## Transformations and Actions in PySpark

In PySpark, **Transformations** and **Actions** are two key types of operations performed on Resilient Distributed Datasets (RDDs) or DataFrames.

### 1. Transformations

- **Definition**: Transformations create a new RDD/DataFrame from an existing one. They are *lazy*, meaning they do not execute immediately but are recorded as part of a lineage graph to be executed when an action is called.
- **Characteristics**:
  - Lazy evaluation
  - Return a new RDD/DataFrame
  - Do not modify the original RDD/DataFrame
- **Examples**:
  - **Map**: Applies a function to each element of the RDD/DataFrame.
    Example: rdd.map(lambda x: x * 2)
  - **Filter**: Selects elements that meet a condition.
    Example: rdd.filter(lambda x: x > 10)
  - **FlatMap**: Similar to map, but can return multiple values for each input.
    Example: rdd.flatMap(lambda x: x.split(" "))
  - **GroupByKey**: Groups elements by key. Example: rdd.groupByKey()
  - **Join**: Joins two datasets based on keys. Example: rdd1.join(rdd2)

### 2. Actions

- **Definition**: Actions trigger the computation of transformations and return results to the driver or write data to an external storage system.
- **Characteristics**:
  - Trigger execution of the lineage graph
  - Produce a result or side-effect
- **Examples**:
  - **Collect**: Brings all elements of the RDD/DataFrame to the driver. Example: rdd.collect()
  - **Count**: Counts the number of elements. Example: rdd.count()
  - **First**: Returns the first element. Example: rdd.first()
  - **Take**: Retrieves a specified number of elements. Example: rdd.take(5)
  - **Reduce**: Aggregates elements using a function.
    Example: rdd.reduce(lambda x, y: x + y)
  - **SaveAsTextFile**: Writes data to a text file.
    Example: rdd.saveAsTextFile("output_path")

## Pair RDD Transformations and Actions in PySpark

**Pair RDDs** are RDDs consisting of key-value pairs, which allow operations that take advantage of the key-value structure. Below is a summary of **transformations** and **actions** specific to pair RDDs.

### Transformations on Pair RDDs

Pair RDD transformations leverage the key-value nature for grouped and aggregated operations.

| Transformation | Description | Example |
|---|---|---|
| reduceByKey | Combines values for each key using a function. | rdd.reduceByKey(lambda x, y: x + y) |
| groupByKey | Groups values by key into a list. | rdd.groupByKey() |
| mapValues | Applies a function to each value for each key. | rdd.mapValues(lambda x: x * 2) |
| flatMapValues | Similar to mapValues, but returns multiple values for each key. | rdd.flatMapValues(lambda x: range(x)) |
| keys | Extracts only the keys from the pair RDD. | rdd.keys() |
| values | Extracts only the values from the pair RDD. | rdd.values() |
| sortByKey | Sorts the RDD by keys. | rdd.sortByKey() |
| join | Joins two pair RDDs by their keys. | rdd1.join(rdd2) |
| cogroup | Groups data from two RDDs by their keys. | rdd1.cogroup(rdd2) |

### Actions on Pair RDDs

These actions trigger computations and return results or save the output.

| Action | Description | Example |
|---|---|---|
| countByKey | Counts the number of elements for each key. | rdd.countByKey() |
| collectAsMap | Returns the key-value pairs as a dictionary. | rdd.collectAsMap() |
| lookup | Returns all values associated with a specified key. | rdd.lookup(key) |

## Select, Rename, and Filter Data in a PySpark DataFrame

### 1. Select Columns

- **Purpose**: Extract specific columns from a DataFrame.
- **Methods**:
    - select: Select one or more columns by name.
    - selectExpr: Use SQL expressions to create or modify columns.

```
# Select specific columns
df.select("col1", "col2").show()
# Use expressions
df.selectExpr("col1", "col2 * 2 AS col2_double").show()
```

