Introduction to Spark DataFrame

A spark data frame can be said to be a distributed data collection organized into named columns and is also used to provide operations such as filtering, computation of aggregations, grouping, and can be used with Spark SQL. Data frames can be created by using structured data files, existing RDDs, external databases, and Hive tables. It is basically termed and known as an abstraction layer which is built on top of RDD and is also followed by the dataset API, which was introduced in later versions of Spark (2.0 +). Moreover, the datasets were not introduced in Pyspark but only in Scala with Spark, but this was not the case in the case of Dataframes. Data frames, popularly known as DFs, are logical columnar formats that make working with RDDs easier and more convenient, also making use of the same functions as RDDs in the same way. If you talk more on the conceptual level, it is equivalent to the relational tables along with good optimization features and techniques.

How to Create a DataFrame?

A Data Frame is generally created by any one of the mentioned methods. It can be created by making use of Hive tables, external databases, Structured data files or even in the case of existing RDDs. These all ways can create these named columns known as Dataframes used for the processing in Apache Spark. By making use of SQLContext or SparkSession, applications can be used to create Dataframes.

Spark DataFrames Operations

In Spark, a data frame is the distribution and collection of an organized form of data into named columns which is equivalent to a relational database or a schema or a data frame in a language such as R or python but along with a richer level of optimizations to be used. It is used to provide a specific domain kind of language that could be used for structured data manipulation.

The below mentioned are some basic Operations of Structured Data Processing by making use of Dataframes.

1. Reading a document which is of type: JSON: We would be making use of the command sqlContext.read.json.

Example: Let us suppose our filename is student.json, then our piece of code will look like:
val sqlContext = spark.sqlContext

```
val dfs= sqlContext.read.json(“student.json”)
```

Output: In this case, the output will be that the field names will be automatically taken from the file student.json.

2. Showing of Data: In order to see the data in the Spark data frames, you will need to use the command:
dfs.show()

Example: Let us suppose our filename is student.json, then our piece of code will look like:

```
val dfs= sqlContext.read.json(“student.json”)
dfs.show()
```

Output: The student data will be present to you in a tabular format.

3. Using printSchema method: If you are interested to see the structure, i.e. schema of the data frame, then make use of the following command: dfs.printSchema()

Example: Let us suppose our filename is student.json, then our piece of code will look like:

```
val dfs= sqlContext.read.json("student.json")
dfs. printSchema ()
```

Output: The structure or the schema will be present to you

4. Use the select method: In order to use the select method, the following command will be used to fetch the names and columns from the list of data frames.
dfs.select("column-name").show()

Example: Let us suppose our filename is student.json, then our piece of code will look like:

```
val dfs= sqlContext.read.json("student.json")
dfs.select("name").show()
```

Output: The values of the name column can be seen.

5. Using Age filter: The following command can be used to find the range of students whose age is more than 23 years.
dfs.filter(dfs("column-name") > value).show()

Example: Let us suppose our filename is student.json, then our piece of code will look like:

```
val dfs= sqlContext.read.json("student.json")
dfs.filter(dfs("age")>23).show()
```

Output: The filtered age for greater than 23 will appear in the results.

6. Using the groupBy method: The following method could be used to count the number of students who have the same age.
dfs.groupBy("column-name").count().show()

Example: Let us suppose our filename is student.json, then our piece of code will look like:

```
val dfs= sqlContext.read.json("student.json")
dfs.groupBy("age").count().show()
```

7. Using SQL function upon a SparkSession: It enables the application to execute SQL type queries programmatically and hence returns the result in the form of a data frame.
spark.sql(query)

Example: Suppose we have to register the SQL data frame as a temp view then:

```
df.createOrReplaceTempView("student")
sqlDF=spark.sql("select * from student")
sqlDF.show()
```

Output: A temporary view will be created by the name of the student, and a spark.sql will be applied on top of it to convert it into a data frame.

8. Using SQL function upon a Spark Session for Global temporary view: This enables the application to execute SQL type queries programmatically and hence returns the result in the form of a data frame.
spark.sql(query)

Example: Suppose we have to register the SQL data frame as a temp view then:

```
df.createGlobalTempView(“student”)
park.sql(“select * from global_temp.student”).show()
spark.newSession().sql(“Select * from global_temp.student”).show()
```

Output: A temporary view will be created by the name of the student, and a spark.sql will be applied on top of it to convert it into a data frame.