Apache Spark: Explode Function

BigDataEnthusiast Follow 6 min read · Aug 15, 2023

Apache Spark built-in function that takes input as an column object (array or map type) and returns a new row for each element in the given array or map type column. Refer official documentation here.

Explode

Here is the example of `explode` function.

```
package com

import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.functions._

object InvokeCode extends App {

  val spark: SparkSession = SparkSession
    .builder()
    .master("local[1]")
    .appName("learn")
    .getOrCreate()

  val sc = spark.sparkContext
  sc.setLogLevel("OFF")

  import spark.implicits._

  val df = List(
    ("1", "A", List("1654123", "2654122", "3654121")),
    ("2", "B", List("1254123", "2354122", "3454121")),
    ("3", "C", List()),
    ("4", "D", null)
  ).toDF("id", "name", "phone_details")

  df.show(false)
  df.printSchema()

  val df_exploded =
    df.withColumn("phone_details_exploded", explode($"phone_details"))

  df_exploded.show(false)
  df_exploded.printSchema()
}
```

```
Input Dataframe: df

+---+----+---------------------------+
|id |name|phone_details              |
+---+----+---------------------------+
|1  |A   |[1654123, 2654122, 3654121]|
|2  |B   |[1254123, 2354122, 3454121]|
|3  |C   |[]                         |
```

```
|4  |D   |null                        |
+---+----+----------------------------+

Output Dataframe: df_exploded

+---+----+----------------------------+----------------------+
|id |name|phone_details               |phone_details_exploded|
+---+----+----------------------------+----------------------+
|1  |A   |[1654123, 2654122, 3654121]|1654123               |
|1  |A   |[1654123, 2654122, 3654121]|2654122               |
|1  |A   |[1654123, 2654122, 3654121]|3654121               |
|2  |B   |[1254123, 2354122, 3454121]|1254123               |
|2  |B   |[1254123, 2354122, 3454121]|2354122               |
|2  |B   |[1254123, 2354122, 3454121]|3454121               |
+---+----+----------------------------+----------------------+
```

If you have noticed here, exploded doesn't include empty list/null values. To include these null values we have to use `explode_outer` function.

```
val df_exploded =
  df.withColumn("phone_details_exploded", explode_outer($"phone_details"))

df_exploded.show(false)
```

```
Output Dataframe: df_exploded using explode_outer

+---+----+----------------------------+----------------------+
|id |name|phone_details               |phone_details_exploded|
+---+----+----------------------------+----------------------+
|1  |A   |[1654123, 2654122, 3654121]|1654123               |
|1  |A   |[1654123, 2654122, 3654121]|2654122               |
|1  |A   |[1654123, 2654122, 3654121]|3654121               |
|2  |B   |[1254123, 2354122, 3454121]|1254123               |
|2  |B   |[1254123, 2354122, 3454121]|2354122               |
|2  |B   |[1254123, 2354122, 3454121]|3454121               |
|3  |C   |[]                          |null                  |
|4  |D   |null                        |null                  |
```

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

Let’s explode if we have array of objects instead array of strings/integers.

