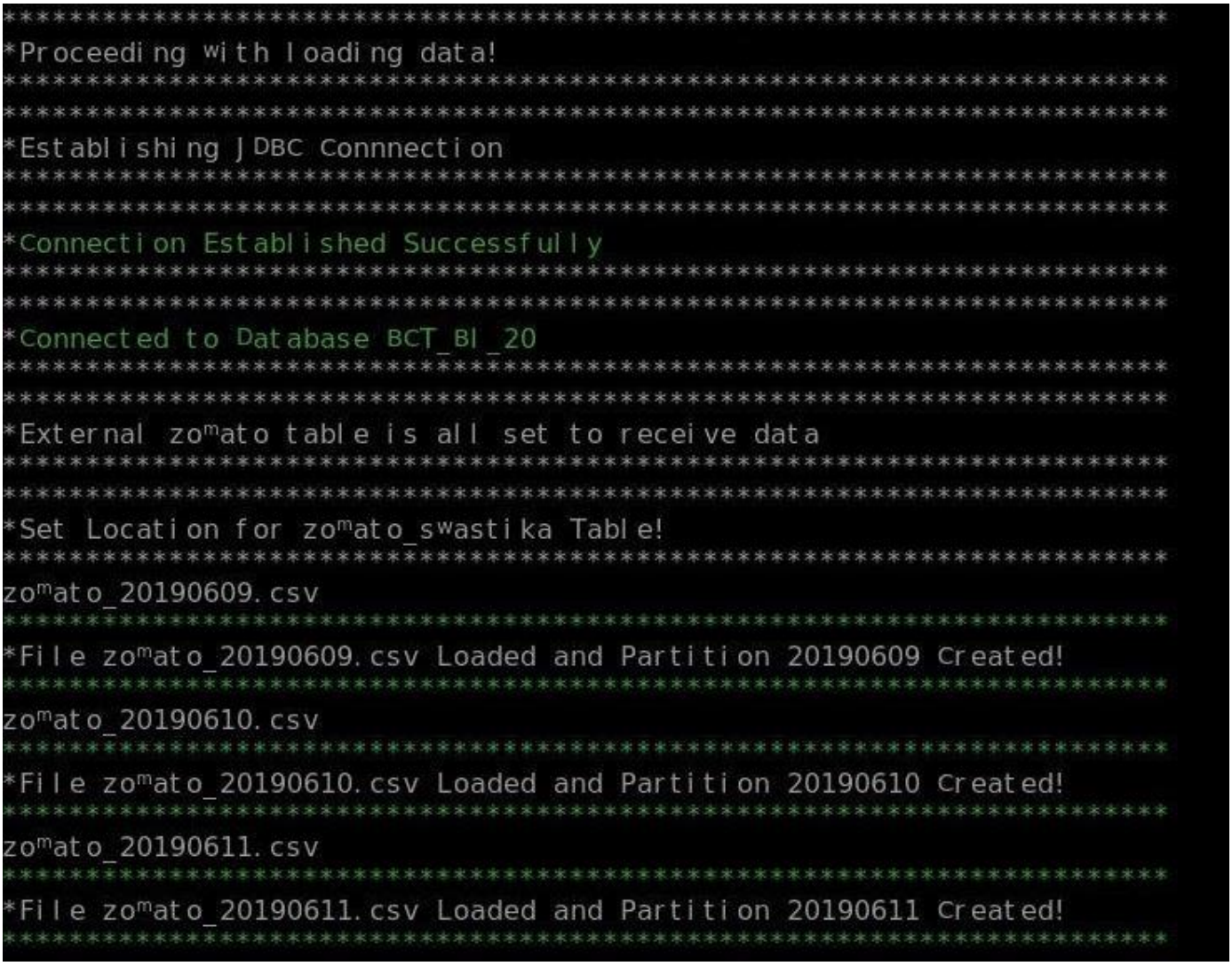
```
val hive_statement = jdbc_connect.createStatement()

//Initialising a query_text var to enable input of query in String format
var query_text = " "

println("*********************************************************************")
println(ansi"*%green{Connection Established Successfully}")
println("*********************************************************************")

query_text = "USE rahul_database"
hive_statement.execute(query_text)

println("*********************************************************************")
println(ansi"*%green{Connected to Database RAHUL_DATABASE}")
println("*********************************************************************")

//Creating zomato table if not exists
query_text = "CREATE EXTERNAL TABLE IF NOT EXISTS rahul_database.zomato_rahul(`Restaurant ID` INT,`Restaurant Name` STRING,`Country Code` INT,`City` STRING, `Address` STRING,`Locality` STRING,`Locality Verbose` STRING,`Longitude` STRING,`Latitude` STRING,`Cuisines` STRING,`Average Cost for two` INT,`Currency` STRING,`Has Table booking` INT,`Has Online delivery` INT,`Is delivering now` INT,`Switch to order menu` INT,`Price range` INT,`Aggregate rating` STRING,`Rating text` STRING,`Votes` STRING) PARTITIONED BY ( filedate int ) ROW FORMAT DELIMITED fields terminated by '\t'stored as textfile"

hive_statement.execute(query_text)

println("*********************************************************************")
println("*External zomato table is all set to receive data")
println("*********************************************************************")

//Setting location of zomato table
query_text = "ALTER TABLE zomato_rahul SET LOCATION 'hdfs://Theflopguyproductions/user/bctfeb20-bigdata- swastika.s/zomato_etl_rahul.anand/zomato_ext/zomato'"
hive_statement.execute(query_text)

println("*********************************************************************")
println("*Set Location for zomato_rahul Table!")
println("*********************************************************************")
```

```
    //Run a for loop to loop through the csv files located in the given location
    val file_status  =  file_system.globStatus(new  Path(load_file_path+"/zomato_*"))
    for (urlStatus <- file_status) {

    //Split the filename to get extension to avoid IOException
    var file_name=urlStatus.getPath.getName
    var dot_split = file_name.split('.')// zomato_20190609, csv || 20190609
    var first_name = dot_split(0).toString//zomato_20190609        ||
    var extension = dot_split(1).toString//csv
    var underscore_split = first_name.split('_')
    var partition_name = underscore_split(1).toString
    var file_path = load_file_path+"/"+file_name
    println(file_name)

    //Loading Data Into zomato Table
    query_text = "LOAD DATA INPATH '"+file_path+"' OVERWRITE INTO TABLE zomato_rahul
PARTITION (filedate = '"+partition_name+"')"
    hive_statement.execute(query_text)

    println(ansi"%green{**********************************************************************}")
    println("*File "+file_name+" Loaded and Partition "+partition_name+" Created!")
    println(ansi"%green{**********************************************************************}")

    }

    //Closing All Connections Safely
    hive_statement.close()
    jdbc_connect.close()
    spark.close()

    //Writing the final status of the execution as "SUCCESS"
    file_writer=new PrintWriter(new File(loadzomato_status_location+"/module_2_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    status = "SUCCESS"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()

    }
    catch{
```

```
    case e: IOException => {
    println("*******************************************************************")
    println(ansi"%red{Had an IOException trying to read that file}")
    println("*******************************************************************")
    var file_writer = new PrintWriter(new File(loadzomato_status_location+"/module_2_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-TO-ZOMATO-TABLE"
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()

    }

    case e: FileNotFoundException => {

    println("*******************************************************************")
    println(ansi"%red{Caught File Not Found Exception!}")
    println("*******************************************************************")

    var file_writer = new PrintWriter(new File(loadzomato_status_location+"/module_2_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-TO-ZOMATO-TABLE"
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()

    }

    case e: ArrayIndexOutOfBoundsException => {

    println("*******************************************************************")
    println(ansi"%red{Caught Array Index Out Of Bounds Exception, Kindly Input Parameters On Invoking This
Script}")
    println("*******************************************************************************")
    var file_writer = new PrintWriter(new File(loadzomato_status_location+"/module_2_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-TO-ZOMATO-TABLE"
```

```
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    case _: Throwable => {
    println("*********************************************************************")
    println(ansi"%red{Caught an Error, Kindly Refer Logs. Failed Status Updated!}")
    println("*********************************************************************")
     var file_writer = new PrintWriter(new File(loadzomato_status_location+"/module_2_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-TO-ZOMATO-TABLE"
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    }
  }
}
```

