Table of contents

1 Spark SQL **1**
1.1 Spark SQL vs Spark RDD APIs . . . 2
1.2 DataFrames . . . 2

2 DataFrames **3**
2.1 Creating DataFrames from csv files . . . 3
2.2 Creating DataFrames from JSON files . . . 4
2.3 Creating DataFrames from other data sources . . . 6
2.4 Creating DataFrames from RDDs or Python lists . . . 6
2.5 From DataFrame to RDD . . . 7
2.5.1 Usage of the Row class . . . 7

3 Operations on DataFrames **8**
3.1 Show method . . . 8
3.2 PrintSchema method . . . 8
3.3 Count method . . . 8
3.4 Distinct method . . . 9
3.5 Select method . . . 9
3.6 SelectExpr method . . . 10
3.7 Filter method . . . 12
3.8 Where method . . . 13
3.9 Join . . . 13
3.10 Aggregate functions . . . 16
3.11 groupBy and aggregate functions . . . 17
3.12 Sort method . . . 19

4 DataFrames and the SQL language **19**

5 Save DataFrames **22**

6 UDFs: User Defines Functions **24**

7 Other notes **25**
7.1 Data warehouse methods: cube and rollup . . . 25
7.2 Set methods . . . 26
7.3 Broadcast join . . . 27
7.4 Execution plan . . . 27

1 Spark SQL

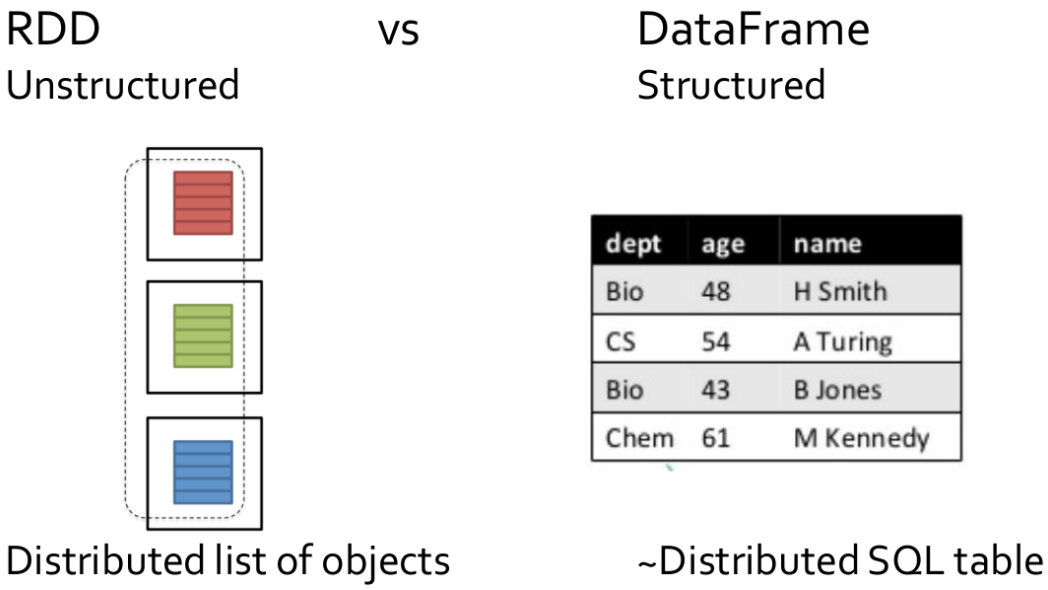
Spark SQL is the Spark component for structured data processing. It provides a programming abstraction called *Dataframe* and can act as a distributed SQL query engine: the input data can be queried by using

- Ad-hoc methods
- Or an SQL-like language

1.1 Spark SQL vs Spark RDD APIs

The interfaces provided by Spark SQL provide more information about the structure of both the data and the computation being performed. Spark SQL uses this extra information to perform extra optimizations based on a "SQL-like" optimizer called **Catalyst**, and so programs based on Dataframe are usually faster than standard RDD-based programs.

Figure 1: Spark SQL vs Spark RDD APIs



1.2 DataFrames

A DataFrame is a distributed collection of structured data. It is conceptually equivalent to a table in a relational database, and it can be created reading data from different types of external sources (CSV files, JSON files, RDBMs,...). A DataFrame benefits from Spark SQL optimized execution engine exploiting the information about the data structure.

