**Name -Monika Gariya**

**Email-** [**monikagariya2023@gmail.com**](mailto:monikagariya2023@gmail.com)

**Data Engineering Batch 1**

**Date – 10-02-2024**

**Topic- PySpark**

PySpark is a Python library for Apache Spark that allows users to interface with Spark using Python. It provides a programming interface to the Spark ecosystem, allowing users to harness the power of Spark from Python.

[PySpark](https://granulate.io/blog/understanding-pyspark-features-ecosystem-optimization/) is particularly useful for data scientists and data engineers who want to use Spark for data processing and analytics tasks, but are more comfortable working with Python than with Scala, the programming language that Spark is written in. PySpark allows these users to leverage their Python skills to work with Spark, and makes it easier to integrate Spark into their existing Python-based workflow

**Key Features of PySpark**

* **Integrated** − Seamlessly mix SQL queries with Spark programs. Spark SQL lets you query structured data as a distributed dataset (RDD) in Spark, with integrated APIs in Python, Scala and Java. This tight integration makes it easy to run SQL queries alongside complex analytic algorithms.
* **Unified Data Access** − Load and query data from a variety of sources. Schema-RDDs provide a single interface for efficiently working with structured data, including Apache Hive tables, parquet files and JSON files.
* **Hive Compatibility** − Run unmodified Hive queries on existing warehouses. Spark SQL reuses the Hive frontend and MetaStore, giving you full compatibility with existing Hive data, queries, and UDFs. Simply install it alongside Hive.
* **Standard Connectivity** − Connect through JDBC or ODBC. Spark SQL includes a server mode with industry standard JDBC and ODBC connectivity.
* **Scalability** − Use the same engine for both interactive and long queries. Spark SQL takes advantage of the RDD model to support mid-query fault tolerance, letting it scale to large jobs too. Do not worry about using a different engine for historical data.

**What Is PySpark RDD?**

Resilient Distributed Datasets, often known as RDDs, are the components used in a cluster's parallel processing that run and operate across numerous nodes. Since RDDs are immutable elements, you cannot alter them after creation. Because RDDs are fault-tolerant, they will immediately recover from any failure. These RDDs allow you to do various operations to complete a certain goal.

**Features of RDD**

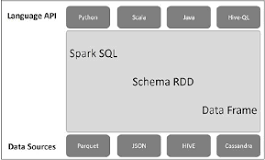
1.  In-Memory - Spark RDD can be used to store data. Data storage in a spark RDD is size and volume-independent. We can save any size of data. The term "in-memory computation" refers to processing data stored in the main RAM. Operating across tasks is necessary, not in intricate databases because running databases slow the drive.

2. Lazy Evaluations - Its name implies that the execution process does not begin immediately after calling a certain operation. There must be an action taken to start the execution. As a result of such action, RDD data cannot be transformed or made public. Spark keeps track of each operation performed through DAG. Referring to a directed acyclic graph, or DAG.

3. Immutable and Read-only - Since RDDs cannot be changed over time, they are immutable. When we carry out more computations, that property helps to maintain consistency.RDDs that have already been generated cannot be changed; they can only be turned into new RDDs. This is made feasible by its processes of transformation.

4. Cacheable or Persistence - All of the data can be kept on disc, memory, and persistent storage. Disk and memory (most preferable) (less Preferred because of its slow access speed). Additionally, we can retrieve it straight from memory. Because of this, RDDs are advantageous for quick computations. As a result, we can run several operations on the same data set. This leads to reusability as well, which speeds up computation.

## ****Spark SQL Architecture****



* **Language API** − Spark is compatible with different languages and Spark SQL. It is also, supported by these languages- API (python, scala, java, HiveQL).
* **Schema RDD** − Spark Core is designed with special data structure called RDD. Generally, Spark SQL works on schemas, tables, and records. Therefore, we can use the Schema RDD as temporary table. We can call this Schema RDD as Data Frame.
* **Data Sources** − Usually the Data source for spark-core is a text file, Avro file, etc. However, the Data Sources for Spark SQL is different. Those are Parquet file, JSON document, HIVE tables, and Cassandra database

**DATAFRAME**

A DataFrame is a distributed collection of data, which is organized into named columns. A DataFrame can be constructed from an array of different sources such as Hive tables, Structured Data files, external databases, or existing RDDs. This API was designed for modern Big Data and data science applications taking inspiration from **DataFrame in R Programming** and **Pandas in Python**.

## Features of DataFrame

* Ability to process the data in the size of Kilobytes to Petabytes on a single node cluster to large cluster.
* Supports different data formats (Avro, csv, elastic search, and Cassandra) and storage systems (HDFS, HIVE tables, mysql, etc).
* State of art optimization and code generation through the Spark SQL Catalyst optimizer (tree transformation framework).
* Can be easily integrated with all Big Data tools and frameworks via Spark-Core.
* Provides API for Python, Java, Scala, and R Programming.

## What is PySpark Optimization and Why Is it Important?

Apache PySpark is the Python API for Apache Spark, an open-source, distributed computing system that is designed for high-speed processing of large data sets. PySpark allows users to leverage the power of Spark using the familiar Python programming language.

PySpark is implemented using the Py4J Java library, which allows Python programs to dynamically call Java code. While this is convenient for Python users, it also adds overhead compared to working with Spark with its native programming language, Scala.

There are also a number of alternatives to UDFs that can be used to extend the functionality of Spark’s SQL and DataFrame APIs, including:

* **Spark SQL functions:**These are built-in functions that are provided by Spark and can be used to perform common data processing tasks.
* **Window functions:** These are functions that can be used to perform operations on a set of rows over a window of data.
* **Aggregate functions:** These are functions that can be used to perform operations on a group of rows, such as calculating the average or sum of a set of values.