Output Screenshot



Module 3: Load Data From zomato Table To zomato_summary table - Spark Application

A spark application that combines all the data from zomato table and matches the country id with the data from dim_country table to add a column with country name from dim_country while creating audit column and derived column as required by constraints. It also gives the user an option to choose between historical loading and criteria/manual loading of data.

Coding

```
import org.apache.spark.sql.SparkSession
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;
import java.io.IOException
import java.io._
import java.io.FileNotFoundException
import java.sql.Timestamp
import java.text.SimpleDateFormat
import org.backuity.ansi.AnsiFormatter.FormattedHelper


object Main{

  //Main Function begins here
  def main(args: Array[String]) = {

    //Initialising timestamp variable to keep track of this running instance
    val loadsummary_status_location = "/home/rahul.anand/zomato_etl/logs" var
    end_timestamp = "NA"
    val timestamp_format = new SimpleDateFormat("yyyy/MM/dd-HH:mm:ss")
    val start_timestamp_retrieve = new Timestamp(System.currentTimeMillis())
    //Building Spark Session
    val start_timestamp = timestamp_format.format(start_timestamp_retrieve).toString
    val spark = SparkSession.builder().master("local").appName("Module3").getOrCreate()
    //Retrieving Spark Application ID
    val raw_id=spark.sparkContext.applicationId
    val app_id=raw_id.toString
```

```
//Initiating Try Method to catch any run time errors
try{
//Opening a file module_1_status.log to write the current status of the application
var file_writer = new PrintWriter(new File(loadsummary_status_location+"/module_3_status.log"))
var job_step = "LOAD-ZOMATO-SUMMARY"

var status = "RUNNING"

file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)

file_writer.close()

val input_con = System.console()
val user = System.getProperty("user.name")

println("**************************************************************************")
println("*Hello, "+user+"!")
println("**************************************************************************")
println("*Proceeding with loading data!")
println("**************************************************************************")


//obtaining HiveDriver package to set Class for JDBC connection
val driver_name = "org.apache.hive.jdbc.HiveDriver";


//Initialise Class with driver_name
Class.forName(driver_name);

println("**************************************************************************")
println("*Establishing JDBC Connnection")
println("**************************************************************************")


//Establishing JDBC Connection
val jdbc_connect =
DriverManager.getConnection("jdbc:hive2://hdp1.infocepts.com:2181,hdp2.infocepts.com:2181,hdp3.infocepts.com
:2181/default;password=rahul.anand;serviceDiscoveryMode=zooKeeper;user=bctfeb20-bigdata-
swastika.s;zooKeeperNamespace=hiveserver2", "rahul.anand", "rahul.anand123")
```

```
//Initialising a jdbc statement to enable execution of queries, etc
val hive_statement = jdbc_connect.createStatement()

//Initialising a query_text var to enable input of query in String format
var query_text = " "

//Initialising a choice variable
var choice = 0

println("**************************************************************************")
println(ansi"*%green{Connection Established Successfully}")
println("**************************************************************************")

query_text = "USE rahul_database"
hive_statement.execute(query_text)

println("**************************************************************************")
println(ansi"*%green{Connected to Database RAHUL_DATABASE}")
println("**************************************************************************")

//Get Input Option From User
if(args.length>0){
if(args(0)=="1"){
println("**************************************************************************")
println("*")
println("*Proceeding with updation with criteria")
println("*")
choice = 1
}
if(args(0)=="2"){

println("**************************************************************************")
println("*")
println("*       Proceeding with historical updation")
println("*")

choice = 2
}
```

```
}

if(args.length==0){
println("**********************************************************************")
println("*")
println("*Enter an Apt Option To Proceed")
println("*")
println(ansi"%yellow{To Load Data With FileData and Country Specified-->1}")
println(ansi"%yellow{To Load Data Historically-->2}")
print  (">>")

choice = Integer.parseInt(input_con.readLine())
}

println("**********************************************************************")

//Create zomato_summary table if it doesn't exist
query_text = "CREATE TABLE  IF  NOT  EXISTS  zomato_summary_rahul(`Restaurant  ID` INT,`Restaurant Name` STRING,`Country Code` INT,`City` STRING, `Address` STRING,`Locality` STRING,`Locality  Verbose` STRING,`Longitude`  STRING,`Latitude`  STRING,`Cuisines`  STRING,`Average  Cost for two` INT,`Currency` STRING,`Has Table booking` INT,`Has Online delivery` INT,`Is delivering now` INT,`Switch to order menu` INT,`Price range` INT,`Aggregate rating` STRING,`Rating text` STRING,`Votes` STRING, `m_rating_colour` STRING,  `m_cuisines`  STRING,  `create_datetime`  TIMESTAMP,  `user_id` STRING) PARTITIONED BY ( p_filedate INT, p_country_name STRING ) stored as ORC"
hive_statement.execute(query_text)

println("**********************************************************************")
println("*Managed zomato_summary table is all set to receive data")
println("**********************************************************************")

choice match {
case 1 => {

//Manual data updation
println("**********************************************************************")
print("*Please Enter Filedate to load >>")
val file_date = input_con.readLine()
println("**********************************************************************")
print("*Please Enter Country Name to load >>")
val country_name = input_con.readLine()
```

