## **Create DataFrame from RDD**

### **1.1 Using toDF() function**

| columns = ["language","users\_count"]  data = [("Java", "20000"), ("Python", "100000"), ("Scala", "3000")] |
| --- |

PySpark RDD’s toDF() method is used to create a DataFrame from the existing RDD. Since RDD doesn’t have columns, the DataFrame is created with default column names “\_1” and “\_2” as we have two columns.

| rdd = spark.sparkContext.parallelize(data)  dfFromRDD1 = rdd.toDF()  dfFromRDD1.printSchema() |
| --- |

root

|-- \_1: string (nullable = true)

|-- \_2: string (nullable = true)

If you want to provide column names to the DataFrame use the toDF() method with column names as arguments as shown below.

| columns = ["language","users\_count"]  dfFromRDD1 = rdd.toDF(columns)  dfFromRDD1.printSchema() |
| --- |

root

|-- language: string (nullable = true)

|-- users: string (nullable = true)

| dfFromRDD1.show() |
| --- |

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

|language|users\_count|

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

| Java| 20000|

| Python| 100000|

| Scala| 3000|

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

# **PySpark StructType & StructField Explained with Examples**

| 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() |
| --- |

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 |

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

## **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.

In 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) |
| --- |

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 |

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

## **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.

| 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() |
| --- |

Here, it copies “gender“, “salary” and “id” to the new struct “otherInfo” and adds 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)

| updatedDF.show(truncate=False) |
| --- |

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

|name |OtherInfo |

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

|{James, , Smith} |{36636, M, 3100, Medium}|

|{Michael, Rose, } |{40288, M, 4300, High} |

|{Robert, , Williams}|{42114, M, 1400, Low} |

|{Maria, Anne, Jones}|{39192, F, 5500, High} |

|{Jen, Mary, Brown} |{, F, -1, Low} |

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

### **Using createDataFrame()**

Using createDataFrame() from SparkSession is another way to create manually and it takes rdd object as an argument. and chain with toDF() to specify name to the columns.

| dfFromRDD2 = spark.createDataFrame(rdd).toDF(\*columns)  dfFromRDD2.show() |
| --- |

### **Using createDataFrame() from SparkSession**

Calling createDataFrame() from SparkSession is another way to create PySpark DataFrame manually, it takes a list object as an argument. and chain with toDF() to specify names to the columns.

| dfFromData2 = spark.createDataFrame(data).toDF(\*columns) |
| --- |