Università di Pisa

Dipartimento di Informatica

Corso di Laurea Magistrale in

Data Science and Business Informatics

Progetto per il corso di

Laboratory of Data Science

Answers Dataset - Building a Data Warehouse

A cura di:

Marco Di Cristo, 636814

Mario Bianchi, 616658

Anno accademico 2022/2023

Part 1: Creating the Data Warehouse

1.1 Introduction

Part 1 of the project requires to create and populate a database starting from two csv files after having performed multiple operations on them. The file *answerdatacorrect.csv* contains the main body of data: a table with data about answers given by students to various multiple-choice questions. In the same table, there are several data regarding the questions, the students and the subject of the questions.

The file *subject_metadata.csv* contains information about the possible subjects of each question. Each subject is described by its name and by its level (ranging from 0 to 3, with 3 being the deepest). In the *answerdatacorrect.csv* file each question is described by a list of subject indexes that allow retrieving the subject of the question from the *subject_metadata.csv* file.

1.2 Assignment 0: Database schema

The Assignment 0 requires to recreate a given database schema in Microsoft SQL Server Management Studio. The database is uploaded to the *lds.unipi.it* server.
The obtained schema is represented in *Figure 1.1*.

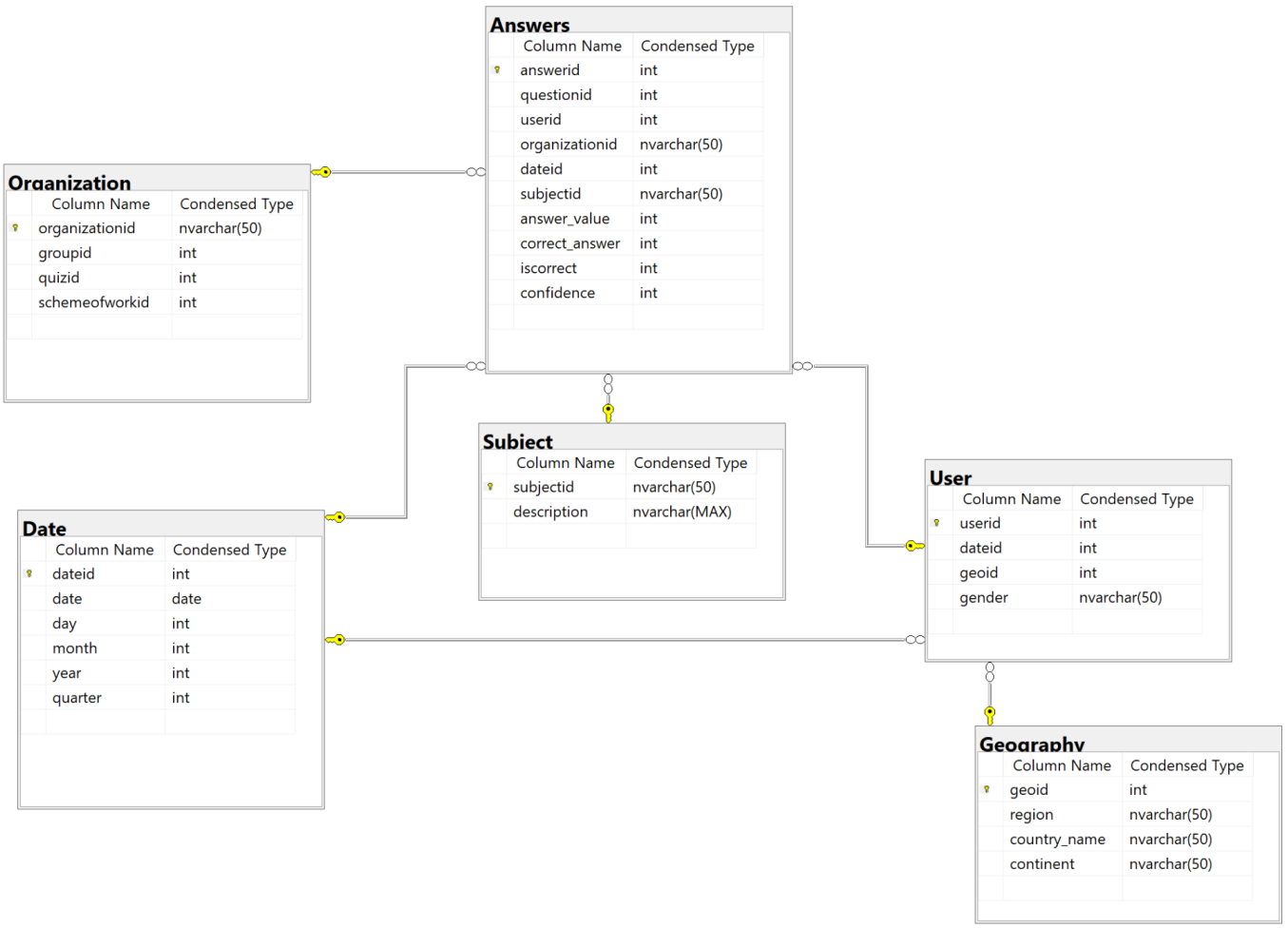

Figure 1.1: Database schema

The chosen primary keys are:

- *answerid* for the fact table *Answer*
- *organizationid* for the *Organization* table
- *subjectid* for the *Subject* table
- *userid* for the *User* table
- *geoid* for the *Geography* table
- *dateid* for the *Date* table

The foreign key relations are the following:

- *FK-Answers-Organization* on the attribute *organizationid*
- *FK-Answers-Date* on the attribute *dateid*
- *FK-Answers-Subject* on the attribute *subjectid*
- *FK-Answers-User* on the attribute *userid*
- *FK-User-Date* on the attribute *dateid*
- *FK-User-Geography* on the attribute *geoid*

The preferred type for numerical attributes has been *int*, while *nvarchar* has been chosen for strings. The attribute *date* has been uploaded in *date type*.

1.3 Assignment 1: Data preparation and table splitting

The two input datasets, *answerdatacorrect.csv* and *subject_metadata.csv*, have been imported through the Python *csv module*. Both files have been saved as lists of lists, where each inner list is a row of the original file. After having explored the data, the project objectives have been tackled in the following order:

