Apache Spark Structured API

- The Structured APIs are a tool for manipulating all sorts of data.
- Unstructured log files to semi-structured CSV files.
- Highly structured Parquet files.
- These APIs refer to the following core types of distributed collection:
- SQL tables and views, DataFrames and Datasets

Spark SQL

- You can run SQL queries against views or tables organized into databases
- You also can use system functions or define user functions and analyze query plans in order to optimize their workloads.
- This integrates directly into the DataFrame and Dataset API
- You can choose to express some of your data manipulations in SQL and others in DataFrames

In [1]:
```
# Import depedicies
from pyspark.sql import SparkSession
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
spark =SparkSession(sc)
```

Creating Tables

- CREATE and DROP tables
- Create table from .json file
- Create table from .csv file

In [2]:
```
# Drop table flights
spark.sql("""
DROP TABLE IF EXISTS flights
""")
```

Out[2]:
```
DataFrame[]
```

In [3]:
```
# Create table 'flights' with columns
# DEST_COUNTRY_NAME STRING, ORIGIN_COUNTRY_NAME STRING, count LONG
# From .json file:path Resources/20015-summary.json
spark.sql("""
CREATE TABLE flights (
DEST_COUNTRY_NAME STRING, ORIGIN_COUNTRY_NAME STRING, count LONG)
USING JSON OPTIONS (path '/Resources/2015-summary.json')
""")
```

Out[3]:
```
DataFrame[]
```

In [4]:
```
# Drop flights_csv
spark.sql("""
DROP TABLE IF EXISTS flights_csv
""")
```

Out[4]:
```
DataFrame[]
```

In [5]:
```
# Create table 'flights_csv' with columns
# DEST_COUNTRY_NAME STRING, ORIGIN_COUNTRY_NAME STRING, count LONG
# From .csv file 20015-summary.csv
spark.sql("""
CREATE TABLE flights_csv (
DEST_COUNTRY_NAME STRING,
ORIGIN_COUNTRY_NAME STRING COMMENT "remember, the US will be most prevalent",
count LONG)
USING csv OPTIONS (header true, path '/Resources/2015-summary.csv')
""")
```

Out[5]:
```
DataFrame[]
```

Views

- Creating Views
- Creating Temporary views that are available only during the curent session
- Overwrite and replace view if one already exists from previously

In [6]:
```
# We create a view(just_usa_view) in which the des_country_name is United States in ord
spark.sql("""
CREATE VIEW just_usa_view AS
SELECT * FROM flights WHERE dest_country_name = 'United States'
""")
```

Out[6]:
```
DataFrame[]
```

In [7]:
```
# Create a temporary view(just_usa_view) in which the destination is United States in o
spark.sql("""
CREATE TEMP VIEW just_usa_view_temp AS
SELECT * FROM flights WHERE dest_country_name = 'United States'
""")
```

Out[7]:
```
DataFrame[]
```

In [8]:
```
# Overwrite and replace view(just_usa_view) if one already exists from previous
spark.sql("""
CREATE OR REPLACE TEMP VIEW just_usa_view_temp AS
SELECT * FROM flights WHERE dest_country_name = 'United States'
""")
```

Out[8]:
```
DataFrame[]
```

DataFrame Transformations

- Adding/removing rows
- Transform row into column(or vice versa)
- Changing the order of rows based on the values in columns

Creating DataFrames

In [9]:
```
# Create a new DataFrame and register as a temporary view to query it in SQL(SQL trasfo
# For Quering in SQL, name the SQL dataset table  (2015-summary-json)
df = spark.read.format("json").load("Resources/2015-summary.json")
df.createOrReplaceTempView("dfTable")
```

In [10]:
```
# Return the schema(StructType) of the dataFrame: Schemas define the name as well as th
# Return the the first five column records
# Return statistics for numeric columns
# Return the logical and physical plans. DataFrame lineage(how Spark executes query)
# Return the datatypes
df.printSchema()
df.show(5)
df.describe().show()
df.explain()
df.dtypes
```

```
root
 |-- DEST_COUNTRY_NAME: string (nullable = true)
 |-- ORIGIN_COUNTRY_NAME: string (nullable = true)
 |-- count: long (nullable = true)

+-----------------+-------------------+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----------------+-------------------+-----+
|    United States|            Romania|   15|
|    United States|            Croatia|    1|
|    United States|            Ireland|  344|
|            Egypt|      United States|   15|
|    United States|              India|   62|
+-----------------+-------------------+-----+
only showing top 5 rows

+-------+-----------------+-------------------+------------------+
|summary|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|             count|
+-------+-----------------+-------------------+------------------+
|  count|              256|                256|               256|
|   mean|             null|               null|       1770.765625|
| stddev|             null|               null|23126.516918551915|
|    min|          Algeria|             Angola|                 1|
|    max|           Zambia|            Vietnam|            370002|
+-------+-----------------+-------------------+------------------+

== Physical Plan ==
*(1) FileScan json [DEST_COUNTRY_NAME#9,ORIGIN_COUNTRY_NAME#10,count#11L] Batched: fals
e, Format: JSON, Location: InMemoryFileIndex[file:/C:/Users/tenle/Documents/Web/Spark AP
I Structured Operations/Resources/20..., PartitionFilters: [], PushedFilters: [], ReadSc
hema: struct<DEST_COUNTRY_NAME:string,ORIGIN_COUNTRY_NAME:string,count:bigint>
```

Out[10]:
```
[('DEST_COUNTRY_NAME', 'string'),
 ('ORIGIN_COUNTRY_NAME', 'string'),
 ('count', 'bigint')]
```

