

## Big data: architectures and data analytics

## Spark SQL

### Spark SQL

- Spark SQL is the Spark component for structured data processing
- It provides a programming abstraction called `DataFrames` and can act as distributed SQL query engine
  - The input data can be queried by using
    - Ad-hoc methods
    - Or the SQL language

3

### Spark Context

- All the Spark SQL commands are based on an instance of the `org.apache.spark.sql.SQLContext` class
- To instance an `SQLContext` object the constructor `org.apache.spark.sql.SQLContext(SparkContext)` of the `SQLContext` must be used
- E.g.,
  - `SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);`

4

## DataFrames

## DataFrames

- DataFrame
  - It is a distributed collection of data organized into named columns
  - It is equivalent to a relational table
- DataFrames are objects of the class `org.apache.spark.sql.DataFrame`
- Also the content of DataFrames is lazily evaluated

6

## DataFrames

- DataFrames can be constructed from different sources
  - Structured (textual) data files
    - E.g., csv files, json files
  - Existing RDDs
  - Hive tables
  - External relational databases

7

## Creating DataFrames from json files

- Spark SQL provides an API that allows creating a DataFrame directly from a textual file where each line contains a json object
  - Hence, the input file is not a "standard" json file
  - It must be properly formatted in order to have one json object (tuple) for each line

8

## Creating DataFrames from json files

- Example of json file compatible with the Spark expected format
 

```
{"name":"Michael"}  
 {"name":"Andy", "age":30}  
 {"name":"Justin", "age":19}
```
- The file contains name and age of three persons
  - The age of the first person is unknown

9

## Creating DataFrames from json files

- The creation of a DataFrame from a json file is based the [DataFrame json\(String path\)](#) method of the [org.apache.spark.sql.DataFrameReader](#) class
  - Path is the path of the input file
- The creation of a [DataFrameReader](#) object is based on the [DataFrameReader read\(\)](#) method of the [SQLContext](#) class

10

## Creating DataFrames from json files: example

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name and age of a person
    - Age can assume the null value

11

## Creating DataFrames from json files: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("Spark SQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame df = dfr.json("persons.json");
```

12

## Creating DataFrames from an existing RDD

- Spark SQL provides an API that allows creating a DataFrame from an existing RDD
  - The RDD must be an RDD of JavaBeans
  - The BeanInfo, obtained using reflection, defines the schema of the DataFrame
  - Spark SQL does not support JavaBeans that contain nested elements or contain complex types such as Lists or Arrays
  - You can create a JavaBean by creating a class that implements the Serializable interface and has getters and setters for all of its fields

13

## Creating DataFrames from an existing RDD

- The creation of a DataFrame from an RDD is based the `DataFrame.createDataFrame(JavaRDD<T> rdd, java.lang.Class<T> beanClass)` method of the `SQLContext` class
  - T is the data type (class) of the input elements

14

## Creating DataFrames from an existing RDD: example

- Create a DataFrame from an RDD of objects of type Person
- Person is a Java class containing name and age of a single person
- Suppose the following persons are part of the input RDD
  - Paolo, 40
  - Giovanni, 30
  - Luca, 32

15

## Creating DataFrames from an existing RDD: example

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

16

## Creating DataFrames from an existing RDD: example

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

    public String getName() {
        return name;
    }

    Each instance of this class is characterized
    by name and age.
    The schema of the DataFrame based on
    objects of this class is: name, age

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

17

## Creating DataFrames from an existing RDD: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("Spark SQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);
```

18

## Creating DataFrames from an existing RDD: example

```
// Create a local array of Persons
ArrayList<Person> persons = new ArrayList<Person>();

Person person;

person = new Person();
person.setName("Paolo");
person.setAge(40);
persons.add(person);

person = new Person();
person.setName("Giovanni");
person.setAge(30);
persons.add(person);

person = new Person();
person.setName("Luca");
person.setAge(32);
persons.add(person);
```

19

## Creating DataFrames from an existing RDD: example

```
// Define and RDD of Persons
JavaRDD<Person> personsRDD = sc.parallelize(persons);

// Define a DataFrame based on personsRDD
DataFrame personsDF =
    sqlContext.createDataFrame(personsRDD, Person.class);
```

20

## Creating DataFrames from other data sources

- The DataFrameReader class (the same we used for reading a json file and store it in a DataFrame) provides other methods to read many standard (textual) formats and read data from external databases
- The `jdbc(url, table, properties)` method allows reading the input data from an external relational database, through a JDBC connection
- The `parquet(path)` method reads data from apache parquet files

21

## Creating DataFrames from other data sources

- The Hive database is also supported
- The specific `org.apache.spark.sql.hive.HiveContext` class is used to create an HiveContext instead of the standard SQLContext
- HiveContext extends the SQLContext class providing specific features related to Hive

22

## Operations on DataFrames

## DataFrame Operations

- A set of methods, implementing a set of operations, are available for the DataFrame class
  - E.g., `show()`, `printSchema()`, `count()`, `distinct()`, `select()`, `filter()`

