databricks PySpark

(https://databricks.com)

```
'''
PySpark Join is used to combine two DataFrames and by chaining these you can join multiple DataFrames; it supports all basic join type
OUTER, RIGHT OUTER, LEFT ANTI, LEFT SEMI, CROSS, SELF JOIN. PySpark Joins are wider transformations that involve data shuffling across

PySpark SQL Joins comes with more optimization by default (thanks to DataFrames) however still there would be some performance issues

In this PySpark SQL Join section, you will learn different Join syntaxes and using different Join types on two or more DataFrames and

PySpark Join Syntax
PySpark Join Types
Inner Join DataFrame
Full Outer Join DataFrame
Left Outer Join DataFrame
Right Outer Join DataFrame
Left Anti Join DataFrame
Left Semi Join DataFrame
Self Join DataFrame
Using SQL Expression

1. PySpark Join Syntax
PySpark SQL join has a below syntax and it can be accessed directly from DataFrame.
join() operation takes parameters as below and returns DataFrame.

join(self, other, on=None, how=None)

param other: Right side of the join
param on: a string for the join column name
param how: default inner. Must be one of inner, cross, outer,full, full_outer, left, left_outer, right, right_outer,left_semi, and lef
You can also write Join expression by adding where() and filter() methods on DataFrame and can have Join on multiple columns.

2. PySpark Join Types
Below are the different Join Types PySpark supports.

Join String                          Equivalent SQL Join
inner                              INNER JOIN
outer, full, fullouter, full_outer   FULL OUTER JOIN
left, leftouter, left_outer          LEFT JOIN
right, rightouter, right_outer       RIGHT JOIN
cross
anti, leftanti, left_anti
semi, leftsemi, left_semi

Before we jump into PySpark SQL Join examples, first, let’s create an "emp" and "dept" DataFrames. here, column "emp_id" is unique on
emp_dept_id from emp has a reference to dept_id on dept dataset.
'''

emp = [(1,"Smith",-1,"2018","10","M",3000), \
    (2,"Rose",1,"2010","20","M",4000), \
    (3,"Williams",1,"2010","10","M",1000), \
    (4,"Jones",2,"2005","10","F",2000), \
    (5,"Brown",2,"2010","40","",-1), \
      (6,"Brown",2,"2010","50","",-1) \
  ]
empColumns = ["emp_id","name","superior_emp_id","year_joined", \
       "emp_dept_id","gender","salary"]

empDF = spark.createDataFrame(data=emp, schema = empColumns)
empDF.printSchema()
empDF.show(truncate=False)

dept = [("Finance",10), \
    ("Marketing",20), \
    ("Sales",30), \
```

```
    ("IT",40) \
  ]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)

'''This prints “emp” and “dept” DataFrame to the console. Refer complete example below on how to create spark object.'''
```

```
root
 |-- emp_id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- superior_emp_id: long (nullable = true)
 |-- year_joined: string (nullable = true)
 |-- emp_dept_id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: long (nullable = true)

+------+--------+---------------+-----------+-----------+------+------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|
+------+--------+---------------+-----------+-----------+------+------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |
|2     |Rose    |1              |2010       |20         |M     |4000  |
|3     |Williams|1              |2010       |10         |M     |1000  |
|4     |Jones   |2              |2005       |10         |F     |2000  |
|5     |Brown   |2              |2010       |40         |      |-1    |
|6     |Brown   |2              |2010       |50         |      |-1    |
+------+--------+---------------+-----------+-----------+------+------+

root
```

```
'''
3. PySpark Inner Join DataFrame
Inner join is the default join in PySpark and it’s mostly used. This joins two datasets on key columns, where keys don’t match
the rows get dropped from both datasets (emp & dept).

When we apply Inner join on our datasets, It drops “emp_dept_id” 50 from “emp” and “dept_id” 30 from “dept” datasets. Below is
the result of the above Join expression.
'''
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") \
     .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

```
'''
4. PySpark Full Outer Join
Outer a.k.a full, fullouter join returns all rows from both datasets, where join expression doesn’t match it returns null on
respective record columns.