- Data preparation: since the *csv module* imports every item as a string, some data type conversions have been performed. In particular, the *subjectid* attribute, containing a list of subject identifiers, was transformed from a string to a list in order to then obtain the *Description* attribute (e.g. *"[3, 101, 115, 342]"* to *[3, 101, 115, 342]*). At the end of this process the only data types present in the data are strings and integers, with the exception of *subjectid*.

- Creation of the *Continent* attribute: to add the continent to each row it has been necessary to get data from the web. The table at *https://country-code.cl/*, which contains a list of every country with continent and country code, has been converted into a csv file (*countrycode_continent.csv*) with the Excel *From Web* tool. The ***getContinent*** function takes each *Region* and looks for a match in the lastly obtained table, adding the corresponding continent.

- Creation of the *isCorrect* attribute: this binary attribute has been created checking the equivalence between *AnswerGiven* and *CorrectAnswer*.

- Creation of the *Description* attribute: the creation of this attribute had to be performed using both the input datasets. The ***getDescription*** functions gives two results: first it sorts the *subjectid* list by level, checking for each id the corresponding subject level in *subject_metadata*; then it creates a string (the *Description*) concatenating the name of each subject listed in *subjectid* (e.g. "*Maths* | *Data and Statistics* | *Data Representation* | *Scatter Diagram"*).

- *Date* format conversion: since both the attributes *DateOfBirth* and *DateAnswered* need to be uploaded to the same table, the attributes have been transformed into identifiers, in order to be used as foreign keys. Both attributes have been converted to the *YYYYMMDD* format.

- *Gender* attribute conversion: since the attribute presented two types of values, *1* and *2*, it has been necessary to infer the gender. Since the people described by the data are too young to attend University degrees, no argument regarding males and females in STEM courses could be made. Given

the data distribution (*1* = 272630, *2* = 266205), and given the fact that the gender ratio in the countries listed in the data is slightly unbalanced towards females (*https://ourworldindata.org/gender-ratio*), *1* has been replaced with *F* and *2* with *M*.

