Day 9 - PySpark, BigData & Apache Spark

📅 Date: 17th June 2025

# Introduction to Apache Spark

Apache Spark is a fast, in-memory, distributed computing framework built for large-scale data processing. Originally developed in Scala, Spark supports multiple languages—Scala, Java, Python (via PySpark), and R.

Features:

* Open-source and cluster-computing based.
* Optimized for speed, thanks to in-memory data storage.
* Designed for scalability, capable of managing workloads across hundreds of nodes.

# What is PySpark?

PySpark is the Python API for Apache Spark, enabling developers to write Spark jobs using Python.  
  
How It Works:  
• Python code is executed locally.  
• Py4J, a gateway library, bridges Python with JVM-based Spark.  
• Actual computation is executed on Spark’s Scala engine, while Python acts as the interface.

# RDD (Resilient Distributed Dataset): Foundation of Spark

Characteristics of RDDs:

* Immutability: Once created, cannot be changed. Operations return new RDDs.
* In-memory: Intermediate results stored in RAM for better speed.
* Lazy Evaluation: Transformations aren't run until an action is triggered.
* Fault-Tolerant: Can recover lost partitions using lineage.
* Partitioned: Data is split across nodes for parallelism.
* Persistable: Frequently used RDDs can be cached.
* Granular Ops: Supports both dataset-wide and element-wise transformations.

# RDD Operations

Transformations (Lazy): map(), filter(), flatMap(), groupByKey(), sortByKey()

Types:

* • Narrow: Data remains within a single partition (e.g., map, filter)
* • Wide: Requires data shuffling across partitions (e.g., groupByKey, reduceByKey)

Actions (Trigger Execution): collect(), count(), first(), take(), reduce(), saveAsTextFile()

# Creating RDDs

1. From Local Collection:  
data = [1, 2, 3, 4, 5]  
rdd = sc.parallelize(data)

2. From External Sources:  
rdd = sc.textFile("path/to/file.txt")

3. From Existing RDDs:  
rdd2 = rdd.map(lambda x: x \* x)

# DataFrames in PySpark: The Modern Approach

DataFrames are high-level, tabular data structures offering structured schema, Catalyst optimizer, and SQL integration.

Create SparkSession:  
from pyspark.sql import SparkSession  
spark = SparkSession.builder.appName("MyApp").getOrCreate()

# PySpark DataFrame Operations – Practical Flow

Step 1: Create a DataFrame  
data = [(10,), (20,), (30,), (40,), (50,)]  
df = spark.createDataFrame(data, ["number"])  
df.show()

Step 2: Filter Values  
filtered\_df = df.filter(df.number > 25)  
filtered\_df.show()

Step 3: Add Transformed Column  
transformed\_df = filtered\_df.withColumn("squared", filtered\_df.number \*\* 2)  
transformed\_df.show()

Step 4: Collect Results  
results = transformed\_df.collect()  
for row in results:  
 print(row)

Step 5: Save as CSV  
transformed\_df.write.mode("overwrite").csv("/FileStore/output/transformed\_data")

# RDD vs DataFrame Comparison

Create Dataset:  
RDD: sc.parallelize([1,2])  
DF: spark.createDataFrame([(1,), (2,)])

Filter:  
RDD: rdd.filter(lambda x: x > 5)  
DF: df.filter(df.col > 5)

Transform:  
RDD: rdd.map(lambda x: x \* x)  
DF: df.withColumn("squared", df.col\*\*2)

Display:  
RDD: rdd.collect()  
DF: df.show()

Save:  
RDD: rdd.saveAsTextFile("path")  
DF: df.write.csv("path")

# Schema Definition (CSV Reading)

from pyspark.sql.types import StructType, StructField, IntegerType, StringType  
  
schema = StructType([  
 StructField("order\_id", IntegerType(), True),  
 StructField("customer\_id", IntegerType(), True),  
 StructField("order\_date", StringType(), True),  
 StructField("product", StringType(), True),  
 StructField("category", StringType(), True),  
 StructField("quantity", StringType(), True),  
 StructField("price\_per\_unit", IntegerType(), True),  
])  
data = spark.read.csv("/path/to/file.csv", header=True, schema=schema)

# SQL Integration with DataFrames

data.createOrReplaceTempView("sales\_data")  
spark.sql("SELECT \* FROM sales\_data WHERE price\_per\_unit > 500").show()

# Summary

* RDDs offer low-level control but are less optimized.
* DataFrames provide high-level abstraction, are faster, and integrate well with SQL.
* In Databricks or cloud environments, DataFrames are preferred over RDDs.
* PySpark allows writing powerful distributed data transformations using familiar Python syntax.