

DATAFRAMES

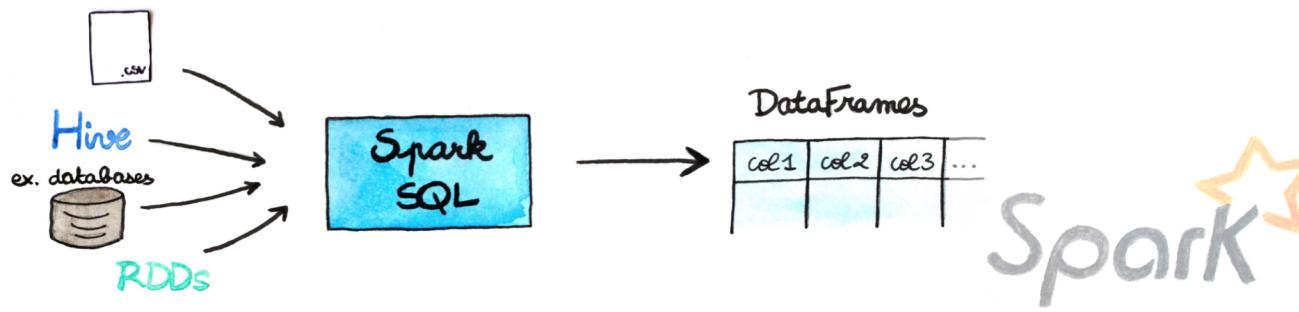


Spark SQL is a component on top of **Spark Core** that facilitates processing of structured and semi-structured data and the integration of several data formats as source (Hive, Parquet, JSON).

<https://spark.apache.org/docs/latest/sql-programming-guide.html>

DataFrames

- DataFrame is an immutable distributed collection of data.
- Unlike an RDD, **data is organized into named columns**, like a table in a relational database or a dataframe in R/Python
- Distributed collection of data grouped into named columns:
 - DataFrames == RDD + Schema (illustration of the structure of data)
- Designed to make large data sets processing even easier.
- Allows developers to impose a structure onto a distributed collection of data, allowing higher-level abstraction



Why DataFrames ?

- DataFrame is one step ahead of **RDD**
 - **Custom Memory Management:**
 - Memory is saved - data is stored in off-heap memory in binary
 - No Garbage Collection overhead
 - Expensive Java serialization is also avoided.
 - data is stored in binary format and the schema of memory is known
 - **Optimized Execution plan:** Once the optimized plan is created final execution takes place on RDDs of Spark

The structured spectrum

Structured

- Relational Databases
- Parquet
- Formatted Messages

Semi-structured

- HTML
- XML
- JSON

Unstructured

- Plain text
- Generic media

RDD vs DataFrames

DataFrames are composed of Row objects, along with a schema that describes the data types of each column in the row.

Person
Person
Person

Person
Person
Person

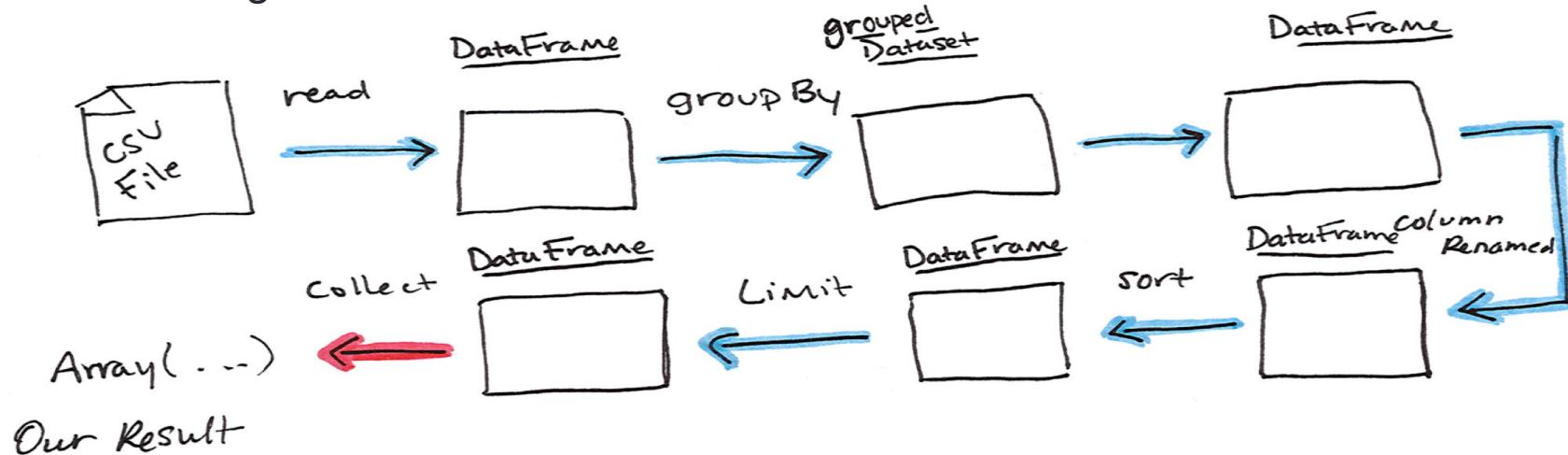
RDD[Person]

