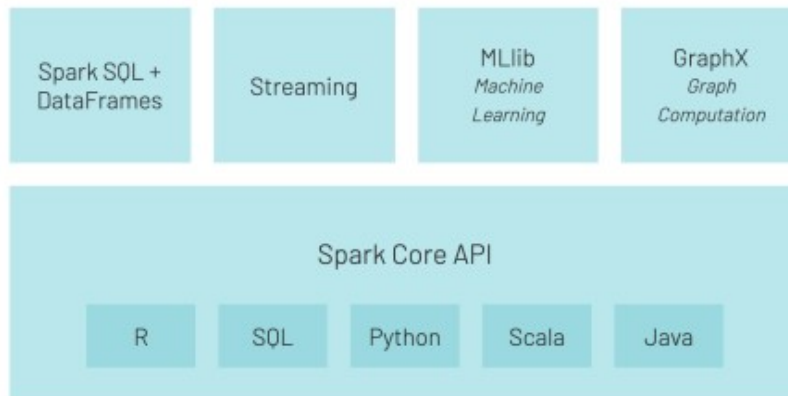


And in the following blogposts we will explore the core engine and services on top:

- Spark SQL+ Dataframes
- Streaming
- MLlib – Machine learning library
- GraphX – Graph computations



Apache Spark is a powerful open-source processing engine built around speed, ease of use, and sophisticated analytics.

Spark Core is underlying general execution engine for the Spark Platform with all other functionalities built-in. It is in memory computing engine that provides variety of language support, as Scala, R, Python for easier data engineering development and machine learning development.

Spark has three key interfaces:

- **Resilient Distributed Dataset (RDD)** – It is an interface to a sequence of data objects that consist of one or more types that are located across a collection of machines (a cluster). RDDs can be created in a variety of ways and are the “lowest level” API available. While this is the original data structure for Apache Spark, you should focus on the DataFrame API, which is a superset of the RDD functionality. The RDD API is available in the Java, Python, and Scala languages.
- **DataFrame** – similar in concept to the DataFrame you will find with the pandas Python library and the R language. The DataFrame API is available in the Java, Python, R, and Scala languages.
- **Dataset** – is combination of RDD and DataFrame. It proved typed interface of RDD and gives you the convenience of the DataFrame. The Dataset API is available only for Scala and Java.

In general, when you will be working with the performance optimisations, either DataFrames or Datasets should be enough. But when going into more advanced components of Spark, it may be necessary to use RDDs. Also the visualisation within Spark UI references directly RDDs.

1.Datasets

Let us start with Databricks datasets, that are available within every workspace and are here mainly for test purposes. This is nothing new; both Python and R come with sample datasets. For example the Iris dataset that is available with Base R engine and Seaborn Python package. Same goes with Databricks and sample dataset can be found in `/databricks-datasets` folder.

Create a new notebook in your workspace and name it Day21_Scala. Language: Scala. And run the following Scala command.

```
display(dbutils.fs.ls("/databricks-datasets"))
```

Cnd 3

```
1 display(dbutils.fs.ls("/databricks-datasets"))
```

	path	name	size
1	dbfs:/databricks-datasets/COVID/	COVID/	0
2	dbfs:/databricks-datasets/README.md	README.md	976
3	dbfs:/databricks-datasets/Rdatasets/	Rdatasets/	0
4	dbfs:/databricks-datasets/SPARK_README.md	SPARK_README.md	3359
5	dbfs:/databricks-datasets/adult/	adult/	0
6	dbfs:/databricks-datasets/airlines/	airlines/	0
7	dbfs:/databricks-datasets/amazon/	amazon/	0
8	dbfs:/databricks-datasets/asa/	asa/	0
9	dbfs:/databricks-datasets/atlas_higgs/	atlas_higgs/	0
10	dbfs:/databricks-datasets/bikeSharing/	bikeSharing/	0
11	dbfs:/databricks-datasets/cctvVideos/	cctvVideos/	0
12	dbfs:/databricks-datasets/credit-card-fraud/	credit-card-fraud/	0
13	dbfs:/databricks-datasets/cs100/	cs100/	0
14	dbfs:/databricks-datasets/cs110x/	cs110x/	0
15	dbfs:/databricks-datasets/cs190/	cs190/	0
16	dbfs:/databricks-datasets/data.gov/	data.gov/	0
17	dbfs:/databricks-datasets/definitive-guide/	definitive-guide/	0
18	dbfs:/databricks-datasets/flights/	flights/	0
19	dbfs:/databricks-datasets/flower_photos/	flower_photos/	0
20	dbfs:/databricks-datasets/flowers/	flowers/	0
21	dbfs:/databricks-datasets/genomics/	genomics/	0
22	dbfs:/databricks-datasets/hail/	hail/	0

Showing all 49 rows.