- Creation of the attributes *geoid and organizationid*: while *geoid* depends on the *Region* attribute, *organizationid* depends on *userid, quizid* and *schemeofworkid*. In simpler terms, an "organization" consists of a group, attending a certain quiz with a certain scheme of work. The ***idCreator*** function works in two phases. The first scan of the data creates a Python dictionary without duplicate values, where each key is an identifier and the value is a list of the values of the attributes that form a distinct entity. This list is formed by scanning each row and considering only the attributes that need to be identified (e.g. *24: [center italy]* for geoid or *1: [3833, 15391, 8409]* for *organizationid*). During the second phase the data is scanned again row by arrow checking the values of interest (e.g. *Region* = *center italy*, and for each row the function looks for the corresponding id in the dictionary and adds it to the row.

- Conversion of *subjectid*: a list type attribute cannot be uploaded to Microsoft SQL Server, therefore *subjectid* has been transformed into a string (e.g. *3-101-342-115*).

- Splitting data into the six tables: the ***createFactTable*** function selects from the data only the columns to be inserted into the fact table. On the other hand, ***createDimensionTable*** does the same thing but it also removes duplicates and sorts the obtained tables by id. As previously mentioned, the *Date* table has been created taking the values of both *DateOfBirth* and *DateAnswered*.

- Creation of the attributes *date*, *day*, *month*, *year* and *quarter* in the *Date* table: *dateid*, saved in the *YYYYMMDD* format, has been transformed back into the *YYYY-MM-DD* format to obtain the *date* attribute, from which the other attributes were extracted.

At the end of this process the obtained tables (lists of rows) have been exported as csv files, ready to be uploaded to the database in ***Assignment2.py***.

1.4 Assignment 2: Uploading the data

After having established the connection to the database with the *pyodbc module*, the six csv files have been imported. The numeric attributes have been converted from strings to integers. The tables have been uploaded starting from the dimension tables using the ***sqlUploader*** function, which cleans the table on the server with the *DELETE FROM TableName* command before uploading the data. Figure 1.2 shows the *sqlUploader* function.

```
def sqlUploader(table, tab_name):
    # Cleaning the table (TRUNCATE returns error because there is a FK constraint)
    # Square brackets to avoid errors with reserved SQL keywords (e.g. User)
    sql_delete = 'DELETE FROM [{}].[{}];'.format(username,tab_name)
    cursor.execute(sql_delete)
    print(tab_name, ': deleted')

    for row in table[1:]: # Skipping header
        sql_insert_into = 'INSERT INTO [{}].[{}] VALUES'.format(username,tab_name)
        sql_values = str(tuple([x for x in row]))
        sql_final = sql_insert_into + sql_values
        cursor.execute(sql_final)

    cnxn.commit()
    print(tab_name, ': uploaded')
    return table
```

Figure 1.2: sqlUploader function

Part 2: SQL Server Integration Services (SSIS)

1.1 Introduction

Part 2 requires solving some problems with the use of SQL Server Integration Services (SSIS) on Visual Studio. In particular, it is required to use standard SSIS nodes and to avoid SQL querying inside the nodes.

1.2 Assignment 0: For every country, the number of total answers.

After connecting to the *Answers* table to retrieve the attributes *answered* and *userid*, two lookup nodes have been used in order to add the attributes *geoid* and *userid* to the selection. After grouping by *country_name* and counting the answers, the result has been sorted in descending order for clarity. The output of the query is stored in *output_assignment0.txt*.

