UNIVERSITY OF PISA

2022/2023

Build a Datawarehouse

Laboratory of Data Science Project

TEAM

Maria Grazia Antico 645005
Alessio Cascione 582765

PROFESSORS

Anna Monreale
Roberto Pellungrini

1. Creation of the Datawarehouse

In this section, will be described the different steps to be carried out for the construction and population of a database, starting from the ***"answerdatacorrect.csv"*** file, containing data about answers given by students to various multiple-choice questions and other data regarding the questions, the students and the subject of the questions, and "***subject_metadata.csv***" where, instead, there are informations about the subject of each question. The steps addressed will be the following, described in detail in the following sections:

1. Defining the database schema, using SQL Server Management Studio
2. Writing python programs to execute the split of the main file *"answerdatacorrect.csv*" in order to obtain six separate tables: **answers**, **organization**, **date**, **subject**, **user** and **geography**. It will also be necessary to use some functions to integrate the data containing in the main file.
3. Populate the database with the data obtained from the previous assignment

1.1 Assignment 0: Definition of the database schema

SQL Server Management Studio was used to build the database schema (Fig.1), consisting of six tables. The types of the different attributes used are as follows:

- *'Nchar'* for string type attributes with fixed number of characters (e.g. for country_name and continent we used nchar (2)).
- '*Nvarchar'* for string-type attributes with a variable number of characters, setting the maximum value present in the data for each attribute as the maximum length. Nvarchar was used over varchar as the former allows to store UNICODE characters.
- For numeric attributes, with the exception of an attribute of type '*float'* (confidence), '*int*' was used in particular for all keys and, for some exceptions, tynint (for example for quarter) in cases where the values of the attributes did not exceed the threshold of 255. Although the attribute "*schemeofworkid*" was represented as a float in the main file, it was decided to transform it to an *int* as it was intended to be considered as a discrete value, unlike the continuous *'confidence'* attribute.
- For the "**iscorrect**" attribute, generated later and not present in the main data, '*bit'* was used as it has values of 1 and 0 based on whether the answer is correct or not.
- In the Date table, '*date'* was used to identify the **date** attribute, concerning the dates of birth and the response dates in the format YYYMMDD.

For each dimension, the following primary keys and foreign keys have been established:

- For the fact table **Answer**: *'answerid'* as primary key; *organizationid (Organization), dateid (Date), subjectid (Subject*), *userid (User)* as foreign keys.
- For the **Organization** dimension: *'organizationid'* as primary key (created by the concatenation of "groupid", "quizid", "schemeofworkid").
- For the **Date** dimension: *'dateid'* as the primary key.
- For the **Subject** dimension: *'subjectid'* as the primary key.
- For the **Geography** dimension: *'geoid'* as primary key (created by concatenating "region", "country_code").
- For the **User** dimension: *'userid'* as primary key*; datebirthid (Date*), *geoid (Geography*) as foreign keys.

