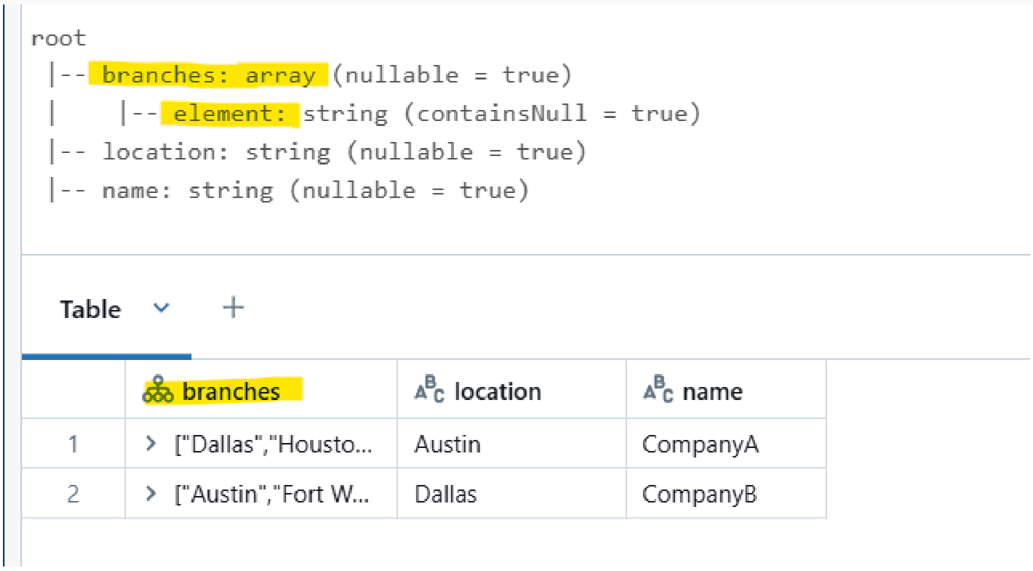
Master Pyspark Zero to Hero:

Working with Json Structure : Part 1

1. Exploding Arrays into Multiple Rows

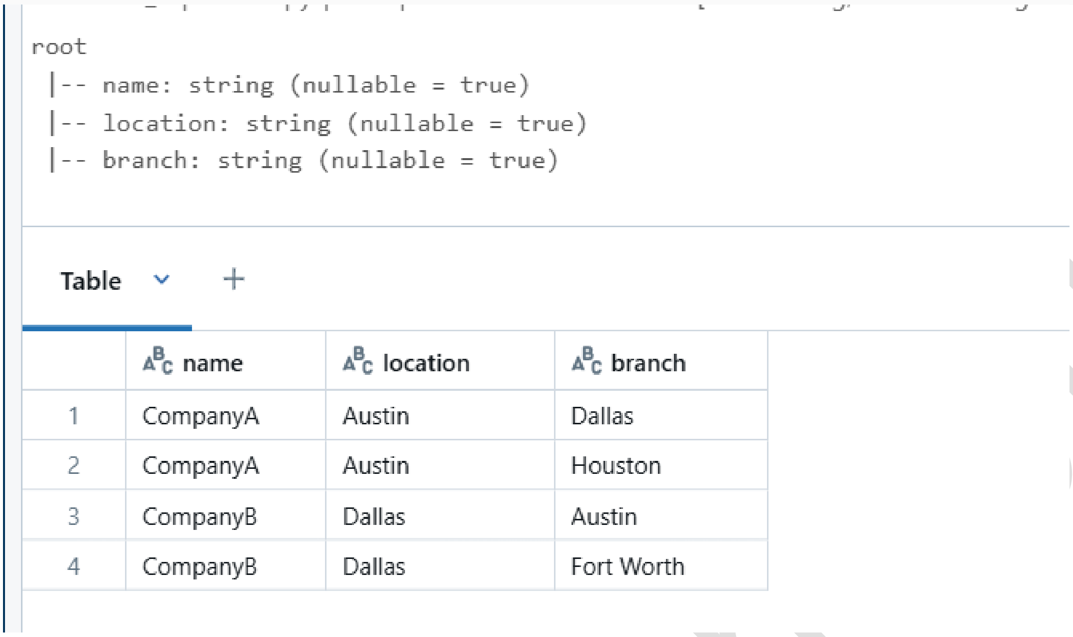
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode

# Initialize Spark session
spark = SparkSession.builder.appName("FlattenJson").getOrCreate()

# Sample JSON data
data = [
    {"name": "CompanyA", "location": "Austin", "branches":
["Dallas", "Houston"]},
    {"name": "CompanyB", "location": "Dallas", "branches":
["Austin", "Fort Worth"]}
]

# Create DataFrame from JSON
df = spark.read.json(spark.sparkContext.parallelize(data))
df.printSchema()
df.display()
```



```
root
 |-- branches: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- location: string (nullable = true)
 |-- name: string (nullable = true)
```

	branches	location	name
1	> ["Dallas","Housto...	Austin	CompanyA
2	> ["Austin","Fort W...	Dallas	CompanyB

