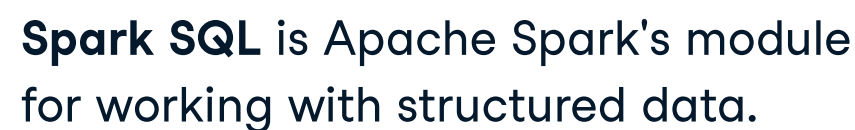




# PySpark & Spark SQL



A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

## > Creating DataFrames

```
>>> from pyspark.sql.types import *

Infer Schema

>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(", "))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopleDF = spark.createDataFrame(people)

Specify Schema

>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
```

name	age
Mine	28
Filip	29
Jonathan	30

```

JSON

>>> df = spark.read.json("customer.json")
>>> df.show()

+-----+-----+-----+-----+-----+
|address|age|firstName|lastName|phoneNumbers|
+-----+-----+-----+-----+-----+
|New York,10021,N...|25|John|Smith|(212 555-1234,ho...|
|New York,10021,N...|21|Jane|Doe|(322 888-1234,ho...|
+-----+-----+-----+-----+-----+

>>> df2 = spark.read.load("people.json", format=

Parquet files

>>> df3 = spark.read.load("users.parquet")

TXT files

>>> df4 = spark.read.text("people.txt")

```

```
#Filter entries of age, only keep those records of which the values are >24
>>> df.filter(df["age"]>24).show()
```

```
>>> df = df.dropDuplicates()
```

```
>>> from pyspark.sql import functions as F

Select

>>> df.select("firstName").show() #Show all entries in firstName column
>>> df.select("firstName", "lastName") \
    .show()
>>> df.select("firstName", #Show all entries in firstName, age and type
    "age",
    explode("phoneNumber") \
    .alias("contactInfo")) \
    .select("contactInfo.type",
    "firstName",
    "age") \
    .show()
>>> df.select(df["firstName"], df["age"] + 1) #Show all entries in firstName and age,
    .show()                                add 1 to the entries of age
>>> df.select(df["age"] > 24).show() #Show all entries where age >24
```

```
>>> df.select("firstName", #Show firstName and 0 or 1 depending on age >30
              F.when(df.age > 30, 1) \
                .otherwise(0)) \
        .show()
>>> df[df.firstName.isin("Jane","Boris")] #Show firstName if in the given options
        .collect()
```

```
>>> df.select("firstName", #Show firstName, and lastName is TRUE if lastName is like Smith
              df.lastName.like("Smith")) \
      .show()
```

```
>>> df.select("firstName", #Show firstName, and TRUE if LastName starts with Sm
             df.lastName \
               .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endsWith("th"))\ #Show last names ending in th
      .show()
```

```
>>> df.select(df.firstName.substr(1, 3) \ #Return substrings of firstName
              .alias("name")) \
      .collect()
```

```
>>> df.select(df.age.between(22, 24)) \ #Show age: values are TRUE if between 22 and 24
      .show()
```

## Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
      .withColumn('postalCode',df.address.postalCode) \
      .withColumn('state',df.address.state) \
      .withColumn('streetAddress',df.address.streetAddress) \
      .withColumn('telephoneNumber', explode(df.phoneNumber.number)) \
      .withColumn('telephoneType', type(df.phoneNumber.type))
```

```
>>> df = df.withColumnRenamed('teLePhoneNuMber', 'phoneNuMber')
```

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

```
>>> df.na.fill(50).show() #Replace null values
>>> df.na.drop().show() #Return new df omitting rows with null values
>>> df.na \ #Return new df replacing one value with another
      .replace(10, 20) \
      .show()
```

```
>>> df.groupBy("age")\ #Group by age, count the members in the groups
      .count() \
      .show()
```

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age", "city"],ascending=[0,1])\
    .collect()
```

```
>>> df.repartition(10)\ #df with 10 partitions
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions() #df with 1 partition
```

## Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
    .show()
```

```
>>> df.dtypes #Return df column names and data types
>>> df.show() #Display the content of df
>>> df.head() #Return first n rows
>>> df.first() #Return first row
>>> df.take(2) #Return the first n rows >>> df.schema Return the schema of df
>>> df.describe().show() #Compute summary statistics >>> df.columns Return the columns of df
>>> df.count() #Count the number of rows in df
>>> df.distinct().count() #Count the number of distinct rows in df
>>> df.printSchema() #Print the schema of df
>>> df.explain() #Print the (logical and physical) plans
```

## Data Structures

```
>>> rdd1 = df.rdd #Convert df into an RDD
>>> df.toJSON().first() #Convert df into a RDD of string
>>> df.toPandas() #Return the contents of df as Pandas DataFrame
```

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json", format="json")
```

```
>>> spark.stop()
```