24

## Show

- The **void show(int numRows)** method of the **DataFrame** class prints on the standard output numRows of the DataFrame

25

## Show: example

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name and age of a person
    - Age can assume the null value
- Print the content of 3 persons (i.e., 3 rows of the DataFrame)

26

## Show: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("SparkSQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame df = dfr.json("persons.json");

// Print, on the standard output, 3 rows of the DataFrame
df.show(3);
```

27

## PrintSchema

- The **void printSchema()** method of the **DataFrame** class prints on the standard output the schema of the DataFrame
  - i.e., the name of the attributes of the data stored in the DataFrame

28

## PrintSchema: example

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name and age of a person
    - Age can assume the null value
- Print the schema of the created DataFrame

29

## PrintSchema: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("SparkSQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame df = dfr.json("persons.json");

// Print, on the standard output, the schema of the DataFrame
df.printSchema();
```

30

## Count

- The `long count()` method of the `DataFrame` class returns the number of rows in the `DataFrame`

33

## Count: example

- Create a `DataFrame` from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name and age of a person
- Print the number of persons (i.e., rows) in the created `DataFrames`

32

## Count: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("SparkSQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame df = dfr.json("persons.json");

// Print, on the standard output, the number of persons
System.out.println("The input file contains "+df.count()+" persons");
```

33

## Distinct

- The `DataFrame distinct()` method of the `DataFrame` class returns a new `DataFrame` that contains only the unique rows of the `DataFrame`
  - Pay attention that the `distinct` operation is always an heavy operation in terms of data sent on the network
  - A shuffle phase is needed

34

## Distinct: example

- Create a `DataFrame` from a json file containing a list of first names
  - Each line of the file contains a json object containing a first name
- Create a new `DataFrame` without duplicates

35

## Distinct: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("SparkSQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame dfNames = dfr.json("names.json");

// Create a new DataFrame without duplicates
DataFrame distinctNames=df.distinct();
```

36

## Select

- The **DataFrame select(String col1, .., String coln)** method of the **DataFrame** class returns a new DataFrame that contains only the specified columns of the DataFrame

37

## Select: example

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name, age, and gender of a person
- Create a new DataFrame containing only name and age of the persons

38

## Select: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("SparkSQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame dfProfiles = dfr.json("profiles.json");

// Create a new DataFrame containing only name and age of the
personsDataFrame names_ages= dfProfiles.select("name", "age");
```

39

## Filter

- The **DataFrame filter(String conditionExpr)** method of the **DataFrame** class returns a new DataFrame that contains only the rows satisfying the specified condition
  - The condition is a Boolean expression where each atom of the expression is a comparison between an attribute and a value or between two attributes

40

## Filter: example

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name, age, and gender of a person
- Create a new DataFrame containing only the persons with an age greater than 17

41

## Filter: example

```
// Create a configuration object and set the name of the application
SparkConf conf=new SparkConf().setAppName("SparkSQL");

// Create a Spark Context object
JavaSparkContext sc = new JavaSparkContext(conf);

// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame dfProfiles = dfr.json("profiles.json");

