Spark SQL Tutorial

Let’s make some practice! ☺

How to use Spark?

- **Databricks notebook** community edition - databricks.com
- **Local mode**

  install python (https://www.python.org/downloads/)

  install pip (https://pip.pypa.io/en/stable/installing/)

  in the **terminal**:

  ```
  pip install findspark
  ```

  then, in the **code**:

  ```
  import findspark
  findspark.init(‘/path/to/spark’)

  from pyspark.sql import SparkSession
  from pyspark.conf import SparkConf

  conf = SparkConf().setAppName(appName).setMaster(master)
  spark = SparkSession.builder.config(conf=conf).getOrCreate()
  ```

SQL

Dataset

- Distributed collection of data
- Strong typed
- SQL’s optimized execution engine
- Only for Scala and Java (but Python already has some features)

DataFrame

- Table in a relational database
- Untyped (Dataset<Row>)
- Many available sources (structured data files, tables, databases, RDDs)
- Scala, Java, Python, and R

Initializing SparkSession

Create a SparkSession (the notebook already has it)

```
from pyspark import SparkSession
spark = SparkSession \
                .builder \
                .appName(appName) \
                .master(master) \
                .getOrCreate()

sc = spark.sparkContext
```

appName = Name of the application to show on the cluster UI

master = Spark URL ('spark://ip-address:7077') or 'local'

Creating DataFrames

```
df = spark.read.csv('files/people.csv')
```

df.show()	outputs the content of the dataframe
df.printSchema()	outputs the schema of the dataframe

Untyped Dataset Operations (aka DataFrame Operations)

```
// Select only the “name” column

df.select("name").show()


// Select only the "name" and "address" column and add 1 to "age"

df.select(col("name"), col("address"), col("age").plus(1)).show()
```

In order to access nested element do

root.child e.g. "address.city"

Untyped Dataset Operations (aka DataFrame Operations)

```
// Select people older than 21

df.filter(col("age").gt(21)).show()


// Count people by age

df.groupBy("age").count().show()
```

Running SQL Queries Programmatically

```
// Register the DataFrame as a SQL temporary view

df.createOrReplaceTempView("people")


sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
```

Running SQL Queries Programmatically

Temporary views are session-scoped -> they disappear after termination
In order to keep it alive, create a global temporary view

```
spark = spark.newSession()  // table people is no longer present

df.createGlobalTempView("people")

spark.newSession().sql("SELECT * FROM global_temp.people").show();
```

Creating Datasets

```
public class Person implements Serializable {
    private String name;
    private int age;
    private Address address;

    /*** Getters and Setters ***/
}

public class Address implements Serializable {
    private String city;
    private String state;

    /*** Getters and Setters ***/
}
```

Creating Datasets

```
Person person = new Person();
person.setName("Andy");
person.setAge(32);
Address address = new Address();
address.setCity("Rome");
address.setState("Italy");
person.setAddress(address);

Encoder<Person> personEncoder = Encoders.bean(Person.class);
Dataset<Person> dataset =
    spark.createDataset(Collections.singletonList(person), personEncoder);

dataset.show();
```

Creating Datasets

```
Encoder<Integer> integerEncoder = Encoders.INT();

Dataset<Integer> primitiveDS =
              spark.createDataset(Arrays.asList(1, 2, 3), integerEncoder);

Dataset<Integer> transformedDS = primitiveDS.map(
         (MapFunction<Integer, Integer>) value -> value + 1,
         integerEncoder);

transformedDS.collect();
```

Creating Datasets

```
Encoder<Person> personEncoder = Encoders.bean(Person.class);

Dataset<Person> personDataset =
                spark.read().json("files/people.json").as(personEncoder);

personDataset.show();
```

Interoperating with RRDs

Inferring the Schema Using Reflection

Concise syntax, schema already known

```
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
```

Interoperating with RRDs

Inferring the Schema Using Reflection

```
schemaPeople = spark.createDataFrame(people)
schemaPeople.createOrReplaceTempView("people")


teenagers = spark.sql("""
        SELECT name FROM people
        WHERE age >= 13 AND age <= 19""")


teenNames = teenagers.rdd.map(lambda p: "Name: " + p.name).collect()

for name in teenNames:
    print(name)
```

Interoperating with RRDs

Programmatically Specifying the Schema

Verbose syntax, schema not known

```
lines = sc.textFile("examples/src/main/resources/people.txt")\
              .map(lambda l: l.split(","))\
              .map(lambda p: (p[0], p[1].strip()))

schemaStr = "name age city state";

fields = [StructField(field, StringType(), True) for field in schemaStr.split()]

schema = StructType(fields)
```

Interoperating with RRDs

Programmatically Specifying the Schema

```
schemaPeople = spark.createDataFrame(people, schema)

schemaPeople.createOrReplaceTempView("people")

results = spark.sql("SELECT name FROM people")


results.show()
```

Data Sources

Generic Load/Save Functions

Default: Parquet file

```
usersDF = spark.read.load("files/users.parquet")

usersDF.select("name", "favorite_color").write.save("files/results.parquet")

sqlDF = spark.sql("SELECT * FROM parquet.`files/users.parquet`")
```

dbTrento

Exercises [both RDD and SparkSQL]

- Use github.com/forons/BigDataExamples/blob/master/files/tweets_cleaned.csv
- Count tweets per user
- Split created_at field and count the number of tweets per hour
- Count the tweets that contain a word that you choose
- Sort the users based on the number of tweets

Exercises [both RDD and SparkSQL]

- Find all the tweets by user
- Find how many tweets each user has
- Find all the persons mentioned on tweets
- Count how many times each person is mentioned
- Find the 10 most mentioned persons
- Find all the hashtags mentioned on a tweet
- Count how many times each hashtag is mentioned
- Find the 10 most popular Hashtags

Contacts

For any problem, send a mail to

daniele.foroni@unitn.it