Name	Age	Height
String	Int	Double

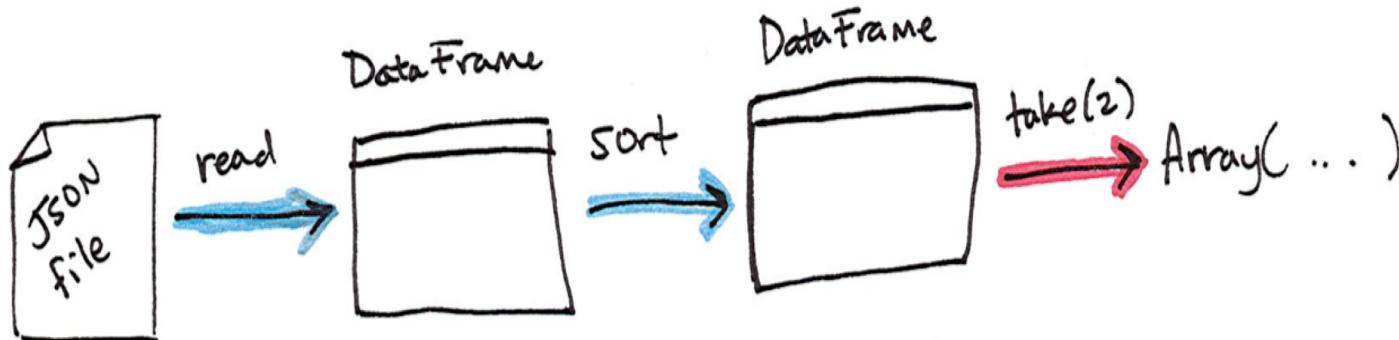
DataFrame

DataFrames and CSV/JSON Files

Working with CSV files



Working with Json files



<https://spark.apache.org/docs/latest/sql-data-sources.html>

DataFrame – read/write formats

- `sqlContext.read.[format]`

```
>> sqlContext.read.parquet(path)
>> sqlContext.read.json(path, [schema])
>> sqlContext.read.jdbc(url, table)
>> sqlContext.read.load(path, [format])
```

- `sqlContext.write.[format]`

```
>> sqlContext.write.parquet(path)
>> sqlContext.write.json(path, [mode])
>> sqlContext.write.jdbc(url, table, [mode])
>> sqlContext.write.save(path, [format], [mode])
```

DataFrames and RDDs

- Create from RDD of tuples

```
>> rdd = sc.parallelize([('a', 1), ('b', 2), ('c', 3)])  
>> df = sqlContext.createDataFrame(rdd, ["name", "id"])  
>> df.show()
```

+-----+-----+		
	name	id
+-----+-----+		
	a	1
	b	2
	c	3
+-----+-----+		

DataFrames and RDDs

- Create from RDD of Rows

```
>> from pyspark.sql import Row  
  
>> ExampleRow = Row("name", "id")  
  
>> rdd = sc.parallelize([ ExampleRow("a", 1), ExampleRow("b", 2),  
                         ExampleRow("c", 3) ])  
  
>> df = sqlContext.createDataFrame(rdd)  
  
>> df.show()  
      >> df.printSchema()  
  
+---+---+  
|name|id |  
+---+---+  
|  a| 1|  
|  b| 2|  
|  c| 3|  
+---+---+  
  
root  
| -- name: string (nullable = false)  
| -- id: integer (nullable = false)
```

