Laboratory of Data Science: Part I
Data Integration

Ludovico Lemma, Davide Innocenti

5 November 2022

1 Introduction

With this report we are going to describe the Data Science project assigned to us. We are going to describe the various preprocessing and ETL steps we performed on a given dataset regarding answers of individuals to various multiple-choice questions delivered by different organizations. The aim is to create detailed tables to synthesize a Data Warehouse into SQL Server Management Studio so that it could be queried for business purposes.

The data was provided through two different CSV files: "answer_full.csv" and "subject_metadata.csv". The former contains the main body of the of data, 538.835 and 17 dimensions, regarding both information about individuals and organizations, it was presented as a single table with the following features:

```
ColumnIndex(['QuestionId', 'UserId', 'AnswerId', 'CorrectAnswer',
             'AnswerValue', 'Gender', 'DateOfBirth', 'PremiumPupil',
             'DateAnswered', 'Confidence', 'GroupId', 'QuizId',
             'SchemeOfWorkId', 'SubjectId', 'RegionId', 'Region',
             'CountryCode'])
```

The latter is instead used to provide details about the "SucjectId" column since it contains an explicit reference to a hierarchy of 4 different levels for a total of 388 subjects. The columns were presented as it follows:

```
ColumnIndex(['SubjectId', 'Name', 'ParentId', 'Level'])
```

A preliminary Data Exploration task has been performed in order to check for incongruities. Two dimensions have been taken into account:

- Missing Values
- Data Integrity

Luckily enough, no missing values have been found in the dataset, while for the data integrity check, every column has been analyzed and no record had consistency issues.

2 Creating the Data Warehouse Schema

For the true first step of the project we decided to design the snow flake schema of the Data Warehouse on Microsoft SQL Server Management Studio, we set integers as the data types of the primary keys of the tables we had to create, we discarded from the original Dataset the features "PremiumPupil" and "RegionId", the first one because it wasn't requested according to the business objectives, the second one because it wasn't meaningful as it was a direct mapping of the country code which wasn't the feature on which we needed to build the key in the Geography table, and it was semantically wrong as the feature Region had an higher grain of detail. For the Description column in the subject table we assigned a place holder at first but later we determined nvarchar(286) by measuring the largest description we had as a result of the explicitation of the ids.

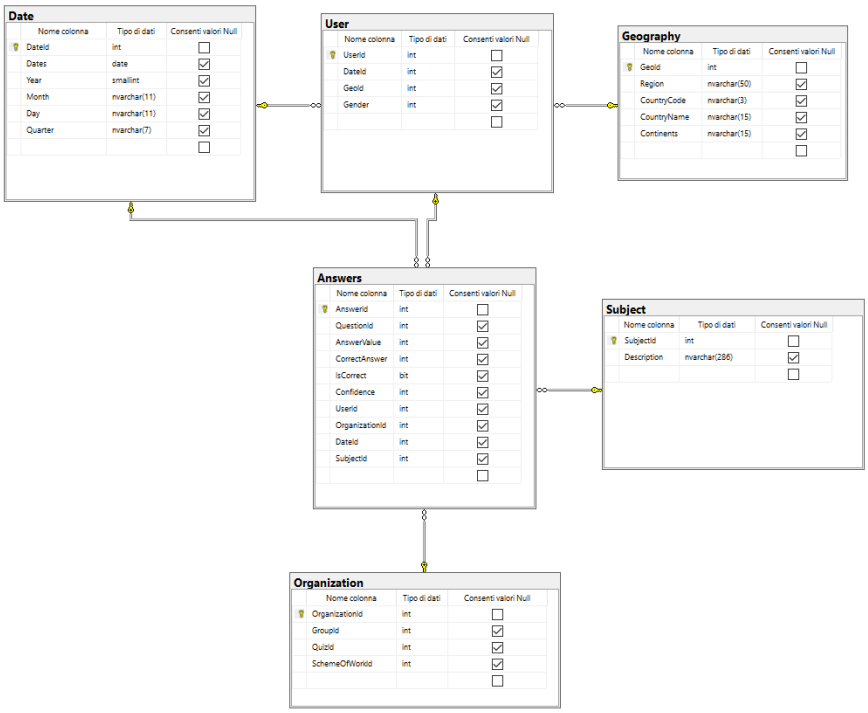

Figure 1: Data Warehouse Schema

3 Preprocessing the Dataset

In this step we dealt with the general tasks of reading and processing the CSV files, generating the tables and the primary keys, managing the peculiarities of the different tables and finally mapping the primary keys to the values they referred to as foreign keys in the original fact table or the tables higher in their hierarchy.
The functions were written with the scope of being as general as possible to mantain and adapt them to other contexts in the future. The only libraries we used in the development of the functions were the python standard "re", used to manage the CSV default reading standard (we implemented only the two cases of "commas" and "double quotes"), and the pycountry_convert library, used to get both the country and continent name from the ISO country codes.
Several tasks have been performed to adapt the tables and satisfy the requirements of the business questions for the Data Warehouse:

- "IsCorrect" measure has been generated to check, for each individual, the correctness in answering the questions it was computed by comparing values of the "CorrectAnswer" and "AnswerValue" columns.
- The "Subject" table has been made by only using the "SubjectId" and "Description" dimensions. The latter is an extended representation of sorted subjects according to the level order, from the most general one (e.g. Math) of the subject list to most specific ones. In order to do that every distinct list of the old "SubjectId" column in the answer_full.csv has been overwritten with the explicit description sorted by level.
- The "Geography" table primary key ("GeoId") has been generated by mapping a concatenation of the "Region" and "CountryName" columns (with Region being the most important having the highest grain). The additional "Continent" and "CountryName" dimensions were also requested. The library "pycountry_convert" has been exploited to map the ISO country codes to the desired features, with only some minor adjustments needed for the United Kingdom.
- In the "Date" table, we accommodated both dates of birth of users and for dates of answers to save space. For this reason, we to performed a set union of both columns, for a total amount of 596 dates. From this column we then built the derived dimensions requested by also preserving the hierachy (Year, Month, Day and Quarter). The corresponding "DateId" was later mapped on the User table "DateOfBirth" dimension and to the Fact Table dimension "DateOfAnswer".

In the end, every table has been assigned its own integer progressive primary key by considering the distinct values of sorted records, then the primary key was mapped back to the original (not distinct) dimensions used to build it, to the source data, so that a foreign key could be obtained for a related table.

4 Uploading the Tables to the Server

In this final step of this part of the project we developed a single class to manage the entire process of loading the table to the server. The class accepts as input a CSV loaded as a dictionary as in the preprocessing step (with each column as a value of the column name), and the destination table in the server which will store a Data Warehouse. In order for the class to work a schema as to be already defined. After establishing a connection (for which a credentials.txt file with user id and password is needed in the same folder) the schema's types will be fetched from the server in order to be casted so that the input data will match the requirements (if possible), there is also a check on the header to be uploaded and the one expected from the server so that an error would be raised if the two wouldn't match.

Then a parametric query is gonna be generated to insert the rows into the remote table, by matching the header's columns and the number parameters "?" to be inserted as in the following example:

```
INSERT INTO [Table_Name] (Column0, Column1, ...) VALUES (?,?, ...)
```

In order to keep things simple we decided to intend the Class object to work as a "first-only upload", indeed we established a routine of deleting all previous data from the table expected to be uploaded to avoid inconsistencies. We managed also to catch possible errors about integrity of possible related tables, so that also data present in the table's hierarchy will be deleted. Finally the class uploads the records, making a maximum of 10 attempts to execute the INSERT INTO query, and committing every 1.000 queries.