UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

# Big Data Management (BDMA & MIRI Masters)
# Session: Apache Spark

Sergi Nadal

## Instructions

In this session, as Spark jobs will run locally, you do not need to start the Spark service and everything can be run from IntelliJ or Eclipse.

Once you have fetched the code, it can be imported as a *Maven Project* into IntelliJ or Eclipse. All executions can be run locally, and thus you do not need to be concerned with the host connection. In the *Main* class you can see that different arguments are expected, and will drive the behaviour of the program. The simplest way to define such arguments is by creating a *Run Configuration* of type *Java Application* from the *Run* menu. In the tab menu called *Arguments* you will be able to edit such arguments, in each exercise you will be guided with the required arguments.

### Delivery

By the end of the session hand in your answers. Code must be delivered via LearnSQL. Prepare a ZIP file and upload it to LearnSQL with the following content: `Exercise_1.java`, `Exercise_2.java`.

## Exercise 1: Analyzing HR data with Spark (40 % weight)

The first exercise consists on exploring a dataset with human resources from Kaggle. The file is already provided in the project's *resources* folder with the anem `HR_comma_sep.csv`. It is a comma-separated CSV file that contains the following attributes:

- Satisfaction Level
- Last evaluation
- Number of projects
- Average monthly hours
- Time spent at the company
- Whether they have had a work accident
- Whether they have had a promotion in the last 5 years
- Departments (column sales)
- Salary
- Whether the employee has left

We are interested in knowing if the number of projects an employee works on affects their satisfaction level. Thus, we precisely ask you to implement in class `Exercise_1.java` the computation of *average satisfaction level per number of projects, ordered from lowest to highest*. You should obtain an output similar to the following:

```
Employees who work in 7 projects have an average satisfaction of 0.11871093750000004
Employees who work in 6 projects have an average satisfaction of 0.27345826235093895
Employees who work in 2 projects have an average satisfaction of 0.4787688442211093
Employees who work in 5 projects have an average satisfaction of 0.6788880840275276
Employees who work in 3 projects have an average satisfaction of 0.6876695437731203
Employees who work in 4 projects have an average satisfaction of 0.6951317296678116
```

## Exercise 2: Optimization of an Spark program (60 % weight)

In this exercise we will analyze data related to the *World Happiness Report* dataset from Kaggle (`https://www.kaggle.com/unsdsn/world-happiness`). It consists of a set of CSV files (one per year) that rank 155 countries by their happiness levels according to metrics about economic production, social support, etc. We will focus on three files, related to the years 2015, 2016 and 2017. Each file provides the ranking of that year, however we are interested on obtaining *the happiest country of Europe in 2015, 2016 and 2017 according to its happiness score*. You can assume there are no duplicates for happiness scores.

Below we provide you with an Spark program that performs such analysis, however it suffers from several performance flaws due to a bad design. First, analyze the following code (also available in class *Exercise_2.java*) and detect such issues:

```
JavaRDD<String> report_2015 = spark.textFile("src/main/resources/2015.csv");
JavaRDD<String> report_2016 = spark.textFile("src/main/resources/2016.csv");
JavaRDD<String> report_2017 = spark.textFile("src/main/resources/2017.csv");

JavaRDD<String> all = report_2015.union(report_2016).union(report_2017);

JavaRDD<String> unified = all.repartition(1);

List<Tuple2<Double,String>> ranking = unified.flatMap(t -> {
    if (t.contains("Country")) return new ArrayList<String>().iterator();
    else return Lists.newArrayList(t).iterator();
  })
  .mapToPair(t -> new Tuple2<Double,String>(Double.parseDouble(t.split(",")[3]),t))
  .groupByKey()
  .sortByKey(false)
  .mapToPair(t -> new Tuple2<Double,String>(t._1, t._2.iterator().next()))
  .mapValues(t -> t.contains("Europe") ? t : null)
  .collect();

Tuple2<Double,String> top = ranking.get(0);
out = top._2.split(",")[0]+" is the happiest country in Europe for 2015, 2016 and 2017 with
    an score of "+top._1;
```

Your task consists on proposing an improved version of this program such that it optimizes its performance and applies the right set of operations. You are free to modify the provided code as long as the output is exactly the same. You are not allowed to use any external library. Optimize your program with the goal of executing it in a distributed cluster, however bear in mind that you are executing it in local mode.

You can replace the current code in *Exercise_2.java* with your proposal. The rationale behind your decisions should also be discussed in the provided answer sheet. Considering that the input files are very small, and in order to test the performance improvements, we provide you with a larger version (i.e., *2015_long.csv*) of each. You can download the files from `https://bit.ly/2FFLmab`.