Spark SQL

Demo

SQLContext

SQLContext is a class and is used for initializing the functionalities of Spark SQL. SparkContext class object (sc) is required for initializing SQLContext class object.

The following command is used for initializing the SparkContext through spark-shell.

```
$ spark-shell
```

By default, the SparkContext object is initialized with the name **sc** when the spark-shell starts.

Use the following command to create SQLContext.

```
scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
```

Example

Let us consider an example of employee records in a JSON file named **employee.json**. Use the following commands to create a DataFrame (df) and read a JSON document named **employee.json** with the following content.

employee.json − Place this file in the directory where the current **scala>** pointer is located.

```
{
   {"id" : "1201", "name" : "satish", "age" : "25"}
   {"id" : "1202", "name" : "krishna", "age" : "28"}
   {"id" : "1203", "name" : "amith", "age" : "39"}
   {"id" : "1204", "name" : "javed", "age" : "23"}
   {"id" : "1205", "name" : "prudvi", "age" : "23"}
}
```

DataFrame Operations

DataFrame provides a domain-specific language for structured data manipulation. Here, we include some basic examples of structured data processing using DataFrames.

Follow the steps given below to perform DataFrame operations −

Read the JSON Document

First, we have to read the JSON document. Based on this, generate a DataFrame named (dfs).

Use the following command to read the JSON document named **employee.json**. The data is shown as a table with the fields − id, name, and age.

```
scala> val dfs = sqlContext.read.json("employee.json")
```

Output − The field names are taken automatically from **employee.json**.

```
dfs: org.apache.spark.sql.DataFrame = [age: string, id: string, name: string]
```

Show the Data

If you want to see the data in the DataFrame, then use the following command.

```
scala> dfs.show()
```

Output − You can see the employee data in a tabular format.

```
<console>:22, took 0.052610 s
+----+------+--------+
|age | id   |  name  |
+----+------+--------+
| 25 | 1201 | satish |
| 28 | 1202 | krishna|
| 39 | 1203 | amith  |
| 23 | 1204 | javed  |
| 23 | 1205 | prudvi |
+----+------+--------+
```

Use printSchema Method

If you want to see the Structure (Schema) of the DataFrame, then use the following command.

```
scala> dfs.printSchema()
```

Output

```
root
   |-- age: string (nullable = true)
   |-- id: string (nullable = true)
   |-- name: string (nullable = true)
```

Use Select Method

Use the following command to fetch **name**-column among three columns from the DataFrame.

```
scala> dfs.select("name").show()
```

Output − You can see the values of the **name** column.

```
<console>:22, took 0.044023 s
+--------+
|  name  |
+--------+
| satish |
| krishna|
| amith  |
| javed  |
| prudvi |
+--------+
```

Use Age Filter

Use the following command for finding the employees whose age is greater than 23 (age > 23).

```
scala> dfs.filter(dfs("age") > 23).show()
```

Output

```
<console>:22, took 0.078670 s
+----+------+--------+
|age | id   | name   |
+----+------+--------+
| 25 | 1201 | satish |
| 28 | 1202 | krishna|
| 39 | 1203 | amith  |
+----+------+--------+
```

Use groupBy Method

Use the following command for counting the number of employees who are of the same age.

```
scala> dfs.groupBy("age").count().show()
```

Output – two employees are having age 23.

```
<console>:22, took 5.196091 s
+----+-----+
|age |count|
+----+-----+
| 23 |  2  |
| 25 |  1  |
| 28 |  1  |
| 39 |  1  |
+----+-----+
```