All the Spark SQL functionalities are based on an instance of the `pyspark.sql.SparkSession` class

To import it in a standalone application use

```
from pyspark.sql import SparkSession
```

To instance a `SparkSession` object use

```
spark = SparkSession.builder.getOrCreate()
```

To close a `SparkSession` use the `SparkSession.stop()` method

```
spark.stop()
```

2 DataFrames

A DataFrame is a distributed collection of data organized into named columns, equivalent to a relational table: DataFrames are lists of **Row objects**.

The classes used to define DataFrames are

- `pyspark.sql.DataFrame`
- `pyspark.sql.Row`

DataFrames can be created from different sources

- Structured (textual) data files (e.g., csv files, json files);
- Existing RDDs;
- Hive tables;
- External relational databases.

2.1 Creating DataFrames from csv files

Spark SQL provides an API that allows creating DataFrames directly from CSV files. The creation of a DataFrame from a csv file is based the `load(path)` method of the `pyspark.sql.DataFrameReader` class, where `path` is the path of the input file. To get a `DataFrameReader` using the the `read()` method of the `SparkSession` class.

```
1  df = spark.read.load(path, options)
```

i Example

Create a DataFrame from a csv file ("people.csv") containing the profiles of a set of people. Each line of the file contains name and age of a person, and age can assume the null value (i.e., it can be missing). The first line contains the header (i.e., the names of the attributes/columns).
Example of csv file

```
name,age
Andy,30
Michael,
Justin,19
```

Notice that the age of the second person is unknown.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)
```

`format="csv"`	This is used to specify the format of the input file
`header=True`	This is used to specify that the first line of the file contains the name of the attributes/columns
`inferSchema=True`	This method is used to specify that the system must infer the data types of each column. Without this option all columns are considered strings

2.2 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. So, the format of the input file must be compliant with the **JSON Lines text format**, also called newline-delimited JSON.

The creation of a DataFrame from JSON files is based on the same method used for reading csv files, that is the `load(path)` method of the `pyspark.sql.DataFrameReader` class, where `path` is the path of the input file. To get a DataFrameReader use the `read()` method of the `SparkSession` class.

```
df = spark.read.load(path, format="json", options)
```

or

```
df = spark.read.json(path, options)
```

The same API allows also reading standard multiline JSON files by setting the multiline option to true by setting the argument `multiLine=True` on the defined `DataFrameReader` for reading standard JSON files (this feature is available since Spark 2.2.0).

> **Danger**
>
> Pay attention that reading a set of small JSON files from HDFS is very slow.

Example 1

Create a DataFrame from a JSON Lines text formatted file ("people.json") containing the profiles of a set of people: each line of the file contains a JSON object containing name and age of a person. Age can assume the null value.
Example of JSON file

```
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

Notice that the age of the first person is unknown.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.json",
    format="json"
)
```

`format="json"`	This method is used to specify the format of the input file.

Example 2

Create a DataFrame from a folder containing a set of standard multiline JSON files: each input JSON file contains the profile of one person, in particular each file contains name and age of a person. Age can assume the null value.
Example of JSON file

```
{"name":"Andy", "age":30}
```

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "folder_JSONFiles/",
    format="json",
    multiLine=True
)
```

`multiLine=True`	This multiline option is set to true to specify that the input files are standard multiline JSON files.

2.3 Creating DataFrames from other data sources

-
-
-
-

2.4 Creating DataFrames from RDDs or Python lists

Example

Create a DataFrame from the following Python list

```
[
    (19,"Justin"),
    (30,"Andy"),
    (None,"Michael")
]
```

The column names must be set to “age” and “name”.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a Python list of tuples
profilesList = [
    (19,"Justin"),
    (30,"Andy"),
    (None,"Michael")
]

# Create a DataFrame from the profilesList
df = spark.createDataFrame(profilesList,["age","name"])
```

2.5 From DataFrame to RDD

2.5.1 Usage of the Row class

-
-
-

Example

1. Create a DataFrame from a csv file containing the profiles of a set of people: each line of the file contains name and age of a person, but the first line contains the header (i.e., the name of the attributes/columns);
2. Transform the input DataFrame into an RDD, select only the name field/column and store the result in the output folder.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Define an RDD based on the content of
# the DataFrame
rddRows = df.rdd

# Use the map transformation to extract
# the name field/column
rddNames = rddRows.map(lambda row: row.name)

# Store the result
rddNames.saveAsTextFile(outputPath)
```

3 Operations on DataFrames

3.1 Show method

Example

1. Create a DataFrame from a csv file containing the profiles of a set of people;
2. Print the content of the first 2 people (i.e., the first 2 rows of the DataFrame).