Command took 1.15 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 16:44:19 on databricks_cli_Standard

You can always store the results to variable and later use is multiple times:

```
// transformation
val textFile = spark.read.textFile("/databricks-datasets/samples/
docs/README.md")
```

and listing the content of the variable by using **show()** function:

```
textFile.show()
```

Cnd 7

```
1 textFile.show()
```

▶ (1) Spark Jobs

value
Welcome to the Sp...
This readme will ...
here with the Spa...
Spark at http://s...
Read on to learn ...
documentation you...
whichever version...
Prerequisites
The Spark documen...
Python and R. To ...
\$ sudo gem in...
\$ sudo gem in...
\$ sudo pip in...
\$ sudo pip in...

Command took 9.02 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020,

And some other useful functions; to count all the lines in textfile, to show the first line and to filter the text file showing only the lines containing the search argument (word sudo).

```
// Count number or lines in textFile
```

```

textFile.count()
// Show the first line of the textFile
textFile.first()
// show all the lines with word Sudo
val linesWithSudo = textFile.filter(line => line.contains("sudo"))

```

```

Cmd 10
1 // show all the lines with word Sudo
2 val linesWithSudo = textFile.filter(line => line.contains("sudo"))

linesWithSudo: org.apache.spark.sql.Dataset[String] = [value: string]
linesWithSudo: org.apache.spark.sql.Dataset[String] = [value: string]
Command took 0.33 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 16:55:07 on databricks_cli_standard

Cmd 11
1 linesWithSudo.show()

(1) Spark Jobs
+-----+
| value |
+-----+
| $ sudo gem in... |
| $ sudo gem in... |
| $ sudo pip in... |
| $ sudo pip in... |
+-----+

Command took 1.18 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 16:55:10 on databricks_cli_standard

Cmd 12

```

And also printing all (first four) lines of with the subset of text containing the word “sudo”. In the second example finding the Line number with most words:

```

// Output the all four lines
linesWithSudo.collect().take(4).foreach(println)
// find the lines with most words
textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)

```

```

Cmd 14
1 // Output the all four lines
2 linesWithSudo.collect().take(4).foreach(println)

(1) Spark Jobs
$ sudo gem install jekyll
$ sudo gem install jekyll-redirect-from
$ sudo pip install Pygments
$ sudo pip install sphinx

Command took 1.25 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 16:56:26 on databricks_cli_standard

Cmd 15
And example of map-reduce formula. This following command will:
• map - get all the lines
• map - split the words
• map - get the number of words (or spaces)
• reduce - compare the lines to reduce it to the largest one (or to find the largest one)

Cmd 16
1 // find the lines with most words
2 textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)

(1) Spark Jobs
res18: Int = 19

Command took 1.62 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 16:58:55 on databricks_cli_standard

Cmd 17

```

2. Create a dataset

Now let's create a dataset (remember the difference between *Dataset* and *DataFrame*) and load some

data from */databricks-datasets* folder.

```
val df = spark.read.json("/databricks-datasets/samples/people/people.json")
```

3. Convert Dataset to DataFrame

We can also convert Dataset to DataFrame for easier operation and usage. We must define a class that represents a type-specific Scala JVM object (like a schema) and now repeat the same process with definition.

```
case class Person (name: String, age: Long)
val ds = spark.read.json("/databricks-datasets/samples/people/
people.json").as[Person]
```

