How can I use GroupBy in PySpark? Can you provide an example?

Authored by

stats writer

June 24, 2024

GroupBy is a function in PySpark that allows you to group data based on specific criteria. It is particularly useful for data analysis and aggregation tasks. To use GroupBy in PySpark, you first need to import the necessary libraries and create a SparkSession. Then, you can use the GroupBy function on a DataFrame to group the data by a particular column or columns. This will result in a GroupedData object, which can then be used to perform various aggregate operations such as count, sum, mean, etc. An example of using GroupBy in PySpark would be to group sales data by product category and then calculate the total revenue for each category.

Similar to SQL `GROUP BY` clause, PySpark `groupBy()` transformation that is used to group rows that have the same values in specified columns into summary rows. It allows you to perform aggregate functions on groups of rows, rather than on individual rows, enabling you to summarize data and generate aggregate statistics.

How to group and aggregate data using Spark and Scala

1. GroupBy() Syntax & Usage

Following is the syntax of the groupby

```
# Syntax
DataFrame.groupBy(*cols)
#or
DataFrame.groupby(*cols)
```

When we perform `groupBy()` on PySpark Dataframe, it returns `GroupedData` object which contains below aggregate functions.

Function	Definition
`count()`	Use groupBy() count() to return the number of rows for each group.
`mean()`	Returns the mean of values for each group.
`max()`	Returns the maximum of values for each group.
`min()`	Returns the minimum of values for each group.
`sum()`	Returns the total for values for each group.
`avg()`	Returns the average for values for each group.
`agg()`	Using groupBy() agg() function, we can calculate more than one aggregate at a time.
`pivot()`	This function is used to Pivot the DataFrame, which I will not cover in this article as I already have a dedicated article for Pivot & Unpivot DataFrame.

PySpark Groupby Functions

Before we proceed, let's construct the DataFrame with columns such as "employee_name", "department", "state", "salary", "age", and "bonus".

We will use this PySpark DataFrame to run groupBy() on "department" columns and calculate aggregates like minimum, maximum, average, and total salary for each group using min(), max(), and sum() aggregate functions, respectively.

```
from pyspark.sql import SparkSession

# Initialize SparkSession
spark = SparkSession.builder
.appName("example")
.getOrCreate()

# Data
simpleData =

# Create DataFrame
schema =
df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)
```

Yields below output.

```
# Output:
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
| James| Sales| NY| 90000| 34|10000|
| Michael| Sales| NY| 86000| 56|20000|
| Robert| Sales| CA| 81000| 30|23000|
| Maria| Finance| CA| 90000| 24|23000|
| Raman| Finance| CA| 99000| 40|24000|
| Scott| Finance| NY| 83000| 36|19000|
| Jen| Finance| NY| 79000| 53|15000|
| Jeff| Marketing| CA| 80000| 25|18000|
| Kumar| Marketing| NY| 91000| 50|21000|
+-------------+----------+-----+------+---+-----+
```

2. PySpark groupBy on DataFrame Columns

Let's do the `groupBy()` on `department` column of DataFrame and then find the sum of salary for each department using `sum()` function.

Here's how the `groupBy()` function works:

Grouping: You specify one or more columns in the `groupBy()` function to define the grouping criteria. Rows with identical values in the specified columns are grouped together into distinct groups.**Aggregation**: After grouping the rows, you can apply aggregate functions such as `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, etc., to each group. These aggregate functions compute summary statistics for the rows within each group.

The groupBy() is commonly used in conjunction with aggregate functions to perform data summarization, generate reports, and derive insights from datasets. It allows you to analyze data at different levels of granularity by grouping rows based on specific attributes and computing aggregate statistics for each group.

```
# Using groupBy().sum()
df.groupBy("department").sum("salary").show(truncate=False)
```

Output:
#+----------+-----------+
#|department|sum(salary)|
#+----------+-----------+
#|Sales |257000 |
#|Finance |351000 |
#|Marketing |171000 |
#+----------+-----------+

Similarly, we can calculate the number of employees in each department using.

```
# Using groupBy().count()
df.groupBy("department").count()
```

Calculate the minimum salary of each department using `min()`

```
# Using groupBy().min()
```

```
df.groupBy("department").min("salary")
```

Calculate the maximin salary of each department using `max()`

```
# Using groupBy().max()
df.groupBy("department").max("salary")
```

Calculate the average salary of each department using `avg()`

```
# Using groupBy().avg()
df.groupBy("department").avg( "salary")
```

Calculate the mean salary of each department using `mean()`

```
# Using groupBy().mean()
df.groupBy("department").mean( "salary")
```

3. Using Multiple columns

Similarly, we can also run groupBy and aggregate on two or more DataFrame columns, below example does group by on `department`,`state` and does sum() on `salary` and `bonus` columns.

```
# GroupBy on multiple columns
df.groupBy("department","state")
.sum("salary","bonus")
.show(false)
```

This yields the below output.

```
# Output:
+----------+-----+-----------+----------+
|department|state|sum(salary)|sum(bonus)|
+----------+-----+-----------+----------+
|Finance |NY |162000 |34000 |
|Marketing |NY |91000 |21000 |
|Sales |CA |81000 |23000 |
```

```
|Marketing |CA |80000 |18000 |
|Finance |CA |189000 |47000 |
|Sales |NY |176000 |30000 |
+----------+-----+-----------+----------+
```

Similarly, we can run group by and aggregate on two or more columns for other aggregate functions; please refer to the example below.

