Introduction to DataFrames in Spark SQL

**Overview**

* **What Are DataFrames?**
  + Structured data representation
  + Rows and columns
* **Spark SQL**
  + Unified data processing engine
* **Outline of Topics**
  + DataFrames to RDDs and Vice Versa
  + Loading DataFrames from CSVs
  + Working with Schemas
  + Loading Parquet and JSON
  + Data Manipulation
  + Querying, Sorting, and Filtering
  + Handling Missing Data
  + Saving DataFrames

**Introduction to DataFrames**

* **Definition:**
  + A DataFrame is a distributed collection of data organized into named columns.
* **Key Features:**
  + Schema of the data
  + Lazy evaluation
  + Integration with SQL queries
* **Example:**

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Example").getOrCreate()

data = [("Alice", 34), ("Bob", 45)]

df = spark.createDataFrame(data, ["Name", "Age"])

df.show()

**DataFrames to RDDs and Vice Versa**

* **Converting DataFrame to RDD:**

rdd = df.rdd

* **Converting RDD to DataFrame:**

from pyspark.sql import Row

rdd = spark.sparkContext.parallelize([Row(Name="Alice", Age=34)])

df = rdd.toDF()

* **Use Cases:**
  + Use RDDs for low-level transformations and actions
  + Use DataFrames for high-level operations and SQL queries

**Loading DataFrames from CSVs**

* **Example:**

df = spark.read.csv("path/to/file.csv", header=True, inferSchema=True)

df.show()

* **Options:**
  + header=True to use the first row as column names
  + inferSchema=True to automatically detect data types

**Working with Schemas**

* **Schema Definition:**
  + Structuring the data types of each column
* **Example:**

from pyspark.sql.types import StructType, StructField, StringType, IntegerType

schema = StructType([

StructField("Name", StringType(), True),

StructField("Age", IntegerType(), True)

])

df = spark.read.csv("path/to/file.csv", schema=schema)

df.printSchema()

**Loading Parquet and JSON**

* **Loading Parquet:**

df = spark.read.parquet("path/to/file.parquet")

df.show()

* **Loading JSON:**

df = spark.read.json("path/to/file.json")

df.show()

* **Benefits:**
  + Parquet: Columnar storage format, efficient for big data
  + JSON: Flexible schema, good for semi-structured data

**Data Manipulation**

* **Rows and Columns:**
  + **Selecting Columns:**

df.select("Name").show()

* + **Adding Columns:**

from pyspark.sql.functions import col

df = df.withColumn("NewAge", col("Age") + 1)

* + **Renaming Columns:**

df = df.withColumnRenamed("Age", "Years")

* + **Casting Columns:**

df = df.withColumn("Age", col("Age").cast("String"))

* + **Dropping Columns:**

df = df.drop("NewAge")

**Querying, Sorting, and Filtering DataFrames**

* **Querying:**

df.createOrReplaceTempView("people")

result = spark.sql("SELECT \* FROM people WHERE Age > 30")

result.show()

* **Sorting:**

df.orderBy(col("Age").desc()).show()

* **Filtering:**

df.filter(col("Age") > 30).show()

**Handling Missing or Corrupt Data**

* **Handling Missing Data:**
  + **Drop rows with missing values:**

df = df.na.drop()

* + **Fill missing values:**

df = df.na.fill({"Age": 0})

* **Handling Corrupt Data:**

df = spark.read.option("mode", "DROPMALFORMED").json("path/to/file.json")

**Saving DataFrames**

* **Saving as CSV:**

df.write.csv("path/to/save.csv", header=True)

* **Saving as Parquet:**

df.write.parquet("path/to/save.parquet")

* **Saving as JSON:**

df.write.json("path/to/save.json")

**Summary**

* **Key Points:**
  + DataFrames offer a high-level API for data processing
  + They can be converted to and from RDDs
  + DataFrames can be loaded from and saved to various formats (CSV, Parquet, JSON)
  + Rich set of operations available for manipulation, querying, and handling data
* **Questions?**

**References**

* **Spark Documentation:** [Apache Spark Documentation](https://spark.apache.org/docs/latest/)
* **Spark SQL Guide:** [Spark SQL Guide](https://spark.apache.org/docs/latest/sql-programming-guide.html)