println("**")

query_text = "INSERT INTO zomato_summary_rahul(`restaurant id`, `restaurant name`, `country code`,`city`,`address`,`locality`,`locality verbose`,`longitude`,`latitude`,`cuisines`,`average cost for two`,`currency`,`has table booking`,`has online delivery`,`is delivering now`,`switch to order menu`,`price range`,`aggregate rating`,`rating text`,`votes`,`p_filedate`, `p_country_name`, `m_cuisines`, `m_rating_colour`, `create_datetime`, `user_id`) SELECT s.`restaurant id`, nvl(s.`restaurant name`,'NA'), s.`country code`, nvl(s.`city`, 'NA'), nvl(s.`address`, 'NA'), nvl(s.`locality`,'NA'), nvl(s.`locality verbose`,'NA'), nvl(s.`longitude`, 'NA'), nvl(s.`latitude`, 'NA'), nvl(s.`cuisines`,'NA'),s.`average cost for two`, nvl(s.`currency`,'NA'), s.`has table booking`, s.`has online delivery`,s.`is delivering now`,s.`switch to order menu`,s.`price range`,nvl(s.`aggregate rating`,'NA'), nvl(s.`rating text`,'NA'), nvl(s.`votes`,'NA'),s.`filedate`, nvl(d.`country`,'NA') , case when cuisines like any ('%Indian%', '%Andhra%', '%Hyderabadi%', '%Goan%', '%Bengali%', '%Bihari%', '%Chettinad%', '%Gujarati%','%Rajasthani%', '%Kerala%', '%Maharashtrian%','%Mangalorean%', '%Mithai%','%Mughlai%') then 'Indian' else 'World Cuisines' end as m_cuisines, case when `aggregate rating` between 1.9 and 2.4 and `rating text` = 'Poor' then 'Red' when `aggregate rating` between 2.5 and 3.4 and `rating text`='Average' then 'Amber' when `aggregate rating` between 3.5 and 3.9 and `rating text`='Good' then 'Light Green' when `aggregate rating` between 4.0 and 4.4 and `rating text`='Very Good' then 'Green' when `aggregate rating` between 4.5 and 5 and `rating text`='Excellent' then 'Gold' when `aggregate rating` = 0 and `rating text`='Not rated' then 'NA' end as m_rating_colour, current_timestamp(), logged_in_user() from zomato_rahul s, dim_country_rahul d where s.cuisines not like '%0%' and s.`country code` = d.`country code` and s.filedate = "'+file_date+"' and d.country = "'+country_name+"'"

hive_statement.execute(query_text)

println("**")
println("*Loaded Data into zomato_summary table With Specified Criteria")
println("**")

query_text = "INSERT OVERWRITE TABLE zomato_summary_rahul SELECT DISTINCT * FROM zomato_summary_rahul"
hive_statement.execute(query_text)

println("**")
println("*Duplicate values removed from table zomato_rahul")
println("**")