```
df_exploded = df.select("name", "location",
explode("branches").alias("branch"))
df_exploded.printSchema()
df_exploded.display()
```



```
root
 |-- name: string (nullable = true)
 |-- location: string (nullable = true)
 |-- branch: string (nullable = true)
```

	name	location	branch
1	CompanyA	Austin	Dallas
2	CompanyA	Austin	Houston
3	CompanyB	Dallas	Austin
4	CompanyB	Dallas	Fort Worth

Explanation

- The branches array in each row is transformed so that each element of the array becomes a separate row. For example, if the array contains "Dallas" and "Houston", they will now appear as individual rows.
- The name and location columns are repeated for each newly created row corresponding to each branch.

Key Points

1. **Exploding:**
    - The explode() function takes an array (or map) column and generates a new row for each element of the array.
    - Each element of the array is assigned to its own row while the other columns remain unchanged.
2. **Resulting Data:**
    - The transformed data becomes a flat structure where the array's elements are split into individual rows.
    - Non-array columns (e.g., name, location) are duplicated for every row.

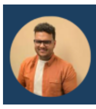
Benefits

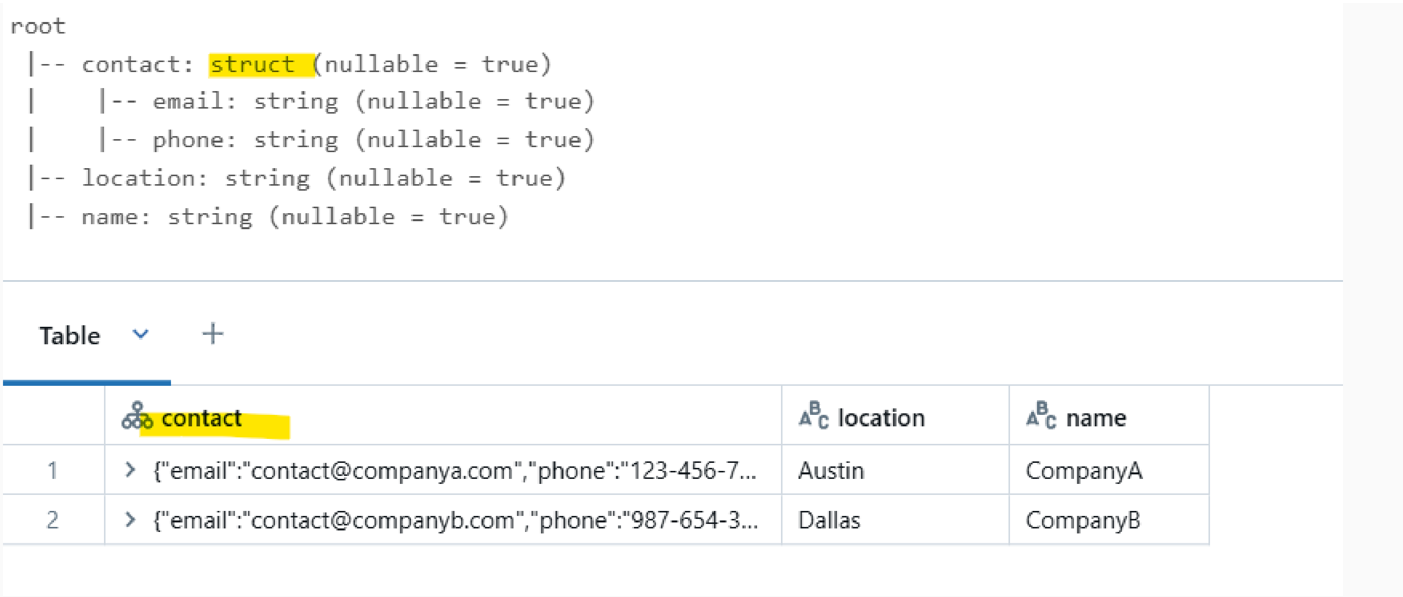
- This transformation is particularly useful when you need to analyze or filter individual elements of an array separately.
- It simplifies downstream data processing tasks by providing a normalized, flat structure.

---

3. Flattening Struct Fields into Columns

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Initialize Spark session
spark =
SparkSession.builder.appName("FlattenStructJson").getOrCreate()

# Sample JSON data
data = [
    {
        "name": "CompanyA",
        "location": "Austin",
        "contact": {
            "phone": "123-456-7890",
            "email": "contact@companya.com"
        }
    },
    {
        "name": "CompanyB",
        "location": "Dallas",
        "contact": {
            "phone": "987-654-3210",
            "email": "contact@companyb.com"
        }
    }
]

# Create DataFrame from JSON
df = spark.read.json(spark.sparkContext.parallelize(data))
```