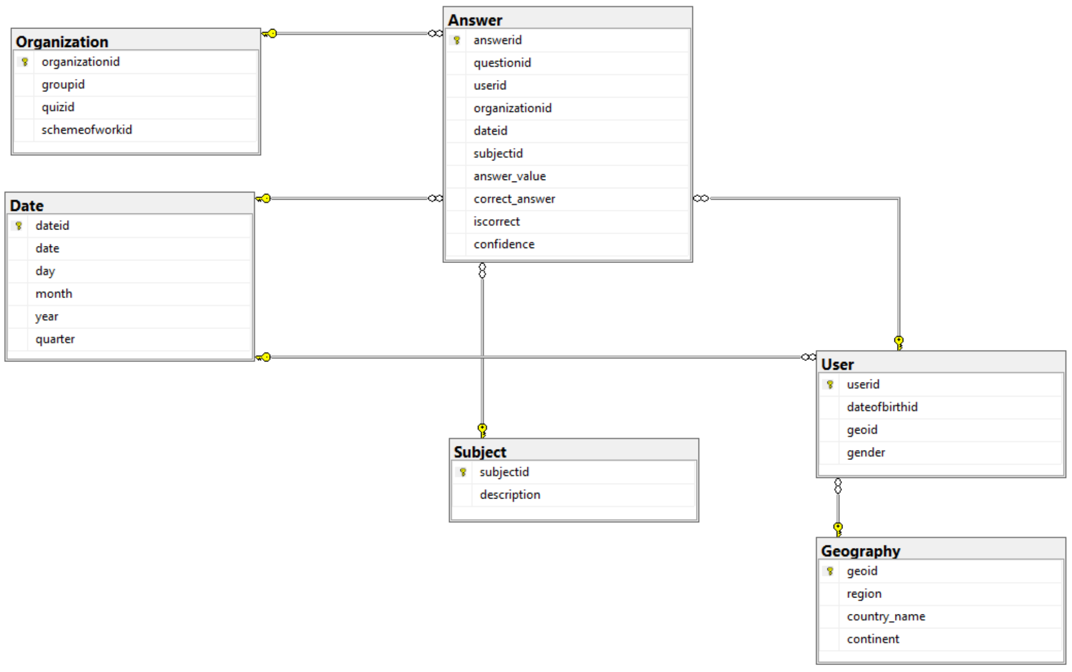

Figure 1. Database schema

1.2 Assignment 1: Generation of the tables

The data contained in the main "*answerdatacorrect.csv*" file have been divided into 6 different files corresponding to the tables of the schema in Figure 1. Using the **csv** library, some general functions have been created in order to support the process of creating the tables and, moreover, , specific functions have been used for each table in order to transform the data where necessary.

1.2.1 General functions

- **create_columns(cols,row,header):**
  When header is True, a row of the form *[attribute_name(1),attribute_name(n),…,attribute_name(n)*] where cols is a sub-set of such row. The aim of the function is finding the indexes corresponding to the sub-set of attributes specified in cols. When header is False, a row of values is specified and we are simply interested in finding out if the cols sub-set is acceptable, i.e. if there are no out of indexes values present inside the list. The final output is stored in a dictionary where every key is the attribute name (if the header is considered, otherwise we just keep the index of the columns) and the respective index of that attribute in the row given as input.
  This function has been used for Date, Geography and Subject tables.

- **table_new_id(file1, file2, header, cols=[], id_name ='id', id_start = 0, increment = 1, output_type = 'csv'):**
  Given a csv file, table_new_id() finds all the unique tuples of the csv file with respect to the values of the set of attributes defined in cols, and assign to each unique tuple an id, starting by default with 0 and incrementing it of 1.
  For instance, given the sub-columns list *['QuizId','GroupId','SchemeOfWorkId']* and considering as input answersdatacorrect.csv, all the unique tuples of values corresponding to those 3 attributes are saved in a new .csv file with a unique integer id.
  If output_type = '*csv*', then the function produces a new .csv file named "file2" with the respective tuples and their ids.
  If output_type = '*dict*', then the function returns a dictionary where every key is of the form *(value_cols[0], value_cols[1],…,value_cols[n])* and the value of each key is the index corresponding to the unique tuple

presented as key (this will result useful when comparing answersdatacorrect.csv with the dictionaries obtained also through this method in order to efficiently build the fact table).
The Organization table has been created with this function only, as seen in create_organization.py.

- **def_dict(csvfile, unique_tuple = [], id_identifier = ''):**
  A simple function that considers a csv file and outputs a dictionary where each key is a tuple of values for a sub-set of attributes defined in the parameter unique_tuple: for instance, an output example is the following for organization.csv: *{('3833', '15391', '8409.0'): '0', ('5906', '15391', '8409.0'): '1',...}* where ('3833', '15391', '8409.0') is a unique tuple of ['QuizId','GroupId','SchemeOfWorkId'] and its value '0' is the respective id in the organization.csv.

1.2.2 Create_date.py

To create the *Date* table, since the "date" attribute consists of the union of the *Dateofbirth* and *DateAnswered* columns present in the main “answerdatacorrect.csv” file, it was necessary to transform dateanswered into the same format as datebirth, that is YYY-MM-DD using the **‘date_handle’** function present in *aux_functions.py* which, in addition to returning the date in the chosen format, also returns the corresponding day, month, year, quarter. This function will be used within the generating function of the Date table **‘create_date_table’** which:

- If the parameter output_style == *'dict'*, it will return a dictionary with the date as key and the associated id as a value (the id is created starting from 0 by increasing by 1 for each key).
- If the output_style == *'csv'* parameter, it will return the *‘date.csv’* file with the dateid, date, day, month, year and quarter columns for each different date. Dateid is generated starting from 0 and also increased by 1.

The records associated with the table, with the removal of duplicates, were found to be 596.