Examples of SparkSQL (1)

```
from pyspark import SparkContext  
  
from pyspark.sql import SQLContext  
  
sc = SparkContext('local', 'Spark SQL')  
  
sqlc = SQLContext(sc)  
  
#We can read a JSON file and create a DataFrame (Spark SQL json reader)  
  
players = sqlc.read.json('players.json') # Print the schema in a tree format  
  
players.printSchema()  
  
#Select only the "FullName" column  
  
players.select("FullName").show(4)  
  
+-----+  
| FullName |  
+-----+  
| ÑAngel Bossio |  
| Juan Botasso |  
| Roberto Cherro |  
| Alberto Chividini |  
+-----+
```

Examples of SparkSQL (1)

```
#Then we can create a view of our DataFrame. The lifetime of this temporary table  
is tied to the SparkSession that was used to create this DataFrame.
```

```
players.registerTempTable("players")
```

```
#We can then query our view; for instance to get the names of all the Teams
```

```
sqlc.sql("select distinct Team from players").show(5)
```

```
+-----+
```

```
| Team |
```

```
+-----+
```

```
|England |
```

```
|Paraguay|
```

```
| POL |
```

```
| Russia |
```

```
| BRA |
```

```
+-----+
```

Examples of SparkSQL (2)

```
# spark is an existing SparkSession
df = spark.read.json("examples/people.json")
# Displays the content of the DataFrame to stdout
df.show()
# +---+-----+
# | age| name|
# +---+-----+
# | null|Michael|
# | 30| Andy|
# | 19| Justin|
# +---+-----+
```

```
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +-----+-----+
# |    name|(age + 1)|
# +-----+-----+
# | Michael|      null|
# | Andy|      31|
# | Justin|      20|
# +-----+-----+
```

Find full example code at "examples/people.py"

```
# Select people older than 21
df.filter(df['age'] > 21).show()
# +---+
# | age|name|
# +---+
# | 30|Andy|
# +---+
```

```
# Count people by age
df.groupBy("age").count().show()
# +---+-----+
# | age|count|
# +---+-----+
# | 19|     1|
# | null|     1|
# | 30|     1|
# +---+-----+
```

Pandas DataFrame

- When in PySpark, there is also an easy option to convert Spark DataFrame to Pandas dataframe.

```
# Convert Spark DataFrame to Pandas
pandas_df = spark_df.toPandas()
```

```
# Create a Spark DataFrame from Pandas
spark_df = context.createDataFrame(pandas_df)
```

- One powerful and easy way to visualize data is
dataframe.toPandas().plot()

Additional Slides – Parsing file formats to RDD

Parsing JSON data to RDD

```
import json

#parse multi-line json file
parsed = sc.textFile("data.json").map(lambda x:
json.loads(x))

#parse directory of json documents
parsed = sc.wholeTextFiles("data/").map(lambda x:
json.loads(x))

#write out json data
data.map(lambda x:json.dumps(x)).saveAsTextFile(output)
```

Parsing CSV data to RDD - Example

```
import csv

from StringIO import StringIO

# Read from CSV

def load_csv(contents):

    return csv.reader(StringIO(contents[1]))

data = sc.wholeTextFile("data/").flatMap(load_csv)

# Write to CSV

def write_csv(records):

    output = StringIO()

    writer = csv.writer()

    for record in records:

        writer.writerow(record)

    return [output.getvalue()]

data.mapPartitions(write_csv).saveAsTextFile("output/")
```

Parsing Structured Objects to RDD

```
import csv

from datetime import datetime
from StringIO import StringIO
from collections import namedtuple

DATE_FMT = "%Y-%m-%d %H:%M:%S" # 2013-09-16 12:23:33
Customer = namedtuple('Customer', ('id', 'name', 'registered'))

def parse(row):
    row[0] = int(row[0]) # Parse ID to an integer
    row[4] = datetime.strptime(row[4], DATE_FMT)
    return Customer(*row)

def split(line):
    reader = csv.reader(StringIO(line))
    return reader.next()

customers = sc.textFile("customers.csv").map(split).map(parse)
```