We can also create and define another dataset, taken from the */databricks-datasets* folder that is in JSON (flattened) format:

```
// define a case class that represents the device data.
case class DeviceIoTData (
  battery_level: Long,
  c02_level: Long,
  cca2: String,
  cca3: String,
  cn: String,
  device_id: Long,
  device_name: String,
  humidity: Long,
  ip: String,
  latitude: Double,
  longitude: Double,
  scale: String,
  temp: Long,
  timestamp: Long
)

val ds = spark.read.json("/databricks-datasets/iot/iot_devices.json"
).as[DeviceIoTData]
```

Add another dataset

Cnd 28

```
1 // define a case class that represents the device data.
2 case class DeviceIoTData (
3   battery_level: Long,
4   c02_level: Long,
5   cca2: String,
6   cca3: String,
7   cn: String,
8   device_id: Long,
9   device_name: String,
10  humidity: Long,
11  ip: String,
12  latitude: Double,
13  longitude: Double,
14  scale: String,
15  temp: Long,
16  timestamp: Long
17 )
```

defined class DeviceIoTData

Command took 0.28 seconds -- by tenaz.kastrun@gmail.com at 20/12/2020, 18:54:49 on databricks_cli_Standard

Cnd 29

```
1 // read the JSON file and create the Dataset from the ``case class`` DeviceIoTData
2 // ds is now a collection of JVM Scala objects DeviceIoTData
3 val ds = spark.read.json("/databricks-datasets/iot/iot_devices.json").as[DeviceIoTData]
```

▶ (1) Spark Jobs

▶  ds: org.apache.spark.sql.Dataset[DeviceIoTData] = [battery_level: long, c02_level: long ... 13 more fields]

ds: org.apache.spark.sql.Dataset[DeviceIoTData] = [battery_level: bigint, c02_level: bigint ... 13 more fields]

Command took 5.77 seconds -- by tenaz.kastrun@gmail.com at 20/12/2020, 18:54:53 on databricks_cli_Standard

and run show() function to see the imported Dataset from JSON file:

1 ds.show()

▶ (1) Spark Jobs

	battery_level	c02_level	cca2	cca3	cn	device_id	device_name	humidity	ip	latitude	lon	longitude	scale	temp	timestamp
1	808	868	US	USA	United States	1	meter-gauge-1xbVRYcj	51	68.161.225.1	38.0	green	-97.0	Celsius	34	1458444054093
2	1473	1473	NO	NOR	Norway	2	sensor-pad-2n2Pea	70	213.161.254.1	62.47	red	6.15	Celsius	13	1458444054119
3	1556	1556	IT	ITA	Italy	3	device-mac-36TK5K1T	44	88.36.5.1	42.83	red	12.83	Celsius	19	1458444054120
4	1080	1080	US	USA	United States	4	sensor-pad-4m2Wkz	32	66.39.173.154	44.06	yellow	-121.32	Celsius	28	1458444054121
5	931	931	PH	PHL	Philippines	5	therm-stick-5gimp...	62	203.82.41.9	14.58	green	120.97	Celsius	25	1458444054122
6	1129	1129	US	USA	United States	6	sensor-pad-6a17RT...	51	204.116.185.67	35.93	yellow	-85.46	Celsius	27	1458444054122
7	1536	1536	CN	CHN	China	7	meter-gauge-70e0amM	26	220.173.179.1	22.82	yellow	108.32	Celsius	18	1458444054123
8	887	887	JP	JPN	Japan	8	sensor-pad-8a1U6p...	35	210.173.177.1	35.69	red	139.69	Celsius	27	1458444054123
9	1470	1470	US	USA	United States	9	device-mac-90cJ22pw	85	118.23.68.227	35.69	green	139.69	Celsius	13	1458444054124
10	1544	1544	IT	ITA	Italy	10	sensor-pad-100syw...	56	200.109.163.218	33.61	red	-111.89	Celsius	26	1458444054125
11	1260	1260	US	USA	United States	11	meter-gauge-11d1W...	85	88.213.193.34	42.83	red	12.83	Celsius	16	1458444054125
12	1087	1087	IN	IND	India	12	sensor-pad-12V2kIm0	92	68.28.93.22	38.0	yellow	-97.0	Celsius	12	1458444054126
13	1346	1346	NO	NOR	Norway	13	meter-gauge-136r0...	92	59.144.134.258	28.6	yellow	77.2	Celsius	13	1458444054127
14	1259	1259	US	USA	United States	14	sensor-pad-14Q193...	90	193.156.90.200	59.95	yellow	18.75	Celsius	1	1458444054127
15	1425	1425	US	USA	United States	15	device-mac-155e6n2	79	67.185.72.1	47.41	yellow	-122.0	Celsius	13	1458444054128
16	1466	1466	US	USA	United States	16	sensor-pad-16a8e1...	53	68.85.85.106	38.0	red	-97.0	Celsius	15	1458444054128
17	1096	1096	CN	CHN	China	17	meter-gauge-172b8...	99	161.188.232.254	39.95	red	-75.16	Celsius	31	1458444054129
18						18	sensor-pad-18XULN9tv	25	221.3.128.242	25.04	yellow	102.72	Celsius	31	1458444054130