// Select the persons with age > 17
DataFrame greaterThan17= dfProfiles.filter("age>17");
```

42

## From DataFrame to RDD

- The `JavaRDD<Row> javaRDD()` method of the `DataFrame` class returns a JavaRDD of Row objects containing the content of the DataFrame on which it is invoked

43

## From DataFrame to RDD

- The most important methods of `Row` are
  - `java.lang.Object get(int position)`
  - `String getString(int position)`
  - `boolean getBoolean(int position)`
  - `double getDouble(int position)`
    - They retrieve the value of the attribute at position `position`
    - The order of the attributes is important**

44

## Operations on DataFrames: Summary

## Operations on DataFrames: Summary

Operation	Purpose	Example	Result															
<code>void show(int numRows)</code>	Print on the standard output numRows of the DataFrame	<code>dataframe1.show(4)</code>	<table border="1"> <thead> <tr> <th>name</th><th>age</th><th>gender</th></tr> </thead> <tbody> <tr> <td>Michael</td><td>30</td><td>Male</td></tr> <tr> <td>Andy</td><td>19</td><td>Male</td></tr> <tr> <td>Marina</td><td>null</td><td>Female</td></tr> <tr> <td>Andy</td><td>19</td><td>Male</td></tr> </tbody> </table>	name	age	gender	Michael	30	Male	Andy	19	Male	Marina	null	Female	Andy	19	Male
name	age	gender																
Michael	30	Male																
Andy	19	Male																
Marina	null	Female																
Andy	19	Male																
<code>void printSchema()</code>	Print the schema of the data	<code>dataframe1.printSchema()</code>	<table border="1"> <thead> <tr> <th>name</th><th>age</th><th>gender</th></tr> </thead> </table>	name	age	gender												
name	age	gender																
<code>long count()</code>	Count the number of rows	<code>dataframe1.count()</code>	4															
<code>DataFrame distinct()</code>	Return a new DataFrame without duplicates	<code>dataframe2= dataframe1.distinct()</code>	<table border="1"> <thead> <tr> <th>name</th><th>age</th><th>gender</th></tr> </thead> <tbody> <tr> <td>Michael</td><td>30</td><td>Male</td></tr> <tr> <td>Andy</td><td>19</td><td>Male</td></tr> <tr> <td>Marina</td><td>null</td><td>Female</td></tr> </tbody> </table>	name	age	gender	Michael	30	Male	Andy	19	Male	Marina	null	Female			
name	age	gender																
Michael	30	Male																
Andy	19	Male																
Marina	null	Female																

47

## Operations on DataFrames: Summary

- All the examples reported in the following tables are applied on a DataFrame (`dataframe1`) associated with the following data

name	age	gender
Michael	30	Male
Andy	19	Male
Marina	null	Female
Andy	19	Male

46

## Operations on DataFrames: Summary

Operation	Purpose	Example	Result										
<code>DataFrame select(String col1,.., String coln)</code>	Return a new DataFrame containing only the specified attributes of each input row	<code>dataframe2= dataframe1.select ("name","age")</code>	<table border="1"> <thead> <tr> <th>name</th><th>age</th></tr> </thead> <tbody> <tr> <td>Michael</td><td>30</td></tr> <tr> <td>Andy</td><td>19</td></tr> <tr> <td>Marina</td><td>null</td></tr> <tr> <td>Andy</td><td>19</td></tr> </tbody> </table>	name	age	Michael	30	Andy	19	Marina	null	Andy	19
name	age												
Michael	30												
Andy	19												
Marina	null												
Andy	19												
<code>DataFrame filter(String conditionExpr)</code>	Return a new DataFrame containing only the rows satisfying the specified constraint	<code>dataframe2= dataframe1.filter ("age&gt;20")</code>	<table border="1"> <thead> <tr> <th>name</th><th>age</th><th>gender</th></tr> </thead> <tbody> <tr> <td>Michael</td><td>30</td><td>Male</td></tr> </tbody> </table>	name	age	gender	Michael	30	Male				
name	age	gender											
Michael	30	Male											
<code>JavaRDD&lt;Row&gt; javaRDD()</code>	Return an RDD of Rows containing the content of the DataFrame	<code>dataframe1.javaRDD()</code>	<code>[[Michael,30, Male], [Andy, 19, Male], [Marina, null, Female], [Andy, 19, Male]]</code>										

48

## DataFrames and the SQL language

### DataFrames and the SQL language

- Sparks allows querying the content of a DataFrame also by using the SQL language
  - In order to do this a “table name” must be assigned to each DataFrame
- The `void registerTempTable(String tableName)` method of the `DataFrame` class can be used to assign a “table name” to the DataFrame on which it is invoked

50

### DataFrames and the SQL language

- Once the DataFrames have been mapped to “table names” standard SQL queries can be executed
  - The executed queries return DataFrame objects
- The `DataFrame sql(String sqlQueryText)` method of the `SQLContext` class can be used to execute an SQL query
  - `sqlQueryText` is an SQL query
- Currently some SQL features are not supported
  - E.g., nested subqueries in the WHERE clause are not allowed

51

### DataFrames and the SQL language: Example 1

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line of the file contains a json object containing name, age, and gender of a person
- Create a new DataFrame containing only the persons with an age greater than 17
  - Use the SQL language to perform this operation

52

### DataFrames and the SQL language: Example 1

```
// Create an SQL Spark Context object
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);

// Create a DataFrame from a json file
DataFrameReader dfr=sqlContext.read();
DataFrame dfProfiles = dfr.json("profiles.json");

// Assign the “table name” people to the dfProfiles DataFrame
dfProfiles.registerTempTable("people");

// Select the persons with age > 17
DataFrame greaterThan17 =
    sqlContext.sql("SELECT * FROM people WHERE age>17");
```

53

### DataFrames and the SQL language: Example 2

- Create a DataFrame from a json file containing the profiles of a set of persons
  - Each line contains a json object containing userId, name, age, and gender of a person
- Create another DataFrame from a json file containing the liked movie genres of each user
  - Each line contains a json object containing userId, genre
- Create a new DataFrame selecting the preferences of the users with age between 15 and 20

54

## DataFrames and the SQL language: Example 2

```
//Create an SQL Spark Context object  
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);  
  
//Create a DataFrame from the users json file  
DataFrame dfUsers = dfr.json("users.json");  
  
//Assign the "table name" users to the dfUsers DataFrame  
dfUsers.registerTempTable("users");  
  
//Create a DataFrame from the preferences json file  
DataFrame dfPreferences = dfr.json("preferences.json");  
  
//Assign the "table name" preferences to the dfPreferences DataFrame  
dfPreferences.registerTempTable("preferences");
```

55

## DataFrames and the SQL language: Example 2

```
//Select the preferences of the users with age between 15 and 20  
DataFrame selectedGenres =  
    sqlContext.sql("SELECT genre "  
        + "FROM users,preferences "  
        + "WHERE users.userId=preferences.userId "  
        + "      and age>=15 and age<=20");
```

56