iscte TECNOLOGIAS E ARQUITETURA

Algoritmos para Big Data
Lab class handout #2

Data joining, data windowing and Spark SQL **2024/2025**

Goals

This lab class aims to get hands-on experience on three topics related to data processing: data joining, data windowing and Spark SQL.

Note: It is acceptable that the work to be undertaken may require some time beyond the class. If that is the case, work should continue outside of class. And if needed, assistance will be provided during the office hours established or to be agreed with the lecturer.

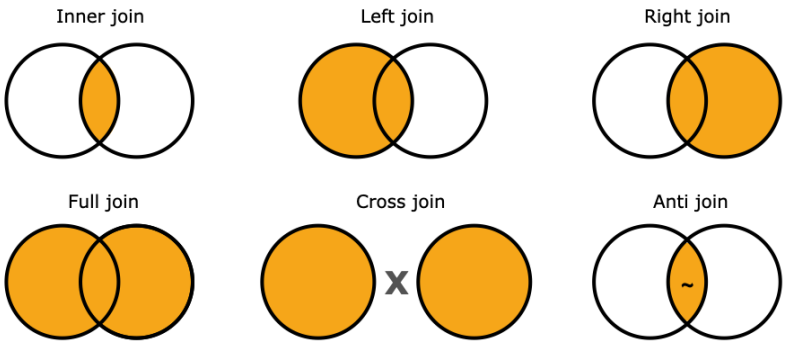
Supporting information

- Course slides
- Apache Spark API Reference
- Spark SQL, DataFrames and Datasets Guide

Introduction

As mentioned above, the issues to tackle in this handout are the following:

1. **Data joining**. It is a process that helps to enrich data by combining rows from two parts/tables, based on a related column condition. Types of data joins to consider include:



   - Inner join - returns only rows that have matching values in both tables.
   - Left join - returns all rows from the left table and the matched rows from the right table. If there is no match, the result is null on the right side.
   - Right join - returns all rows from the right table and the matched rows from the left table. If there is no match, the result is null on the left side.
   - Full join - returns all rows when there is a match in either the left or right table. It combines the results of both the left and right outer joins.

- Cross join - returns the Cartesian product of the two tables, which means it combines each row of the first table with each row of the second table..
- Anti join - returns rows from the first table where no matches are found in the second table. It is the opposite of an inner join.

2. **Data windowing**. It is mostly used in the context of temporal or sequential data. Basically data is split into smaller sequential series to gather insights from each of those data partitions (windows). Examples of operations are:
   - Performing calculations across rows.
   - Ranking and row numbering.
   - Running totals and moving averages.
   - Accessing previous and next row values.
3. **Spark SQL**. As a contribution to have a unified foundation for data processing, there is a programmatic interface available to interact with structured data stored as tables or views in a database. Therefore one can move from dataframes to SQL tables and then use SQL queries should that be appropriated.

Finally, alongside this handout describing the tasks to accomplish, a notebook is also provided, that it is expected to be completed with the implementation of the tasks. And as for the datasets needed, the required file (just one) can be downloaded from:

https://bigdata.iscte-iul.eu/datasets/retail-data.csv

Tasks to accomplish

A. Data ingestion

In this initial task, do execute the following operations:

1. Download the data file already mentioned and save it in a proper directory, within the working directory of your Pyspark instalation.
2. After creating a *SparkSession* to work with, read the data into a DataFrame and then check it. Make sure the data is prepared/cleaned to be worked with. Let us consider the name of such DataFrame as `df_retail`.

B. Data joining

In this task, we will work on how to combine rows from two distinct but related dataframes. Execute the instructions as follows:

1. Using a data aggregation operation on `df_retail` from the previous task, figure out the number of items/rows billed per each country.
2. Create the followings dataframes, after `df_retail`.
   (a) `df_sales`, as `df_retail` but excluding the column `Country`;
   (b) `df_customers`, with just two columns: `CustomerID` and `Country`;
   (c) `df_customers_nonUK`, likewise `df_customers` but filtering out the customers from United Kingdom.
3. Considering the left and right dataframes as respectively `df_sales` and `df_customers_nonUK`, and data joining condition related to the column `CustomerID`, show the outcome of each of the following data joining types of operation:

(a) Inner join (*inner*).

(b) Left join (*left*).

(c) Full join (*full*).

(d) Cross join.

(e) Anti join (*anti*)

C. Data windowing

Do execute the following operations:

1. First, create a DataFrame based upon `df_retail` but with a new column, to be called `Value` and set as `Quantity*UnitPrice`. The new DataFrame should contain the columns `InvoiceDate`, `CustomerID`, `Country` and `Value`, ordered by `InvoiceDate`.
2. Show a new column related to `Value` that, for each invoice/row, would contain the sum of the two previous rows (of column `Value`).
3. Show a new column related to `Value` that, for each invoice/row, would contain the value corresponding to a lag of size 2 (of column `Value`).

D. Spark SQL

In this task, we will use SQL queries to get information from both persistent and temporary tables. Execute the following operations:

1. Store the content of `df_retail` from the initial task as a persistent table, to be called `RetailTable`, into the Hive metastore.
2. List (i) the catalogue of databases available; (ii) the tables in the default database and (iii) the columns of `RetailTable`.
3. Use the default managed table (the one initially stored) and with a SQL query show the content of `RetailTable`.
4. Create a temporary view with the content of `df_customers` from the task about data joining, to be called `CustomersTables`, and then show the outcome of the following queries (with a SQL query):

   (a) CustomerID and country in all rows, with rows order by customerID.

   (b) Counting of rows regarding each country registered.