Chapter 5: Data Pre-processing

Ex2: AA data

Cho dữ liệu về thông tin các chuyến bay trong thư mục "AA_data"

Yêu cầu:

1. Đọc dữ liệu => df
2. Cho biết dữ liệu có bao nhiêu dòng, in scheme. Hiển thị 5 dòng dữ liệu đầu tiên.
3. Kiểm tra dữ liệu NaN, null
4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.
5. Trong df, thêm cột 'airport' lấy dữ liệu từ cột 'Destination Airport', định dạng chữ thường cho nội dung
6. Trong df, thêm cột 'date' lấy dữ liệu từ cột 'Date (MM/DD/YYYY)', sau đó xóa bỏ cột 'Date (MM/DD/YYYY)'
7. Trong df, đổi tên cột "Flight Number" thành "flight_num", cột "Actual elapsed time (Minutes)" thành "actual_time"
8. Lưu df dưới dạng Parquet format với tên là "AA_DFW_ALL.parquet"
9. Đọc parquet "AA_DFW_ALL.parquet" => df_new
10. Tạo một bảng tạm 'flights'. Cho biết trung bình của 'actual_time' trong 'flight'
11. Caching các dòng dữ liệu duy nhất của df_new. Đếm số dòng. Cho biết thời gian thực hiện các công việc này.
12. Đếm lại số dòng. Cho biết thời gian thực hiện các công việc này.
13. Kiểm tra xem df_new có trong cache hay không? Nếu có thì bỏ df_new ra khỏi cache.

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark import SparkContext
from pyspark.conf import SparkConf
from pyspark.sql import SparkSession
```

In [3]:
```
sc =SparkContext()
```

In [4]:
```
spark = SparkSession(sc)
```

In [5]:
```
#1.
df = spark.read.csv(["AA_data"], header=True, inferSchema=True)
```

In [6]:
```
#2.
df.count()
```

Out[6]:
```
583718
```

In [7]:
```
df.show(5)
```

```
+----------------+------------+------------------+-------------------------
---+
|Date (MM/DD/YYYY)|Flight Number|Destination Airport|Actual elapsed time (Minut
es)|
+----------------+------------+------------------+-------------------------
---+
|       01/01/2014|           5|               HNL|
519|
|       01/01/2014|           7|               OGG|
505|
|       01/01/2014|          35|               SLC|
174|
|       01/01/2014|          43|               DTW|
153|
|       01/01/2014|          52|               PIT|
137|
+----------------+------------+------------------+-------------------------
---+
only showing top 5 rows
```

In [8]:
```
df.printSchema()
```

```
root
 |-- Date (MM/DD/YYYY): string (nullable = true)
 |-- Flight Number: integer (nullable = true)
 |-- Destination Airport: string (nullable = true)
 |-- Actual elapsed time (Minutes): integer (nullable = true)
```

In [9]:
```
from pyspark.sql.functions import col, udf
from pyspark.sql.functions import isnan, when, count, col
```

In [10]:
```
#3. Kiểm tra dữ liệu NaN, null
df.select([count(when(isnan(c), c)).alias(c) for c in df.columns]).toPandas().T
```

Out[10]:

	0
Date (MM/DD/YYYY)	0
Flight Number	0
Destination Airport	0
Actual elapsed time (Minutes)	0

In [11]:
```
# => Không có dữ liệu NaN
```

In [12]:
```
df.select([count(when(col(c).isNull(), c)).alias(c) for c in
           df.columns]).toPandas().T
```

Out[12]:

	0
Date (MM/DD/YYYY)	0
Flight Number	0
Destination Airport	0
Actual elapsed time (Minutes)	0

In [13]:
```
# Không có dữ liệu null
```

In [14]:
```
#4. Kiểm tra dữ liệu trùng. Xóa dữ liệu trùng.
```

In [15]:
```
num_rows = df.count()
```

In [16]:
```
num_dist_rows = df.distinct().count()
```

In [17]:
```
dup_rows = num_rows - num_dist_rows
```

In [18]:
```
dup_rows
```

Out[18]:
```
0
```

In [19]:
```
# Không có dữ liệu trùng
```

Lazy processing operations

In [20]:
```
from pyspark.sql.functions import *
```

In [21]:
```
#5. Add the airport column using the F.lower() method
df = df.withColumn('airport', lower(df['Destination Airport']))
```

In [22]:
```
df = df.drop('Destination Airport')
```

In [23]:
```
df.show(5)
```