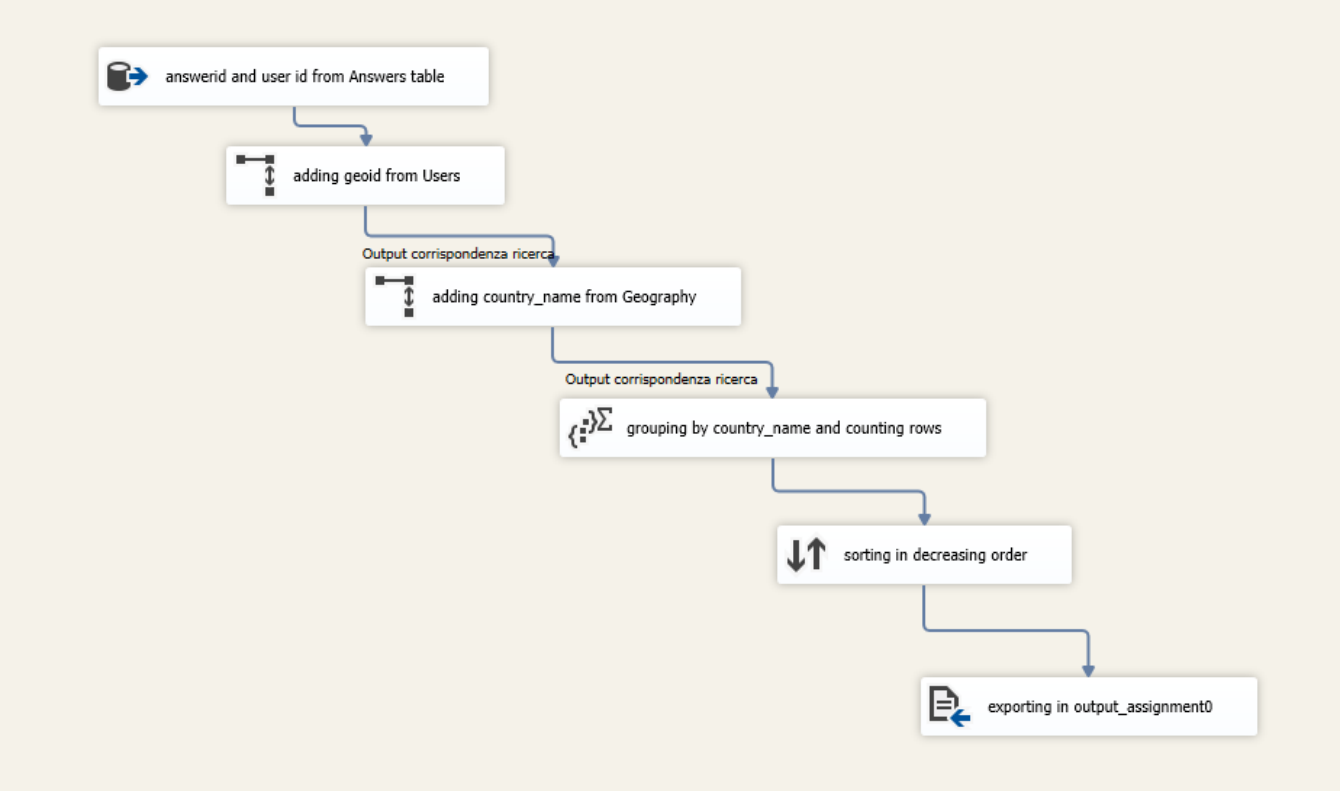

Figure 2.1: SSIS, Assignment 0

1.3 Assignment 1: List all the groups with an age mismatch.

A group (identified by *GroupId*) is said to have an age mismatch if the difference between the date of birth of the youngest participating student and the oldest is greater than 365 days. The output of the query is stored in *output_assignment1.txt*.

After connecting to the *Answers* table, the selection has been joined with the *Organization* table in order to take the *groupid* field, then with the *User* table to take the *dateid* field and lastly with the *Date* table to take the *date* field. Then, after grouping by *groupid*, it was possible to obtain the maximum and the minimum date for each group. After that, the column *age_mismatch* has been obtained using the *DATEDIFF* function. The column contains a boolean value (*TRUE* if the difference between the minimum date and maximum date is greater than 365, otherwise *FALSE*). The flow has been filtered keeping the rows where *age_mismatch=TRUE* and the result has been exported into a flat file.