}

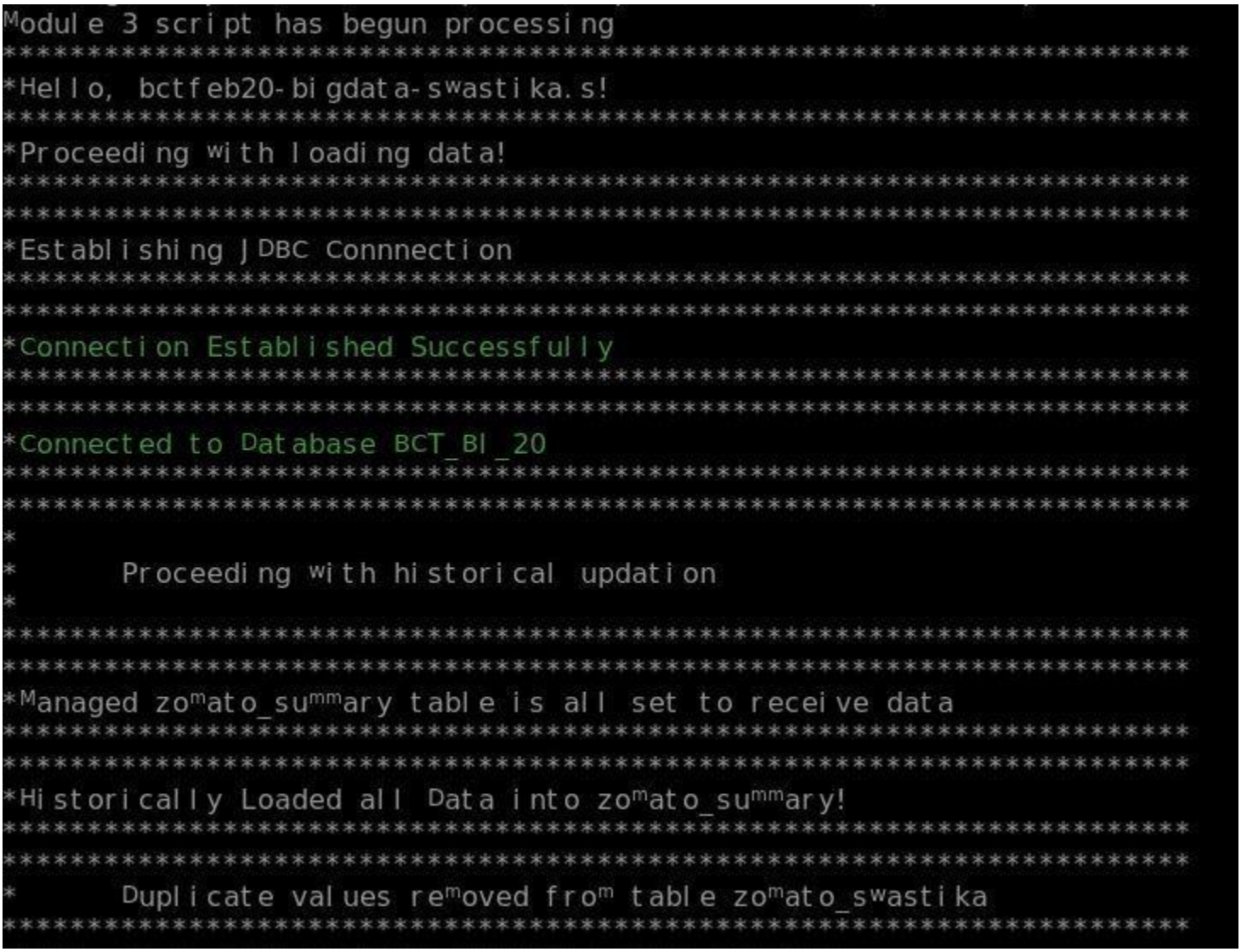
case 2 => {

```
//Historical data updation
query_text = "INSERT INTO zomato_summary_rahul(`restaurant id`, `restaurant  name`, `country code`,`city`,`address`,`locality`,`locality verbose`,`longitude`,`latitude`,`cuisines`,`average cost for two`,`currency`,`has table booking`,`has online delivery`,`is delivering now`,`switch to order menu`,`price range`,`aggregate rating`,`rating text`,`votes`,`p_filedate`, `p_country_name`, `m_cuisines`, `m_rating_colour`, `create_datetime`, `user_id`) SELECT  s.`restaurant id`, nvl( s.`restaurant name`,'NA'), s.`country code`, nvl(s.`city`, 'NA'), nvl(s.`address`, 'NA'), nvl( s.`locality`,'NA'), nvl(s.`locality verbose`,'NA'), nvl( s.`longitude`, 'NA'), nvl( s.`latitude`, 'NA'), nvl(s.`cuisines`,'NA'),s.`average cost for two`, nvl(s.`currency`,'NA'), s.`has table booking`, s.`has online delivery`,s.`is delivering now`,s.`switch to order menu`,s.`price range`,nvl(s.`aggregate rating`,'NA'), nvl(s.`rating text`,'NA'),  nvl(s.`votes`,'NA'),s.`filedate`, nvl(d.`country`,'NA') , case when cuisines like any ('%Indian%', '%Andhra%', '%Hyderabadi%', '%Goan%', '%Bengali%', '%Bihari%', '%Chettinad%', '%Gujarati%','%Rajasthani%', '%Kerala%', '%Maharashtrian%','%Mangalorean%', '%Mithai%','%Mughlai%') then 'Indian' else 'World Cuisines' end as m_cuisines, case when `aggregate rating` between 1.9 and 2.4 and `rating text` = 'Poor' then 'Red' when `aggregate rating` between 2.5 and 3.4 and `rating text`='Average' then 'Amber' when `aggregate rating` between 3.5 and 3.9and `rating text`='Good' then 'Light Green' when `aggregate rating` between 4.0 and 4.4 and `rating text`='Very Good' then 'Green' when `aggregate rating` between 4.5 and 5 and `rating text`='Excellent' then 'Gold' when `aggregate rating` = 0 and `rating text`='Not rated' then 'NA' end as m_rating_colour, current_timestamp(), logged_in_user() from zomato_rahul s, dim_country_rahul d where s.cuisines not like '%0%' and s.`country code` = d.`country code`"

hive_statement.execute(query_text)

println("**********************************************************************")
println("*Historically Loaded all Data into zomato_summary!")
println("**********************************************************************")

query_text = "INSERT OVERWRITE TABLE zomato_summary_rahul SELECT DISTINCT * FROM zomato_summary_rahul"
hive_statement.execute(query_text)

println("**********************************************************************")
println("*         Duplicate values removed from table zomato_rahul")
println("**********************************************************************")

}

case _ => {
```

```
//Default case called if wrong input given
println("*************************************************************************")
println(ansi"*%red{Invalid Option Chosen!}")
println("*************************************************************************")
}

println("*************************************************************************")
println(ansi"*%green{Module 3 Completed!}")
