IN SUPREMÆ DIGNITATIS
1343

UNIVERSITÀ DI PISA

LDS Project – part 1

Group 8:

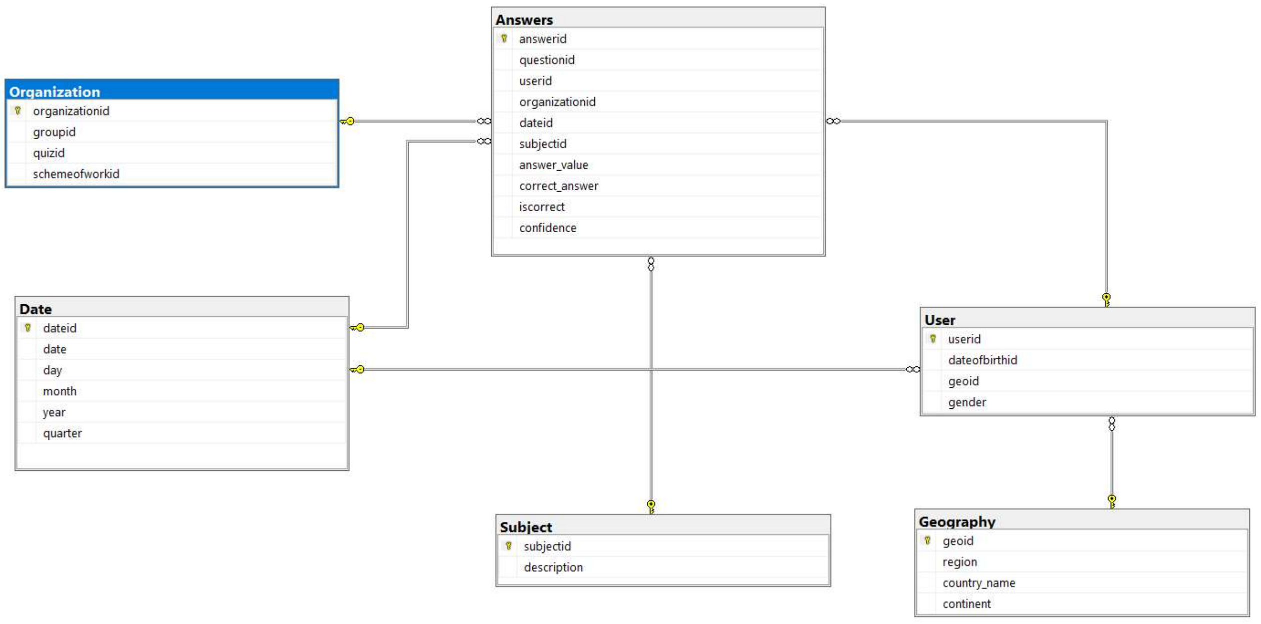
Sabatini Emanuele 637756

Volpi Federico 648069

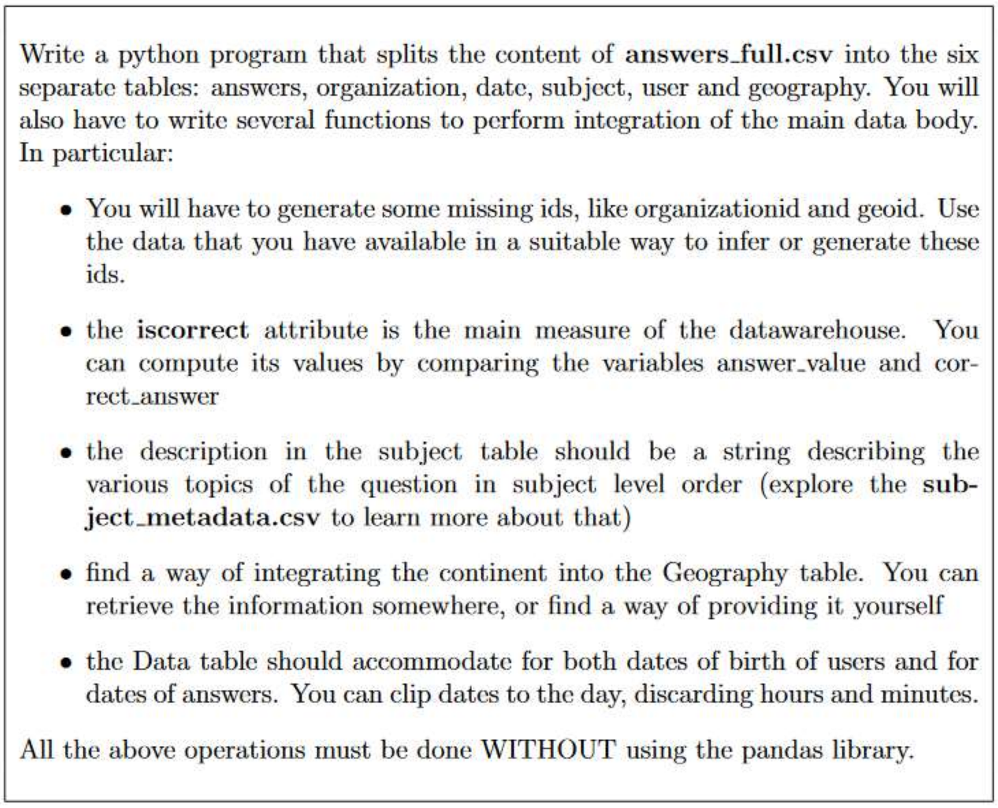
Assignment 0

Create the database schema in Figure 1 using SQL Server Management Studio in server lds.unipi.it. The name of the database must be *GroupID_DB* (example: Group_01_DB).

To do this task, we have created a Project Diagram to correctly replicate the schema. We have firstly defined the primary keys of each tab, namely the dimension that ends with "-id". Next, we have proceeded defining the relations between each tab. The Answers tab has as external keys the primary keys of all the tables except for the Geography tab one, *geoid*, so it is related to all of them except Geography tab. Other than the one with Answers, the User tab has two additional relationships because of two external keys, *dateofbirthid* and *geoid*. The former is the primary key of Date tab, the latter the primary key of Geography tab.



Assignment 1


Write a python program that splits the content of **answers_full.csv** into the six separate tables: answers, organization, date, subject, user and geography. You will also have to write several functions to perform integration of the main data body. In particular:

- You will have to generate some missing ids, like organizationid and geoid. Use the data that you have available in a suitable way to infer or generate these ids.
- the **iscorrect** attribute is the main measure of the datawarehouse. You can compute its values by comparing the variables answer_value and correct_answer
- the description in the subject table should be a string describing the various topics of the question in subject level order (explore the **subject_metadata.csv** to learn more about that)
- find a way of integrating the continent into the Geography table. You can retrieve the information somewhere, or find a way of providing it yourself
- the Data table should accommodate for both dates of birth of users and for dates of answers. You can clip dates to the day, discarding hours and minutes.

All the above operations must be done WITHOUT using the pandas library.

The solution we have found to support maintainability consists in the creation of a class for each table we are interested in. In each class there are functions that can be used to create each csv, initialize metadata, load data, retrieve the id and terminate. We have used a dictionary to make the process of retrieving smoother and to avoid duplicates. This solution with classes appeared to us preferable in respect to one with just different functions because it makes the code easier to maintain and more readable, while keeping a coherent and easy way to change code if different necessities emerge.

To solve the **first point** of Assignment 1, in the *__init__* function of each class we have created an instance of the class with the name of each missing id and it was initialized at 1. In the case of *geoid*, it becomes the value of a key-value pair of the geography dictionary where the key is a tuple containing *region*, *country_name* and *continent_name*. The value of geoid is increased for each tuple not present in the dictionary, thus creating an id dimension that univocally links one tuple with one id. The same procedure is used for *organizationid*.

