What are the operators and functions available in the PySpark Column Class?

Authored by

stats writer

June 24, 2024

The PySpark Column Class is a powerful tool for manipulating and transforming data in PySpark. It offers a wide range of operators and functions that allow users to perform various operations on the data, such as filtering, sorting, and aggregating. Some of the commonly used operators in the Column Class include logical, arithmetic, and comparison operators, which enables users to perform logical and mathematical operations on the data. Additionally, the Column Class also provides various functions, such as string manipulation, date and time manipulation, and mathematical functions, to further enhance data transformation capabilities. Overall, the operators and functions available in the PySpark Column Class provide a comprehensive set of tools for data manipulation and transformation in PySpark.

`pyspark.sql.Column` class provides several functions to work with DataFrame to manipulate the Column values, evaluate the boolean expression to filter rows, retrieve a value or part of a value from a DataFrame column, and to work with list, map & struct columns.

In this article, I will cover how to create Column object, access them to perform operations, and finally most used PySpark Column Functions with Examples.

Related Article: PySpark Row Class with Examples

Key Points:

Note: Most of the pyspark.sql.functions return Column type hence it is very important to know the operation you can perform with Column type.

1. Create Column Class Object

One of the simplest ways to create a Column class object is by using PySpark lit() SQL function, this takes a literal value and returns a Column object.

```
from pyspark.sql.functions import lit
colObj = lit("arabpsychology.com")
```

You can also access the Column from DataFrame by multiple ways.

```
data=
df=spark.createDataFrame(data).toDF("name.fname","gender")
df.printSchema()
#root
# |-- name.fname: string (nullable = true)
# |-- gender: long (nullable = true)
```

```
# Using DataFrame object (df)
df.select(df.gender).show()
df.select(df).show()
#Accessing column name with dot (with backticks)
df.select(df).show()

#Using SQL col() function
from pyspark.sql.functions import col
df.select(col("gender")).show()
#Accessing column name with dot (with backticks)
df.select(col("`name.fname`")).show()
```

Below example demonstrates accessing struct type columns. Here I have use PySpark Row class to create a struct type. Alternatively you can also create it by using PySpark StructType & StructField classes

```
#Create DataFrame with struct using Row class
from pyspark.sql import Row
data=
df=spark.createDataFrame(data)
df.printSchema()
#root
# |-- name: string (nullable = true)
# |-- prop: struct (nullable = true)
# | |-- hair: string (nullable = true)
# | |-- eye: string (nullable = true)

#Access struct column
df.select(df.prop.hair).show()
df.select(df).show()
df.select(col("prop.hair")).show()

#Access all columns from struct
df.select(col("prop.*")).show()
```

2. PySpark Column Operators

PySpark column also provides a way to do arithmetic operations on columns using operators.

```
data=
df=spark.createDataFrame(data).toDF("col1","col2","col3")
```

#Arthmetic operations
df.select(df.col1 + df.col2).show()
df.select(df.col1 - df.col2).show()
df.select(df.col1 * df.col2).show()
df.select(df.col1 / df.col2).show()
df.select(df.col1 % df.col2).show()

df.select(df.col2 > df.col3).show()
df.select(df.col2 < df.col3).show()
df.select(df.col2 == df.col3).show()

3. PySpark Column Functions

Let's see some of the most used Column Functions, on below table, I have grouped related functions together to make it easy, click on the link for examples.

Column Function	Function Description
`alias`(*alias, **kwargs) `name`(*alias, **kwargs)	Provides alias to the column or expressions `name()` returns same as `alias()`.
`asc`() `asc_nulls_first`() `asc_nulls_last`()	Returns ascending order of the column. `asc_nulls_first`() Returns null values first then non-null values. `asc_nulls_last`() - Returns null values after non-null values.
`astype`(dataType) `cast`(dataType)	Used to cast the data type to another type. `astype()` returns same as `cast()`.
`between`(lowerBound, upperBound)	Checks if the columns values are between lower and upper bound. Returns boolean value.
`bitwiseAND`(other) `bitwiseOR`(other) `bitwiseXOR`(other)	Compute bitwise AND, OR & XOR of this expression with another expression respectively.
`contains`(other)	Check if String contains in another string.
`desc`() `desc_nulls_first`() `desc_nulls_last`()	Returns descending order of the column. `desc_nulls_first`() -null values appear before non-null values. `desc_nulls_last`() - null values appear after non-null values.
`startswith`(other) `endswith`(other)	String starts with. Returns boolean expression String ends with. Returns boolean expression

Column Function	Function Description
`eqNullSafe`(other)	Equality test that is safe for null values.
`getField`(name)	Returns a field by name in a StructField and by key in Map.
`getItem`(key)	Returns a values from Map/Key at the provided position.
`isNotNull`() `isNull`()	isNotNull() - Returns True if the current expression is NOT null. isNull() - Returns True if the current expression is null.
`isin`(*cols)	A boolean expression that is evaluated to true if the value of this expression is contained by the evaluated values of the arguments.
`like`(other) `rlike`(other)	Similar to SQL like expression. Similar to SQL RLIKE expression (LIKE with Regex).
`over`(window)	Used with window column
`substr`(startPos, length)	Return a Column which is a substring of the column.
`when`(condition, value) `otherwise`(value)	Similar to SQL CASE WHEN, Executes a list of conditions and returns one of multiple possible result expressions.
`dropFields`(*fieldNames)	Used to drops fields in StructType by name.
`withField`(fieldName, col)	An expression that adds/replaces a field in `StructType` by name.