Command took 1.33 seconds -- by tenaz.kastrun@gmail.com at 20/12/2020, 18:55:18 on databricks_cli_Standard

Cnd 31

Now let's play with the dataset using Scala Dataset API with following frequently used functions:

- display(),
- describe(),
- sum(),
- count(),
- select(),
- avg(),
- filter(),
- map() or where(),
- groupBy(),
- join(), and
- union().

display()

You can also view the dataset using display() (similar to .show() function):

```
display(ds)
```

display()

You can also view the dataset using display()

Cmd 32

```
1 display(ds)
```

(1) Spark Jobs

	battery_level	c02_level	cca2	cca3	cn	device_id	device_name	humidity	ip	latitude
1	8	868	US	USA	United States	1	meter-gauge-1xbYFYcj	51	88.161.225.1	38
2	7	1473	NO	NOR	Norway	2	sensor-pad-2n2Pee	70	213.161.294.1	62.47
3	2	1556	IT	ITA	Italy	3	device-mac-36TWSKt	44	88.36.5.1	42.83
4	6	1080	US	USA	United States	4	sensor-pad-4mZWkz	32	66.39.173.154	44.06
5	4	931	PH	PHL	Philippines	5	them-stick-5gimpUrBB	62	203.82.41.9	14.58
6	3	1210	US	USA	United States	6	sensor-pad-6aI7RTAcobR	51	204.116.105.67	35.93
7	3	1129	CN	CHN	China	7	meter-gauge-7GaDoanM	26	220.173.179.1	22.82

Showing the first 1000 rows.

Command took 2.12 seconds -- by tonaz.kastrun@gmail.com at 28/12/2020, 19:23:15 on databricks_cli_Standard

describe()

Describe() function is great for exploring the data and the structure of the data:

```
ds.describe()
```

Cmd 36

```
1 ds.describe()
```

(2) Spark Jobs

res77: org.apache.spark.sql.DataFrame

```
summary: string
battery_level: string
c02_level: string
cca2: string
cca3: string
cn: string
device_id: string
device_name: string
humidity: string
ip: string
latitude: string
lcd: string
longitude: string
scale: string
temp: string
timestamp: string

res77: org.apache.spark.sql.DataFrame = [summary: string, battery_level: string ..
```

Command took 3.68 seconds -- by tonaz.kastrun@gmail.com at 28/12/2020, 19:36:55 on databricks_cli_Standard

or for getting descriptive statistics of the Dataset or of particular column:

```
display(ds.describe())
// or for column
display(ds.describe("c02_level"))
```

Cod 37

```
1 display(ds.describe())
```

↳ (2) Spark Jobs

	summary	battery_level	c02_level	cca2	cca3	cn	device_id
1	count	198164	198164	198164	198164	198164	198164
2	mean	4.4997678690377665	1199.7639429967098	null	null	null	99062.5
3	stddev	2.8733916884106137	231.06002562900773	null	null	null	57205.163705
4	min	0	800	AD	ABW		1
5	max	9	1599	ZW	ZWE	Åland	198164

Showing all 5 rows.

