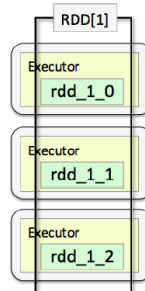


# PARTITIONING AND BUCKETING IN PYSPARK

## LOGICAL PARTITIONING ON EXECUTOR

### Data Partitioning

- Data in Datasets and DataFrames is managed by underlying RDDs
- Data in an RDD is partitioned across executors
  - This is what makes RDDs distributed
  - Spark assigns tasks to process a partition to the executor managing that partition
- Data Partitioning is done automatically by Spark
  - In some cases, you can control how many partitions are created
  - More partitions = more parallelism



Spark determines how to partition data in an RDD, Dataset, or DataFrame when

- The data source is read
- An operation is performed on a DataFrame, Dataset, or RDD
- Spark optimizes a query
- You call repartition or coalesce

Catalyst optimizer manages partitioning of RDDs that implement DataFrames and Datasets.

**By Sai Kumar**

## EXECUTE THE BELOW ON DATABRICKS COMMUNITY EDITION

**Ex: Create DataFrame with default partitions**

```
empDF=spark.read.json("/FileStore/EMP.json")
```

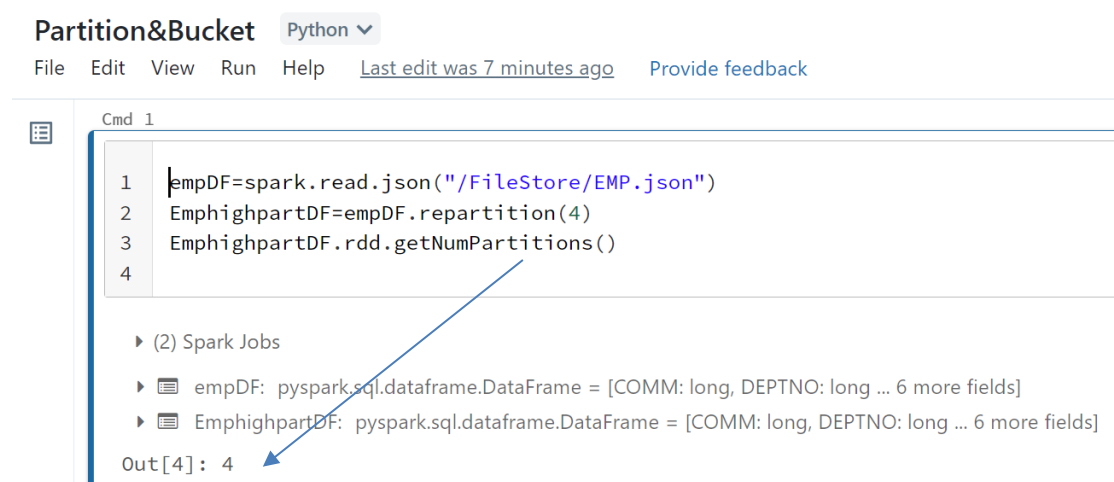
```
empDF.rdd.getNumPartitions()
```

**Create DataFrame with custom partitions**

```
empDF=spark.read.json("/FileStore/EMP.json")
```

```
EmphighpartDF=empDF.repartition(4)
```

```
EmphighpartDF.rdd.getNumPartitions()
```



The screenshot shows the Databricks IDE interface. At the top, there's a header with 'Partition&Bucket' and a 'Python' dropdown menu. Below this is a menu bar with 'File', 'Edit', 'View', 'Run', and 'Help'. A status bar indicates 'Last edit was 7 minutes ago' and a link to 'Provide feedback'. The main area is a code editor with a command prompt 'Cmd 1' at the top. It contains four lines of Python code: 1. `empDF=spark.read.json("/FileStore/EMP.json")`, 2. `EmphighpartDF=empDF.repartition(4)`, 3. `EmphighpartDF.rdd.getNumPartitions()`, and 4. (empty line). Below the code editor, there's a section for '(2) Spark Jobs' with two entries: 'empDF: pyspark.sql.dataframe.DataFrame = [COMM: long, DEPTNO: long ... 6 more fields]' and 'EmphighpartDF: pyspark.sql.dataframe.DataFrame = [COMM: long, DEPTNO: long ... 6 more fields]'. At the bottom, the output 'Out[4]: 4' is displayed, with a blue arrow pointing from the third line of code to this output.

```
Partition&Bucket Python
File Edit View Run Help Last edit was 7 minutes ago Provide feedback

Cmd 1
1 empDF=spark.read.json("/FileStore/EMP.json")
2 EmphighpartDF=empDF.repartition(4)
3 EmphighpartDF.rdd.getNumPartitions()
4

▶ (2) Spark Jobs
▶ empDF: pyspark.sql.dataframe.DataFrame = [COMM: long, DEPTNO: long ... 6 more fields]
▶ EmphighpartDF: pyspark.sql.dataframe.DataFrame = [COMM: long, DEPTNO: long ... 6 more fields]

Out[4]: 4
```

## PHYSICAL PARTITIONING ON DISK

We can physically partition based on column/s to store values in different sub-directories based on the values. For example, based on each department value we can create separate sub-directories and put the data of each department number separately. It gives performance benefit on queries and read.

**Example:**

```
empDF=spark.read.json("/FileStore/EMP.json")
```

```
empDF.write.option("header", True).partitionBy("DEPTNO") \  
.mode("overwrite").csv("EMP_DEPTNOS")
```

Go to data option and see **EMP\_DEPTNOS** directory

/EMP_DEPTNOS	
Q Prefix search	Q Prefix search
EMP_DEPTNOS	DEPTNO=10
FileStore	DEPTNO=20
user	DEPTNO=30
	_SUCCESS

PySpark **partitionBy()** with Multiple Columns:

```
empDF=spark.read.json("/FileStore/EMP.json")
empDF.write.option("header", True).partitionBy("DEPTNO",
"JOB") \
.mode("overwrite").csv("DEPTNO_JOBS")
```

/DEPTNO_JOBS/DEPTNO=10	
Q Prefix search	Q Prefix search
DEPTNO=10	JOB=CLERK
DEPTNO=20	JOB=MANAGER
DEPTNO=30	JOB=PRESIDENT
_SUCCESS	

## BUCKETING

```
empDF=spark.read.json("/FileStore/EMP.json")
empDF.write.bucketBy(2,"SALARY") .mode("overwrite")
.saveAsTable('bucketed_table')
```