println("*************************************************************************")

}

//Closing All Connections Safely
hive_statement.close()
jdbc_connect.close()
spark.close()

//Writing the status log file to "SUCCESS" status after closing the spark session
file_writer = new PrintWriter(new File(loadsummary_status_location+"/module_3_status.log"))
var end_time_retrieve = new Timestamp(System.currentTimeMillis())
end_timestamp = timestamp_format.format(end_time_retrieve).toString
job_step = "LOAD-ZOMATO-SUMMARY-TABLE"
status = "SUCCESS"
file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
file_writer.close()

}
catch{
case e: IOException => {
println("*************************************************************************")
println(ansi"%red{Had an IOException trying to read that file}")
println("*************************************************************************")
var file_writer = new PrintWriter(new File(loadsummary_status_location+"/module_3_status.log"))
var end_time_retrieve = new Timestamp(System.currentTimeMillis())
end_timestamp = timestamp_format.format(end_time_retrieve).toString
var job_step = "LOAD-ZOMATO-SUMMARY-TABLE"
var status = "FAILED"
file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
file_writer.close()
}
```

```
    case e: FileNotFoundException => {
    println("*************************************************************************")
    println(ansi"%red{Caught File Not Found Exception!}")
    println("*************************************************************************")
    var file_writer = new PrintWriter(new File(loadsummary_status_location+"/module_3_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-ZOMATO-SUMMARY-TABLE"
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    case _: Throwable => {
    println("*************************************************************************")
    println(ansi"%red{Got some other kind of exception, Exiting!}")
    println("*************************************************************************")
    var file_writer = new PrintWriter(new File(loadsummary_status_location+"/module_3_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-ZOMATO-SUMMARY-TABLE"
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    case e: ArrayIndexOutOfBoundsException => {
    println("*************************************************************************")
    println(ansi"%red{Caught Array Index Out Of Bounds Exception, Kindly Input Parameters On Invoking This
Script}")
    println("*************************************************************************")
    var file_writer = new PrintWriter(new File(loadsummary_status_location+"/module_3_status.log"))
    var end_time_retrieve = new Timestamp(System.currentTimeMillis())
    end_timestamp = timestamp_format.format(end_time_retrieve).toString
    var job_step = "LOAD-ZOMATO-SUMMARY-TABLE"
    var status = "FAILED"
    file_writer.write(app_id +"\t"+job_step+"\t"+start_timestamp+"\t"+end_timestamp+"\t"+status)
    file_writer.close()
    }
    }
  }
}
```

