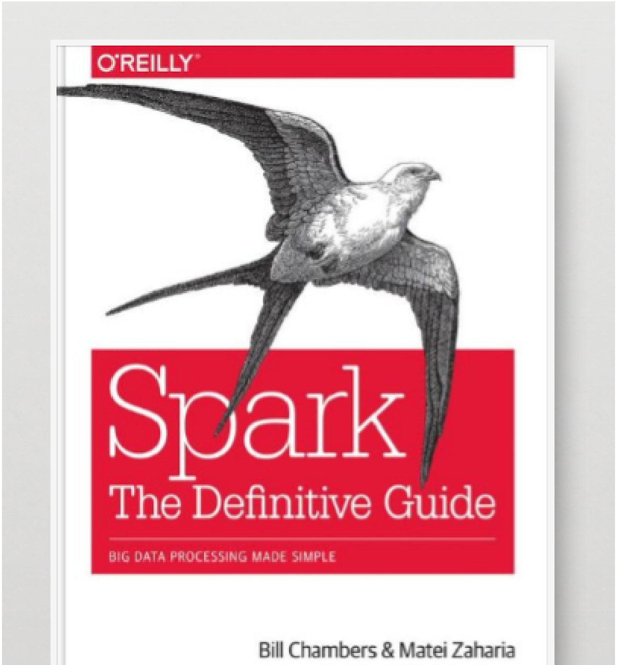
Spark the Definitive Guide 2nd Edition

Chapter 05

Basic Structured Operations

Basic Structured Operations

Text Book


O'REILLY
Spark
The Definitive Guide
BIG DATA PROCESSING MADE SIMPLE
Bill Chambers & Matei Zaharia

Objectives and Outcomes

- Introduce the tools we will use to manipulate DataFrames
- Focus on fundamental DataFrame operations

Review

So far:

- We were introduced to Spark's Structured APIs, Datasets, DataFrames, and SQL Views
- We learned how Spark transforms a logical plan into a physical execution plan on a cluster
- Learned how DataFrames consist of a series of records
- Learned how DataFrames are of type `Row` and have a number of columns
- Learned that schemas define the name and type of data in each column
- Learned that `Partitioning` of the DataFrame defines the layout of the DataFrames physical distribution on the cluster

Create a DataFrame

```
df = spark.read.format("json").load("Spark-The-Definitive-(

# Let's tale a look at the defined schema
df.printSchema()
```

- JSON
  - is a lightweight, text-based data interchange format.

Schemas

- Schemas tie everything together
- Schema defines the column names and column types of a DataFrame
  - Schema can be applied on read or inferred or declared
- For Ad-hoc data usually *schema-on-read* is good enough
  - Though it can be a bit slow when dealing with text-based file formats like:
    - CSV
    - JSON
- Schema-on-read can lead to precision problems
  - If a column is really of type `LONG` but the numbers are smaller and interpreted as type `INT`
- Spark can be used for **ETL**:
  - Extraction
  - Transform
  - Load In these cases it is best to provide the schema to ensure type matches

JSON Object

```
spark.read.format("json").load("Spark-The-Definitive-Guide/

# This datatype is returned from the previous command
# StructType(List(StructField
# (DEST_COUNTRY_NAME,StringType,true),
# StructField
# (ORIGIN_COUNTRY_NAME,StringType,true),
# StructField(count,LongType,true)))
```

- A schema is a `StructType` made up of a number of fields
  - `StructFields` have a name, type, and b a Boolean flag indicating if they take `nulls`
  - If types in the data at run-time do not match the schema, Spark will thrown and error

Declare a Schema

```
from pyspark.sql.types import StructField, StructType, Str

myManualSchema = StructType([StructField("DEST_COUNTRY_NAM
StructField("ORIGIN_COUTNRY_NAME",StringType(),True),Struc

df = spark.read.format("json").schema(myManualSchema)
.load("Spark-The-Definitive-Guide/data/flight-data/json/20
```

Columns and Expressions

- Columns can be selected, manipulated, and removed from DataFrames
  - These operations are referred to as *expressions*
  - Must use Spark to manipulate Rows (logical collection of Rows is a column)
  - Must be in the context of a DataFrame
  - To work on columns use the *col* or *column* functions
  - We will stick to using the *col* function
  - Columns are not resolved until compared to the *catalog* at run-time
  - Column and table resolution happen in the *analyzer* phase

```
from pyspark.sql.functions import col, column

col("someColumnName")
column("someColumnName")
```

Column Reference

- If you need to explicitly reference a column you can
- Think of it as a namespace way to reference columns in different DataFrames that have the same name
  - `df.col("count")`

Columns as Expressions

- What is an *expression*?
  - A set of *transformations* on one or more values in a *record* in a DataFrame
- You can use a `col()` and perform a transformation on a column
- You can use an `expr()` to parse transformations and column references
  - These references can subsequently be passed into further transformations
  - `expr("someCol - 5")` and `col("someCol") - 5` and `expr("someCol") - 5` all evaluate the same
  - Spark compiles these to the same logic tree
- Columns are just expressions
- Columns and transformations of those columns compile to the same logical plan `(((col("someCol") + 5 ) * 200 ) - 6 ) < col("otherCol")`
- This is also represented by in Python (64):
  - `python from pyspark.sql.functions import expr expr("(((someCol + 5) * 200) -6) < "otherCol")`

Conclusion

- Conclusion here

Questions

- Any questions?
- Read Chapter 06 and do any exercises in the book.