Explode — Array of Struct

In this example we will explore array of struct.

```
package com

import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.sql.functions.explode
import org.apache.spark.sql.types._

object InvokeCode extends App {

  val spark: SparkSession = SparkSession
    .builder()
    .master("local[1]")
    .appName("learn")
    .getOrCreate()

  val sc = spark.sparkContext
  sc.setLogLevel("OFF")

  import spark.implicits._

  val dataSchema = new StructType()
    .add("name", StringType)
    .add("id", StringType)
    .add(
      "phone_details",
      ArrayType(
        new StructType()
```

```
            .add("phone_type", StringType)
            .add("number", StringType)
            .add("primary", BooleanType)
        )
      )

    val data = List(
      Row(
        "1",
        "A",
        List(Row("home", "+1 1254", true), Row("office", "+1 12345", false))
      ),
      Row(
        "2",
        "B",
        List(Row("home", "+1 1254", false), Row("office", "+1 12345", true))
      ),
      Row(
        "3",
        "C",
        List(Row("home", "+1 1254", true), Row("office", "+1 12345", false))
      ),
      Row("4", "D", List()),
      Row("5", "D", null)
    )
    val rdd = spark.sparkContext.parallelize(data)
    val df = spark.createDataFrame(rdd, dataSchema)

    df.show(false)
    df.printSchema()

    val df_exploded =
      df.withColumn("phone_details_exploded", explode($"phone_details"))

    df_exploded.show(false)
    df_exploded.printSchema()
```

```
Input Dataframe: df
+----+---+-----------------------------------------------------+
```

```
|name|id |phone_details                                          |
+----+---+-------------------------------------------------------+
|1   |A  |[[home, +1 1254, true], [office, +1 12345, false]]|
|2   |B  |[[home, +1 1254, false], [office, +1 12345, true]]|
|3   |C  |[[home, +1 1254, true], [office, +1 12345, false]]|
|4   |D  |[]                                                     |
|5   |D  |null                                                   |
+----+---+-------------------------------------------------------+

Output Dataframe: df_exploded
+----+---+-------------------------------------------------------+---------------
|name|id |phone_details                                          |phone_details_e
+----+---+-------------------------------------------------------+---------------
|1   |A  |[[home, +1 1254, true], [office, +1 12345, false]]|[home, +1 1254,
|1   |A  |[[home, +1 1254, true], [office, +1 12345, false]]|[office, +1 123
|2   |B  |[[home, +1 1254, false], [office, +1 12345, true]]|[home, +1 1254,
|2   |B  |[[home, +1 1254, false], [office, +1 12345, true]]|[office, +1 123
|3   |C  |[[home, +1 1254, true], [office, +1 12345, false]]|[home, +1 1254,
|3   |C  |[[home, +1 1254, true], [office, +1 12345, false]]|[office, +1 123
+----+---+-------------------------------------------------------+---------------
```

```
Input Dataframe Schema:

root
 |-- name: string (nullable = true)
 |-- id: string (nullable = true)
 |-- phone_details: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- phone_type: string (nullable = true)
 |    |    |-- number: string (nullable = true)
 |    |    |-- primary: boolean (nullable = true)

Output Dataframe Schema:
root
 |-- name: string (nullable = true)
 |-- id: string (nullable = true)
 |-- phone_details: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- phone_type: string (nullable = true)
 |    |    |-- number: string (nullable = true)
 |    |    |-- primary: boolean (nullable = true)
 |-- phone_details_exploded: struct (nullable = true)
 |    |-- phone_type: string (nullable = true)
 |    |-- number: string (nullable = true)
 |    |-- primary: boolean (nullable = true)
```

Notice in the above schema comparison, how the array of struct converted to struct after explode.

Elegant way to select all columns from `struct` see below example.

```
val df_exploded =
    df.withColumn("phone_details_exploded", explode($"phone_details"))

df_exploded
    .select("id", "name", "phone_details_exploded.*")
    .show(false)
```

```
+---+----+----------+--------+-------+
|id |name|phone_type|number  |primary|
+---+----+----------+--------+-------+
|A  |1   |home      |+1 1254 |true   |
|A  |1   |office    |+1 12345|false  |
|B  |2   |home      |+1 1254 |false  |
|B  |2   |office    |+1 12345|true   |
|C  |3   |home      |+1 1254 |true   |
|C  |3   |office    |+1 12345|false  |
+---+----+----------+--------+-------+
```

Other way of selecting columns by explicitly giving column names from struct.

```
df_exploded
    .select(
      "id",
      "phone_details_exploded.phone_type",
      "phone_details_exploded.number"
    )
    .show(false)
```

