UE Large Scale Processing @ ESIGELEC
2019/2020

14 - Spark SQL - Data Format/Sources and UDF

Abdel Dadouche
DJZ Consulting

adadouche@hotmail.com
@adadouche

Data Format / Sources

- Supported **data formats**:
  - CSV: comma separated values
  - JSON: JavaScript Object Notation
  - Avro: row-based self-describing binary storage data format with nested data
  - Parquet: columnar-based binary storage format with nested data
  - ORC (Optimized Row Columnar): columnar-based groups of rows in stripes
  - Table: access external RDBMS tables and view
  - and possibly more …

Data Format / Sources – Semi-Structured (JSON)

- SparkSQL can automatically infers a schema from a set of JSON records
- Schema inference algorithm works in one pass over the data
- Can run a sample of the data if desired

```
CREATE TABLE tweets USING json OPTIONS (path "my.json")
```

```
SELECT loc.lat , loc.long FROM tweets
WHERE text LIKE '%Spark%' AND tags IS NOT NULL
```

```
{
    "text ": "This is a tweet about #Spark",
    "tags ": ["# Spark "],
    "loc ": {" lat ": 45.1 , "long ": 90}
}{
    "text ": "This is another tweet",
    "tags ": [],
    "loc ": {" lat ": 39, "long ": 88.5}
}{
    "text ": "A #tweet without #location",
    "tags ": ["# tweet", "# location "]
}
```

Data Format / Sources

- Supported **data sources**:
  - Local & external files
  - HDFS
  - JDBC
  - RDD, DataSet or DataFrame
  - And many other connectors…

Check : Databricks Data Sources documentation

for more informations

- Most are common connectors provided as Spark packages:
  - spark-avro
  - spark-redshift
  - spark-csv
  - deep-spark
  - spark-mongdb
  - spark-cassandra-connector
  - couchbase-spark-connector
  - pyspark-csv
  - pyspark-elastic
  - elasticsearch-hadoop
  - and more…

Data Format / Sources - spark-packages.org

- Community-Managed
- References to:
  - third-party libraries
  - add-ons
  - Applications…

- Easy to use via Maven coordinates:

```
spark-submit \
  --packages <package id>:<version> \
  --conf key=value
```

- Or via JARs:

```
spark-submit \
  --jars "spark-csv_2.10-1.0.0.jar, commons-csv-1.1.jar" \
  --conf key=value
```

DataFrameReader – one class to read all!

- Package:
  - org.apache.spark.sql
- Class
  - DataFrameReader
- Use the “read” attribute/method from:
  - SQLContext
  - HiveContext
  - SparkSession

Note: You can also use the SparkContext to read data but only into a RDD

- Then you can use directly the corresponding input format method :
  - json()
  - csv()
  - parquet()
  - orc()
  - text()
- Or the format / load methods for more flexibility

Note: You can use DataStreamReader for streaming data

DataFrameReader – PySpark Examples

- Read a CSV file:

```
from os.path import expanduser
home = expanduser("~") + "/esigelec-ue-lsp-hdp/spark-3.0.0"
path = "file://" + home + "/examples/src/main/resources"

# read with the csv method
df = spark.read \
  .option("sep", ";") \
  .option("inferSchema", "true") \
  .option("header", "true") \
  .csv(path + "/people.csv")

# read with the format/load method
df = spark.read \
  .option("sep", ";") \
  .option("inferSchema", "true") \
  .option("header", "true") \
  .format("csv") \
  .load(path + "/people.csv")
```

- Read a PARQUET file:

```
from os.path import expanduser
home = expanduser("~") + "/esigelec-ue-lsp-hdp/spark-3.0.0"
path = "file://" + home + "/examples/src/main/resources"

# read with the parquet method
df = spark.read.parquet(path + "/users.parquet")

# read with the format/load method
df = spark.read.format("parquet").load(path + "/users.parquet")

# read with the load method
df = spark.read.load(path + "/users.parquet")
```

DataFrameWriter – one class to write all!

- Package:
  - org.apache.spark.sql
- Class
  - DataFrameWriter
- Use the “write” attribute/method from:
  - DataFrame
  - Dataset
- Then you can use directly the corresponding output format method :
  - json()
  - csv()
  - parquet()
  - orc()
  - text()
- Or the save / saveAsTable methods for more flexibility

DataFrameWriter – PySpark Examples

- Write a CSV file:

```
from os.path import expanduser
home = expanduser("~") + "/esigelec-ue-lsp-hdp/spark-3.0.0"
path = "file://" + home + "/examples/src/main/resources"

# write with the csv method
df.write \
  .option("sep", ";") \
  .option("inferSchema", "true") \
  .option("header", "true") \
  .csv(path + "/df-csv-folder")

# write with the format/save method
df. write \
  .option("sep", ";") \
  .option("inferSchema", "true") \
  .option("header", "true") \
  .format("csv") \
  .save(path + "/df-csv-save-folder")
```

- Write a PARQUET file:

```
from os.path import expanduser
home = expanduser("~") + "/esigelec-ue-lsp-hdp/spark-3.0.0"
path = "file://" + home + "/examples/src/main/resources"

# write with the parquet method
df.write.parquet(path + "/df-parquet-folder")

# read with the format/load method
df.write.format("parquet").save(path + "/df-parquet-save-folder")

# write with the load method
df.write.save(path + "/df-parquet-save-direct-folder")
```

SparkSQL Caching

- SparkSQL can materialize (cache) data in memory (columnar storage)
- Columnar cache reduce the memory
- In interactive mode, iterative algorithms will benefit from caching
- Caching can be invoked using:
  - cache() API on DataFrame/Dataset
  - cacheTable() API on tables
  - CACHE TABLE sql statement on tables

SparkSQL UDF (User Defined Function)

- Supports existing Apache Hive UDFs
- SparkSQL provides a series of native UDFs (pyspark.sql.functions)
- Can be developed in :
  - Java
  - Scala
  - Python
- **Note**: SparkSQL will not try to optimize the code

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

from os.path import expanduser
home = expanduser("~") + "/esigelec-ue-lsp-hdp/spark-3.0.0“
path = "file://" + home + "/examples/src/main/resources“

sparkSession = SparkSession.builder \
    .master("spark://localhost:7077") \
    .appName("SparkSQL App") \
    .getOrCreate()

def squaredPy(s): return s * s

sparkSession.udf.register("squaredPy", squaredPy, LongType())

sparkSession.range(1, 20).createOrReplaceTempView("test")

sparkSession.sql("select id, squaredPy(id) from test").show()
```