1.2.3 Create_geography.py

To create the *Geography* table, the “**countryInfo.txt”**[1] file containing information relating to the various countries and the associated continent was used to generate the "**continent**" attribute not present in the main data. Therefore, the ‘**generate_continent’** function was created from which a dictionary is obtained having the countrycode as key and the associated continent as its value. To better adapt the function to the selected text file, it was decided to replace the continents 'NA' and 'SA', corresponding to North America and South America, with the continent 'AM'. In the ‘**create_geography**’ function, generator of the ‘*geography.csv’* file, the dictionary obtained from the previous function is given as input and, by scrolling the CountryCode column of the main file “answerdatacorrect.csv”, the associated continent is found in the dictionary. For each distinct Region and CountryCode, an id is generated starting from 0 corresponding to the *'geoid'* attribute.

In total, the records of the Geography table, without duplicates, are 76.

1.2.4 Create_Subject.py

To create the *Subject* table, three different functions were used here too: ‘**string_tolist’**, ‘**ordered_subjects’** and ‘**subject_id_table’**. The first function, present in *aux_functions.py*, was found to be necessary to transform the SubjectId of the main file from the string format to the list format. The second function was used to reorder the SubjectId lists (containing the different subjects) according to the level. The function, in particular, using the data present in “**subject_metadata.csv**”, initially creates a dictionary in which the keys are the subjectid (which in the reference file are the individual subjects) and as values the tuples with the name of the subject and the level associated, for example: *'34': ('Upper and Lower Bounds', 3)*. Using this last dictionary, it was possible, using a zip of three different lists (subjectid, levels associated with each subject, description associated with each subject) for each list,

1 The file can be downloaded here: http://download.geonames.org/export/dump/countryInfo.txt – we specify that comments preceeding the actual table have been manually removed in order to simplify the script used to extract

reordering them by level. Not considering duplicates of descriptions, the result of the function is a list of tuples having an id, the ordered list of subjectid, and the associated ordered description.

Finally, in the third and last '*subject_id_table'* function, using the output of the previous function, only the id and the associated description are transcribed in the "*subject.csv*" file if the parameter of the function output_style == *'csv'*. If output_style == *'dict'*, on the other hand, the function generates a dictionary having as a key a tuple of the ordered elements of subjectid and the associated id as a value.

The total records of the Subject table, without duplicates, are 412.

1.2.5 Create_Organization.py

As mentioned in the 'General Functions' section, only the *table_new_id* function was used to create the *Organization* table, associating an id for each distinct *groupid*, *quizid*, *schemeofworkid* from the main data, generating the '*Organization.csv'* file as output, containing in total 24,640 records.

1.2.6 Create_User.py and Create_Answers.py

To create the User and Answer tables (in temporary version, the "iscorrect" column will then be added), the following function was used: **table_dict(file1,new_csvfile,id_subs={},additional_del_cols=[])**. It considers a csv file and a dictionary id_subs in input where the input dictionary is, for instance: *{('organizationid','QuizId','GroupId','SchemeOfWorkId') : org_dic, ('subjectid', 'SubjectId') : subj_dic,('dateid', 'DateAnswered') : date_dic}*.

In the case in example, for every row of the .csv file in input, the function looks for the tuple of values corresponding to the attributes specified in the tuple, then substitutes those values with the corresponding id found in the dictionary passed as value of the tuple in question. For instance, in the case of ('organizationid','QuizId','GroupId','SchemeOfWorkId'), for every row, it looks for the three values of 'QuizId','GroupId','SchemeOfWorkId' and replace them with the value of the corresponding tuple in org_dic. *Additional_del_cols* identifies additional column we wish to remove from the row that should not be replaced with any id.

The User table contains, in total, 13,630 records and the final Answer table will contain the same number of records as the original file, which is 538,835.

1.3 Assignment 2: Loading data into the database

The file "*upload_data.py*" describes the phase of uploading the data to the database. Initially, through the **pyodbc** library, a connection was opened to the reference database (Group_7_DB) and subsequently two functions were used: **'define_dictionarty'** and **'general db'**. The first, using the *.DictReader ()* method, returns the data and header of a given file. The second, using the cursor as a parameter:

- If the delete_all parameter == True it executes the initial query 'DELETE FROM table_name', after having checked if the object_id related to that table_name is null or not, in order to delete values inside the table in question, emptying it.
- After the optional delete phase, values are inserted one by one using the cursor passed as parameter in the table, also checking if the object_id related to the table name is null or no.