Apache Spark 2 using SQL - Pre -Defined Functions

Overview

Spark SQL provides a rich set of pre-defined (built-in) functions for data manipulation, transformation, and querying. These functions make it easier to work with DataFrames and run SQL queries efficiently using Spark's distributed processing capabilities.

Step 1: Initialize SparkSession in Scala

To use Spark SQL functions, start with creating a SparkSession:

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder()
  .appName("Spark SQL Functions Tutorial")
  .master("local[*]")
  .getOrCreate()

import spark.implicits._
import org.apache.spark.sql.functions._
```

Step 2: Common Pre-Defined Functions and Usage Examples

2.1 String Functions

- `upper(col)`: Converts a string column to uppercase.
- `split(col, pattern)`: Splits a string column by a regex pattern.
- `reverse(col)`: Reverses the string column.

Example:

```
val df = Seq(
  (1, "hello world"),
  (2, "spark sql")
).toDF("id", "text")

val transformedDF = df.withColumn("upper_text", upper($"text"))
  .withColumn("split_text", split($"text", " "))
  .withColumn("reversed_text", reverse($"text"))

transformedDF.show(false)
```

Output:

```
+---+-----------+----------+------------+
|id |text       |upper_text|split_text  |
+---+-----------+----------+------------+
|1  |hello world|HELLO WORLD|[hello, world]|
|2  |spark sql  |SPARK SQL |[spark, sql]|
+---+-----------+----------+------------+
```

2.2 Math Functions

- `factorial(col)`: Calculates factorial of a number.
- `sin(col)`, `cos(col)`, `sqrt(col)`: Various math functions.

Example:

```
val numbersDF = Seq(2, 3, 4).toDF("number")
```

```
val mathDF = numbersDF.withColumn("factorial",
factorial($"number"))
mathDF.show()
```

Output:

```
+------+---------+
|number|factorial|
+------+---------+
|2     |2        |
|3     |6        |
|4     |24       |
+------+---------+
```

2.3 Date and Time Functions

- `current_date()`: Returns current date.
- `date_add(col, days)`: Adds days to a date.

Example:

```
val dateDF = Seq(("2025-08-10")).toDF("date_str")
  .withColumn("date", to_date($"date_str"))
  .withColumn("date_plus_5", date_add($"date", 5))

dateDF.show()
```

Step 3: Using SQL Queries with Functions

You can register a DataFrame as a temporary SQL view and use Spark SQL's built-in functions:

```
df.createOrReplaceTempView("text_table")

val sqlDF = spark.sql("""
  SELECT id, upper(text) AS upper_text, length(text) AS len
```

```
  FROM text_table
""")

sqlDF.show()
```

Step 4: Defining and Using UDFs (User Defined Functions)

Sometimes you need custom logic beyond pre-defined functions. You can create UDFs in Scala:

```
import org.apache.spark.sql.functions.udf

val addPrefix = udf((s: String) => "PREFIX_" + s)

val dfWithPrefix = df.withColumn("prefixed_text",
addPrefix($"text"))
dfWithPrefix.show()
```

Step 5: Exercises and Quizzes

1. Write a Spark SQL function to convert a string column to lowercase.
2. Using pre-defined math functions, calculate the square root of numbers 16, 25, 36.
3. Create a UDF that returns "even" or "odd" based on an integer column.
4. Write a SQL query to select rows where the length of a text column is greater than 5.

GitHub References for Spark SQL Functions in Scala

- Repository with practical Scala and Spark examples including SQL and functions: [ruslanmv/Scala-and-Spark-in-Practice-]
- Official Apache Spark GitHub repo (explore sql/core for function implementations): [apache/spark]
- Example Notebooks on Spark SQL and UDFs: [Databricks SparkSQLAndUDFs]