From our “emp” dataset’s “emp_dept_id” with value 50 doesn’t have a record on “dept” hence dept columns have null and “dept_id”
30 doesn’t have a record in “emp” hence you see null’s on emp columns. Below is the result of the below Join expression.
'''
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"outer") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"full") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"fullouter") \
    .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|null  |null    |null           |null       |null       |null  |null  |Sales    |30     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
|6     |Brown   |2              |2010       |50         |      |-1    |null     |null   |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+

+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|null  |null    |null           |null       |null       |null  |null  |Sales    |30     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
```

```
  '''
  5. PySpark Left Outer Join
Left a.k.a Leftouter join returns all rows from the left dataset regardless of match found on the right dataset when join
expression doesn’t match, it assigns null for that record and drops records from right where match not found.

From our dataset, “emp_dept_id” 50 doesn’t have a record on “dept” dataset hence, this record contains null on “dept” columns
(dept_name & dept_id). and “dept_id” 30 from “dept” dataset dropped from the results. Below is the result of the above Join
expression.
  '''
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"left") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftouter") \
    .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
|6     |Brown   |2              |2010       |50         |      |-1    |null     |null   |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+

+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

```
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
|6     |Brown   |2              |2010       |50         |      |-1    |null     |null   |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

```
'''
6. Right Outer Join
Right a.k.a Rightouter join is opposite of left join, here it returns all rows from the right dataset regardless of math found
on the left dataset, when join expression doesn’t match, it assigns null for that record and drops records from left where match
not found.

From our example, the right dataset “dept_id” 30 doesn’t have it on the left dataset “emp” hence, this record contains null on
“emp” columns. and “emp_dept_id” 50 dropped as a match not found on left. Below is the result of the above Join expression.
'''
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"right") \
   .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"rightouter") \
   .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|null  |null    |null           |null       |null       |null  |null  |Sales    |30     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+