Output Screenshot



build.sbt

A common build.sbt file used to add dependencies as well as specify app-name/version in root dir of each module

Coding

```
crossPaths:=false

name := "<Module Name>"
version := "1.0"
scalaVersion := "2.11.8"

//Default libraryDependencies for spark sql as well as hadoop operations
libraryDependencies +="org.apache.spark" %% "spark-sql" % "2.1.0"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.1.0"
libraryDependencies += "org.apache.hadoop" % "hadoop-common" % "2.6.0"
libraryDependencies += "org.apache.commons" % "commons-io" % "1.3.2"
libraryDependencies += "org.apache.hadoop" % "hadoop-hdfs" % "2.6.0"

//ColoredFontFunctionality
libraryDependencies +="org.backuity" %% "ansi-interpolator" % "1.1" % "provided"

//LoggingFunctionality
libraryDependencies += "com.typesafe.scala-logging" %% "scala-logging" % "3.9.2"
libraryDependencies += "log4j" % "apache-log4j-extras" % "1.1"
```

log4j-spark.properties

Default log4j.properties file as a common file for all the modules that specifies the details and category of information that needs to be stored in the log file of the specificmodule_log

Coding

log_location = /home/rahul.anand/zomato_etl/logs app_log_name = <modulename>_log

log4j.rootCategory=INFO,FILE

log4j.appender.FILE=org.apache.log4j.rolling.RollingFileAppender

log4j.appender.FILE.RollingPolicy=org.apache.log4j.rolling.TimeBasedRollingPolicy log4j.appender.FILE.RollingPolicy.FileNamePattern=${log_location}/${app_log_name}_%d{ddMMyyyy_HHmm}.log log4j.appender.FILE.layout=org.apache.log4j.PatternLayout log4j.appender.FILE.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

Module1 Shell Script: Call Module1 Spark Application

A shell script that calls the spark-submit command to execute the first module after checking for current instances and previously failed/successful instances. It also updates to the zomato_summary_log table the status of the current execution after the execution and sends a mail to the user post completion of the execution or if the previous instance had failed with relevant details.

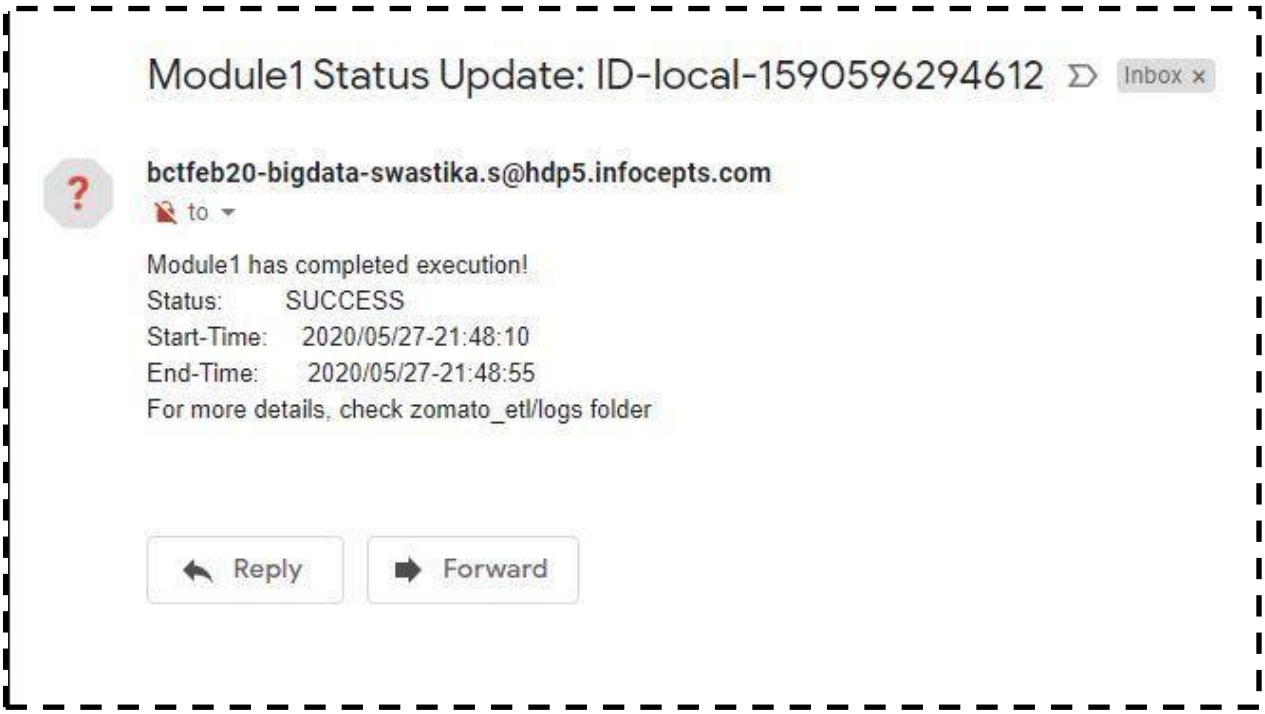
Coding

```
#!/bin/sh

#Initialising default variables
file_status_path=/home/rahul.anand/zomato_etl/logs/module_1_status.log
mail_list="swastikachaubey2122@gmail.com rahul.anand@hdp5" module_nomen="Module1"

#Mail function to send mail on status updation
function mail(){
   module_name=$1
   module_status=$2
   module_starttime=$3
   module_endtime=$4
   module_id=$5
   echo -e "Subject: $module_name Status Update: ID-$module_id\n\n$module_name has completed execution!\nStatus:\t\t$module_status\nStart-Time:\t$module_starttime\nEnd-Time:\t$module_endtime\nFor more details, check zomato_etl/logs folder" | /usr/sbin/sendmail $mail_list
}

#Spark submit function to call the spark submit command and update the status
function spark_submit() {

   echo "Module 1 script has begun processing"
   json_source_path=/home/rahul.anand/zomato_etl/source/json/*
   spark_submit_value="spark-submit --driver-java-options -Dlog4j.configuration=file:/home/bctfeb20-bigdata-swastika.s/zomato_etl/spark/scala/jsoncsv/src/main/resources/log4j-spark.properties --class Main --master yarn --deploy-mode client /home/rahul.anand/zomato_etl/spark/scala/jsoncsv/target/module1-1.0.jar"
   current_date=$(date +"%Y%m%d")
   $spark_submit_value $current_date
```

```
    updation "$spark_submit_value"
}

#Update function to update the status log and call the mail function
function updation(){

    spark_value=$1

    declare -a update_value
    if test -f "$file_status_path"; then

        update_value=(`cat $file_status_path`)

        #Beeline command to load status log into the zomato_summary_log table
        beeline -u
"jdbc:hive2://hdp1.infocepts.com:2181,hdp2.infocepts.com:2181,hdp3.infocepts.com:2181/;serviceDiscoveryMode=z
ooKeeper;zooKeeperNamespace=hiveserver2" -n "rahul.anand" -e "insert into
rahul_database.zomato_summary_log_rahul
values('${update_value[0]}','${update_value[1]}','$spark_value','${update_value[2]}','${update_value[3]}','${update_value
[4]}')"

        if [ ${update_value[4]}="SUCCESSFUL" ]; then {
        mail $module_nomen "SUCCESS" ${update_value[2]} ${update_value[3]} ${update_value[0]}
        }
        elif [ ${update_value[4]}="FAILED" ]; then {
        mail $module_nomen "FAILED" ${update_value[2]} ${update_value[3]} ${update_value[0]}
        }
        elif [ ${update_value[4]}="RUNNING" ]; then {
        mail $module_nomen "Unsuccessfully RUNNING" ${update_value[2]} ${update_value[3]} ${update_value[0]}
        }
        else {
        mail $module_nomen "Unknown" ${update_value[2]} ${update_value[3]} ${update_value[0]}
        }
        fi

        else
        echo "Unable to get updated instance"
        echo "Could not update zomato_summary_log table"
        fi
}
```

```
#Array to hold status.log file name
declare -a file_value
if test -f "$file_status_path"; then

   file_value=(`cat $file_status_path`)

   #Case to run application based on running instance check
   case "${file_value[4]}" in
      "SUCCESS")
      spark_submit
      ;;
      "FAILED")
      echo "Previous Instance had failed"
      mail $module_nomen "Previous Instance Had Failed" ${file_value[2]} ${file_value[3]} ${ file_value[0]}
      spark_submit
      ;;
      "RUNNING")
      echo "Previous instance is still running!"
      echo "Aborting"
      mail $module_nomen "Previous Instance is still running" ${file_value[2]} ${file_value[3]} ${ file_value[0]}

      ;;
      *)
      mail $module_nomen "Something went wrong, Removing status logs!" ${file_value[2]} ${file_value[3]}
${ file_value[0]}
      echo "Something went wrong, restarting this module!"
      echo "Removing corrupted status log"
      rm "$file_status_path"
      spark_submit
      ;;
   esac
else
   echo "Status file not found, Running Spark Application"
   spark_submit
fi
```