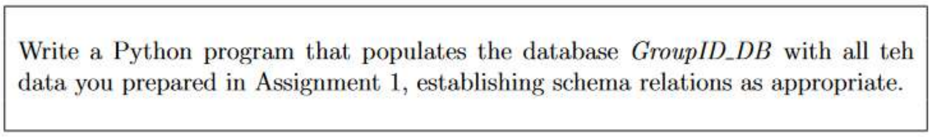
To solve the **second point**, we have created an *is_correct* function in the class *answer_csv*. This function has as arguments the *give_value* variable, that is the answer of the student and *correct_value*, the correct answer. If the answer of the student is the correct one, the string "YES" is returned, otherwise a "NO" one. This function is called in the load function, where the value returned is saved in a variable called *correct*, that will populate the '*iscorrect*' dimension.

To solve the **third point**, we move to the class subject. With the function *recover_metadata*, the subject_metadata.csv file is opened and a dictionary *subject_dict* is created to store the information to create the descriptions. This dictionary is populated with key-value pairs in a for with each line of the subject metadata, having the subject id as key and a tuple with name, *parentid* and level as value. The descriptions are generated with the function *generate_description*, starting with a string variable that is used to convert in a string list, with the string passed as argument. For each element of the list the subject is retrieved from the code and stored in the list *subjects_list*. This latter list is sorted in increasing order based on the third element of the tuple, that is *level*. Finally, the description is generated in the output variable calling the elements from the list and then returned. In the load function, the *generate_description* function is called with the list of ids passed as argument and the value returned in the variable description.

To solve the **fourth point**, we move to class geo_csv. In the function load, the variables *country_c* (country code) and region are passed as arguments. We used *pycountry_convert* library to find the name of the continents, with three different functions: the first is *country_alpha2_to_country_name*, to find the country name from the country code and saving it in the *country_name* variable. The second is *country_alpha2_to_continent_code*, to convert the country code in the continent code and return it in the variable *continent_code*. Finally, the third one is *convert_continent_code_to_continent_name*, to convert the *continent_code* in the continent name and saving it in the variable *continent_name*. Then, the variable self.geo is created as a tuple of the variables region, *country_name* and *continent_name*. We also introduced an if on the variable *country_c* in the case if the code is that of the UK. Since the library recognizes the code as 'GB' and not 'UK', we have corrected it.

To solve the **fifth point**, we move to class *date_csv*. We have created a *find_quarter* function with month passed as argument that returns which quarter of the year the date refers to. Then the *load_birth* function with *dateofbirth* passed as argument is used to split the date of birth in its original format to create a tuple *birth_date* with the elements that have been split (day, month, year), also containing quarter. The tuple is then searched in the date dictionary, if it is not found a new key-value pair is created with the *birth_date* variable as key and a string in the format DDMMYYY as value. Finally, with the function *load_answer* a similar procedure is done with the variable *dateofanswer*, passed as argument. In the *retrieve_id* function, a variable choose is passed as argument: if 'birth' is chosen, it returns the *birth_date*, otherwise the *answer_date*.

Assignment 2



Write a Python program that populates the database *GroupID_DB* with all teh data you prepared in Assignment 1, establishing schema relations as appropriate.

This assignment has been solved with a large upload class with many functions. In the init we created the *connectionString* to connect to the DB, with a cursor variable. The *fast_executemany* method has been added to make the upload faster, as all the list is updated in memory before the database. With the *check_empty* function we check the content of the table: if it is not empty, the function deletes all the rows, otherwise it does nothing. Moving to the *check_row* function, that accept as arguments the table the *csv_row* variable, we execute a query to check the number of rows, that should be equal to *csv_row*. In this case, we print that the table has been loaded correctly, otherwise we point out that there is a problem. Then, for each table we have a function that calls *check_empty* and a query to update the database. Finally, with the close function we close the cursor and the connection.