Command took 3.49 seconds -- by tomasz.kastrun@gmail.com at 28/12/2020, 19:38:14 on databricks_cli_standard

Cod 38

or get more descriptive statistics over one particular column

Cod 39

```
1 display(ds.describe("c02_level"))
```

↳ (2) Spark Jobs

	summary	c02_level
1	count	198164
2	mean	1199.7639429967098
3	stddev	231.06002562900773
4	min	800
5	max	1599

Showing all 5 rows.

sum()

Let's sum all c02_level values:

```
//create a variable sum_c02_1 and sum_c02_2;  
// both are correct and return same results
```

```
val sum_c02_1 = ds.select("c02_level").groupBy().sum()  
val sum_c02_2 = ds.groupBy().sum("c02_level")
```

```
display(sum_c02_1)
```

```
Cnd 43
1 //create a variable sum_c02_1 and sum_c02_2;
2 // both are correct and return same results
3
4 val sum_c02_1 = ds.select("c02_level").groupBy().sum()
5 val sum_c02_2 = ds.groupBy().sum("c02_level")
6
7 display(sum_c02_1)

▶ (2) Spark Jobs
▶ sum_c02_1: org.apache.spark.sql.DataFrame = [sum(c02_level): long]
▶ sum_c02_2: org.apache.spark.sql.DataFrame = [sum(c02_level): long]

sum(c02_level)
1 237750022
Showing all 1 rows.

Command took 2.92 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 19:08:27 on databricks_cli_Standard

Cnd 44
1 display(sum_c02_2)

▶ (2) Spark Jobs

sum(c02_level)
1 237750022
Showing all 1 rows.

Command took 2.79 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 19:08:19 on databricks_cli_Standard

Cnd 45
```

And we can also double check the result of this sum with SQL. Just because it is fun. But first We need to create a SQL view (or it could be a table) from this dataset.

```
ds.createOrReplaceTempView("SQL_iot_table")
```

And then define cell as SQL statement, using **%sql**. Remember, complete code today is written in Scala, unless otherwise stated with **%{lang}** and the beginning.

```
%sql
SELECT sum(c02_level) as Total_c02_level FROM SQL_iot_table
```

```
Cnd 46
1 ds.createOrReplaceTempView("SQL_iot_table")

Command took 0.18 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 19:10:21 on databricks_cli_Standard

Cnd 47
1 %sql
2 SELECT sum(c02_level) as Total_c02_level FROM SQL_iot_table

▶ (2) Spark Jobs

Total_c02_level
1 237750022
Showing all 1 rows.

Command took 2.67 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 19:11:56 on databricks_cli_Standard

Cnd 48
```

And for sure, we get the same result (!).

select()

Select() function will let you show only the columns you want to see.


```
// Both will return same results
ds.select("cca2","cca3", "c02_level").show()
// or
display(ds.select("cca2","cca3","c02_level"))
```

Cmd 49

```
1 // Both will return same results
2 ds.select("cca2","cca3", "c02_level").show()
3 // or
4 display(ds.select("cca2","cca3","c02_level"))
```

▶ (2) Spark Jobs

	cca2	cca3	c02_level
1	US	USA	868
2	NO	NOR	1473
3	IT	ITA	1556
4	US	USA	1080
5	PH	PHL	931
6	US	USA	1210
7	CN	CHN	1129

Showing the first 1000 rows.

Command took 2.69 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 19:28:13 on databb

avg()

Avg() function will let you aggregate a column (let us take: c02_level) over another column (let us take: countries in variable cca3). First we want to calculate average value over the complete dataset:

```
val avg_c02 = ds.groupBy().avg("c02_level")
```

```
display(avg_c02)
```

And then also the average value for each country:

```
val avg_c02_byCountry = ds.groupBy("cca3").avg("c02_level")
```

```
display(avg_c02_byCountry)
```

```

1 val avg_c02 = ds.groupBy().avg("c02_level")
2
3 display(avg_c02)

```

⚡ (2) Spark Jobs

avg_c02: org.apache.spark.sql.DataFrame = [avg(c02_level): double]

	avg(c02_level)
1	1199.7639429967068

Showing all 1 rows.

Command took 3.01 seconds -- by tomas.kastrun@gmail.com at 29/12/2020, 19:38:22 on databricks_cli_standard

```

1 val avg_c02_byCountry = ds.groupBy("cca3").avg("c02_level")
2
3 display(avg_c02_byCountry)

```

⚡ (2) Spark Jobs

avg_c02_byCountry: org.apache.spark.sql.DataFrame = [cca3: string, avg(c02_level): double]

	cca3	avg(c02_level)
1	HTI	1291.3333333333333
2	PSE	1244.64
3	POL	1193.7452629906706
4	LVA	1189.1340782122904
5	BRB	1257.5526315789473
6	JAM	1162.2272727272727
7	ZMB	1151.857142857143

Showing all 205 rows.

filter()

Filter() function will shorten or filter out the values that will not comply with the condition. Filter() function can also be replaced by where() function; they both have similar behaviour.

