

# Spark – Big Data Processing

## Aula 5



**Semantix<sup>®</sup>**

All about data

# Quem sou eu?



## Rodrigo Augusto Rebouças

Engenheiro de dados da Semantix  
Instrutor do Semantix Academy

### Contatos

[rodrigo.augusto@semantix.com.br](mailto:rodrigo.augusto@semantix.com.br)  
[linkedin.com/in/rodrigo-reboucas](https://www.linkedin.com/in/rodrigo-reboucas)



# Opções de Leitura em CSV

- DataSet e DataFrames

# Leitura de Arquivo CSV

```
data = spark.read.  
    option("sep", "|").  
    option("header", "true").  
    option("quote", "\"").  
    option("mode", "DROPMALFORMED").  
    csv("hdfs:///user/teste/")
```

# Comandos com WithColumn

- DataSet e DataFrames

# WithColumn

- WithColumn
  - Criar Colunas
  - Sintaxe:

`<dataframe>.withColumn("<nomeColuna>", <Coluna>)`

```
data = spark.read.  
    option("sep", "|").  
    option("header", "true").  
    option("quote", "\"").  
    option("mode", "DROPMALFORMED").  
    csv("hdfs:///user/teste/")
```

```
addColumn = data.withColumn("Novo Campo", col("id"))
```

# Acesso a coluna de DataFrame/DataSet

## Scala

- `col("campo")`
- `dataframe("campo")`
- `$"campo"`

## Python

- `from pyspark.sql.functions import col`
- `col("campo")`
- `dataframe["campo"]`
- `dataframe.campo`

```
dataframe.select(*cols)
```

```
dataframe.withColumn(colName, col)
```

# Comandos com WithColumn

- **Funções**
  - Timestamp
  - Substring
  - Split



# WithColumn – Trabalhando com Timestamp

- Converter Coluna para timestamp
  - `unix_timestamp(col("<ColString>"), "<FormatoAtual>"),`
- Alterar formato de Coluna de timestamp
  - `from_unixtime(<ColTimestamp>, "<FormatoConversão>"))`

```
from pyspark.sql.functions import unix_timestamp, from_unixtime
formato = data.select("data").show(1) // formato = 2020/10/25
convUnix = formato.withColumn("timestamp", unix_timestamp(col("data"), "yyyy/MM/dd"))
convUnixData = convUnix.withColumn("new data", from_unixtime("timestamp", "MM-dd-yyyy"))

convDataDireto = formato.withColumn("mes-dia-ano",
from_unixtime(unix_timestamp(col("data"), "yyyy/MM/dd"), "MM-dd-yyyy"))
```

# WithColumn – Trabalhando com Substring

- Extrair dados de uma coluna de acordo com uma posição
- Sintaxe:

`<dataframe>.withColumn("<nomeColuna>", substring("<Coluna>", <posicaoInicial>, <tamanho>))`

- Muito usado com concat para concatenar as colunas e strings

```
formato = data.select("data").show(1) // data = 2020/10/25
```

```
mes= formato.withColumn("mes" ,substring(col("data"),6,2)) // mes = 10
```

```
codsDF = data.select("codigo") // código = AA150000CCS
```

```
resumoCod = codsDF.withColumn("pedido" , concat(substring(col("codigo"), 1, 2),lit("-"),  
substring(col("codigo"), 9, 3)) // pedido = AA-CCS
```

# WithColumn – Trabalhando com Split

- Split

- Criar um array de acordo com um delimitar
- Sintaxe:

```
<dataframe>.withColumn("<nomeColuna>", split("<Coluna>", "<delimitador" ))
```

```
nomesDF = data.select("nome").show(1) // nome = Rodrigo Augusto Rebouças
```

```
sepNomesDF = nomesDF.withColumn("sepNome", split(col("nome"), " "))
```

```
sepNomesDF.printSchema()
```

```
|-- nome: string (nullable = true)
```

```
|-- sepNome: array (nullable = true)
```

```
valpNome = sepNomesDF.withColumn("pNome", col("sepNome").getItem(0)).drop("sepNome")
```

## Exercícios - WithColumn

1. Criar um dataframe para ler o arquivo no HDFS /user/<nome/data/juros\_selic/juros\_selic
2. Alterar o formato do campo data para “MM/dd/yy”
3. Com uso da função from\_unixtime crie o campo “ano\_unix”, com a informação do ano do campo data
4. Com uso de substring crie o campo “ano\_str”, com a informação do ano do campo data
5. Com uso da função split crie o campo “ano\_str”, com a informação do ano do campo data
6. Salvar no hdfs /user/rodrigo/juros\_selic\_americano no formato CSV, incluindo o cabeçalho.