+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|null  |null    |null           |null       |null       |null  |null  |Sales    |30     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

```
'''
7. Left Semi Join
leftsemi join is similar to inner join difference being leftsemi join returns all columns from the left dataset and ignores all
columns from the right dataset. In other words, this join returns columns from the only left dataset for the records match in
the right dataset on join expression, records not matched on join expression are ignored from both left and right datasets.

The same result can be achieved using select on the result of the inner join however, using this join would be efficient.
Below is the result of the above join expression.
'''
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftsemi") \
   .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|
+------+--------+---------------+-----------+-----------+------+------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |
|3     |Williams|1              |2010       |10         |M     |1000  |
|4     |Jones   |2              |2005       |10         |F     |2000  |
|2     |Rose    |1              |2010       |20         |M     |4000  |
|5     |Brown   |2              |2010       |40         |      |-1    |
```

```
+------+--------+---------------+-----------+-----------+------+------+
```

```
'''
8. Left Anti Join
leftanti join does the exact opposite of the leftsemi, leftanti join returns only columns from the left dataset for non-matched records.

Yields below output
'''
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftanti") \
   .show(truncate=False)
```

```
+------+-----+---------------+-----------+-----------+------+------+
|emp_id|name |superior_emp_id|year_joined|emp_dept_id|gender|salary|
+------+-----+---------------+-----------+-----------+------+------+
|6     |Brown|2              |2010       |50         |      |-1    |
+------+-----+---------------+-----------+-----------+------+------+
```

```
'''
Joins are not complete without a self join, Though there is no self-join type available, we can use any of the above-explained join types to join DataFrame to itself. below example use inner self join.

Here, we are joining emp dataset with itself to find out superior emp_id and name for all employees.
'''
from pyspark.sql.functions import col
empDF.alias("emp1").join(empDF.alias("emp2"), \
    col("emp1.superior_emp_id") == col("emp2.emp_id"),"inner") \
    .select(col("emp1.emp_id"),col("emp1.name"), \
      col("emp2.emp_id").alias("superior_emp_id"), \
      col("emp2.name").alias("superior_emp_name")) \
   .show(truncate=False)
```

```
+------+--------+---------------+-----------------+
|emp_id|name    |superior_emp_id|superior_emp_name|
+------+--------+---------------+-----------------+
|2     |Rose    |1              |Smith            |
|3     |Williams|1              |Smith            |
|4     |Jones   |2              |Rose             |
|5     |Brown   |2              |Rose             |
|6     |Brown   |2              |Rose             |
+------+--------+---------------+-----------------+
```

```
'''
4. Using SQL Expression
Since PySpark SQL support native SQL syntax, we can also write join operations after creating temporary tables on DataFrames and
use these tables on spark.sql().


'''
empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")

joinDF = spark.sql("select * from EMP e, DEPT d where e.emp_dept_id == d.dept_id") \
  .show(truncate=False)

joinDF2 = spark.sql("select * from EMP e INNER JOIN DEPT d ON e.emp_dept_id == d.dept_id") \
  .show(truncate=False)
```

```
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+

+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|dept_name|dept_id|
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |Finance  |10     |
|3     |Williams|1              |2010       |10         |M     |1000  |Finance  |10     |
|4     |Jones   |2              |2005       |10         |F     |2000  |Finance  |10     |
|2     |Rose    |1              |2010       |20         |M     |4000  |Marketing|20     |
|5     |Brown   |2              |2010       |40         |      |-1    |IT       |40     |
+------+--------+---------------+-----------+-----------+------+------+---------+-------+
```

```
'''
6. PySpark SQL Join Complete Example
'''
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

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

emp = [(1,"Smith",-1,"2018","10","M",3000), \
    (2,"Rose",1,"2010","20","M",4000), \
    (3,"Williams",1,"2010","10","M",1000), \
    (4,"Jones",2,"2005","10","F",2000), \
    (5,"Brown",2,"2010","40","",-1), \
      (6,"Brown",2,"2010","50","",-1) \
  ]
empColumns = ["emp_id","name","superior_emp_id","year_joined", \
       "emp_dept_id","gender","salary"]

empDF = spark.createDataFrame(data=emp, schema = empColumns)
empDF.printSchema()
empDF.show(truncate=False)


dept = [("Finance",10), \
    ("Marketing",20), \
    ("Sales",30), \
    ("IT",40) \
  ]
deptColumns = ["dept_name","dept_id"]
deptDF = spark.createDataFrame(data=dept, schema = deptColumns)
deptDF.printSchema()
deptDF.show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"inner") \