The content of people.csv is

```
name,age
Andy,30
Michael,
Justin,19
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

df.show(2)
```

3.2 PrintSchema method

3.3 Count method

3.4 Distinct method

Danger

Pay attention that the distinct operation is always an heavy operation in terms of data sent on the network.

Example

1. Create a DataFrame from a csv file containing the names of a set of people. The first line is the header.
2. Create a new DataFrame without duplicates.

The content of "names.csv" is

```
name
Andy
Michael
Justin
Michael
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from names.csv
df = spark.read.load(
    "names.csv",
    format="csv",
    header=True,
    inferSchema=True
)

df_distinct = df.distinct()
```

3.5 Select method

Danger

Pay attention that the select method can generate errors at runtime if there are mistakes in the names of the columns.

Example

1. Create a DataFrame from the "people2.csv" file that scontains the profiles of a set of people
    - The first line contains the header;
    - The others lines contain the users' profiles: one line per person, and each line contains name, age, and gender of a person.
2. Create a new DataFrame containing only name and age of the people.

The "people2.csv" has the following structure

```
name,age,gender
Paul,40,male
John,40,male
```

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people2.csv
df = spark.read.load(
    "people2.csv",
    format="csv",
    header=True,
    inferSchema=True
)

dfNamesAges = df.select("name","age")
```

3.6 SelectExpr method

Example 1

1. Create a DataFrame from the "people2.csv" file that scontains the profiles of a set of people
    - The first line contains the header;
    - The others lines contain the users' profiles: one line per person, and each line contains name, age, and gender of a person.
2. Create a new DataFrame containing only name and age of the people.
3. Create a new DataFrame containing only the name of the people and their age plus one. Call the age column as "new_age".

The "people2.csv" has the following structure

```
name,age,gender
Paul,40,male
John,40,male
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people2.csv
df = spark.read.load(
    "people2.csv",
    format="csv",
    header=True,
    inferSchema=True
)

dfNamesAges = df.selectExpr("name","age")

dfNamesAgesMod = df.selectExpr("name", "age + 1 AS new_age")
```

Example 2

1. Create a DataFrame from the "people2.csv" file that contains the profiles of a set of people

- The first line contains the header;
- The others lines contain the users' profiles: each line contains name, age, and gender of a person.

2. Create a new DataFrame containing the same columns of the initial dataset plus an additional column called "newAge" containing the value of age incremented by one.

The "people2.csv" has the following structure

```
name,age,gender
Paul,40,male
John,40,male
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people2.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Create a new DataFrame with four columns:
# name, age, gender, newAge = age +1
dfNewAge = df.selectExpr(
    "name",
    "age",
    "gender",
    "age+1 as newAge"
)
```

`"... as newAge"`	This part of the expression is used to specify the name of the column associated with the result of the first part of the expression in the returned DataFrame. Without this part of the expression, the name of the returned column would be "age+1".

3.7 Filter method

Danger

Pay attention that this version of the filter method can generate errors at runtime if there are errors in the filter expression: the parameter is a string and the system cannot check the correctness of the expression at compile time.

Example

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people
   - The first line contains the header;
   - The others lines contain the users' profiles: each line contains name and age of a person.
2. Create a new DataFrame containing only the people with age between 20 and 31.

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

df_filtered = df.filter("age>=20 and age<=31")
```

3.8 Where method

3.9 Join

Danger

Pay attention that this method: can generate errors at runtime if there are errors in the join expression.