### 2. Rename Columns

- **Purpose**: Rename one or more columns in the DataFrame.
- **Methods**:
    - withColumnRenamed: Rename a single column.

```
# Rename a single column
df_renamed = df.withColumnRenamed("old_col_name", "new_col_name")
df_renamed.show()
```

### 3. Filter Rows

- **Purpose**: Filter rows based on conditions.
- **Methods**:
    - filter or where: Accept SQL-like expressions or column-based conditions.

```
# Filter using a condition
df_filtered = df.filter(df["col1"] > 10)
# Filter with multiple conditions
df_filtered = df.filter((df["col1"] > 10) & (df["col2"] == "value"))
# Using SQL-like syntax
df_filtered = df.filter("col1 > 10 AND col2 = 'value'")
```

## Views and Temporary Views in PySpark

**1. View**

- A **view** is a logical representation of a DataFrame.
- Useful for querying data using SQL syntax.
- Created using the createOrReplaceTempView() or createOrReplaceGlobalTempView() methods.

**2. Temporary View**

- **Temporary View (createOrReplaceTempView)**:
  - The view is tied to the current session.
  - Automatically removed when the session ends.
  - Accessible only in the session that created it.

```
# Create a temporary view
df.createOrReplaceTempView("temp_view")
# Query the temporary view using SQL
spark.sql("SELECT * FROM temp_view WHERE column > 10").show()
```

**3. Global Temporary View**

- **Global Temporary View (createOrReplaceGlobalTempView)**:
  - The view is accessible across multiple sessions.
  - Prefixed with the database name global_temp.
  - Removed only when the Spark application terminates.

```
# Create a global temporary view
df.createOrReplaceGlobalTempView("global_temp_view")
# Query the global temporary view
spark.sql("SELECT * FROM global_temp.global_temp_view").show()
```

```
11:10 AM (<1s)                                                    5

#to create rdds and  dataframe
from pyspark import SparkContext
from pyspark.sql import  SparkSession

sc =SparkContext.getOrCreate()
spark = SparkSession.builder.appName('pyspark first program').getOrCreate()

#create the rdd

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)], 4)
mydata = ['Division','English','Mathematics','Physics','Chemistry']
marks_df = spark.createDataFrame(rdd, schema=mydata)

print(rdd.collect())
print("\n")
print(rdd) #---Transformation which gives rdd value
print("\n")
rdd.collect() #----Action gives non rdd value

(3) Spark Jobs
marks_df:  pyspark.sql.dataframe.DataFrame = [Division: string, English: long ... 3 more fields]
[('C', 85, 76, 87, 91), ('B', 85, 76, 87, 91), ('A', 85, 78, 96, 92), ('A', 92, 76, 89, 96)]


ParallelCollectionRDD[158] at readRDDFromInputStream at PythonRDD.scala:435


Out[44]: [('C', 85, 76, 87, 91),
 ('B', 85, 76, 87, 91),
 ('A', 85, 78, 96, 92),
 ('A', 92, 76, 89, 96)]
```

collect()

## first()

It is an action that returns the first element from an RDD or DataFrame.

```
11:10 AM (<1s)                                                    7

first_rdd = sc.parallelize([1,2,3,4,5,5,6,7,8,9])
first_rdd.first() #First method is action

(1) Spark Jobs
Out[45]: 1
```

## count()

It is an action that returns the total number of elements in an RDD or DataFrame.

```
11:10 AM (<1s)                                                    9

rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)], 4)
mydata = ['Division','English','Mathematics','Physics','Chemistry']
marks_df = spark.createDataFrame(rdd, schema=mydata)
print(rdd.count())

(2) Spark Jobs
marks_df:  pyspark.sql.dataframe.DataFrame = [Division: string, English: long ... 3 more fields]
4
```

## take(n)

It is an action that returns the first n elements from an RDD or DataFrame as a list.