4. PySpark Column Functions Examples

Let's create a simple DataFrame to work with PySpark SQL Column examples. For most of the examples below, I will be referring DataFrame object name (df.) to get the column.

```
data=
columns=
df=spark.createDataFrame(data,columns)
```

4.1 alias() - Set's name to Column

On below example `df.fname` refers to Column object and `alias()` is a function of the Column to give alternate name. Here, `fname` column has been changed to `first_name` & `lname` to `last_name`.

In the second example I have use PySpark expr() function to concatenate columns and named column as `fullName`.

```
#alias
from pyspark.sql.functions import expr
df.select(df.fname.alias("first_name"),
df.lname.alias("last_name")
).show()
```

#Another example
df.select(expr(" fname ||','|| lname").alias("fullName")
).show()

4.2 asc() & desc() - Sort the DataFrame columns by Ascending or Descending order.

```
#asc, desc to sort ascending and descending order repsectively.
df.sort(df.fname.asc()).show()
df.sort(df.fname.desc()).show()
```

4.3 cast() & astype() - Used to convert the data Type.

```
#cast
df.select(df.fname,df.id.cast("int")).printSchema()
```

4.4 between() - Returns a Boolean expression when a column values in between lower and upper bound.

```
#between
df.filter(df.id.between(100,300)).show()
```

4.5 contains() -

Check if a PySpark DataFrame column value contains a string value specified in this function.

```
#contains
df.filter(df.fname.contains("Cruise")).show()
```

4.6 startswith() & endswith() -

Checks if the value of the DataFrame Column startsWith() and endsWith() a String. startsWith() filters rows where a specified substring exists at the beginning while `endsWith()` filter rows where the specified substring presents at the end.

```
#startswith, endswith()
df.filter(df.fname.startswith("T")).show()
df.filter(df.fname.endswith("Cruise")).show()
```

4.7 eqNullSafe() -

4.8 isNull & isNotNull() - Checks if the DataFrame column has NULL or non NULL values.

Refer to

```
#isNull & isNotNull
df.filter(df.lname.isNull()).show()
df.filter(df.lname.isNotNull()).show()
```

4.9 like() & rlike() - Similar to SQL LIKE expression

```
#like , rlike
df.select(df.fname,df.lname,df.id)
.filter(df.fname.like("%om"))
```

4.10 substr() - Returns a Column after getting sub string from the Column

```
df.select(df.fname.substr(1,2).alias("substr")).show()
```

4.11 when() & otherwise() - It is similar to SQL Case When, executes sequence of expressions until it matches the condition and returns a value when match.

```
#when & otherwise
from pyspark.sql.functions import when
df.select(df.fname,df.lname,when(df.gender=="M","Male")
.when(df.gender=="F","Female")
.when(df.gender==None ,"")
.otherwise(df.gender).alias("new_gender")
).show()
```

4.12 isin() - Check if value presents in a List.

```
#isin
li=
df.select(df.fname,df.lname,df.id)
.filter(df.id.isin(li))
.show()
```

4.13 getField() - To get the value by key from MapType column and by stuct child name from StructType column

Rest of the below functions operates on List, Map & Struct data structures hence to demonstrate these I will use another DataFrame with list, map and struct columns. For more explanation how to use Arrays refer to PySpark ArrayType Column on DataFrame Examples & for map refer to PySpark MapType Examples

```
#Create DataFrame with struct, array & map
from pyspark.sql.types import
StructType,StructField,StringType,ArrayType,MapType
data=,{'hair':'black','eye':'brown'}),
(("Ann","Varsa"),,{'hair':'brown','eye':'black'}),
(("Tom Cruise",""),,{'hair':'red','eye':'grey'}),
(("Tom Brand",None),,{'hair':'black','eye':'blue'})]
```

schema = StructType()),
StructField('languages', ArrayType(StringType()),True),
StructField('properties', MapType(StringType(),StringType()),True)
])
df=spark.createDataFrame(data,schema)
df.printSchema()

#Display's to console

root
|-- name: struct (nullable = true)
| |-- fname: string (nullable = true)
| |-- lname: string (nullable = true)
|-- languages: array (nullable = true)
| |-- element: string (containsNull = true)
|-- properties: map (nullable = true)
| |-- key: string
| |-- value: string (valueContainsNull = true)

getField Example

```
#getField from MapType
df.select(df.properties.getField("hair")).show()
```

#getField from Struct
df.select(df.name.getField("fname")).show()

4.14 getItem() - To get the value by index from MapType or ArrayTupe & ny key for MapType column.

```
#getItem() used with ArrayType
df.select(df.languages.getItem(1)).show()
```

#getItem() used with MapType
df.select(df.properties.getItem("hair")).show()

4.15 dropFields -

```
# TO-DO, getting runtime error
```

4.16 withField() -

```
# TO-DO getting runtime error
```

4.17 over() - Used with Window Functions

`TO-DO`

Happy Learning !!

Related Article