Example 1

1. Create two DataFrames
   - One based on the "people_id.csv" file that contains the profiles of a set of people, the schema is: uid, name, age;
   - One based on the liked_sports.csv file that contains the liked sports for each person, the schema is: uid, sportname. 2.Join the content of the two DataFrames (uid is the join column) and show it on the standard output.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Read people_id.csv and store it in a DataFrame
dfPeople = spark.read.load(
    "people_id.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Read liked_sports.csv and store it in a DataFrame
dfUidSports = spark.read.load(
    "liked_sports.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Join the two input DataFrames
dfPersonLikes = dfPeople.join(
    dfUidSports,
    dfPeople.uid == dfUidSports.uid
)

# Print the result on the standard output
dfPersonLikes.show()
```

`dfPeople.uid == dfUidSports.uid`	Specify the join condition on the uid columns.

Example 2

1. Create two DataFrames
   - One based on the "people_id.csv" file that contains the profiles of a set of people, the schema is: uid, name, age;
   - One based on the banned.csv file that contains the banned users, the schema is: uid, bannedmotivation.
2. Select the profiles of the non-banned users and show them on the standard output.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Read people_id.csv and store it in a DataFrame
dfPeople = spark.read.load(
    "people_id.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Read banned.csv and store it in a DataFrame
dfBannedUsers = spark.read.load(
    "banned.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Apply the Left Anti Join on the two input DataFrames
dfSelectedProfiles = dfPeople.join(
    dfBannedUsers,
    dfPeople.uid == dfBannedUsers.uid,
    "left_anti"
)

# Print the result on the standard output
dfSelectedProfiles.show()
```

`dfPeople.uid == dfUidSports.uid`	Specify the (anti) join condition on the uid columns.
`"left_anti"`	Use Left Anti Join.

3.10 Aggregate functions

-
-
-
-
-

Danger

Pay attention that this methods can generate errors at runtime (e.g., wrong attribute name, wrong data type).

Example

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people (each line contains name and age of a person)
   - The first line contains the header;
   - The others lines contain the users' profiles.
2. Create a Dataset containing the average value of age.

Input file example

```
name,age
Andy,30
Michael,15
Justin,19
Andy,40
```

Expected output example

```
avg(age)
26.0
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Compute the average of age
averageAge = df.agg({"age": "avg"})
```

3.11 groupBy and aggregate functions

Danger

Pay attention that this methods can generate errors at runtime if there are semantic errors (e.g., wrong attribute names, wrong data types).

-
-
-
-
-

Example 1

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people

- The first line contains the header;
- The others lines contain the users' profiles: each line contains name and age of a person.

2. Create a DataFrame containing the for each name the average value of age.

Input file example

```
name,age
Andy,30
Michael,15
Justin,19
Andy,40
```

Expected output example

```
name,avg(age)
Andy,35
Michael,15
Justin,19
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

grouped = df.groupBy("name").avg("age")
```

Example 2

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people
   - The first line contains the header
   - The others lines contain the users' profiles: each line contains name and age of a person
2. Create a DataFrame containing the for each name the average value of age and the number of person with that name

Input file example

```
name,age
Andy,30
Michael,15
Justin,19
Andy,40
```

Expected output example

```
name,avg(age),count(name)
Andy,35,2
Michael,15,1
Justin,19,1
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

grouped = df.groupBy("name") \
    .agg({"age": "avg", "name": "count"})
```

3.12 Sort method

4 DataFrames and the SQL language

Example 1

1. Create a DataFrame from a JSON file containing the profiles of a set of people: each line of the file contains a JSON object containing name, age, and gender of a person;
2. Create a new DataFrame containing only the people with age between 20 and 31 and print

them on the standard output (use the SQL language to perform this operation).

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.json",
    format="json"
)

# Assign the "table name" people to the df DataFrame
df.createOrReplaceTempView("people")

# Select the people with age between 20 and 31
# by querying the people table
selectedPeople = spark.sql(
    "SELECT * FROM people WHERE age>=20 and age<=31"
)

# Print the result on the standard output
selectedPeople.show()
```

Example 2

1. Create two DataFrames
   - One based on the "people_id.csv" file that contains the profiles of a set of people, the schema is: uid, name, age;
   - One based on the "liked_sports.csv" file that contains the liked sports for each person, the schema is: uid, sportname.
2. Join the content of the two DataFrames and show it on the standard output.

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Read people_id.csv and store it in a DataFrame
dfPeople = spark.read.load(
    "people_id.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Assign the "table name" people to the dfPerson
dfPeople.createOrReplaceTempView("people")

# Read liked_sports.csv and store it in a DataFrame
dfUidSports = spark.read.load(
    "liked_sports.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Assign the "table name" liked to dfUidSports
dfUidSports.createOrReplaceTempView("liked")

# Join the two input tables by using the
#SQL-like syntax
dfPersonLikes = spark.sql(
    "SELECT * from people, liked where people.uid=liked.uid"
)

# Print the result on the standard output
dfPersonLikes.show()
```

Example 3

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people
   - The first line contains the header;
   - The others lines contain the users' profiles: each line contains name and age of a person.
2. Create a DataFrame containing for each name the average value of age and the number of person with that name. Print its content on the standard output.

