Practice3

March 25, 2021

In [ ]:
```
#%run -i 'etl.py'
```

In [11]:
```
import configparser
from datetime import datetime
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col
from pyspark.sql.functions import year, month, dayofmonth, hour, weekofyear, date_forma


config = configparser.ConfigParser()
config.read('pp_test.cfg')

os.environ['AWS_ACCESS_KEY_ID']=config['AWS']['AWS_ACCESS_KEY_ID']
os.environ['AWS_SECRET_ACCESS_KEY']=config['AWS']['AWS_SECRET_ACCESS_KEY']


spark = SparkSession \
        .builder \
        .config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:2.7.0") \
        .getOrCreate()
```

In [12]:
```
input_data = "s3a://udacity-dend/song_data/A/B/C/"
#input_data = "s3a://datalakepp/song_data" # my bucket

song_data = input_data + "*.json"
#song_data = "s3a://udacity-dend/song_data/A/B/C/TRABCEI128F424C983.json"
```

In [13]:
```
df = spark.read.json(song_data)
```

In [14]:
```

```

```
success
```

In [15]:
```
df.printSchema()
df.show(5)
```

```
root
 |-- artist_id: string (nullable = true)
 |-- artist_latitude: double (nullable = true)
 |-- artist_location: string (nullable = true)
 |-- artist_longitude: double (nullable = true)
 |-- artist_name: string (nullable = true)
 |-- duration: double (nullable = true)
 |-- num_songs: long (nullable = true)
 |-- song_id: string (nullable = true)
 |-- title: string (nullable = true)
 |-- year: long (nullable = true)

+------------------+---------------+--------------------+----------------+--------------------+-
|         artist_id|artist_latitude|     artist_location|artist_longitude|         artist_name|
+------------------+---------------+--------------------+----------------+--------------------+-
|ARLTWXK1187FB5A3F8|       32.74863|      Fort Worth, TX|       -97.32925|         King Curtis|3
|ARIOZCU1187FB3A3DC|           null|          Hamlet, NC|            null|       JOHN COLTRANE|2
|ARPFHN61187FB575F6|       41.88415|         Chicago, IL|       -87.63241|         Lupe Fiasco|2
|AR5S9OB1187B9931E3|       34.05349|     Los Angeles, CA|      -118.24532|         Bullet Boys|1
|AR5T4OY1187B9996C6|           null|       Lulea, Sweden|            null|    The Bear Quartet|
|AR9OEB71187B9A97C6|           null|Edmonton, Alberta...|            null|              Faunts|3
|ARBDJHO1252CCFA6FC|           null|                    |            null|The Band of HM Ro...|1
|ARAADXM1187FB3ECDB|        34.1688|  Woodland Hills, CA|      -118.61092|    Styles Of Beyond|
|ARZJDBC1187FB52056|       27.94017|    Brandon, Florida|       -82.32547|        Nasty Savage|3
|AROSPS51187B9B481F|           null|                    |            null| Vince Guaraldi Trio|1
|AROIAWL1187B9A96D0|         8.4177|              Panama|       -80.11278|        Danilo Perez|1
|ARCWVUK1187FB3C71A|           null|                    |            null|     Brigitte Bardot|
|ARZGTK71187B9AC7F5|           null|     California, USA|            null|                Eels|1
|ARWB3G61187FB49404|           null|      Hamilton, Ohio|            null|         Steve Morse|3
|ARCKOJF1241B9C75B4|           null|                    |            null|        Eddie Sierra|2
+------------------+---------------+--------------------+----------------+--------------------+-
only showing top 15 rows
```

```
In [16]: print("success")
```

```
success
```

```
In [32]: output_data = "s3a://udacity-dend/"
         #df = spark.read.json(song_data)

             # extract columns to create songs table
             # Using dataframe property, create a new dataframe with required fields
         songs_table = df['song_id', 'title', 'artist_id', 'year', 'duration']
             # Drop duplicates
         songs_table = songs_table.dropDuplicates()
```

```
songs_table.head()
```

```
Out[32]: Row(song_id='SOQFYBD12AB0182188', title='Intro', artist_id='ARAADXM1187FB3ECDB', year=1
```

```
In [34]:     # write songs table to parquet files partitioned by year and artist
         songs_table.write.partitionBy('year', 'artist_id').parquet(os.path.join(output_data, 's
```

```
        ---------------------------------------------------------------------------
        Py4JJavaError                             Traceback (most recent call last)

        <ipython-input-34-da64ebc0ad49> in <module>()
          1 # write songs table to parquet files partitioned by year and artist
        ----> 2 songs_table.write.partitionBy('year', 'artist_id').parquet(os.path.join(output_data,