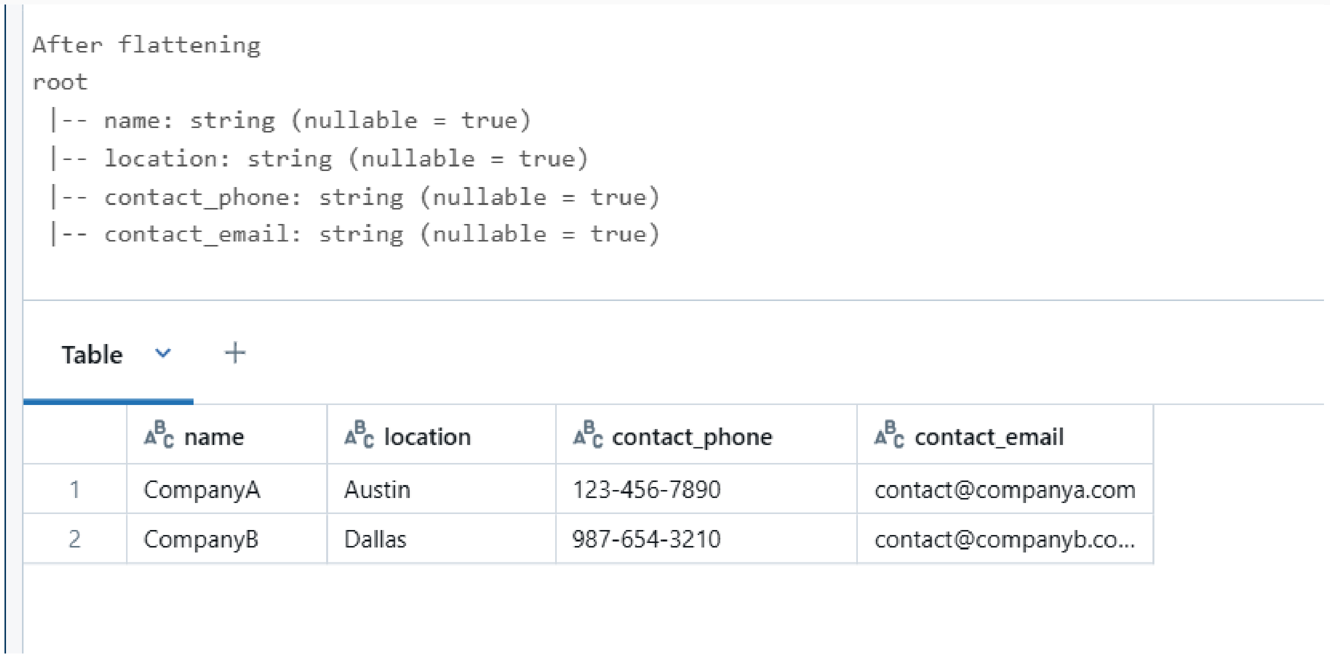




```
root
 |-- contact: struct (nullable = true)
 |    |-- email: string (nullable = true)
 |    |-- phone: string (nullable = true)
 |-- location: string (nullable = true)
 |-- name: string (nullable = true)
```

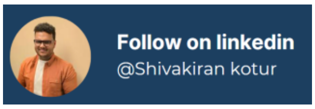
	contact	location	name
1	> {"email":"contact@companya.com","phone":"123-456-7...	Austin	CompanyA
2	> {"email":"contact@companyb.com","phone":"987-654-3...	Dallas	CompanyB

```
# Flatten the struct 'contact' into separate columns
flattened_df = df.select(
    "name",
    "location",
    col("contact.phone").alias("contact_phone"),
    col("contact.email").alias("contact_email")
)
df.printSchema()
df.display()

print("After flattening")
# Show the flattened DataFrame
flattened_df.printSchema()
flattened_df.display()
```



```
After flattening
root
 |-- name: string (nullable = true)
 |-- location: string (nullable = true)
 |-- contact_phone: string (nullable = true)
 |-- contact_email: string (nullable = true)
```

	name	location	contact_phone	contact_email
1	CompanyA	Austin	123-456-7890	contact@companya.com
2	CompanyB	Dallas	987-654-3210	contact@companyb.co...



Follow on linkedin
@Shivakiran kotur

Explanation

- A struct column, such as contact, may contain nested fields like phone and email. These fields can be accessed and transformed into individual top-level columns.
- For example, the contact struct containing contact.phone and contact.email can be flattened into two new columns: contact_phone and contact_email.

Key Points

1. **Struct Fields:**
   - A struct is a complex data type that encapsulates multiple fields. In this case, the contact struct contains phone and email.
2. **Flattening:**
   - Using the col() function, you can extract individual fields from the struct and create separate top-level columns for each field.
3. **Resulting Data:**
   - The nested structure is "flattened," making each field (e.g., phone, email) accessible as its own column.
   - This results in a cleaner, more accessible dataset suitable for analysis or reporting.

Benefits

- Flattening a struct allows easier access and manipulation of nested fields.
- It simplifies querying and ensures the data is ready for further transformations or aggregations.

Conclusion

These transformations—**exploding arrays** and **flattening structs**—are crucial steps in normalizing and simplifying data for analysis. They enhance the usability of complex data structures by converting them into a more accessible and flatter format.