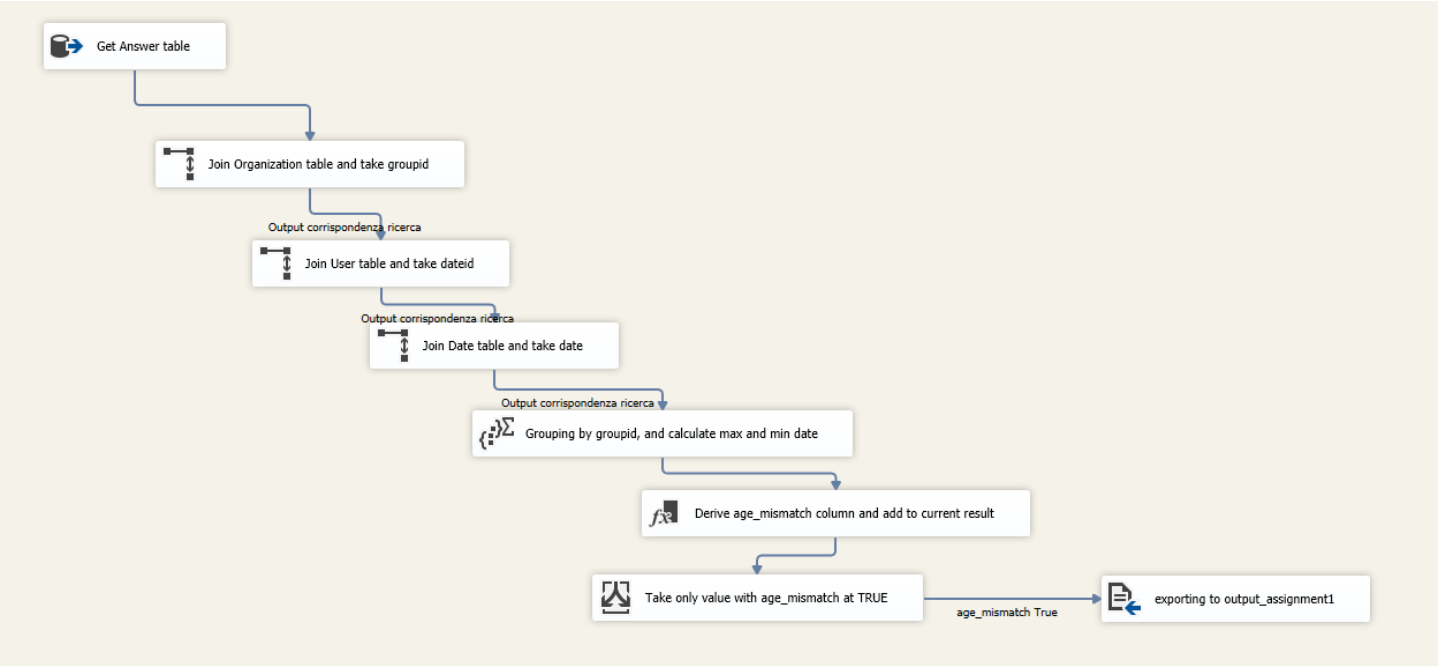

Figure 2.2: SSIS, Assignment 1

1.4 Assignment 2: For each continent the ratio between correct answers of males and correct answers of females.

After connecting to the *Answers* table to retrieve the attributes *userid* and *is correct*, a filter node has been used in order to exclude the wrong answers. Then, two lookup nodes have been used to attach the attributes *gender* and *continent*. The rows have been splitted by gender, and for each gender the answers have been grouped by continent in order to obtain the count of correct answers for each continent and for each gender. The two obtained columns (Correct answers by males and Correct Answers by females) have been used to obtain the column *Ratio*. The output of the query is stored in *output_assignment2.txt*. The result includes the two additional columns *Correct answers by males* and *Correct answers by females*. The ratio between these columns gives the result.

