Lab 8

In this lab, we analyze historical data about the stations of Barcelona's bike sharing system. The data are the same you already analyzed during the previous practice, as well as the main goal of your task: computing the "criticality" for the pairs (station, timeslot) and selecting the most critical ones. However, in this practice, you are requested to use the Spark SQL APIs, and the final selection process differs slightly from the one you implemented during the previous practice.

The analysis is based on two files (available in the shared folder of the BigData@Polito cluster - /share/students/bigdata/Dati/Lab8/datiCompleti/ - and on the webpage of the course):

1. register.csv
2. stations.csv

1) **register.csv** contains the historical information about the number of used and free slots for ~3000 stations from May 2008 to September 2008. Each line in register.csv corresponds to a reading about the situation of one station at a specific timestamp. Each line has the following format:
   - *station*\t*timestamp*\t*used_slots*\t*free_slots*

   For example, the line

   23 2008-05-15 19:01:00 5 13

   means that there were **5** used slots and **13** free slots at station **23** on **May 15, 2008** at **19:01:00**.

   The first line of register.csv contains the header of the file.

   Pay attention that some of the lines of register.csv contain wrong data due to temporary problems with the monitoring system. Specifically, some lines are characterized by used_slots = 0 and free_slots = 0. Those lines must be filtered before performing the analysis.

2) **stations.csv** contains the description of the stations. Each line of registers.csv has the following format:
   - *id*\t*longitude*\t*latitude*\t*name*

   For example, the line

   1 2.180019 41.397978 Gran Via Corts Catalanes

   contains the information about station **1**. The coordinates of station 1 are 2.180019,41.397978 and its name is **Gran Via Corts Catalanes**.

Task 1

Write a single Spark application that selects the pairs (station, timeslot) that are characterized by a high "criticality" value. The first part of this practice is similar to the one that you already solved during the previous practice. However, in this case you are requested to **solve the problem by using two different sets of Spark SQL APIs.**

(i) Implement **a first version** of the application based on **Dataframes** and the associated APIs

(ii) Implement **a second version** of the application based on the use of **SQL queries** within the Spark application**,** i.e., use **SparkSession.sql("SELECT …").**

In this practice, each pair "day of the week – hour" is a timeslot and is associated with all the readings associated with that pair, independently of the date. For instance, the timeslot "Wednesday - 15" corresponds to all the readings made on Wednesday from 15:00:00 to 15:59:59.

A station Si is in the critical state if the number of free slots is equal to 0 (i.e., the station if full).

The "criticality" of a station Si in the timeslot Tj is defined as

$$\frac{number\ of\ readings\ with\ num.of\ free\ slot\ equal\ to\ 0\ for\ the\ pair\ (Si,Tj)}{total\ number\ of\ readings\ for\ the\ pair\ (Si,Tj)}$$

Write an application, based on the Spark SQL APIs, that:

- Removes the lines with used_slots = 0 and free_slots = 0.
- Computes the **criticality value** for each pair (Si,Tj).
- Selects only the combinations having a criticality value greater than a minimum criticality threshold. The **minimum criticality threshold** is an argument of the application.
- **Join** the content of the previous selected combinations with the content of stations.csv to retrieve **the coordinates of the stations**.
- Store in the output folder the selected records, by using **csv files (with header)**. Store only the following attributes:
    - station
    - day of week
    - hour
    - criticality
    - station longitude
    - station latitude
- Store the results **by decreasing criticality**. If there are two or more records characterized by the same criticality value, consider the station (in ascending order). If also the station is the same, consider the weekday (ascending) and finally the hour (ascending).

Hints

- The SQL-like language available in Spark SQL is characterized by a predefined function called **hour(timestamp)** that can be used in SQL queries, or in the **selectExpr** transformation, to select the "hour part" of a given timestamp. The **date_format(timestamp,'EE')** is another predefined SQL function available in Spark SQL that can be used to extract the weekday from a timestamp column.
    - Example

      …

      resDF= myDataframe.selectExpr("date_format(timestamp,'EE') as weekday",\
      "hour(timestamp) as hour")

      …

- To specify that the separator of the input CSV files is "tab", set the delimiter option to **\\t**, i.e., invoke **.option("delimiter", "\\t")** during the reading of the input data.

Shut down JupyterHub container

As soon as you complete all the tasks and activities on JupyterHub environment, please remember to shut down the container to let all your colleagues in all the sessions connect on JupyterHub and do all the lab activities.

1. Go into File -> Hub Control Panel menu
2. A new browser tab opens with the "Stop My Server" button. Click on it and wait till it disappears.