**PySpark StructType & StructField Explained with Examples**

**1. StructType – Defines the structure of the Dataframe**

PySpark provides from pyspark.sql.types import StructType class to define the structure of the DataFrame.

StructType is a collection or list of StructField objects.

printSchema() method on the DataFrame shows StructType columns as “struct”.

**2. StructField – Defines the metadata of the DataFrame column**

PySpark provides pyspark.sql.types import StructField class to define the columns which includes column name(String), column type (DataType), nullable column (Boolean) and metadata (MetaData)

**3. Using PySpark StructType & StructField with DataFrame**

While creating a PySpark DataFrame we can specify the structure using StructType and StructField classes. As specified in the introduction, StructType is a collection of StructField’s which is used to define the column name, data type, and a flag for nullable or not. Using StructField we can also add nested struct schema, ArrayType for arrays, and MapType for key-value pairs which we will discuss in detail in later sections.

The below example demonstrates a very simple example of how to create a StructType & StructField on DataFrame and it’s usage with sample data to support it.

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType,StructField, StringType, IntegerType

data = [("James","","Smith","36636","M",3000),

("Michael","Rose","","40288","M",4000),

("Robert","","Williams","42114","M",4000),

("Maria","Anne","Jones","39192","F",4000),

("Jen","Mary","Brown","","F",-1)

]

schema = StructType([ \

StructField("firstname",StringType(),True), \

StructField("middlename",StringType(),True), \

StructField("lastname",StringType(),True), \

StructField("id", StringType(), True), \

StructField("gender", StringType(), True), \

StructField("salary", IntegerType(), True) \

])

df = spark.createDataFrame(data=data,schema=schema)

df.printSchema()

df.show(truncate=False)

By running the above snippet, it displays below outputs.

root

|-- firstname: string (nullable = true)

|-- middlename: string (nullable = true)

|-- lastname: string (nullable = true)

|-- id: string (nullable = true)

|-- gender: string (nullable = true)

|-- salary: integer (nullable = true)

+---------+----------+--------+-----+------+------+

|firstname|middlename|lastname|id |gender|salary|

+---------+----------+--------+-----+------+------+

|James | |Smith |36636|M |3000 |

|Michael |Rose | |40288|M |4000 |

|Robert | |Williams|42114|M |4000 |

|Maria |Anne |Jones |39192|F |4000 |

|Jen |Mary |Brown | |F |-1 |

+---------+----------+--------+-----+------+------+

**4. Defining Nested StructType object struct**

While working on DataFrame we often need to work with the nested struct column and this can be defined using StructType.

On the below example column “name” data type is StructType which is nested.

structureData = [

(("James","","Smith"),"36636","M",3100),

(("Michael","Rose",""),"40288","M",4300),

(("Robert","","Williams"),"42114","M",1400),

(("Maria","Anne","Jones"),"39192","F",5500),

(("Jen","Mary","Brown"),"","F",-1)

]

structureSchema = StructType([

StructField('name', StructType([

StructField('firstname', StringType(), True),

StructField('middlename', StringType(), True),

StructField('lastname', StringType(), True)

])),

StructField('id', StringType(), True),

StructField('gender', StringType(), True),

StructField('salary', IntegerType(), True)

])

df2 = spark.createDataFrame(data=structureData,schema=structureSchema)

df2.printSchema()

df2.show(truncate=False)

Outputs below schema and the DataFrame

root

|-- name: struct (nullable = true)

| |-- firstname: string (nullable = true)

| |-- middlename: string (nullable = true)

| |-- lastname: string (nullable = true)

|-- id: string (nullable = true)

|-- gender: string (nullable = true)

|-- salary: integer (nullable = true)

+--------------------+-----+------+------+

|name |id |gender|salary|

+--------------------+-----+------+------+

|[James, , Smith] |36636|M |3100 |

|[Michael, Rose, ] |40288|M |4300 |

|[Robert, , Williams]|42114|M |1400 |

|[Maria, Anne, Jones]|39192|F |5500 |

|[Jen, Mary, Brown] | |F |-1 |

+--------------------+-----+------+------+

**5. Adding & Changing struct of the DataFrame**

Using PySpark SQL function struct(), we can change the struct of the existing DataFrame and add a new StructType to it. The below example demonstrates how to copy the columns from one structure to another and adding a new column. PySpark Column Class also provides some functions to work with the StructType column.

from pyspark.sql.functions import col,struct,when

updatedDF = df2.withColumn("OtherInfo",

struct(col("id").alias("identifier"),

col("gender").alias("gender"),

col("salary").alias("salary"),

when(col("salary").cast(IntegerType()) < 2000,"Low")

.when(col("salary").cast(IntegerType()) < 4000,"Medium")

.otherwise("High").alias("Salary\_Grade")

)).drop("id","gender","salary")

updatedDF.printSchema()

updatedDF.show(truncate=False)

Here, it copies “gender“, “salary” and “id” to the new struct “otherInfo” and add’s a new column “Salary\_Grade“.

root

|-- name: struct (nullable = true)

| |-- firstname: string (nullable = true)

| |-- middlename: string (nullable = true)

| |-- lastname: string (nullable = true)

|-- OtherInfo: struct (nullable = false)

| |-- identifier: string (nullable = true)

| |-- gender: string (nullable = true)

| |-- salary: integer (nullable = true)

| |-- Salary\_Grade: string (nullable = false)

**6. Using SQL ArrayType and MapType**

SQL StructType also supports ArrayType and MapType to define the DataFrame columns for array and map collections respectively. On the below example, column hobbies defined as ArrayType(StringType) and properties defined as MapType(StringType,StringType) meaning both key and value as String.