4. Running more aggregates at a time

Alternatively, you can use `groupBy().agg()` to perform aggregation on DataFrame columns after grouping them based on one or more keys. It allows you to compute aggregate functions on grouped data, generating summary statistics for each group.

After grouping, you call the `agg()` method, which takes one or more pairs of column names and aggregation functions. The aggregation functions can include built-in functions like `count()`, `sum()`, `avg()`, `min()`, `max()`, etc., as well as user-defined functions.

```
from pyspark.sql.functions import sum,avg,max

# Running more aggregations
df.groupBy("department")
.agg(sum("salary").alias("sum_salary"),
avg("salary").alias("avg_salary"),
sum("bonus").alias("sum_bonus"),
max("bonus").alias("max_bonus")
)
.show(truncate=False)
```

This code will produce a DataFrame with the aggregated statistics for each `department`, including the sum of salaries, average salary, sum of bonuses, and maximum bonus. The `show()` method displays the DataFrame content in tabular format, with the columns "department", "sum_salary", "avg_salary", "sum_bonus", and "max_bonus".

```
# Output:
+----------+----------+-----------------+---------+---------+
|department|sum_salary|avg_salary       |sum_bonus|max_bonus|
+----------+----------+-----------------+---------+---------+
|Sales |257000 |85666.66666666667|53000 |23000 |
```

```
|Finance |351000 |87750.0 |81000 |24000 |
|Marketing |171000 |85500.0 |39000 |21000 |
+----------+----------+-----------------+---------+---------+
```

5. Using filter on aggregate data

```
from pyspark.sql.functions import sum,avg,max
```

```
# Using filter on aggregate data
df.groupBy("department")
.agg(sum("salary").alias("sum_salary"),
avg("salary").alias("avg_salary"),
sum("bonus").alias("sum_bonus"),
max("bonus").alias("max_bonus"))
.where(col("sum_bonus") >= 50000)
.show(truncate=False)
```

This removes the sum of a bonus that has less than 50000 and yields below output.

```
# Output:
+----------+----------+-----------------+---------+---------+
|department|sum_salary|avg_salary |sum_bonus|max_bonus|
+----------+----------+-----------------+---------+---------+
|Sales |257000 |85666.66666666667|53000 |23000 |
|Finance |351000 |87750.0 |81000 |24000 |
+----------+----------+-----------------+---------+---------+
```

6. PySpark SQL GROUP BY Query

To run SQL queries in PySpark, you have to register the DataFrame as a temporary table/view. The `createOrReplaceTempView()` is a method available on DataFrame objects that allows you to register a DataFrame as a temporary view in the SparkSession. This temporary view acts as a pointer to the DataFrame and enables you to run SQL queries against it using Spark SQL.

After registering the DataFrame as a temporary view, you can write SQL queries against it using Spark SQL. These SQL queries can include various operations such as selection, filtering, grouping, joining, and aggregation, allowing you to perform data analysis and manipulation

The `HAVING` clause is used in conjunction with the `GROUP BY` clause to filter groups based on

aggregate conditions. While the `WHERE` clause filters individual rows before grouping, the `HAVING` clause filters groups after grouping and aggregation have been performed.

```
# Register DataFrame as a temporary view
df.createOrReplaceTempView("employees")
```

```
# Using SQL Query
sql_string = """SELECT department,
SUM(salary) AS sum_salary,
AVG(salary) AS avg_salary,
SUM(bonus) AS sum_bonus,
MAX(bonus) AS max_bonus
FROM employees
GROUP BY department
HAVING SUM(bonus) >= 50000"""

# Execute SQL query against the temporary view
df2 = spark.sql(sql_string)
df2.show()
```

6. PySpark groupBy Example Source code

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col,sum,avg,max

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData =

schema =
df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)

df.groupBy("department").sum("salary").show(truncate=False)

df.groupBy("department").count().show(truncate=False)

df.groupBy("department","state")
.sum("salary","bonus")
```

```
.show(truncate=False)

df.groupBy("department")
.agg(sum("salary").alias("sum_salary"),
avg("salary").alias("avg_salary"),
sum("bonus").alias("sum_bonus"),
max("bonus").alias("max_bonus")
)
.show(truncate=False)

df.groupBy("department")
.agg(sum("salary").alias("sum_salary"),
avg("salary").alias("avg_salary"),
sum("bonus").alias("sum_bonus"),
max("bonus").alias("max_bonus"))
.where(col("sum_bonus") >= 50000)
.show(truncate=False)
```

This example is also available at GitHub PySpark Examples project for reference.

7. Conclusion

In this tutorial, you have learned how to use `groupBy()` functions on PySpark DataFrame and also learned how to run these on multiple columns and finally filter data on the aggregated columns.

Thanks for reading. If you like it, please do share the article by following the below social links and any comments or suggestions are welcome in the comments sections!

Happy Learning !!

Related Articles