Output Mail Screenshot



Module2 Shell Script: Call Module2 Spark Application

A shell script that calls the spark-submit command to execute the second module after checking for current instances and previously failed/successful instances. It also updates to the zomato_summary_log table the status of the current execution after the execution and sends a mail to the user post completion of the execution or if the previous instance had failed with relevant details.

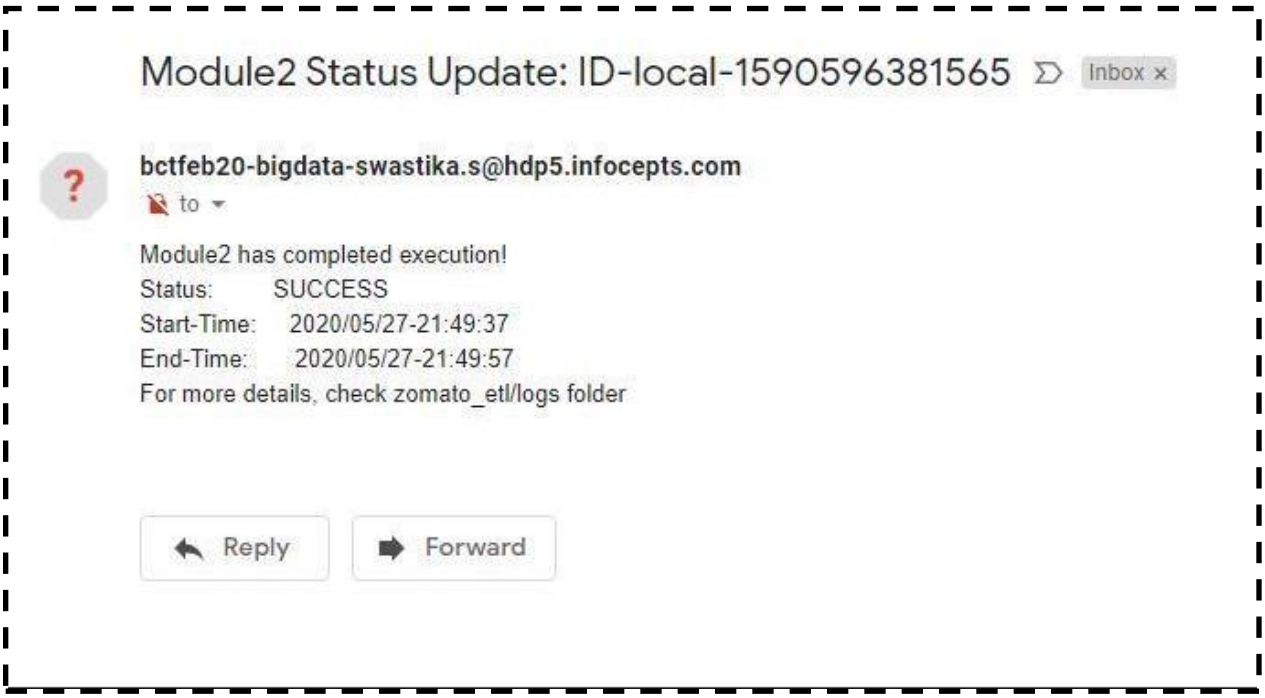
Coding

```
#!/bin/sh

#Initialising default variables
file_status_path=/home/rahul.anand/zomato_etl/logs/module_2_status.log spark_submit_value="spark-submit --driver-java-options -Dlog4j.configuration=file:/home/bctfeb20-bigdata-swastika.s/zomato_etl/spark/scala/Module2/src/main/resources/log4j-spark.properties --class Main --master yarn --deploy-mode client /home/rahul.anand/zomato_etl/spark/scala/Module2/target/module2-1.0.jar"
mail_list="swastikachaubey2122@gmail.com rahul.anand@hdp5"
module_nomen="Module2"

#Mail function to send mail on status updation
function mail(){

   module_name=$1
   module_status=$2
   module_starttime=$3
   module_endtime=$4
   module_id=$5
   echo -e "Subject: $module_name Status Update: ID-$module_id\n\n$module_name has completed execution!\nStatus:\t\t$module_status\nStart-Time:\t$module_starttime\nEnd-Time:\t$module_endtime\nFor more details, check zomato_etl/logs folder" | /usr/sbin/sendmail $mail_list

}

#Spark submit function to call the spark submit command and update the status
function spark_submit() {
   echo "Module 2 script has begun processing"
   $spark_submit_value
   updation
}
```

```
#Update function to update the status log and call the mail function
function updation(){

   declare -a update_value
   if test -f "$file_status_path"; then
      update_value=(`cat $file_status_path`)

      #Beeline command to load status log into the zomato_summary_log table
      beeline -u
"jdbc:hive2://hdp1.infocepts.com:2181,hdp2.infocepts.com:2181,hdp3.infocepts.com:2181/;serviceDiscoveryMode=z
ooKeeper;zooKeeperNamespace=hiveserver2" -n "rahul.anand" -e "insert into
rahul_database.zomato_summary_log_rahul
values('${update_value[0]}','${update_value[1]}','$spark_submit_value','${update_value[2]}','${update_value[3]}','${upda
te_value[4]}')"

      if [ ${update_value[4]}="SUCCESSFUL" ]; then {
      mail $module_nomen "SUCCESS" ${update_value[2]} ${update_value[3]} ${update_value[0]}
      }
      elif [ ${update_value[4]}="FAILED" ]; then {
      mail $module_nomen "FAILED" ${update_value[2]} ${update_value[3]} ${update_value[0]}
      }
      elif [ ${update_value[4]}="RUNNING" ]; then {
      mail $module_nomen "Unsuccessfully exited with RUNNING" ${update_value[2]} ${update_value[3]}
${update_value[0]}
      }
      else {
      mail $module_nomen "Unknown" ${update_value[2]} ${update_value[3]} ${update_value[0]}
      }

      fi



      else
      echo "Unable to get updated instance"
      echo "Could not update zomato_summary_log table"
   fi
}

#Array to hold status.log file name
```

```
declare -a file_value

if test -f "$file_status_path"; then
   file_value=(`cat "$file_status_path"`)

   #Case to run application based on running instance check
   case "${file_value[4]}" in
      "SUCCESS")
      echo "Found SUCCESS"
      spark_submit
      ;;
      "FAILED")
      echo "Previous instance failed, sent more details in mail!"
      mail $module_nomen "Previous Instance had failed!" ${file_value[2]} ${file_value[3]} ${ file_value[0]}
      spark_submit
      ;;
      "RUNNING")
      mail $module_nomen "Previous Instance is still running!" ${file_value[2]} ${file_value[3]} ${ file_value[0]} echo
      "Previous Instance is still running"
      ;;
      *)
      mail $module_nomen "Corrupted Status Logs Found, Restarting Module!" ${file_value[2]} ${file_value[3]}
${ file_value[0]}
      echo "Something went wrong, restarting this module!"
      echo "Removing corrupted Status files"
      rm "$file_status_path"
      spark_submit
      ;;
   esac
   else
   echo "Status file not found, Running Spark Application"
   spark_submit
fi
```

Output Mail Screenshot



Module3 Shell Script: Call Module3 Spark Application

A shell script that calls the spark-submit command to execute the third module after checking for current instances and previously failed/successful instances. It also updates to the zomato_summary_log table the status of the current execution after the execution and sends a mail to the user post completion of the execution or if the previous instance had failed with relevant details.