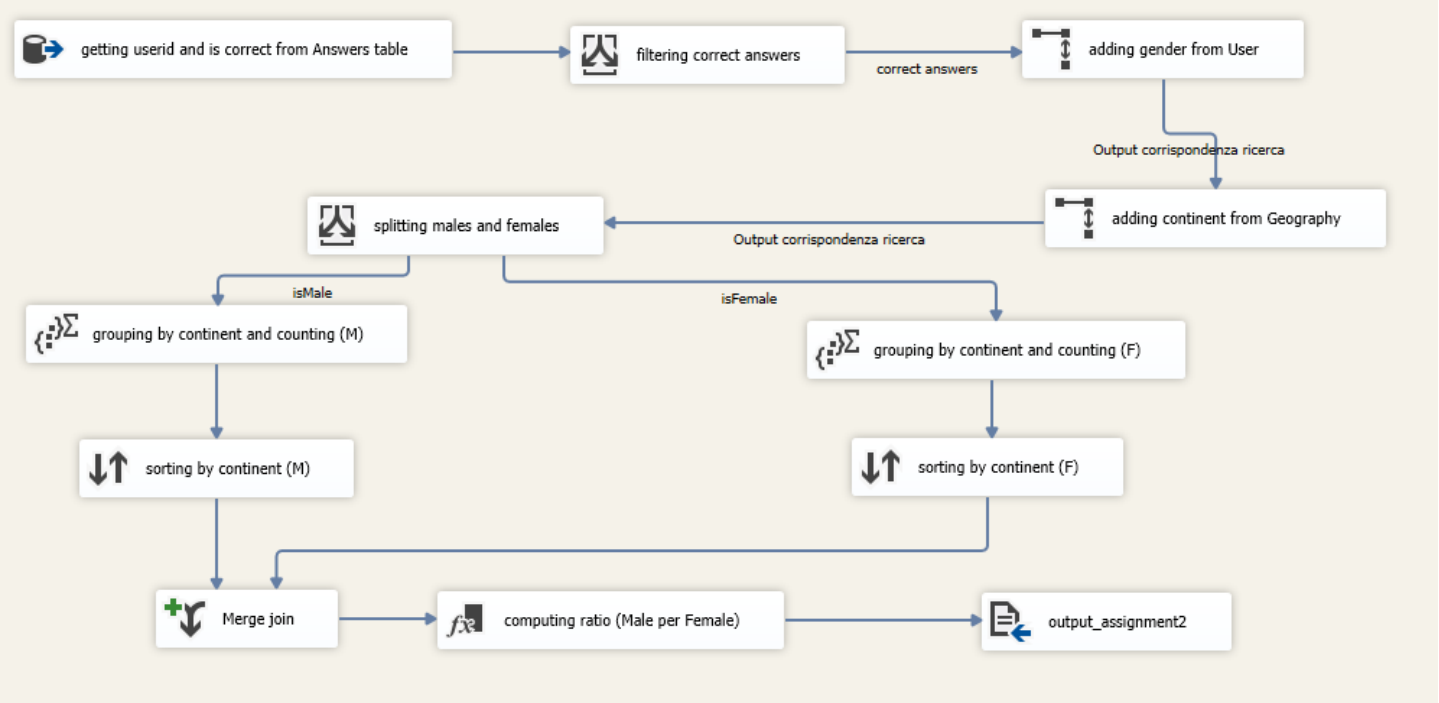

Figure 2.3: SSIS, Assignment 2

Part 3: SQL Server Analysis Services (SSAS)

1.1 Introduction

Part 3 requires to create an OLAP cube in Visual Studio, to write three MDX queries in SQL Management Studio and to create two interactive dashboards with Power BI.

1.2 Assignment 0: Build a datacube from the data of the tables in your database, defining the appropriate hierarchies for time and geography. Create the needed measures based on the queries you need to answer.

After establishing a connection to *Group_18_DB* and creating a view of said database, it has been necessary to create two *New named calculations*:

- *notcorrect*: boolean attribute with value 0 if *iscorrect* is 1, 0 otherwise. It has been used both for the MDX queries and the dashboards.
- *ContinentExtended*: string attribute with value *'North America'* when *Continent* is *'NA'*, *'Europe'* when it is *'EU'*, *'Oceania'* when it is *'OC'*. This extension has been created for the dashboard described at paragraph 1.6 (Assignment 4), which could not locate the abbreviations.

The following step has been creating the dimensions, two of which were hierarchies:

- The *Geography* table has been used to create the hierarchy in Figure 3.1.

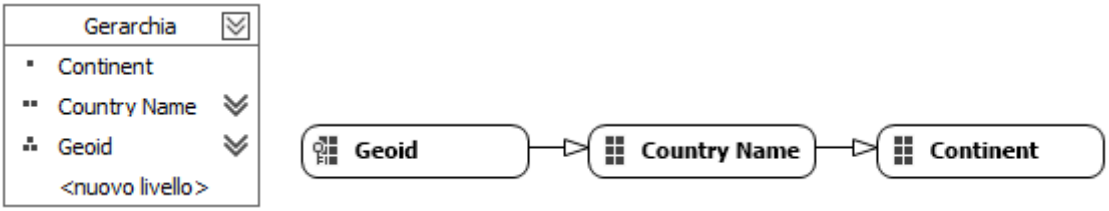

Figure 3.1: Geography hierarchy

- The *Date* table has been used to create the hierarchy in Figure 3.2. This hierarchy does not contain functional dependencies among the attributes.