     .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"outer") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"full") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"fullouter") \
    .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"left") \
    .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftouter") \
   .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"right") \
   .show(truncate=False)
empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"rightouter") \
   .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftsemi") \
   .show(truncate=False)

empDF.join(deptDF,empDF.emp_dept_id ==  deptDF.dept_id,"leftanti") \
   .show(truncate=False)

empDF.alias("emp1").join(empDF.alias("emp2"), \
    col("emp1.superior_emp_id") == col("emp2.emp_id"),"inner") \
    .select(col("emp1.emp_id"),col("emp1.name"), \
      col("emp2.emp_id").alias("superior_emp_id"), \
      col("emp2.name").alias("superior_emp_name")) \
   .show(truncate=False)
```

```
empDF.createOrReplaceTempView("EMP")
deptDF.createOrReplaceTempView("DEPT")

joinDF = spark.sql("select * from EMP e, DEPT d where e.emp_dept_id == d.dept_id") \
  .show(truncate=False)

joinDF2 = spark.sql("select * from EMP e INNER JOIN DEPT d ON e.emp_dept_id == d.dept_id") \
  .show(truncate=False)
```

```
root
 |-- emp_id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- superior_emp_id: long (nullable = true)
 |-- year_joined: string (nullable = true)
 |-- emp_dept_id: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: long (nullable = true)

+------+--------+---------------+-----------+-----------+------+------+
|emp_id|name    |superior_emp_id|year_joined|emp_dept_id|gender|salary|
+------+--------+---------------+-----------+-----------+------+------+
|1     |Smith   |-1             |2018       |10         |M     |3000  |
|2     |Rose    |1              |2010       |20         |M     |4000  |
|3     |Williams|1              |2010       |10         |M     |1000  |
|4     |Jones   |2              |2005       |10         |F     |2000  |
|5     |Brown   |2              |2010       |40         |      |-1    |
|6     |Brown   |2              |2010       |50         |      |-1    |
+------+--------+---------------+-----------+-----------+------+------+

root
```

```
'''
PySpark distinct() function is used to drop/remove the duplicate rows (all columns) from DataFrame and dropDuplicates() is used
to drop rows based on selected (one or multiple) columns. In this article, you will learn how to use distinct() and
dropDuplicates() functions with PySpark example.

Before we start, first let’s create a DataFrame with some duplicate rows and values on a few columns. We use this DataFrame to
demonstrate how to get distinct multiple columns.

On the above table, record with employer name James has duplicate rows, As you notice we have 2 rows that have duplicate values
on all columns and we have 4 rows that have duplicate values on department and salary columns.

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

data = [("James", "Sales", 3000), \
    ("Michael", "Sales", 4600), \
    ("Robert", "Sales", 4100), \
    ("Maria", "Finance", 3000), \
    ("James", "Sales", 3000), \
    ("Scott", "Finance", 3300), \
    ("Jen", "Finance", 3900), \
    ("Jeff", "Marketing", 3000), \
    ("Kumar", "Marketing", 2000), \
    ("Saif", "Sales", 4100) \
  ]
columns= ["employee_name", "department", "salary"]
df = spark.createDataFrame(data = data, schema = columns)
df.printSchema()
df.show(truncate=False)
'''
1. Get Distinct Rows (By Comparing All Columns)
On the above DataFrame, we have a total of 10 rows with 2 rows having all values duplicated, performing distinct on this
DataFrame should get us 9 after removing 1 duplicate row.
distinct() function on DataFrame returns a new DataFrame after removing the duplicate records. This example yields the below
output.
'''
#Distinct
distinctDF = df.distinct()
print("Distinct count: "+str(distinctDF.count()))
distinctDF.show(truncate=False)
'''
Alternatively, you can also run dropDuplicates() function which returns a new DataFrame after removing duplicate rows.
'''
#Drop duplicates
df2 = df.dropDuplicates()
print("Distinct count: "+str(df2.count()))
df2.show(truncate=False)

'''
2. PySpark Distinct of Selected Multiple Columns
PySpark doesn’t have a distinct method which takes columns that should run distinct on (drop duplicate rows on selected multiple
columns) however, it provides another signature of dropDuplicates() function which takes multiple columns to eliminate
duplicates.

Note that calling dropDuplicates() on DataFrame returns a new DataFrame with duplicate rows removed.

Yields below output. If you notice the output, It dropped 2 records that are duplicate.
'''
#Drop duplicates on selected columns
dropDisDF = df.dropDuplicates(["department","salary"])
print("Distinct count of department salary : "+str(dropDisDF.count()))
dropDisDF.show(truncate=False)

'''
```

```
In this PySpark SQL article, you have learned distinct() method which is used to get the distinct values of rows (all columns)
and also learned how to use dropDuplicates() to get the distinct and finally learned using dropDuplicates() function to get
distinct of multiple columns.
'''
```

```
root
 |-- employee_name: string (nullable = true)
