groupByKey( ) Transformation

By definition, When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable) pairs.

To group the values for each key in the RDD into a single sequence.

In [1]:

```python
from pyspark.sql import SparkSession

spark = SparkSession \
        .builder \
        .master("local[*]") \
        .appName("groupByKey Transformation") \
        .enableHiveSupport() \
        .getOrCreate()

#path of the data file on the local machine
data_file = '/Users/vaishaliyasala/Desktop/Github/Spark/Exercise_Dependencies/sales_data.csv'

#Read the csv into a dataframe
df = spark.read.csv(data_file, header = True, )

df1 = df.select(df["InvoiceNo"],df["UnitPrice"],df["Quantity"]).repartition(4)

print(df1.printSchema())

#Creating view of the dataframe of with 3 required columns and sample of 3% of data
sample_df = df1.sample(0.03,25)

sample_df.show()
```

```
22/10/13 18:02:30 WARN Utils: Your hostname, Vaishalis-MacBook-Pro.local resolves to a loopback address: 127.0.
0.1; using 192.168.0.105 instead (on interface en0)
22/10/13 18:02:30 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/10/13 18:02:30 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builti
n-java classes where applicable
22/10/13 18:02:31 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
22/10/13 18:02:31 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
22/10/13 18:02:31 WARN Utils: Service 'SparkUI' could not bind on port 4042. Attempting port 4043.

root
 |-- InvoiceNo: string (nullable = true)
 |-- UnitPrice: string (nullable = true)
 |-- Quantity: string (nullable = true)

None
+---------+---------+--------+
|InvoiceNo|UnitPrice|Quantity|
+---------+---------+--------+
|   536409|     0.85|       6|
|   536500|     3.95|       2|
|   536384|     2.95|       6|
|   536381|     2.95|       1|
|   536398|     3.95|       4|
|   536409|     2.95|       1|
|   536406|     2.55|       8|
|   536415|     1.25|       2|
|   536375|     3.39|       6|
|   536395|     3.75|       8|
|   536477|      2.1|      48|
|   536415|      2.1|       5|
|   536378|     0.55|      24|
|   536415|     1.65|       3|
|   536396|     4.95|       2|
|   536409|      2.1|       1|
|   536408|     5.95|       3|
|   536389|     4.95|       8|
|   536415|     1.65|       5|
|   536378|     1.65|      10|
+---------+---------+--------+
only showing top 20 rows
```

In [2]:

```python
# apply a map() transformation to rdd to create (K, V) pairs

#In this key-value pair, key is the InvoiceN0 and the tuple (UnitPrice, Quantity) is the value

rdd1 = sample_df.rdd.map(lambda x : (x[0],(x[1],x[2])))
print("Number of elements =",len(rdd1.collect()))
print("Number of Partitions =",rdd1.getNumPartitions())
```

```
Number of elements = 24
Number of Partitions = 4
```

In [3]:

```python
# apply groupByKey() transformation to rdd2 to return a dataset of (K, Iterable<V>) pairs.

rdd2 =rdd1.groupByKey()
print("Number of elements =",len(rdd2.collect()))
print("Number of Partitions =",rdd2.getNumPartitions())
print("----------")

print(rdd2.take(5))
```

```
Number of elements = 16
Number of Partitions = 4
----------
[('536409', <pyspark.resultiterable.ResultIterable object at 0x1060878b0>), ('536406', <pyspark.resultiterable.
ResultIterable object at 0x1061f2290>), ('536389', <pyspark.resultiterable.ResultIterable object at 0x1061f2b30
>), ('536398', <pyspark.resultiterable.ResultIterable object at 0x1061f28f0>), ('536375', <pyspark.resultiterab
le.ResultIterable object at 0x1061f2950>)]
```

From result of Input block 2 and 3, we can see the number of elements decreased because they are grouped together when they have the same key.

In [4]:

```python
# RDD.mapValues - Pass each value in the key-value pair RDD through a map function without changing the keys
# mapValues operates on the values only
# this also changes the original RDD's partitioning.

print("rdd2.mapValues().collect() = ", rdd2.mapValues(lambda values: list(values)).take(10))
```

```
rdd2.mapValues().collect() =  [('536409', [('0.85', '6'), ('2.95', '1'), ('2.1', '1')]), ('536406', [('2.55',
'8')]), ('536389', [('4.95', '8')]), ('536398', [('3.95', '4')]), ('536375', [('3.39', '6'), ('6.95', '4'),
('2.55', '6')]), ('536477', [('2.1', '48')]), ('536378', [('0.55', '24'), ('1.65', '10')]), ('536396', [('4.9
5', '2')]), ('536408', [('5.95', '3')]), ('536382', [('0.85', '12')])]
```

groupByKey() Transformation cannot be used on large datasets.

Because all the elements from each partition will be sent over the network to the Task performing the reduce operation. Since the data is not combined or reduced on the map side, we transferred all the elements over the network during shuffle. Since all elements are sent to the task performing the aggregate operation, the number of elements to be handled by the task will be more and could possibly result in an Out of Memory exception.