How can I use the expr() function in PySpark SQL to create an Expression?

Authored by

stats writer

June 24, 2024

The expr() function in PySpark SQL is a powerful tool that allows users to create expressions for data manipulation and analysis. By using this function, users can write complex expressions using SQL syntax and functions to perform various calculations and transformations on their data. This feature is particularly useful for data scientists and analysts who need to process large datasets efficiently. With the expr() function, users can easily create custom expressions tailored to their specific needs, making PySpark SQL a versatile and efficient tool for data analysis.

PySpark `expr()` is a SQL function to execute SQL-like expressions and to use an existing DataFrame column value as an expression argument to Pyspark built-in functions. Most of the commonly used SQL functions are either part of the PySpark Column class or built-in `pyspark.sql.functions` API, besides these PySpark also supports many other SQL functions, so in order to use these, you have to use `expr()` function.

Below are 2 use cases of **PySpark expr() funcion**.

1. PySpark expr() Syntax

Following is syntax of the expr() function.

```
expr(str)
```

`expr()` function takes SQL expression as a string argument, executes the expression, and returns a PySpark Column type. Expressions provided with this function are not a compile-time safety like DataFrame operations.

2. PySpark SQL expr() Function Examples

Below are some of the examples of using expr() SQL function.

2.1 Concatenate Columns using || (similar to SQL)

If you have SQL background, you pretty much familiar using || to concatenate values from two string columns, you can use expr() expression to do exactly same.

```
#Concatenate columns using || (sql like)
data=
df=spark.createDataFrame(data).toDF("col1","col2")
df.withColumn("Name",expr(" col1 ||','|| col2")).show()
```

```
+-----+-----+-----------+
| col1| col2| Name|
+-----+-----+-----------+
|James| Bond| James,Bond|
|Scott|Varsa|Scott,Varsa|
+-----+-----+-----------+
```

2.2 Using SQL CASE WHEN with expr()

PySpark doesn't have SQL Like CASE WHEN so in order to use this on PySpark DataFrame withColumn() or select(), you should use `expr()` function with expression as shown below.

Here, I have used `CASE WHEN` expression on `withColumn()` by using `expr()`, this example updates an existing column gender with the derived values, **M for male, F for Female,** and unknown **for others**

```
from pyspark.sql.functions import expr
data =
columns =
df = spark.createDataFrame(data = data, schema = columns)
```

```
#Using CASE WHEN similar to SQL.
from pyspark.sql.functions import expr
df2=df.withColumn("gender", expr("CASE WHEN gender = 'M' THEN 'Male' " +
"WHEN gender = 'F' THEN 'Female' ELSE 'unknown' END"))
df2.show()
+-------+-------+
| name| gender|
+-------+-------+
| James| Male|
|Michael| Female|
| Jen|unknown|
+-------+-------+
```

If you have any errors in the expression you will get the run time error but not during the compile time.

2.3 Using an Existing Column Value for Expression

Most of the PySpark function takes constant literal values but sometimes we need to use a value from an existing column instead of a constant and this is not possible without expr() expression. The below example adds a number of months from an existing column instead of a Python constant.

```
from pyspark.sql.functions import expr
data=
df=spark.createDataFrame(data).toDF("date","increment")

#Add Month value from another column
df.select(df.date,df.increment,
expr("add_months(date,increment)")
.alias("inc_date")).show()
```

```
+----------+---------+----------+
| date|increment| inc_date|
+----------+---------+----------+
|2019-01-23| 1|2019-02-23|
|2019-06-24| 2|2019-08-24|
|2019-09-20| 3|2019-12-20|
+----------+---------+----------+
```

Note that Importing SQL functions are not required when using them with expr(). You see above add_months() is used without importing.

2.4 Giving Column Alias along with expr()

You can also use SQL like syntax to provide the alias name to the column expression.

```
# Providing alias using 'as'
from pyspark.sql.functions import expr
df.select(df.date,df.increment,
expr("""add_months(date,increment) as inc_date""")
).show()
# This yields same output as above
```

2.5 cast Function with expr()

The below example converts long data type to String type.

```
# Using Cast() Function
df.select("increment",expr("cast(increment as string) as str_increment"))
.printSchema()
```

root
|-- increment: long (nullable = true)
|-- str_increment: string (nullable = true)

2.7 Arithmetic operations

expr() is also used to provide arithmetic operations, below examples add value 5 to `increment` and creates a new column `new_increment`

```
# Arthemetic operations
df.select(df.date,df.increment,
expr("increment + 5 as new_increment")
).show()
```

```
+----------+---------+-------------+
| date|increment|new_increment|
+----------+---------+-------------+
|2019-01-23| 1| 6|
|2019-06-24| 2| 7|
|2019-09-20| 3| 8|
+----------+---------+-------------+
```

2.8 Using Filter with expr()

Filter the DataFrame rows can done using expr() expression.

```
#Use expr() to filter the rows
from pyspark.sql.functions import expr
data=
df=spark.createDataFrame(data).toDF("col1","col2")
df.filter(expr("col1 == col2")).show()
```

+----+----+
|col1|col2|
+----+----+
| 500| 500|
+----+----+

3. Complete Example of PySpark SQL expr() Function

This example is also available at GitHub PySpark Examples Project.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
```

from pyspark.sql.functions import expr
#Concatenate columns
data=
df=spark.createDataFrame(data).toDF("col1","col2")
df.withColumn("Name",expr(" col1 ||','|| col2")).show()

#Using CASE WHEN sql expression
data =
columns =
df = spark.createDataFrame(data = data, schema = columns)
df2 = df.withColumn("gender", expr("CASE WHEN gender = 'M' THEN 'Male' " +
"WHEN gender = 'F' THEN 'Female' ELSE 'unknown' END"))
df2.show()

#Add months from a value of another column
data=
df=spark.createDataFrame(data).toDF("date","increment")
df.select(df.date,df.increment,
expr("add_months(date,increment)")
.alias("inc_date")).show()

Providing alias using 'as'
df.select(df.date,df.increment,
expr("""add_months(date,increment) as inc_date""")
).show()

Add

```
df.select(df.date,df.increment,
expr("increment + 5 as new_increment")
).show()

# Using cast to convert data types
df.select("increment",expr("cast(increment as string) as str_increment"))
.printSchema()

#Use expr() to filter the rows
data=
df=spark.createDataFrame(data).toDF("col1","col2")
df.filter(expr("col1 == col2")).show()
```

Conclusion

PySpark expr() function provides a way to run SQL like expression with DataFrames, here you have learned how to use expression with select(), withColumn() and to filter the DataFrame rows.

Happy Learning !!

Related Articles

References