 |-- department: string (nullable = true)
 |-- salary: long (nullable = true)

+-------------+----------+------+
|employee_name|department|salary|
+-------------+----------+------+
|James        |Sales     |3000  |
|Michael      |Sales     |4600  |
|Robert       |Sales     |4100  |
|Maria        |Finance   |3000  |
|James        |Sales     |3000  |
|Scott        |Finance   |3300  |
|Jen          |Finance   |3900  |
|Jeff         |Marketing |3000  |
|Kumar        |Marketing |2000  |
|Saif         |Sales     |4100  |
+-------------+----------+------+

Distinct count: 9
```

```
'''
PySpark JSON Functions with Examples
PySpark JSON functions are used to query or extract the elements from JSON string of DataFrame column by path, convert it to
struct, mapt type e.t.c, In this article, I will explain the most used JSON SQL functions with Python examples.

1. PySpark JSON Functions
from_json() – Converts JSON string into Struct type or Map type.

to_json() – Converts MapType or Struct type to JSON string.

json_tuple() – Extract the Data from JSON and create them as a new columns.

get_json_object() – Extracts JSON element from a JSON string based on json path specified.

schema_of_json() – Create schema string from JSON string

1.1. Create DataFrame with Column contains JSON String
In order to explain these JSON functions first, let’s create DataFrame with a column contains JSON string.
'''

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

jsonString="""{"Zipcode":704,"ZipCodeType":"STANDARD","City":"PARC PARQUE","State":"PR"}"""
df=spark.createDataFrame([(1, jsonString)],["id","value"])
df.show(truncate=False)

'''
2.1. from_json()
PySpark from_json() function is used to convert JSON string into Struct type or Map type. The below example converts JSON string
to Map key-value pair. I will leave it to you to convert to struct type. Refer, Convert JSON string to Struct type column.
'''

#Convert JSON string column to Map type
from pyspark.sql.types import MapType,StringType
from pyspark.sql.functions import from_json
df2=df.withColumn("value",from_json(df.value,MapType(StringType(),StringType())))
df2.printSchema()
df2.show(truncate=False)

'''
2.2. to_json()
to_json() function is used to convert DataFrame columns MapType or Struct type to JSON string. Here, I am using df2 that created
from above from_json() example.
'''

from pyspark.sql.functions import to_json,col
df2.withColumn("value",to_json(col("value"))) \
   .show(truncate=False)

'''
2.3. json_tuple()
Function json_tuple() is used the query or extract the elements from JSON column and create the result as a new columns.
'''

from pyspark.sql.functions import json_tuple
df.select(col("id"),json_tuple(col("value"),"Zipcode","ZipCodeType","City")) \
    .toDF("id","Zipcode","ZipCodeType","City") \
    .show(truncate=False)
'''
2.4. get_json_object()
get_json_object() is used to extract the JSON string based on path from the JSON column.
'''

from pyspark.sql.functions import get_json_object
df.select(col("id"),get_json_object(col("value"),"$.ZipCodeType").alias("ZipCodeType")) \
    .show(truncate=False)
```

```
'''
2.5. schema_of_json()
Use schema_of_json() to create schema string from JSON string column.
'''

from pyspark.sql.functions import schema_of_json,lit
schemaStr=spark.range(1) \
    .select(schema_of_json(lit("""{"Zipcode":704,"ZipCodeType":"STANDARD","City":"PARC PARQUE","State":"PR"}"""))) \
    .collect()[0][0]
print(schemaStr)
```

```
+---+--------------------------------------------------------------------------+
|id |value                                                                     |
+---+--------------------------------------------------------------------------+
|1  |{"Zipcode":704,"ZipCodeType":"STANDARD","City":"PARC PARQUE","State":"PR"}|
+---+--------------------------------------------------------------------------+

root
 |-- id: long (nullable = true)
 |-- value: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)

+---+---------------------------------------------------------------------------+
|id |value                                                                      |
+---+---------------------------------------------------------------------------+
|1  |{Zipcode -> 704, ZipCodeType -> STANDARD, City -> PARC PARQUE, State -> PR}|
+---+---------------------------------------------------------------------------+

+---+----------------------------------------------------------------------------+
|id |value                                                                       |
+---+----------------------------------------------------------------------------+
```

```
root
 |-- employee_name: string (nullable = true)
 |-- department: string (nullable = true)
 |-- salary: long (nullable = true)

+-------------+----------+------+
|employee_name|department|salary|
+-------------+----------+------+
|James        |Sales     |3000  |
|Michael      |Sales     |4600  |
|Robert       |Sales     |4100  |
|Maria        |Finance   |3000  |
|James        |Sales     |3000  |
|Scott        |Finance   |3300  |
|Jen          |Finance   |3900  |
|Jeff         |Marketing |3000  |
|Kumar        |Marketing |2000  |
|Saif         |Sales     |4100  |
+-------------+----------+------+

+-------------+----------+------+----------+
```