Explode Multiple Columns

Suppose we want to explode multiple columns: If we go with one by one approach for exploding multiple columns, it can create bunch of redundant data. We can do first `zip` columns & then explode. See below example.

```
package com

import org.apache.spark.sql.{Row, SparkSession}
import org.apache.spark.sql.functions.{explode, arrays_zip}
import org.apache.spark.sql.types._

object InvokeCode extends App {

  val spark: SparkSession = SparkSession
    .builder()
    .master("local[1]")
    .appName("learn")
    .getOrCreate()

  val sc = spark.sparkContext
  sc.setLogLevel("OFF")

  import spark.implicits._

  val dataSchema = new StructType()
    .add("name", StringType)
    .add("id", StringType)
```

```
      .add(
        "phone_details",
        ArrayType(
          new StructType()
            .add("phone_type", StringType)
            .add("number", StringType)
            .add("primary", BooleanType)
        )
      )
      .add(
        "address_details",
        ArrayType(
          new StructType()
            .add("address_type", StringType)
            .add("address", StringType)
            .add("primary", BooleanType)
        )
      )

    val data = List(
      Row(
        "1",
        "A",
        List(
          Row("home", "+1 1254", true),
          Row("home", "+1 1233", false),
          Row("office", "+1 12345", false)
        ),
        List(Row("home", "XYZ Lane", true), Row("office", "YZ CITY", false))
      ),
      Row(
        "2",
        "B",
        List(Row("home", "+1 1254", false), Row("office", "+1 12345", true)),
        List(Row("home", "ADC Lane", true), Row("office", "AYZ CITY", false))
      ),
      Row(
        "3",
        "C",
        List(Row("home", "+1 1254", true), Row("office", "+1 12345", false)),
        List(Row("home", "XSD Lane", true), Row("office", "XS CITY", false))
```

```
      ),
      Row("4", "D", List(), null),
      Row("5", "D", null, List())
    )
    val rdd = spark.sparkContext.parallelize(data)
    val df = spark.createDataFrame(rdd, dataSchema)

    df.show(false)
    df.printSchema()

    val df_exploded = df
      .withColumn(
        "zip_col",
        explode(arrays_zip($"phone_details", $"address_details"))
      )
      .drop("phone_details", "address_details")
      .select("id", "name", "zip_col.*")

    df_exploded.printSchema()

    df_exploded.show(false)
```

```
Input Dataframe:
+----+---+----------------------------------------------------------------
|name|id |phone_details
+----+---+----------------------------------------------------------------
|1   |A  |[[home, +1 1254, true], [home, +1 1233, false], [office, +1 12345,
|2   |B  |[[home, +1 1254, false], [office, +1 12345, true]]
|3   |C  |[[home, +1 1254, true], [office, +1 12345, false]]
|4   |D  |[]
|5   |D  |null
+----+---+----------------------------------------------------------------

root
 |-- name: string (nullable = true)
 |-- id: string (nullable = true)
 |-- phone_details: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- phone_type: string (nullable = true)
 |    |    |-- number: string (nullable = true)
 |    |    |-- primary: boolean (nullable = true)
 |-- address_details: array (nullable = true)
```

```
 |    |-- element: struct (containsNull = true)
 |    |    |-- address_type: string (nullable = true)
 |    |    |-- address: string (nullable = true)
 |    |    |-- primary: boolean (nullable = true)

Output Dataframe:

+---+----+------------------------+------------------------+
|id |name|phone_details           |address_details         |
+---+----+------------------------+------------------------+
|A  |1   |[home, +1 1254, true]   |[home, XYZ Lane, true]  |
|A  |1   |[home, +1 1233, false]  |[office, YZ CITY, false] |
|A  |1   |[office, +1 12345, false]|null                    |
|B  |2   |[home, +1 1254, false]  |[home, ADC Lane, true]  |
|B  |2   |[office, +1 12345, true] |[office, AYZ CITY, false]|
|C  |3   |[home, +1 1254, true]   |[home, XSD Lane, true]  |
|C  |3   |[office, +1 12345, false]|[office, XS CITY, false] |
+---+----+------------------------+------------------------+

root
 |-- id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- phone_details: struct (nullable = true)
 |    |-- phone_type: string (nullable = true)
 |    |-- number: string (nullable = true)
 |    |-- primary: boolean (nullable = true)
 |-- address_details: struct (nullable = true)
 |    |-- address_type: string (nullable = true)
 |    |-- address: string (nullable = true)
 |    |-- primary: boolean (nullable = true)
```