Following command will return dataset that meet the condition where batter_level is greater than 7.

```
display(ds.filter(d => d.battery_level > 7))
```

And the following command will filter the database on same condition, but only return the specify columns (in comparison with previous command which returned all columns):

```
display(ds.filter(d => d.battery_level > 7).select("battery_level",
"c02_level", "cca3"))
```

```

1 display(ds.filter(d => d.battery_level > 7))
2 // [[ d.cca3 => "USA" ]]

```

⚡ (10) Spark Jobs

	battery_level	c02_level	cca2	cca3	en	device_id	device_name	humidity	ip	latitude
1	8	806	US	USA	United States	1	meter-gauge-1x0YRrcj	51	95.161.225.1	35
2	9	1009	US	USA	United States	15	device-mac-15ee5m2	73	87.185.72.1	47.41
3	9	1031	US	USA	United States	19	meter-gauge-18eq18yKCG	76	94.124.182.216	36
4	9	1511	CA	CAN	Canada	24	sensor-pad-04PzcD06Cp	73	69.06.67.1	43.85
5	9	1000	JP	JPN	Japan	26	sensor-pad-06AyCZ000H9	27	216.156.147.11	35.69
6	8	1062	US	USA	United States	32	sensor-pad-0298vYt	30	128.83.9.148	30.3
7	8	857	US	USA	United States	48	sensor-pad-48Mq33Uk0aa	63	95.5.48.255	36.17

Showing the first 1000 rows.

Command took 2.28 seconds -- by tomas.kastrun@gmail.com at 30/12/2020, 20:42:53 on databricks_cli_standard

```

1 display(ds.filter(d => d.battery_level > 7).select("battery_level", "c02_level", "cca3"))

```

⚡ (10) Spark Jobs

	battery_level	c02_level	cca3
1	8	801	CHV
2	8	802	USA
3	8	802	USA
4	8	802	USA
5	8	802	DEU
6	8	804	USA
7	9	804	USA

Showing the first 1000 rows.

groupBy()

Adding aggregation to filtered data (avg() function) and grouping dataset based on cca3 variable:

```
display(ds.filter(d => d.battery_level > 7).select("c02_level",
"cca3").groupBy("cca3").avg("c02_level"))
```

Note that there is explicit definition of internal subset in filter function. Part where "d => d.battery_level > 7"

is creating a separate subset of data that can also be used with `map()` function, as part of map-reduce Hadoop function.

```
Cmd 57
1 display(ds.filter(d => d.battery_level > 7).select("c02_level", "cca3").groupBy("cca3").avg("c02_level"))
```

↳ (2) Spark Jobs

	cca3	avg(c02_level)
1	PSE	1314.090909090909
2	POL	1204.747863561402
3	LWA	1213.597014925373
4	BBB	1210.25
5	ZMB	948.5
6	BFA	1189.929552905111
7	MOZ	1451

Showing all 188 rows.

Command took 2.98 seconds -- by tonaz.kastrun@gmail.com at 28/12/2020, 20:49:52 on databricks_cli_standard

join()

Join() function will combine two objects. So let us create two simple DataFrames and create a join between them.

```
val df_1 = Seq((0, "Tom"), (1, "Jones")).toDF("id", "first")
val df_2 = Seq((0, "Tom"), (2, "Jones"), (3, "Martin")).toDF("id", "second")
```

Using function `Seq()` to create a sequence and `toDF()` to save it as DataFrame.

