Exploding Complex Types in PySpark

📌 Overview

In PySpark, **complex types** such as **arrays**, **maps**, and **structs** often require **flattening** or **exploding** into multiple rows to make data easier to process. This is achieved using `functions like explode(), posexplode(), explode_outer(), and inline() from pyspark.sql.functions.`

🔹 1. explode()

Purpose:

explode() converts each element in an **array** or each key-value pair in a **map** into a separate row.

If the value is null, it returns **no rows**.

Syntax:

```
from pyspark.sql.functions import explode
explode(col)
```

Example:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode

spark = SparkSession.builder.appName("ExplodeExample").getOrCreate()

data = [

    (1, ["red", "blue"]),
    (2, ["green"]),
    (3, None)
```

```
]
df = spark.createDataFrame(data, ["id", "colors"])
df_exploded = df.select("id", explode("colors").alias("color"))
df_exploded.show()
```

Output:

```
+--------+-------+
|   id   |color  |
+--------+-------+
|   1    | red   |
|   1    | blue  |
|   2    | green |
+--------+-------+
```

Key Notes:

- Null arrays result in **zero rows**.
- Useful for **flattening arrays into rows**.

🔹 2. explode_outer()

Purpose:

Similar to `explode()` but **keeps null values**, producing a row with `null` instead of removing it.

Syntax:

```
from pyspark.sql.functions import explode_outer
explode_outer(col)
```

Example:

```
from pyspark.sql.functions import explode_outer

df_exploded_outer = df.select("id",
explode_outer("colors").alias("color"))
df_exploded_outer.show()
```

Output:

```
+------+-----+
| id   |color|
+------+-----+
| 1    | red |
| 1    | blue|
| 2    |green|
| 3    | null|
+------+-----+
```

Key Notes:

- Retainsnull arrays by returning a **null row**.
- Usefulin**outer join-like behavior** when flattening.

🔹 3. posexplode()

Purpose:

Like `explode()`but also includes the **position (index)** of each element.

Syntax:

```
from pyspark.sql.functions import posexplode
posexplode(col)
```

Example:

```
from pyspark.sql.functions import posexplode

df_posexploded = df.select("id", posexplode("colors").alias("pos",
"color"))
df_posexploded.show()
```

Output:

```
+---+---+-----+
| id|pos|color|
+---+---+-----+
| 1 | 0 | red |
| 1 | 1 | blue|
| 2 | 0 |green|
+---+---+-----+
```

Key Notes:

- Adds an **integer position index** for ordering or referencing.
- Great when **array order matters**.

🔹 4. posexplode_outer()

Purpose:

Same as `posexplode()` but **retains null arrays**, producing a row with `null` values.

Example:

```
from pyspark.sql.functions import posexplode_outer

df_posexploded_outer = df.select("id",
posexplode_outer("colors").alias("pos", "color"))
df_posexploded_outer.show()
```

Output:

```
+---+----+-----+
| id| pos|color|
+---+----+-----+
| 1 | 0  | red |
| 1 | 1  | blue|
| 2 | 0  |green|
| 3 |null| null|
+---+----+-----+
```

🔹 5. inline()

Purpose:

Flattens an **array of structs** into **multiple rows with separate columns**.

Example:

```
from pyspark.sql.functions import inline
from pyspark.sql.types import StructType, StructField, StringType

data2 = [

    (1, [{"name": "apple", "color": "red"}, {"name": "banana",
"color": "yellow"}]),
    (2, [{"name": "grape", "color": "green"}])
]
schema = "id INT, fruits ARRAY<STRUCT<name: STRING, color: STRING>>"
df2 = spark.createDataFrame(data2, schema)

df2_inline = df2.select("id", inline("fruits"))
df2_inline.show()
```

Output:

```
+---+------+------+
| id| name| color|
+---+------+------+
| 1 | apple|   red|
|  1|banana|yellow|
| 2| grape| green|
+---+------+------+
```

🔹 When to Use Which?

Function	Includes Nulls?	Shows Position?	Works with Maps?	Works with Array of Struct?
`explode()`	No	No	Yes	Yes
`explode_outer()`	Yes	No	Yes	Yes
`posexplode()`	No	Yes	Yes	Yes
`posexplode_outer()`	Yes	Yes	Yes	Yes
`inline()`	No (but empty array = 0 rows)	No	No	Yes

Let's build your Data Engineering journey together!

Call us directly at: 9989454737

https://seekhobigdata.com/