        /opt/spark-2.4.3-bin-hadoop2.7/python/pyspark/sql/readwriter.py in parquet(self, path, m
        837             self.partitionBy(partitionBy)
        838         self._set_opts(compression=compression)
        --> 839         self._jwrite.parquet(path)
        840
        841     @since(1.6)


        /opt/spark-2.4.3-bin-hadoop2.7/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py in __
       1255         answer = self.gateway_client.send_command(command)
       1256         return_value = get_return_value(
       -> 1257             answer, self.gateway_client, self.target_id, self.name)
       1258
       1259         for temp_arg in temp_args:


        /opt/spark-2.4.3-bin-hadoop2.7/python/pyspark/sql/utils.py in deco(*a, **kw)
         61     def deco(*a, **kw):
         62         try:
        ---> 63             return f(*a, **kw)
         64         except py4j.protocol.Py4JJavaError as e:
         65             s = e.java_exception.toString()


        /opt/spark-2.4.3-bin-hadoop2.7/python/lib/py4j-0.10.7-src.zip/py4j/protocol.py in get_re
        326                 raise Py4JJavaError(
        327                     "An error occurred while calling {0}{1}{2}.\n".
        --> 328                     format(target_id, ".", name), value)
        329             else:
        330                 raise Py4JError(
```

```
    Py4JJavaError: An error occurred while calling o440.parquet.
: com.amazonaws.services.s3.model.AmazonS3Exception: Status Code: 403, AWS Service: Amazon S
        at com.amazonaws.http.AmazonHttpClient.handleErrorResponse(AmazonHttpClient.java:798
        at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.java:421)
        at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:232)
        at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:3528)
        at com.amazonaws.services.s3.AmazonS3Client.putObject(AmazonS3Client.java:1393)
        at org.apache.hadoop.fs.s3a.S3AFileSystem.createEmptyObject(S3AFileSystem.java:1194)
        at org.apache.hadoop.fs.s3a.S3AFileSystem.createFakeDirectory(S3AFileSystem.java:117
        at org.apache.hadoop.fs.s3a.S3AFileSystem.mkdirs(S3AFileSystem.java:871)
        at org.apache.hadoop.fs.FileSystem.mkdirs(FileSystem.java:1881)
        at org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter.setupJob(FileOutputCom
        at org.apache.spark.internal.io.HadoopMapReduceCommitProtocol.setupJob(HadoopMapRedu
        at org.apache.spark.sql.execution.datasources.FileFormatWriter$.write(FileFormatWrit
        at org.apache.spark.sql.execution.datasources.InsertIntoHadoopFsRelationCommand.run(
        at org.apache.spark.sql.execution.command.DataWritingCommandExec.sideEffectResult$lz
        at org.apache.spark.sql.execution.command.DataWritingCommandExec.sideEffectResult(co
        at org.apache.spark.sql.execution.command.DataWritingCommandExec.doExecute(commands.
        at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala
        at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala
        at org.apache.spark.sql.execution.SparkPlan$$anonfun$executeQuery$1.apply(SparkPlan.
        at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
        at org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:152)
        at org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:127)
        at org.apache.spark.sql.execution.QueryExecution.toRdd$lzycompute(QueryExecution.sca
        at org.apache.spark.sql.execution.QueryExecution.toRdd(QueryExecution.scala:80)
        at org.apache.spark.sql.DataFrameWriter$$anonfun$runCommand$1.apply(DataFrameWriter.
        at org.apache.spark.sql.DataFrameWriter$$anonfun$runCommand$1.apply(DataFrameWriter.
        at org.apache.spark.sql.execution.SQLExecution$$anonfun$withNewExecutionId$1.apply(S
        at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecution.s
        at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.scal
        at org.apache.spark.sql.DataFrameWriter.runCommand(DataFrameWriter.scala:676)
        at org.apache.spark.sql.DataFrameWriter.saveToV1Source(DataFrameWriter.scala:285)
        at org.apache.spark.sql.DataFrameWriter.save(DataFrameWriter.scala:271)
        at org.apache.spark.sql.DataFrameWriter.save(DataFrameWriter.scala:229)
        at org.apache.spark.sql.DataFrameWriter.parquet(DataFrameWriter.scala:566)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:238)
```

```
        at java.lang.Thread.run(Thread.java:748)
```

In [ ]:
```
print("done")
```

In [ ]:
```
```