11:10 AM (<1s) 12

```
rdd = sc.parallelize([('C',85,76,87,91), ('B',85,76,87,91), ("A", 85,78,96,92), ("A", 92,76,89,96)], 4)
mydata = ['Division','English','Mathematics','Physics','Chemistry']
marks_df = spark.createDataFrame(rdd, schema=mydata)
rdd.take(2)
```

(3) Spark Jobs

marks_df: pyspark.sql.dataframe.DataFrame = [Division: string, English: long ... 3 more fields]

```
Out[48]: [('C', 85, 76, 87, 91), ('B', 85, 76, 87, 91)]
```

## 1. map()

It is a transformation that applies a function to each element of an RDD or DataFrame, creating a new RDD or DataFrame with the transformed elements.

11:10 AM (1s) 17

```
from pyspark import SparkContext
sc = SparkContext.getOrCreate()
map_rdd = sc.parallelize([1,2,3])
print(map_rdd.map(lambda x:x+10))
print(map_rdd.map(lambda x:x+10).collect())
```

(1) Spark Jobs

```
PythonRDD[193] at RDD at PythonRDD.scala:58
[11, 12, 13]
```

## 2. reduce()

It is an action that aggregates elements of an RDD or DataFrame into a single value.

11:10 AM (<1s) 19

```
reduce_rdd = sc.parallelize([1,2,3])
print(reduce_rdd.reduce(lambda x,y:x+y))
```

(1) Spark Jobs

```
6
```

### 3. filter()

It is a transformation that creates a new RDD or DataFrame containing only the elements that satisfy a given condition.

```
11:10 AM (<1s)                                                    21

filter_rdd = sc.parallelize([1,2,3])
print(filter_rdd.filter(lambda x:x%2==0))
print(filter_rdd.filter(lambda x:x%2==0).collect())
```

(1) Spark Jobs

```
PythonRDD[198] at RDD at PythonRDD.scala:58
[2]
```

### 3.a startswith('')

It is a function used to check if a string or column starts with a specific prefix. It returns a boolean value: True if the string starts with the prefix, False otherwise.

```
11:10 AM (<1s)                                                    23

from pyspark import SparkContext
# the code filters the filter_rdd_2 RDD to keep only strings starting with 'R'
# and then prints the filtered elements to the console.
sc = SparkContext.getOrCreate()
filter_rdd_2 = sc.parallelize(['Rahul', 'Swati', 'Rohan', 'Shreya', 'Priya'])
print(filter_rdd_2.filter(lambda x: x.startswith('R')).collect())
```

(1) Spark Jobs

```
['Rahul', 'Rohan']
```

## Pair RDD

### collect() - action

```
05:20 PM (<1s)                                                    33

marks = [('Rahul', 88), ('Swati', 92), ('Shreya', 83), ('Abhay', 93), ('Rohan', 78)]
sc.parallelize(marks).collect()
```

(1) Spark Jobs

```
Out[7]: [('Rahul', 88), ('Swati', 92), ('Shreya', 83), ('Abhay', 93), ('Rohan', 78)]
```

### reduceByey() transformation

```
05:22 PM (2s)                                                     35

marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Shreya', 22), ('Abhay', 29), ('Rohan', 22),
('Rahul', 23), ('Swati', 19), ('Shreya', 28), ('Abhay', 26), ('Rohan', 22)])
print(marks_rdd.reduceByKey(lambda x, y: x + y).collect())
```

(1) Spark Jobs

```
[('Shreya', 50), ('Swati', 45), ('Rahul', 48), ('Abhay', 55), ('Rohan', 44)]
```