posexplode

Creates a new row for each element with position in the given array or map column. Uses the default column name `pos` for position, and `col` for elements in the array and `key` and `value` for elements in the map unless specified otherwise.

```
package com

import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.sql.functions._

object InvokeCode extends App {

  val spark: SparkSession = SparkSession
    .builder()
    .master("local[1]")
    .appName("learn")
    .getOrCreate()

  val sc = spark.sparkContext
  sc.setLogLevel("OFF")

  import spark.implicits._

  val df = List(
    ("1", "A", List("1654123", "2654122", "3654121")),
    ("2", "B", List("1254123", "2354122", "3454121")),
    ("3", "C", List()),
    ("4", "D", null)
  ).toDF("id", "name", "phone_details")

  df.show(false)
  df.select($"id",$"name", posexplode($"phone_details")).show(false)
}
```

```
Input Dataframe:
+---+----+---------------------------+
|id |name|phone_details              |
+---+----+---------------------------+
|1  |A   |[1654123, 2654122, 3654121]|
|2  |B   |[1254123, 2354122, 3454121]|
|3  |C   |[]                         |
|4  |D   |null                       |
+---+----+---------------------------+

Output Dataframe:
+---+----+---+-------+
|id |name|pos|col    |
+---+----+---+-------+
```

```
|1  |A   |0  |1654123|
|1  |A   |1  |2654122|
|1  |A   |2  |3654121|
|2  |B   |0  |1254123|
|2  |B   |1  |2354122|
|2  |B   |2  |3454121|
+---+----+---+-------+
```

If you are using `posexplode` in `withColumn` it might fail with this exception. Checkout this issue for more details: SPARK-20174

```
val df_exploded =
    df.withColumn("phone", posexplode($"phone_details"))

Exception in thread "main" org.apache.spark.sql.AnalysisException: The numbe
```

So better to use `posexplode` with `select` or `selectExpr`.

```
import spark.implicits._

val df = List(
  ("1", "A", List("1654123", "2654122", "3654121")),
  ("2", "B", List("1254123", "2354122", "3454121")),
  ("3", "C", List()),
  ("4", "D", null)
).toDF("id", "name", "phone_details")

df.selectExpr("*", "posexplode(phone_details) as (p,c)")
  .drop("phone_details")
  .show(false)
```

```
Output:

+---+----+---+-------+
|id |name|p  |c      |
+---+----+---+-------+
|1  |A   |0  |1654123|
|1  |A   |1  |2654122|
|1  |A   |2  |3654121|
|2  |B   |0  |1254123|
|2  |B   |1  |2354122|
|2  |B   |2  |3454121|
+---+----+---+-------+
```

Refer other blogs:

Apache Spark — createDataFrame

createDataFrame is an overloaded method present in SparkSession class (org.apache.spark.sql) and there are…

bigdataenthusiast.medium.com

Apache Spark — implicits Object (Implicits Conversions)

In the SparkSession class there is one object defined as implicits, which extends SQLImplicits abstract class. So once…

bigdataenthusiast.medium.com

References:

- https://spark.apache.org
- SPARK-20174

Apache Spark | Spark Scala Tutorial

Written by BigDataEnthusiast

91 followers · 4 following

AWS Certified Data Engineer | Databricks Certified Apache Spark 3.0 Developer | Databricks Certified Data Engineer | Oracle Certified SQL Expert

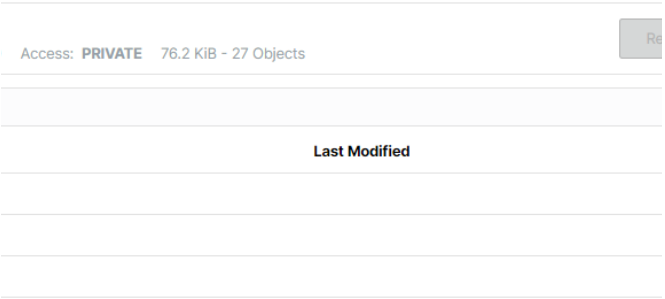
Follow

No responses yet

A Aaditya Adhikari

What are your thoughts?

