An Insufficient Introduction to Spark

Part 3: DataFrames

Riccardo Murri <riccardo.murri@gmail.com>

2019-02-11

User-defined functions

Column expressions

Like in SQL, wherever a column name is wanted, a column expression may be used instead.

A column expression is made of:

- column references, e.g., `df['colname']`
- constants, logical/relation operators `==`, `<=`, etc. and algebraic operators `+`, `-`, `*`, `/` (for numeric-valued columns only)
- One of the many column functions provided in the pyspark.sql.functions module

Examples of column functions (1)

A sample from the pyspark.sql.functions module:

cos(), sin(), log(), . . .
Numerical functions.

concat(*s1, s2,* . . .), locate(*str, substr*),
lower(*col*), lpad(*col, len, pad*), . . .
String functions.

year(), month(), day(), hour(). minute(), second(),
dayofmonth(), dayofyear(), weekofyear()
Extract the corresponding part from a date or timestamp value.

Examples of column functions (2)

A sample from the pyspark.sql.functions module:

asc(), desc()
Sort value in the column in ascending/descending order.

column(*name*)
Values from the named column.

explode(*name*)
When the named column is a composite-type one (e.g. values are lists), return one value at a time.

User-defined functions

It is possible to make user code into a column function by passing it to the `udf` function:

```
def classify(x):
  return ['a', 'b', 'c', 'd'][x % 4]

# make it into a column function
classify_col = udf(classify)

df2 = df.withColumn(df, classify_col.alias('class'))
```

Note that the return type of a UDF defaults to string!

Window Functions

Window functions

"Built-in functions or UDFs, [...] take values from a single row as input, and they generate a single return value for every input row. Aggregate functions, [...] operate on a group of rows and calculate a single return value for every group. [...]

At its core, ***a window function calculates a return value for every input row of a table based on a group of rows****, called the Frame."*

Reference: https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html

Examples of Window functions

rank(), percentRank()
Rank (resp. percentile) of the row within the current frame.

lead(*col*, *offset*, *default*), lag(*col*, *offset*, *default*)
Return the row that comes `offset` before (`lead`) or after (`lag`) the current row. If the frame does not extend that far, return the `default` value.

Any aggregate function can be used as a window function.

Reference: https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html

Using window functions

In order to use a window function one must "instanciate" it with a *window specification* in the `.over()` method:

```
wf = rank().over(wspec)
```

A window specification defines which rows are included in the frame associated with a given input row.

A window specification includes three parts:

1. partitioning specification
2. ordering specification
3. frame specification

Partitioning specification

Partitioning Specification: controls which rows will be in the same partition with the given row.

This is done by calling the '`Window.partitionBy()`' method with an expression whose values will be used in the ordering and framing steps.

Example:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import day

wspec = Window.partitionBy(day('date'))
```

Ordering specification

Ordering Specification: controls the way that rows in a partition are ordered, determining the position of the given row in its partition.

This is done by calling the '`Window.orderBy()`' method with either `pyspark.sql.functions.asc()` or '`desc()`' as argument.

Example:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import desc

wspec = wspec.orderBy(desc('count'))
```

Note that the ordering expression and the partitioning expression can (and often will!) be different.

Frame specification

Frame Specification: states which rows will be included in the frame for the current input row, based on their relative position to the current row.

rowsBetween(*before*, *after*)
Include the *before* rows preceding the current one and the *after* rows following it in the frame.

rangeBetween(*start*, *end*)
Include in the frame all rows on which the ordering expression takes a value in between *start* and *end*.