To join two DataFrames, we use

```
display(df_1.join(df_2, "id"))
```

Name of the first DataFrame – `df_1` (on left-hand side) joined by second DataFrame – `df_2` (on the right-hand side) by a column “`id`”.

```
Cmd 59
1 >val df_1 = Seq((0, "Tom"), (1, "Jones")).toDF("id", "first")
2 val df_2 = Seq((0, "Tom"), (2, "Jones"), (3, "Martin")).toDF("id", "second")
3
4 display(df_1)
```

↳ df_1: org.apache.spark.sql.DataFrame = [id: integer, first: string]

↳ df_2: org.apache.spark.sql.DataFrame = [id: integer, second: string]

	id	first
1	0	Tom
2	1	Jones

Showing all 2 rows.

Command took 0.48 seconds -- by tonaz.kastrun@gmail.com at 28/12/2020, 20:55:44 on databricks_cli_standard

```
Cmd 60
1 display(df_1.join(df_2, "id"))
```

↳ (2) Spark Jobs

	id	first	second
1	0	Tom	Tom

Showing all 1 rows.

Command took 0.29 seconds -- by tonaz.kastrun@gmail.com at 28/12/2020, 20:57:20 on databricks_cli_standard

Cmd 61

Join() implies inner.join and returns all the rows where there is a complete match. If interested, you can also explore the execution plan of this join by adding `explain` at the end of command:

```
df_1.join(df_2, "id").explain
```

```
Cmd 62

1 df_1.join(df_2, "id").explain

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- == Current Plan ==
   Project [id#7211, first#7212, second#7223]
   +- BroadcastHashJoin [id#7211], [id#7222], Inner, BuildLeft
      :- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint))), [id#42321]
      :- +- LocalTableScan [id#7211, first#7212]
         +- LocalTableScan [id#7222, second#7223]
+- == Initial Plan ==
   Project [id#7211, first#7212, second#7223]
   +- BroadcastHashJoin [id#7211], [id#7222], Inner, BuildLeft
      :- BroadcastExchange HashedRelationBroadcastMode(List(cast(input[0, int, false] as bigint))), [id#42321]
      :- +- LocalTableScan [id#7211, first#7212]
         +- LocalTableScan [id#7222, second#7223]

Command took 0.13 seconds -- by tomas.kastrun@gmail.com at 20/12/2020, 20:50:30 on databricks_cli_Standard
```

and also create left/right join or any other semi-, anti-, cross- join.

```
df_1.join(df_2, Seq("id"), "LeftOuter").show
df_1.join(df_2, Seq("id"), "RightOuter").show
```

```
Cmd 64

1 df_1.join(df_2, Seq("id"), "LeftOuter").show

* (2) Spark Jobs
+-----+
| id|first|second|
+-----+
| 0| Tom| Tom|
| 1|Jones| null|
+-----+

Command took 0.23 seconds -- by tomas.kastrun@gmail.com at 20/12/2020, 21:01:50 on databricks_cli_Standard

Cmd 65

1 df_1.join(df_2, Seq("id"), "RightOuter").show

* (2) Spark Jobs
+-----+
| id|first|second|
+-----+
| 0| Tom| Tom|
| 2| null| Jones|
| 3| null| Martin|
+-----+

Command took 0.34 seconds -- by tomas.kastrun@gmail.com at 20/12/2020, 21:01:55 on databricks_cli_Standard
```

union()

To append two datasets (or DataFrames), union() function can be used.

```
val df3 = df_1.union(df_2)

display(df3)
// df3.show(true)
```

```
Cnd 67
1  val df3 = df_1.union(df_2)
2
3  display(df3)
4  // df3.show(true)
```

df3: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [id: integer, first: string]

	id	first
1	0	Tom
2	1	Jones
3	0	Tom
4	2	Jones
5	3	Martin

Showing all 5 rows.

Command took 0.24 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 21:05:20 on databricks_cli

distinct()

Distinct() function will return only the unique values, and it can also be used with union() function to achieve *union all* type of behaviour:

```
display(df3.distinct())
```

```
1  display(df3.distinct())
```

(2) Spark Jobs

	id	first
1	0	Tom
2	3	Martin
3	1	Jones
4	2	Jones

Showing all 4 rows.

Command took 0.39 seconds -- by tomaz.kastrun@gmail.com at 20/12/2020, 21:05:58 on databricks_cli

Tomorrow we will Spark SQL and DataFrames with Spark Core API in Azure Databricks. Today's post was a little bit longer, but it is important to get a good understanding on Spark API, get your hands wrapped around Scala and start working with Azure Databricks.