Input file example

```
name,age
```

```
Andy,30
Michael,15
Justin,19
Andy,40
```

Expected output example

```
name,avg(age),count(name)
Andy,35,2
Michael,15,1
Justin,19,1
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.json",
    format="json"
)

# Assign the "table name" people to the df DataFrame
df.createOrReplaceTempView("people")

# Define groups based on the value of name and
# compute average and number of records for each group
nameAvgAgeCount = spark.sql(
    "SELECT name, avg(age), count(name) FROM people GROUP BY name"
)

# Print the result on the standard output
nameAvgAgeCount.show()
```

5 Save DataFrames

-
-

Example 1

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people
   - The first line contains the header;
   - The others lines contain the users' profiles: each line contains name, age, and gender of a person.
2. Store the DataFrame in the output folder by using the `saveAsTextFile()` method.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Save it
df.rdd.saveAsTextFile(outputPath)
```

Example 2

1. Create a DataFrame from the "people.csv" file that contains the profiles of a set of people
   - The first line contains the header;
   - The others lines contain the users' profiles: each line contains name, age, and gender of a person.
2. Store the DataFrame in the output folder by using the `write()` method, with the CSV format.

```
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Create a DataFrame from people.csv
df = spark.read.load(
    "people.csv",
    format="csv",
    header=True,
    inferSchema=True
)

# Save it
df.write.csv(outputPath, header=True)
```

6 UDFs: User Defines Functions

-
-
-

-
-
  - –
  - –
-

> Example
>
> Define a UDFs that, given a string, returns the length of the string.
>
> ```
> # Create a Spark Session object
> spark = SparkSession.builder.getOrCreate()
>
> # Define the UDF
> # name: length
> # output: integer value
> spark.udf.register("length", lambda x: len(x))
> ```
>
> Use of the defined UDF in a selectExpr transformation.
>
> ```
> result = inputDF.selectExpr("length(name) as size")
> ```
>
> Use of the defined UDF in a SQL query.
>
> ```
> result = spark.sql("SELECT length(name) FROM profiles")
> ```

7 Other notes

7.1 Data warehouse methods: cube and rollup

Example

1. Create a DataFrame from the "purchases.csv" file
   - The first line contains the header;
   - The others lines contain the quantities of purchased products by users: each line contains userid, productid, quantity.
2. Create a first DataFrame containing the result of the cube method. Define one group for each pair userid, productid and compute the sum of quantity in each group; 3.Create a second DataFrame containing the result of the rollup method. Define one group for each pair userid, productid and compute the sum of quantity in each group.

Input file

```
userid,productid,quantity
u1,p1,10
u1,p1,20
u1,p2,20
u1,p3,10
u2,p1,20
u2,p3,40
u2,p3,30
```

Expected output - cube

```
userid,productid,sum(quantity)
null    null    150
null    p1      50
null    p2      20
null    p3      80
u1      null    60
u1      p1      30
u1      p2      20
u1      p3      10
```

```
u2      null    90
u2      p1      20
u2      p3      70
```

Expected output - rollup

```
userid,productid,sum(quantity)
null    null    150
u1      null    60
u1      p1      30
u1      p2      20
u1      p3      10
u2      null    90
u2      p1      20
u2      p3      70
```

```python
# Create a Spark Session object
spark = SparkSession.builder.getOrCreate()

# Read purchases.csv and store it in a DataFrame
dfPurchases = spark.read.load(
    "purchases.csv",
    format="csv",
    header=True,
    inferSchema=True
)

dfCube=dfPurchases \
    .cube("userid","productid") \
    .agg({"quantity": "sum"})

dfRollup=dfPurchases \
    .rollup("userid","productid")\
    .agg({"quantity": "sum"})
```

7.2 Set methods

-
-
-

7.3 Broadcast join

Example

```
dfPersonLikesBroadcast = dfUidSports\
    .join(
        broadcast(dfPersons),
        dfPersons.uid == dfUidSports.uid
    )
```

`broadcast(dfPersons)`	In this case we specify that `dfPersons` must be broadcasted and hence Spark will execute the join operation by using a broadcast join.

7.4 Execution plan