# Comandos com WithColumn

- **Funções**

- cast
  - regexp\_replace
  - when

# WithColumn – Trabalhando com Cast

- Cast: Alterar o tipo do dado
  - Sintaxe:
  - `<dataframe>.withColumn("<nomeColuna>", <coluna>.cast("Tipo" ))`
- Alterar as casas decimais  
`format_number("<coluna>", <numeroCasasDecimais>))`

```
medida = data.select("total").show(1) // total = 1000.00 (String)
```

```
from pyspark.sql.types import *
```

```
converter = medidaR.withColumn("Total real", col("total").cast(FloatType()))
```

```
converter2c = converter.withColumn("Total real", format_number(  
col("Total real").cast(FloatType()),2)
```

# WithColumn – Trabalhando com Cast e regexp\_replace

- Regexp
  - Alterar um padrão com uso de regex  
`regexp_replace("<coluna>", "<padrão_atual>", "<novo_padrão>")`
- Usado para fazer cast de decimais para substituir , por .

```
medida = data.select("total").show(1) // total = 1.000,00 (String)
medidaR = medida.withColumn("total", regexp_replace(col("total"), "\.", ""))
// total = 1000,00 (String)
medidaR = medidaR.withColumn("total", regexp_replace(col("total"), "\,", "."))
// total = 1000.00 (String)
from pyspark.sql.types import *
converter = medidaR.withColumn("Total real", col("total").cast(FloatType() ))
// total = 1000.00 (Float)
```

# WithColumn – Trabalhando com when

- When
  - Responsável por fazer condicional em colunas
  - Sintaxe:

```
<dataframe>.withColumn("<nomeColuna>", when(<condição>,  
<valorVerdadeiro>).otherwise(<valorFalso>))
```

```
codigos = data.select("cod").take(5)
```

```
// cod= { AABB, ACBB, 00ABCC, AACC, 00BBCC}
```

```
remover_zeros = codigos.withColumn("cod_sem_0", when(length(col("cod")) > 4,  
substring(col("cod"), 3,6)).otherwise(col("cod")))
```

```
// cod_sem_0 = { AABB, ACBB, ABCC, AACC, BBCC}
```



# Agregações



# Dataframe – Agregações

`<dataframe>.groupBy(<coluna>).agg(<f_agg>)`

- count
- avg
- sum
- min
- max
- first
- last
- countDistinct
- approx\_count\_distinct
- stddev
- var\_sample
- var\_pop
- covar\_samp
- covar\_pop
- corr

```
peopleDF.groupBy("setor").sum("gastos").sort(desc("gastos"))
```

```
peopleDF.groupBy("setor").agg(avg("gastos"),sum("gastos").alias("total_gastos"))
```

# Exercícios

1. Criar um dataframe para ler o arquivo no HDFS /user/<nome>/data/juros\_selic/juros\_selic
2. Agrupar todas as datas pelo ano em ordem decrescente e salvar a quantidade de meses ocorridos, o valor médio, mínimo e máximo do campo valor com a seguinte estrutura:

Anual	Meses	Valor médio	Valor Mínimo	Valor Máximo
2019	2	00.00	00.00	00.00
2018	12	00.00	00.00	00.00
2017	12	00.00	00.00	00.00
...	...	00.00	00.00	00.00
1986	2	00.00	00.00	00.00

3. Salvar no hdfs:///user/<nome>/relatorioAnual com compressão zlib e formato orc



# Semantix<sup>®</sup>

All about data

[contato@semantix.com.br](mailto:contato@semantix.com.br)

[www.semantix.com.br](http://www.semantix.com.br)