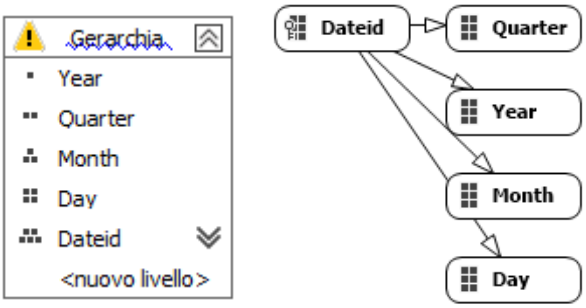

Figure 3.2: Date hierarchy

After these steps the cube has been created and uploaded to the server with name ***Group_18_cube***.

1.3 Assignment 1: Show the percentage increase or decrease in correct answers with respect to the previous year for each student.

After creating the members *correct2019* and *correct2020* with the correct answers of each year it has been possible to calculate the percentage increase or decrease in correct answers for each student. It is worth pointing out that the dataset contains answers only for the years 2019 and 2020 and that there are some missing values. The query is reported in Figure 3.3, while a sample of the results is shown in Figure 3.4.

```
with member correct2019 as
([Measures].[Iscorrect],[Date].[Year].&[2019])

member correct2020 as
([Measures].[Iscorrect],[Date].[Year].&[2020])

member percentage_improvement as
case when correct2020 = 0 or correct2020 = NULL then 'Missing data'
when correct2019 = 0  or correct2020 = NULL then 'Missing data'
else ROUND(((correct2020-correct2019)/(correct2019)),2) end,
format_string="percent"

select {correct2019, correct2020, percentage_improvement} on columns,
[User].[Userid].[Userid] on rows
from [Group_18_cube];
```

Figure 3.3: MDX query for Assignment 1

Messages Results

	correct2019	correct2020	percentage_improvement
47545	74	22	-70.00%
47552	29	(null)	Missing data
47559	11	(null)	Missing data
47570	6	(null)	Missing data
47583	(null)	1	Missing data
47590	68	38	-44.00%
47599	20	1	-95.00%
47611	10	2	-80.00%
47612	81	41	-49.00%
47629	46	6	-87.00%
47686	9	(null)	Missing data
47738	(null)	3	Missing data
47743	9	(null)	Missing data
47746	37	(null)	Missing data
47763	21	0	Missing data
47765	31	(null)	Missing data
47804	10	11	10.00%

Figure 3.4: Result of Assignment 1

1.4 Assignment 2: For each subject show the total correct answers in percentage with respect to the total answers of that subject.

After creating the member *percentage_of_correct_answers* as the ratio between correct answers and total answers, the measure has been calculated for each subject. The query is reported in Figure 3.5 and a sample of the results is presented in Figure 3.6.

```
with member percentage_of_correct_answers as
([Measures].[Iscorrect])/([Measures].[Conteggio di Answers]),
format_string="percent"

select {[Measures].[Conteggio di Answers],
[Measures].[Iscorrect], percentage_of_correct_answers} on columns,
([Subject].[Subjectid].[Subjectid], [Subject].[Description].[Description]) on rows
from [Group_18_cube];
```

Figure 3.5: MDX query for Assignment 2

Messages Results

		Conteggio di Answers	Iscorrect	percentage_of_correct_answer
3-101-113-1188	Maths | Data and Statistics | Data Collection | Data Collection-...	426	229	53.76%
3-101-113-336	Maths | Data and Statistics | Data Collection | Tally Charts	451	380	84.26%
3-101-113-337	Maths | Data and Statistics | Data Collection | Types of Data a...	1537	817	53.16%
3-101-338-102	Maths | Data and Statistics | Data Processing | Averages (me...	5781	3789	65.54%
3-101-338-102-339	Maths | Data and Statistics | Data Processing | Averages (me...	89	47	52.81%
3-101-338-103	Maths | Data and Statistics | Data Processing | Averages and...	1372	756	55.10%
3-101-338-114	Maths | Data and Statistics | Data Processing | Sampling and ...	422	216	51.18%
3-101-338-339	Maths | Data and Statistics | Data Processing | Range and Int...	275	188	68.36%
3-101-338-340	Maths | Data and Statistics | Data Processing | Averages and...	573	265	46.25%
3-101-338-341	Maths | Data and Statistics | Data Processing | Interpreting an...	141	57	40.43%
3-101-338-342-102-106	Maths | Data and Statistics | Data Processing | Data Represe...	74	44	59.46%
3-101-338-342-102-116	Maths | Data and Statistics | Data Processing | Data Represe...	62	26	41.94%
3-101-338-342-104-339	Maths | Data and Statistics | Data Processing | Data Represe...	78	54	69.23%
3-101-338-342-106-340	Maths | Data and Statistics | Data Processing | Data Represe...	112	47	41.96%

Figure 3.6: Result of Assignment 2

1.5 Assignment 3: Show the students having a total incorrect answers greater or equal than the average incorrect answers in each continent.

