# **Union**

# union() and unionAll() transformations are used to merge two or more DataFrame’s of the same schema or structure.

Dataframe union() – union() method of the DataFrame is used to merge two DataFrames of the same structure/schema. If schemas are not the same it returns an error.

DataFrame unionAll() – unionAll() is deprecated since Spark “2.0.0” version and replaced with union().

Note: In other SQL languages, Union eliminates the duplicates but UnionAll merges two datasets including duplicate records. But, in PySpark both behave the same and recommend using DataFrame duplicate() function to remove duplicate rows.

First DataFrame

| import pyspark  from pyspark.sql import SparkSession  spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()  simpleData = [("James","Sales","NY",90000,34,10000), \  ("Michael","Sales","NY",86000,56,20000), \  ("Robert","Sales","CA",81000,30,23000), \  ("Maria","Finance","CA",90000,24,23000) \  ]  columns= ["employee\_name","department","state","salary","age","bonus"]  df = spark.createDataFrame(data = simpleData, schema = columns)  df.printSchema()  df.show(truncate=False) |
| --- |

Second DataFrame

Now, let’s create a second Dataframe with the new records and some records from the above Dataframe but with the same schema.

| simpleData2 = [("James","Sales","NY",90000,34,10000), \  ("Maria","Finance","CA",90000,24,23000), \  ("Jen","Finance","NY",79000,53,15000), \  ("Jeff","Marketing","CA",80000,25,18000), \  ("Kumar","Marketing","NY",91000,50,21000) \  ]  columns2= ["employee\_name","department","state","salary","age","bonus"]  df2 = spark.createDataFrame(data = simpleData2, schema = columns2)  df2.printSchema()  df2.show(truncate=False) |
| --- |

## **Merge two or more DataFrames using union**

DataFrame union() method merges two DataFrames and returns the new DataFrame with all rows from two Dataframes regardless of duplicate data.

| unionDF = df.union(df2)  unionDF.show(truncate=False) |
| --- |

## **Merge without Duplicates**

Since the union() method returns all rows without distinct records, we will use the distinct() function to return just one record when duplicate exists.

| disDF = df.union(df2).distinct()  disDF.show(truncate=False) |
| --- |

## **Loop Through Rows in DataFrame**

PySpark map (map()) is an RDD transformation that is used to apply the transformation function (lambda) on every element of RDD/DataFrame and returns a new RDD. In this article, you will learn the syntax and usage of the RDD map() transformation with an example and how to use it with DataFrame.

* Note1: DataFrame doesn’t have map() transformation to use with DataFrame hence you need to DataFrame to RDD first.
* Note2: If you have a heavy initialization use PySpark mapPartitions() transformation instead of map(), as with mapPartitions() heavy initialization executes only once for each partition instead of every record.

| from pyspark.sql import SparkSession  spark = SparkSession.builder.master("local[1]") \  .appName("SparkByExamples.com").getOrCreate()  data = ["Project","Gutenberg’s","Alice’s","Adventures",  "in","Wonderland","Project","Gutenberg’s","Adventures",  "in","Wonderland","Project","Gutenberg’s"]  rdd=spark.sparkContext.parallelize(data) |
| --- |

| rdd2=rdd.map(lambda x: (x,1))  for element in rdd2.collect():  print(element) |
| --- |

## **map() Example with DataFrame**

PySpark DataFrame doesn’t have map() transformation to apply the lambda function, when you wanted to apply the custom transformation, you need to convert the DataFrame to RDD and apply the map() transformation.

| data = [('James','Smith','M',30),  ('Anna','Rose','F',41),  ('Robert','Williams','M',62),  ]  columns = ["firstname","lastname","gender","salary"]  df = spark.createDataFrame(data=data, schema = columns)  df.show() |
| --- |

| # Refering columns by index.  rdd2=df.rdd.map(lambda x:  (x[0]+","+x[1],x[2],x[3]\*2)  )  df2=rdd2.toDF(["name","gender","new\_salary"] )  df2.show() |
| --- |

Note that aboveI have used index to get the column values, alternatively, you can also refer to the DataFrame column names while iterating.