In [11]:
```
# From the df table return the DES_COUNTRY_Name column(return 2 rows)
df.selectExpr("DEST_COUNTRY_NAME").show(2)
```

```
+-----------------+
|DEST_COUNTRY_NAME|
+-----------------+
|    United States|
|    United States|
+-----------------+
only showing top 2 rows
```

Adding,Renaming and Dropping Columns

- using as [column_name]
- using withColumn method
- using the withColumnRenamed method

In [12]:
```
# Add a new column withinCountry to our DataFrame that specifies whether the destinatio
df.selectExpr(
"*", # all original columns
"(DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) as withinCountry")\
.show(2)
```

```
+-----------------+-------------------+-----+-------------+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|withinCountry|
+-----------------+-------------------+-----+-------------+
|    United States|            Romania|   15|        false|
|    United States|            Croatia|    1|        false|
+-----------------+-------------------+-----+-------------+
only showing top 2 rows
```

In [13]:
```
# Add a column name [numberOne] with values of 1, us withColumn method
df.withColumn("numberOne", lit(1)).show(2)
```

```
+-----------------+-------------------+-----+---------+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|numberOne|
+-----------------+-------------------+-----+---------+
|    United States|            Romania|   15|        1|
|    United States|            Croatia|    1|        1|
+-----------------+-------------------+-----+---------+
only showing top 2 rows
```

In [14]:
```
# set a Boolean flag for when the origin country is the same as the destination country
df.withColumn("withinCountry", expr("ORIGIN_COUNTRY_NAME == DEST_COUNTRY_NAME"))\
.show(2)
```

```
+-----------------+-------------------+-----+-------------+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|withinCountry|
+-----------------+-------------------+-----+-------------+
|    United States|            Romania|   15|        false|
|    United States|            Croatia|    1|        false|
+-----------------+-------------------+-----+-------------+
only showing top 2 rows
```

In [15]:
```
# Rename the DEST_COUNTRY_NAME column to desc
df.withColumnRenamed("DEST_COUNTRY_NAME", "dest").columns
```

Out[15]:
```
['dest', 'ORIGIN_COUNTRY_NAME', 'count']
```

In [16]:
```
# Remove the ORIGIN_COUNTRY_NAME columns
df.drop("ORIGIN_COUNTRY_NAME").columns
```

Out[16]:
```
['DEST_COUNTRY_NAME', 'count']
```

Changing a Column's Type (cast)

In [17]:
```
# Python let's convert our count column from an integer to a type Long:
df.withColumn("count2", col("count").cast("long"))
```

Out[17]:
```
DataFrame[DEST_COUNTRY_NAME: string, ORIGIN_COUNTRY_NAME: string, count: bigint, count2:
bigint]
```

In [18]:
```
# SQL let's convert our count column from an integer to a type Long:
spark.sql("""
SELECT *, cast(count as long) AS count2 FROM dfTable
""").show(3)
```

```
+-----------------+-------------------+-----+------+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|count2|
+-----------------+-------------------+-----+------+
|    United States|            Romania|   15|    15|
|    United States|            Croatia|    1|     1|
|    United States|            Ireland|  344|   344|
+-----------------+-------------------+-----+------+
only showing top 3 rows
```

Filtering & Sorting Rows

In [19]:
```
# Python - filter the count column returning values < 2
# Origin_country_name is not equal to Crotia, show two rows
df.where(col("count") < 2).where(col("ORIGIN_COUNTRY_NAME") != "Croatia")\
.show(2)
```

```
+-----------------+-------------------+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----------------+-------------------+-----+
|    United States|          Singapore|    1|
|          Moldova|      United States|    1|
+-----------------+-------------------+-----+
only showing top 2 rows
```

In [20]:
```
# SQL - filter the count column returning values < 2
# Origin_country_name is not equal to Crotia, show two rows
spark.sql("""
SELECT *
FROM dfTable
WHERE count < 2
AND ORIGIN_COUNTRY_NAME != "Croatia"
LIMIT 2
""").show()
```

```
+-----------------+-------------------+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----------------+-------------------+-----+
|    United States|          Singapore|    1|
|          Moldova|      United States|    1|
+-----------------+------------------+-----+
```

In [21]:
```
# Return the number of Unique rows from ORIGIN_COUNTRY_NAME
df.select("ORIGIN_COUNTRY_NAME","DEST_COUNTRY_NAME").distinct().count()
```

Out[21]:
```
256
```

In [22]:
```
spark.sql("""
SELECT COUNT(DISTINCT(ORIGIN_COUNTRY_NAME, DEST_COUNTRY_NAME))
FROM dfTable
""").show()
```

```
+-------------------------------------------------------------------------------------------
---------------------+
|count(DISTINCT named_struct(ORIGIN_COUNTRY_NAME, ORIGIN_COUNTRY_NAME, DEST_COUNTRY_NAM
E, DEST_COUNTRY_NAME))|
+-------------------------------------------------------------------------------------------
---------------------+
|
256|
+-------------------------------------------------------------------------------------------
---------------------+
```

In [23]:
```
# Python - Return the 'count' column in 'desc order' and the DEST_COUNTRY_NAME column i
df.orderBy(col("count").desc(), col("DEST_COUNTRY_NAME").asc()).show(2)
```

```
+-----------------+------------------+------+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME| count|
+-----------------+------------------+------+
|    United States|      United States|370002|
|    United States|             Canada|  8483|
+-----------------+------------------+------+
only showing top 2 rows
```