# Select, Rename, Filter Data in a Pandas DF

## withColumnRenamed()

04:32 PM (11s) 44

```
# Importing necessary libraries
from pyspark.sql import SparkSession

# Create a spark session
spark = SparkSession.builder.appName('pyspark - example join').getOrCreate()

# Create data in dataframe
data = [(('Ram'), '1991-04-01', 'M', 3000),
        (('Mike'), '2000-05-19', 'M', 4000),
        (('Rohini'), '1978-09-05', 'M', 4000),
        (('Maria'), '1967-12-01', 'F', 4000),
        (('Jenis'), '1980-02-17', 'F', 1200)]

# Column names in dataframe
columns = ["Name", "DOB", "Gender", "salary"]

# Create the spark dataframe
df = spark.createDataFrame(data=data,
                           schema=columns)
df.withColumnRenamed("DOB","date of birth").show()
df.withColumnRenamed("DOB","date of birth").withColumnRenamed("Name","personname").show()
```

(6) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [Name: string, DOB: string ... 2 more fields]

```
+------+-------------+------+------+
|  Name|date of birth|Gender|salary|
+------+-------------+------+------+
|   Ram|   1991-04-01|     M|  3000|
|  Mike|   2000-05-19|     M|  4000|
|Rohini|   1978-09-05|     M|  4000|
| Maria|   1967-12-01|     F|  4000|
| Jenis|   1980-02-17|     F|  1200|
+------+-------------+------+------+

+----------+-------------+------+------+
|personname|date of birth|Gender|salary|
+----------+-------------+------+------+
|       Ram|   1991-04-01|     M|  3000|
|      Mike|   2000-05-19|     M|  4000|
|    Rohini|   1978-09-05|     M|  4000|
|     Maria|   1967-12-01|     F|  4000|
|     Jenis|   1980-02-17|     F|  1200|
+----------+-------------+------+------+
```

## alias() for column

04:34 PM (1s)

```
from pyspark.sql.functions import col

# Select the 'salary' as 'Amount' using aliasing
# Select remaining with their original name
data = df.select(col("Name"),col("DOB"),
                 col("Gender"),
                 col("salary").alias('Amount'))
data.show()
```

(3) Spark Jobs

data: pyspark.sql.dataframe.DataFrame = [Name: string, DOB: string ... 2 more fields]

```
+------+----------+------+------+
|  Name|       DOB|Gender|Amount|
+------+----------+------+------+
|   Ram|1991-04-01|     M|  3000|
|  Mike|2000-05-19|     M|  4000|
|Rohini|1978-09-05|     M|  4000|
| Maria|1967-12-01|     F|  4000|
| Jenis|1980-02-17|     F|  1200|
+------+----------+------+------+
```

## create temporary view

04:35 PM (2s)

```
from pyspark.sql import SparkSession
# Create spark session
spark = SparkSession \
.builder \
.appName("SparkByExamples.com") \
.enableHiveSupport() \
.getOrCreate()
data = [("James","Smith","USA","CA"),
("Michael","Rose","USA","NY"),
("Robert","Williams","USA","CA"),
("Maria","Jones","USA","FL")
]
columns = ["firstname","lastname","country","state"]
# Create dataframe
sampleDF = spark.sparkContext.parallelize(data).toDF(columns)
sampleDF.createOrReplaceTempView("Person")
sampleDF.createOrReplaceTempView("mydata")
sampleDF.show()
```

(5) Spark Jobs

sampleDF: pyspark.sql.dataframe.DataFrame = [firstname: string, lastname: string ... 2 more fields]

```
+---------+--------+-------+-----+
|firstname|lastname|country|state|
+---------+--------+-------+-----+
|    James|   Smith|    USA|   CA|
|  Michael|    Rose|    USA|   NY|
|   Robert|Williams|    USA|   CA|
|    Maria|   Jones|    USA|   FL|
+---------+--------+-------+-----+
```

```
spark.sql("select * from person").show()
spark.sql("select * from mydata").show()
```

(6) Spark Jobs

```
+---------+--------+-------+-----+
|firstname|lastname|country|state|
+---------+--------+-------+-----+
|    James|   Smith|    USA|   CA|
|  Michael|    Rose|    USA|   NY|
|   Robert|Williams|    USA|   CA|
|    Maria|   Jones|    USA|   FL|
+---------+--------+-------+-----+
```