| # Referring Column Names  rdd2=df.rdd.map(lambda x:  (x["firstname"]+","+x["lastname"],x["gender"],x["salary"]\*2)  )  rdd2.collect() |
| --- |

| # Referring Column Names  rdd2=df.rdd.map(lambda x:  (x.firstname+","+x.lastname,x.gender,x.salary\*2)  )  rdd2.collect() |
| --- |

You can also create a custom function to perform an operation. Below func1() function executes for every DataFrame row from the lambda function.

| # By Calling function  def func1(x):  firstName=x.firstname  lastName=x.lastname  name=firstName+","+lastName  gender=x.gender.lower()  salary=x.salary\*2  return (name,gender,salary)  rdd2=df.rdd.map(lambda x: func1(x))  rdd2.collect() |
| --- |

## **Using foreach() to Loop Through Rows in DataFrame**

Similar to map(), foreach() also applied to every row of DataFrame, the difference being foreach() is an action and it returns nothing. Below are some examples to iterate through DataFrame using for each.

| # Foreach example  def f(x): print(x)  df.foreach(f)  # Another example  df.foreach(lambda x:  print("Data ==>"+x["firstname"]+","+x["lastname"]+","+x["gender"]+","+str(x["salary"]\*2))  ) |
| --- |

## **Using pandas() to Iterate**

If you have a small dataset, you can also Convert PySpark DataFrame to Pandas and use pandas to iterate through.

| # Using pandas  import pandas as pd  pandasDF = df.toPandas()  for index, row in pandasDF.iterrows():  print(row['firstname'], row['gender']) |
| --- |

## **Collect Data As List and Loop Through**

You can also Collect the PySpark DataFrame to Driver and iterate through Python, you can also use toLocalIterator().

| # Collect the data to Python List  dataCollect = df.collect()  for row in dataCollect:  print(row['firstname'] + "," +row['lastname'])  #Using toLocalIterator()  dataCollect=df.rdd.toLocalIterator()  for row in dataCollect:  print(row['firstname'] + "," +row['lastname']) |
| --- |

## **sample()**

PySpark sampling (pyspark.sql.DataFrame.sample()) is a mechanism to get random sample records from the dataset, this is helpful when you have a larger dataset and wanted to analyze/test a subset of the data for example 10% of the original file.

fraction – Fraction of rows to generate, range [0.0, 1.0].

| df=spark.range(100)  print(df.sample(0.5).collect()) |
| --- |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In this article, I will use both fill() and fillna() to replace null/none values with an empty string, constant value, and zero(0) on Dataframe columns integer, string with Python examples.

| id,zipcode,type,city,state,population  1,704,STANDARD,,PR,30100  2,704,,PASEO COSTA DEL SUR,PR,  3,709,,BDA SAN LUIS,PR,3700  4,76166,UNIQUE,CINGULAR WIRELESS,TX,84000  5,76177,STANDARD,,TX, |
| --- |

| df = spark.read.options(header='true', inferSchema='true') \  .csv("file:///home/zidane/small\_zipcode.csv")  df.printSchema()  df.show(truncate=False) |
| --- |

fillna(value, subset=None)

fill(value, subset=None)

* value – Value should be the data type of int, long, float, string, or dict. Value specified here will be replaced for NULL/None values.
* subset – This is optional, when used it should be the subset of the column names where you wanted to replace NULL/None values.

## **Replace NULL/None Values with Zero (0)**

PySpark fill(value:Long) signatures that are available in DataFrameNaFunctions is used to replace NULL/None values with numeric values either zero(0) or any constant value for all integer and long datatype columns of PySpark DataFrame or Dataset.

| #Replace 0 for null for all integer columns  df.na.fill(value=0).show()  #Replace 0 for null on only population column  df.na.fill(value=0,subset=["population"]).show() |
| --- |

Above both statements yields the same output, since we have just an integer column population with null values Note that it replaces only Integer columns since our value is 0.

## **Replace Null/None Value with Empty String**

Now let’s see how to replace NULL/None values with an empty string or any constant values String on all DataFrame String columns.

| df.na.fill("").show() |
| --- |

Now, let’s replace NULL’s on specific columns, below example replace column type with empty string and column city with value “unknown”.

| df.na.fill("unknown",["city"]) \  .na.fill("",["type"]).show() |
| --- |

Alternatively you can also write the above statement as

| df.na.fill({"city": "unknown", "type": ""}) \  .show() |
| --- |