Lab 4.2: Spark SQL Basics

Overview

In this lab, we will use some of the basic functionality of Spark SQL, including: Read and

- write data files in varying formats
- Create DataFrames

Builds on

Lab 4.1

Run time

15-20 minutes

Lab Preparation

Tasks

- Load data from the following files, if you haven't already
  - *people.json* as JSON
  - *wiki-pageviews.txt* as text
  - *github.json* as JSON

```
// Scala
> val folksDF = // Load people.json here
> val viewsDF = // Load wiki-pagesviews.txt here
> val githubDF = // Load github.txt here
```

```
# Python
> folksDF = // Load people.json here
> viewsDF = // Load wiki-pagesviews.txt here
> githubDF = // Load github.txt here
```

View the Schema

Tasks

- Examine the schema of the resulting dataframes, e.g.

```
> folksDF.printSchema()
// Similarly for others
```

- The githubDF schema is quite complex. We'll look at some ways of dealing with it soon.

Declare a Schema Explicitly

Tasks

- Declare a schema for folksDF. We
  - illustrate this below.
  - We also supply this code in a helper file - *peopleSchema.txt*:
    - **Scala:** All helpers in *spark-labs/helpers/*
    - **Python:** All helpers in *spark-labs/helpersPython/*
  - You can cat this file in Zeppelin, copy it, then paste it back into Zeppelin to execute.
    - You'll use hdfs commands to cat the file, as shown below. You
    - can then paste it back into your Zeppelin spark shell

```
%sh
# Scala
hdfs dfs -cat spark-labs/helpers/peopleSchema.txt

import org.apache.spark.sql.types._
val mySchema = (new StructType).add("name", StringType).add("gender",
StringType).add("age", IntegerType)
```

```
%sh
# Python
hdfs dfs -cat spark-labs/helpersPython/peopleSchema.txt

from pyspark.sql.types import *
mySchema = StructType().add("name", StringType()).add("gender",
StringType()).add("age", IntegerType())
```

- Look at the Web UI Jobs tab and note the last job shown.
- Read *spark-labs/data/people.json* again, but this time supply the schema you declared.

- Look at the Web UI Jobs tab again.
- Do you see an additional job? You shouldn't - if Spark is not inferring the schema, it doesn't need to scan the data ahead of time.

Work with More Complex Data

Tasks

- Read *spark-labs/data/people-with-address.json*, as shown below.

```
// Scala
> var folksAddressDF = spark.read.json("spark-labs/data/people-with-
address.json")
```

```
# Python
> folksAddressDF = spark.read.json("spark-labs/data/people-with-
address.json")
```

- View the schema (use `printSchema`).
  - Note the nested structure for the address.
- Declare a schema for this DataFrame, and then read it again using the schema you've declared.
  - To add a nested schema structure for the address, first create the schema for the address separately, then add it in as a field to the DataFrame schema.

```
// Scala
> val addressSchema = ...
> val schemaWithAddress = (new StructType).add("address",
addressSchema)...
```

```
# Python
> addressSchema = ...
> schemaWithAddress = StructType().add("address", addressSchema)...
```

Summary

The DataFrame schema is core to the Spark SQL capabilities. We've seen how schema inference can be a great help, but does have some drawbacks. For instance, it can result in an extra data scan. The schema can also be very complex with complex data.

We will look at more techniques to work with the data in later labs.