```
+----------------+------------+---------------------------+-------+
|Date (MM/DD/YYYY)|Flight Number|Actual elapsed time (Minutes)|airport|
+----------------+------------+---------------------------+-------+
|      01/01/2014|           5|                        519|    hnl|
|      01/01/2014|           7|                        505|    ogg|
|      01/01/2014|          35|                        174|    slc|
|      01/01/2014|          43|                        153|    dtw|
|      01/01/2014|          52|                        137|    pit|
+----------------+------------+---------------------------+-------+
only showing top 5 rows
```

In [24]:
```
#6. Add column date, using column Date (MM/DD/YYYY), drop  Date (MM/DD/YYYY)
```

In [25]:
```
df = df.withColumn('date', df['Date (MM/DD/YYYY)'])
```

In [26]:
```
df = df.drop('Date (MM/DD/YYYY)')
```

In [27]:
```
df.show(5)
```

```
+------------+---------------------------+-------+----------+
|Flight Number|Actual elapsed time (Minutes)|airport|      date|
+------------+---------------------------+-------+----------+
|           5|                        519|    hnl|01/01/2014|
|           7|                        505|    ogg|01/01/2014|
|          35|                        174|    slc|01/01/2014|
|          43|                        153|    dtw|01/01/2014|
|          52|                        137|    pit|01/01/2014|
+------------+---------------------------+-------+----------+
only showing top 5 rows
```

In [28]:
```
#7.
df = df.withColumnRenamed("Flight Number", "flight_num")
df = df.withColumnRenamed("Actual elapsed time (Minutes)", "actual_time")
```

In [29]:
```
df.show(5)
```

```
+----------+-----------+-------+----------+
|flight_num|actual_time|airport|      date|
+----------+-----------+-------+----------+
|         5|        519|    hnl|01/01/2014|
|         7|        505|    ogg|01/01/2014|
|        35|        174|    slc|01/01/2014|
|        43|        153|    dtw|01/01/2014|
|        52|        137|    pit|01/01/2014|
+----------+-----------+-------+----------+
only showing top 5 rows
```

Parquet format

In [30]:
```
#8. Save the df DataFrame in Parquet format
df.write.parquet('AA_DFW_ALL.parquet.1', mode='overwrite')
```

In [31]:
```
#9 Read the Parquet file into a new DataFrame
df_new = spark.read.parquet('AA_DFW_ALL.parquet.1')
```

In [32]:
```
print(df_new.count())
```

```
583718
```

SQL and Parquet

In [33]:
```
#10. Register the temp table
df_new.createOrReplaceTempView('flights')
```

In [34]:
```
# Run a SQL query of the average Actual elapsed time
avg_duration = spark.sql('SELECT avg(actual_time) from flights').collect()[0]
print('The average flight time is: %d' % avg_duration)
```

```
The average flight time is: 147
```

Improving Performance

Caching a DataFrame

- Caching can improve performance when reusing DataFrames

In [35]:
```
import time
```

In [36]:
```
11.
start_time = time.time()

# Add caching to the unique rows in df_new
df_new = df_new.distinct().cache()

# Count the unique rows in df_new, noting how long the operation takes
print("Counting %d rows took %f seconds" %
      (df_new.count(), time.time() - start_time))
```

```
Counting 583718 rows took 2.336898 seconds
```

In [37]:
```
# Count the rows again, noting the variance in time of a cached DataFrame
start_time = time.time()
print("Counting %d rows again took %f seconds" %
      (df_new.count(), time.time() - start_time))
```

```
Counting 583718 rows again took 0.877655 seconds
```

Removing a DataFrame from cache

In [38]:
```
# Determine if df_new is in the cache
print("Is df_new cached?: %s" % df_new.is_cached)
print("Removing df_new from cache")

# Remove df_new from the cache
df_new.unpersist()

# Check the cache status again
print("Is df_new cached?: %s" % df_new.is_cached)
```

```
Is df_new cached?: True
Removing df_new from cache
Is df_new cached?: False
```

- Note: Converting to a larger number of files with approximately equal quantity of rows lets Spark decide how best to read the data.

Cluster configurations

In [39]:
```
# Name of the Spark application instance
app_name = spark.conf.get('spark.app.name')

# Driver TCP port
driver_tcp_port = spark.conf.get('spark.driver.port')

# Number of join partitions
num_partitions = spark.conf.get('spark.sql.shuffle.partitions')

# Show the results
print("Name: %s" % app_name)
print("Driver TCP port: %s" % driver_tcp_port)
print("Number of partitions: %s" % num_partitions)
```

```
Name: pyspark-shell
Driver TCP port: 62414
Number of partitions: 200
```

In [40]:
```
# Store the number of partitions in variable
before = df_new.rdd.getNumPartitions()

# Configure Spark to use 500 partitions
spark.conf.set('spark.sql.shuffle.partitions', 500)

# Recreate the DataFrame using the departures data file
df_new = spark.read.parquet('AA_DFW_ALL.parquet').distinct()

# Print the number of partitions for each instance
print("Partition count before change: %d" % before)
print("Partition count after change: %d" % df_new.rdd.getNumPartitions())
```

```
Partition count before change: 200
Partition count after change: 500
```

In [41]:
```
### save data to json file
```

In [42]:
```
#df_new.write.json('AA_DFW_ALL.json')
```