By Sai Kumar

PARTITIONING AND BUCKETING IN PYSPARK

LOGICAL PARTITONING ON EXECUTOR

Data Partitioning

▪ Data in Datasets and DataFrames is managed by underlying RDDs

▪ Data in an RDD is partitioned across executors

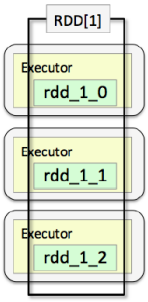
— This is what makes RDDs distributed

— 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

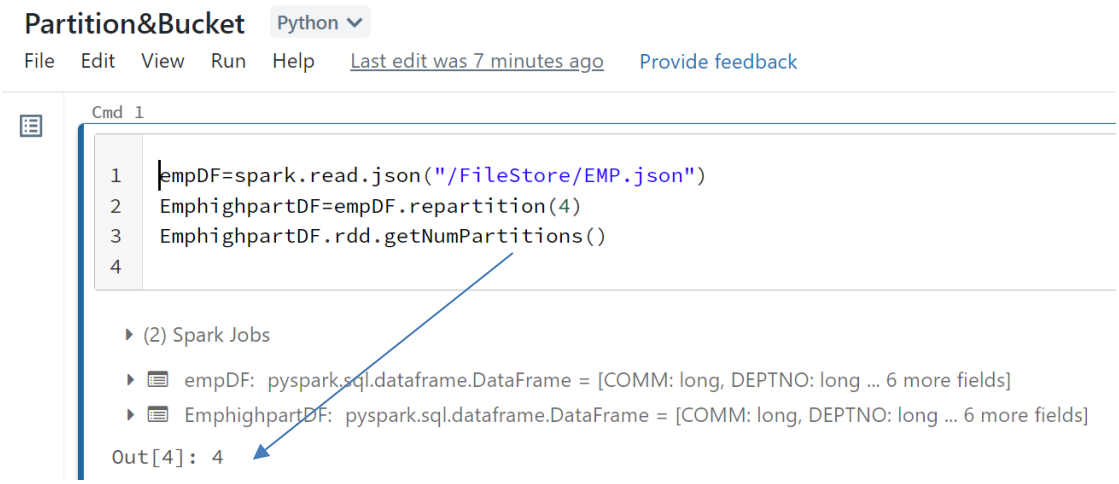
EXECUTE THE BELOW ON DATABRICKS COMMUNITY EDITION

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()
```



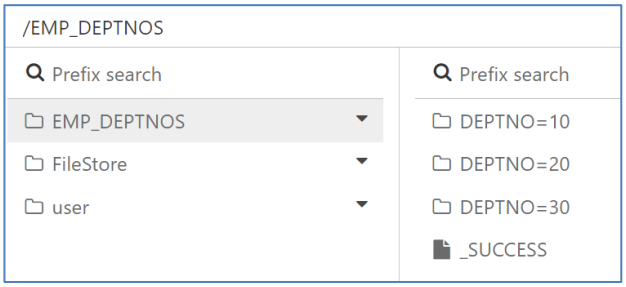
PHYSICAL PARTITIONING ON DISK

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:

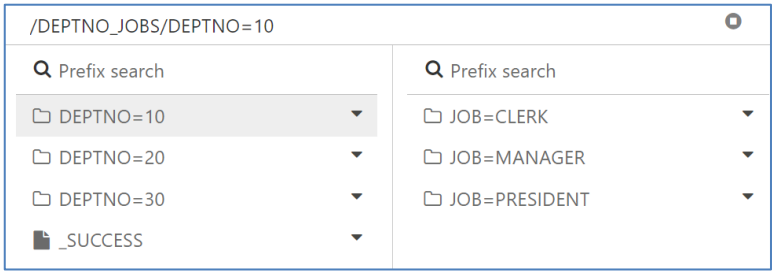
```
empDF=spark.read.json("/FileStore/EMP.json")
empDF.write.option("header", True).partitionBy("DEPTNO") \
      .mode("overwrite").csv("EMP_DEPTNOS")
```

Go to data option and see **EMP_DEPTNOS** directory



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")
```



BUCKETING

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

empDF.write.bucketBy(2,"SALARY") .mode("overwrite")
.saveAsTable('bucketed_table')
```