More from BigDataEnthusiast



BigDataEnthusiast

Apache Iceberg—Hidden Partitioning

In this blog we will explore “Hidden Partitioning” concept in Apache Iceberg.

Mar 31, 2024 74

BigDataEnthusiast

Apache Iceberg—Insert Overwrite

INSERT OVERWRITE can replace/overwrite the data in iceberg table, depending on…

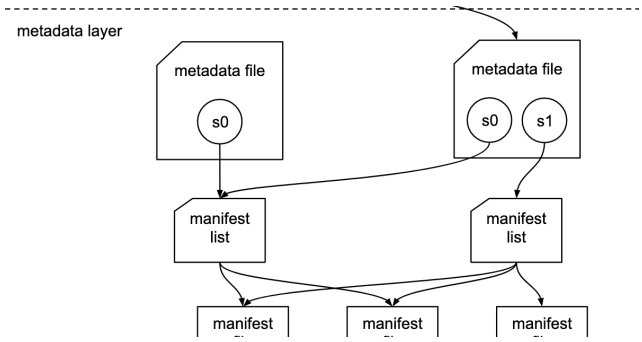
Jul 9, 2023 5

BigDataEnthusiast

How to create StructType schema from JSON schema | PySpark

Using Apache Spark class pyspark.sql.types.StructType method…

Nov 25, 2024 3



BigDataEnthusiast

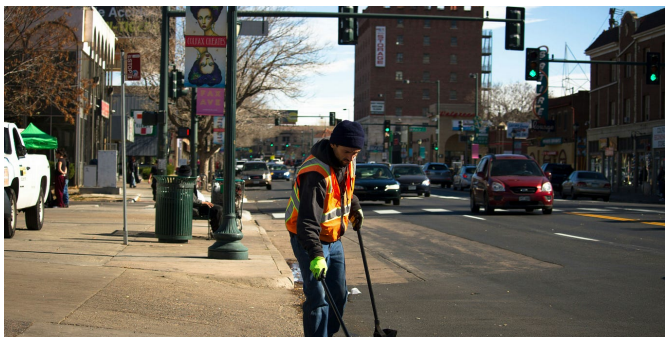
Apache Iceberg Table Format Versions

In this blog we will explore mainly these things.

Jun 25, 2023 10

See all from BigDataEnthusiast

Recommended from Medium



In Art of Data Engineering by Santosh Joshi

Understanding Collect, Take, Limit, Show, Head and Display i…

A Quick and Crisp Guide to Inspecting DataFrames Efficiently in PySpark

Jan 13 7

Sriw World of Coding

Optimizing Spark Resource Allocation with spark-submit

While there are several ways to optimize Spark jobs, one of the most fundamental…

May 4 2

Siddharth Ghosh

Spark Interview Series—Different Optimization Techniques(RBO,…

Apache Spark has evolved into a powerful big data processing engine with several…

Apr 8

Diogo Santos

Mastering Apache Spark Shuffle Optimization: Techniques for…

Explore the power of bucketing, repartitioning, and broadcast joins to…

Jan 26 64

Krishnatej Sreeramula

SQL Made Simple: Spark SQL Basics 🚀🗄️

Introduction

Jan 5 3

Overwrite Entire Table	Small tables	Simple	Inefficient for large tables
Update Specific Partitions	Partitioned tables	Efficient for large tables	Limited to partition-level updates
Merge Using Delta Lake	Frequent updates, ACID compliance	Scalable, ACID-compliant	Requires Delta Lake
Use a Join to Update Records	Standard Spark tables	Flexible	Requires overwriting the entire table
Use a Temporary Table	Atomicity without Delta Lake	Ensures atomicity	Requires additional storage
SQL `UPDATE`	Delta Lake, SQL-based	Simple and	Requires Delta Lake

Sujatha Mudadla

Section 2: Data Processing—Updating Records in a Spark…

In data processing, Type 1 updates refer to overwriting existing records with new data…

Jan 11 1

See more recommendations