arrayStructureSchema = StructType([

StructField('name', StructType([

StructField('firstname', StringType(), True),

StructField('middlename', StringType(), True),

StructField('lastname', StringType(), True)

])),

StructField('hobbies', ArrayType(StringType()), True),

StructField('properties', MapType(StringType(),StringType()), True)

])

Outputs the below schema. Note that field Hobbies is array type and properties is map type.

root

|-- name: struct (nullable = true)

| |-- firstname: string (nullable = true)

| |-- middlename: string (nullable = true)

| |-- lastname: string (nullable = true)

|-- hobbies: array (nullable = true)

| |-- element: string (containsNull = true)

|-- properties: map (nullable = true)

| |-- key: string

| |-- value: string (valueContainsNull = true)

**7. Creating StructType object struct from JSON file**

If you have too many columns and the structure of the DataFrame changes now and then, it’s a good practice to load the SQL StructType schema from JSON file. You can get the schema by using df2.schema.json() , store this in a file and will use it to create a the schema from this file.

print(df2.schema.json())

{

"type" : "struct",

"fields" : [ {

"name" : "name",

"type" : {

"type" : "struct",

"fields" : [ {

"name" : "firstname",

"type" : "string",

"nullable" : true,

"metadata" : { }

}, {

"name" : "middlename",

"type" : "string",

"nullable" : true,

"metadata" : { }

}, {

"name" : "lastname",

"type" : "string",

"nullable" : true,

"metadata" : { }

} ]

},

"nullable" : true,

"metadata" : { }

}, {

"name" : "dob",

"type" : "string",

"nullable" : true,

"metadata" : { }

}, {

"name" : "gender",

"type" : "string",

"nullable" : true,

"metadata" : { }

}, {

"name" : "salary",

"type" : "integer",

"nullable" : true,

"metadata" : { }

} ]

}

Alternatively, you could also use df.schema.simpleString(), this will return an relatively simpler schema format.

Now let’s load the json file and use it to create a DataFrame.

import json

schemaFromJson = StructType.fromJson(json.loads(schema.json))

df3 = spark.createDataFrame(

spark.sparkContext.parallelize(structureData),schemaFromJson)

df3.printSchema()

This prints the same output as the previous section. You can also, have a name, type, and flag for nullable in a comma-separated file and we can use these to create a StructType programmatically, I will leave this to you to explore.

**8. Creating StructType object struct from DDL String**

Like loading structure from JSON string, we can also create it from DLL ( by using fromDDL() static function on SQL StructType class StructType.fromDDL). You can also generate DDL from a schema using toDDL(). printTreeString() on struct object prints the schema similar to printSchemafunction returns.

ddlSchemaStr = "`fullName` STRUCT<`first`: STRING, `last`: STRING,

`middle`: STRING>,`age` INT,`gender` STRING"

ddlSchema = StructType.fromDDL(ddlSchemaStr)

ddlSchema.printTreeString()

**9. Checking if a Column Exists in a DataFrame**

If you want to perform some checks on metadata of the DataFrame, for example, if a column or field exists in a DataFrame or data type of column; we can easily do this using several functions on SQL StructType and StructField.

print(df.schema.fieldNames.contains("firstname"))

print(df.schema.contains(StructField("firstname",StringType,true)))

This example returns “true” for both scenarios. And for the second one if you have IntegerType instead of StringType it returns false as the datatype for first name column is String, as it checks every property in a field. Similarly, you can also check if two schemas are equal and more.

**10. Complete Example of PySpark StructType & StructField**

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType,StructField, StringType, IntegerType,ArrayType,MapType

from pyspark.sql.functions import col,struct,when

data = [("James","","Smith","36636","M",3000),

("Michael","Rose","","40288","M",4000),

("Robert","","Williams","42114","M",4000),

("Maria","Anne","Jones","39192","F",4000),

("Jen","Mary","Brown","","F",-1)

]

schema = StructType([

StructField("firstname",StringType(),True),

StructField("middlename",StringType(),True),

StructField("lastname",StringType(),True),

StructField("id", StringType(), True),

StructField("gender", StringType(), True),

StructField("salary", IntegerType(), True)

])

df = spark.createDataFrame(data=data,schema=schema)

df.printSchema()

df.show(truncate=False)

structureData = [

(("James","","Smith"),"36636","M",3100),

(("Michael","Rose",""),"40288","M",4300),

(("Robert","","Williams"),"42114","M",1400),

(("Maria","Anne","Jones"),"39192","F",5500),

(("Jen","Mary","Brown"),"","F",-1)

]

structureSchema = StructType([

StructField('name', StructType([

StructField('firstname', StringType(), True),

StructField('middlename', StringType(), True),

StructField('lastname', StringType(), True)

])),

StructField('id', StringType(), True),

StructField('gender', StringType(), True),

StructField('salary', IntegerType(), True)

])

df2 = spark.createDataFrame(data=structureData,schema=structureSchema)

df2.printSchema()

df2.show(truncate=False)

updatedDF = df2.withColumn("OtherInfo",

struct(col("id").alias("identifier"),

col("gender").alias("gender"),

col("salary").alias("salary"),

when(col("salary").cast(IntegerType()) < 2000,"Low")

.when(col("salary").cast(IntegerType()) < 4000,"Medium")

.otherwise("High").alias("Salary\_Grade")

)).drop("id","gender","salary")

updatedDF.printSchema()

updatedDF.show(truncate=False)

""" Array & Map"""

arrayStructureSchema = StructType([

StructField('name', StructType([

StructField('firstname', StringType(), True),

StructField('middlename', StringType(), True),

StructField('lastname', StringType(), True)

])),

StructField('hobbies', ArrayType(StringType()), True),

StructField('properties', MapType(StringType(),StringType()), True)

])

T