Coding

```
#!/bin/sh

#Get argument from user
opt=$1

#Initialising default variables
file_status_path=/home/rahul.anand/zomato_etl/logs/module_3_status.log spark_submit_value="spark-submit --driver-java-options -Dlog4j.configuration=file:/home/bctfeb20-bigdata-swastika.s/zomato_etl/spark/scala/Module3/src/main/resources/log4j-spark.properties --class Main --master yarn --deploy-mode client /home/rahul.anand/zomato_etl/spark/scala/Module3/target/module3-1.0.jar"
mail_list="swastikachaubey2122@gmail.com rahul.anand@hdp5"
module_nomen="Module3"

#Mail function to send mail on status updation
function mail(){
   module_name=$1
   module_status=$2
   module_starttime=$3
   module_endtime=$4
   module_id=$5
   echo -e "Subject: $module_name Status Update: ID-$module_id\n\n$module_name has completed execution!\nStatus:\t\t$module_status\nStart-Time:\t$module_starttime\nEnd-Time:\t$module_endtime\nFor moredetails, check zomato_etl/logs folder" | /usr/sbin/sendmail $mail_list
}

#Default usage function called with incorrect option is given
function usage() {
   echo "usage:${0} OPTION"
   echo "1 <- Update Zomato Summary With Filedate/Country Criteria"
   echo "2 <- Update Zomato Summary With Historical Criteria"
   exit 1
```

```
}

#Spark submit function to call the spark submit command and update the status
function spark_submit() {
   case $opt in
      1)
      echo "Module 3 script has begun processing"
      $spark_submit_value "1"
      updation
      ;;
      2)
      echo "Module 3 script has begun processing"
      $spark_submit_value "2"
      updation
      ;;
      "")
      echo "Module 3 script has begun processing"
      $spark_submit_value
      updation
      ;;
      *)
      usage
      exit 1
      ;;
   esac
}

#Update function to update the status log and call the mail function
function updation(){
   declare -a update_value
   if test -f "$file_status_path"; then
      update_value=(`cat $file_status_path`)
      beeline -u
"jdbc:hive2://hdp1.infocepts.com:2181,hdp2.infocepts.com:2181,hdp3.infocepts.com:2181/;serviceDiscoveryMode=z
ooKeeper;zooKeeperNamespace=hiveserver2" -n "rahul.anand" -e "insert into
rahul_database.zomato_summary_log_rahul
values('${update_value[0]}','${update_value[1]}','$spark_submit_value','${update_value[2]}','${update_value[3]}','${upda
te_value[4]}')"
      if [ ${update_value[4]}="SUCCESSFUL" ]; then {
      mail $module_nomen "SUCCESS" ${update_value[2]} ${update_value[3]} ${update_value[0]}
```

```
    }
    elif [ ${update_value[4]}="FAILED" ]; then {
    mail $module_nomen "FAILED" ${update_value[2]} ${update_value[3]} ${update_value[0]}
    }
    elif [ ${update_value[4]}="RUNNING" ]; then {
    mail $module_nomen "Unsuccessfully RUNNING" ${update_value[2]} ${update_value[3]} ${update_value[0]}
    }
    else {
    mail $module_nomen "Unknown" ${update_value[2]} ${update_value[3]} ${update_value[0]}
    }
    fi
    else
    echo "Unable to get updated instance"
    echo "Could not update zomato_summary_log table"
  fi
}

#Array to hold status.log file name
declare -a file_value
if test -f "$file_status_path"; then
  file_value=(`cat "$file_status_path"`)

  #Case to run application based on running instance check
  case "${file_value[4]}" in
    "SUCCESS")
    spark_submit
    ;;
    "FAILED")
    echo "Previous Instance had failed, Details sent in mail!"
    mail $module_nomen "Previous Instance had failed." ${file_value[2]} ${file_value[3]} ${ file_value[0]}
    spark_submit
    ;;
    "RUNNING")
    echo "Previous Instance is still running!"
    mail $module_nomen "Previous Instance is still running." ${file_value[2]} ${file_value[3]} ${ file_value[0]}
    ;;
    *)
    mail $module_nomen "Status log file seems corrupted. Deleting and proceesing!" ${file_value[2]}${file_value[3]}
${ file_value[0]}
    echo "Something went wrong, restarting this module!"
```

```
        rm "$file_status_path"
        spark_submit
        ;;
    esac
    else
    echo "Status file not found, Running Spark Application"
    spark_submit
fi
```

Output Mail Screenshot

Purge_Logs Shell Script:

A shell script that traverses through all the files present in the logs folder and checks the file date difference between the creation date of the log file based on the nomenclature of the log file and the current date and runs an if condition to delete the files that are under 7 days old and more than 1 day old in order to save the logs of the most recent execution for reference purposes.

Coding

```
#!/bin/sh
#Initialising variable with default data
log_path=/home/rahul.anand/zomato_etl/logs/
current_date=$(date +"%d%m%Y")
echo "Purging Begins"
#Looping through the list of all files in the path
for file in `ls $log_path`;
do
   #Retrieving only the file name from the list
   file_name="$(basename "$file")"

   #Searching of the specific naming pattern log
   file_name_pattern=".{7}_log_.{8}_.*.log"
   if [[ $file_name =~ $file_name_pattern ]]; then

      #Retreiving the creation date from the name of the log file
      log_date=$(awk -F '_' '{print $3}' <<< $file_name)
      #Calculating difference
      difference="$(($current_date - $log_date))"
      #Checking the constraint
      if (($difference>0 && $difference<70000000)); then
      echo "Deleting Log :" $file_name
      #Deleting the log file
      rm "$log_path$file_name"
      fi
   fi
done
echo "Purge Complete!"
```

Wrapper Shell Script:

A wrapper shell script that gets one argument from the user with a default case that calls the usage function to help the user choose the argument correctly in case otherwise. It calls the modules and the purge_logs shell script to clean the logs directory after the execution of the module.

Coding

```
#!/bin/sh

#Gets option from User
opt=$1

#Default usage function to help the user incase of a wrong argument
function usage() {
   echo "usage:${0} OPTION"
   echo "1 <- Execute Module 1 [Load CSV File]"
   echo "2 <- Execute Module 2 [Load into zomato table]"
   echo "3 <- Execute Module 3 [Load into zomato_summary table]"
   echo "4 <- Execute Module 4 [Run all the modules in preset sequence]"
   exit 1
}

#Runs a case statement on the argument given by user
case $opt in

   #Module 1 is called along with purge_logs
   1)
      bash /home/rahul.anand/zomato_etl/script/module_1.sh bash
      /home/rahul.anand/zomato_etl/script/purge_logs.sh
      ;;

   #Module 2 is called along with purge_logs
   2)
      bash /home/rahul.anand/zomato_etl/script/module_2.sh bash
      /home/rahul.anand/zomato_etl/script/purge_logs.sh
      ;;

   #Module 3 is called along with purge_logs
```

```
    3)
        bash /home/rahul.anand/zomato_etl/script/module_3.sh bash
        /home/rahul.anand/zomato_etl/script/purge_logs.sh
        ;;



    #All the modules are called along with purge_logs
    4)
        echo "Automated Execution Begins"
        bash /home/rahul.anand/zomato_etl/script/module_1.sh bash
        /home/rahul.anand/zomato_etl/script/module_2.sh bash
        /home/rahul.anand/zomato_etl/script/module_3.sh "2" bash
        /home/rahul.anand/zomato_etl/script/purge_logs.sh
        ;;

    #Default Module is called with usage function
    *)
        usage
        exit 1
        ;;
esac
```

Question 6

Moving the files from source folder to archive folder once the execution is complete. This step has been covered within the spark application to ensure the removal of redundancy and ease of execution.

Question 7

Add new source files into the source folder to execute the complete workflow again. This step has been completed using the “cp” command in terminal.

Question 8 - Crontab cronjob setting

Creating a cronjob using crontab to run all the three modules automatically at 1am everyday.

Coding

```
>> crontab -e

0    1        *        *        *        bash /home/rahul.anand/zomato_etl/script/wrapper_script.sh 4
```