This assignment required some changes to the *Group_18_cube*, therefore an alternative cube has been built solely for solving this query, called ***Group_18_cube_alt***. In particular, this query required that the attribute *Continent* and the attribute *Userid* were in the same dimension: for this reason the new cube includes a new dimension, called *User-Geo*, which includes the attributes from both the *User* table and the *Geography* table. Figure 3.7 shows the new hierarchy built ad-hoc for this query. Including both *Continent* and *Userid* in the same hierarchy allowed the use of the membership functions *currentmember* and *parent*. As for the main cube, the alternative cube includes measures for both the *Answers* table and the *User* table, since it is allowed by Visual Studio. The measure *[Conteggio di User]* (i.e. count of *User*) has been used to calculate the average of incorrect answers in each continent.

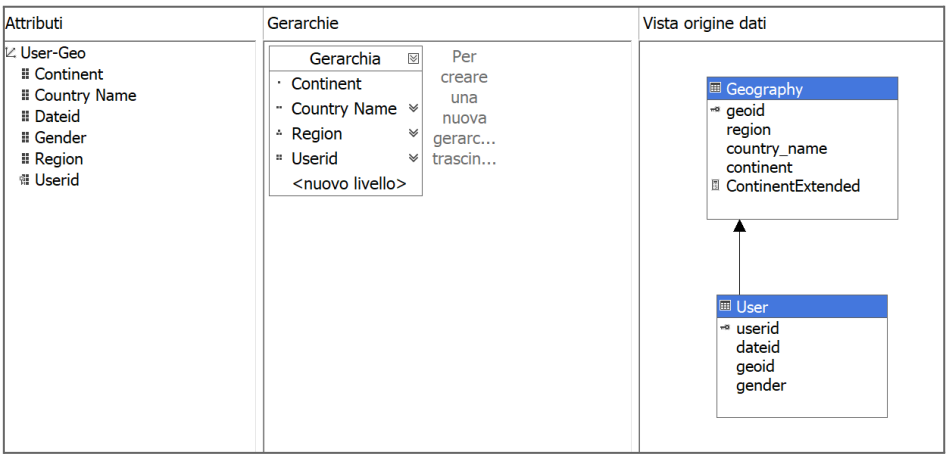

Figure 3.7: User-Geo dimension

```
with member avg_incorrect_answers as
[Measures].[Notcorrect]/[Measures].[Conteggio di User]

member continent_avg as
([User-Geo].[Gerarchia].currentmember.parent.parent.parent, avg_incorrect_answers)

select {[Measures].[Notcorrect], continent_avg} on columns,
filter(([User-Geo].[Gerarchia].[Userid],[User-Geo].[Continent].[Continent]),
[Measures].[Notcorrect] >= continent_avg) on rows
from [Group_18_cube_alt]
```

Figure 3.8: Result of Assignment 3

		Notcorrect	continent_avg
1345	EU	53	14.3128047679249
1363	EU	58	14.3128047679249
3697	EU	15	14.3128047679249
5030	EU	48	14.3128047679249
11359	EU	144	14.3128047679249
11380	EU	21	14.3128047679249
13300	EU	19	14.3128047679249
25059	EU	27	14.3128047679249
31893	EU	87	14.3128047679249
37734	EU	17	14.3128047679249
43394	EU	32	14.3128047679249
52163	EU	125	14.3128047679249
53017	EU	40	14.3128047679249
53364	EU	131	14.3128047679249
56673	EU	30	14.3128047679249
56812	EU	19	14.3128047679249
59098	EU	29	14.3128047679249
60227	EU	24	14.3128047679249
62380	EU	28	14.3128047679249
65803	EU	32	14.3128047679249
73738	EU	37	14.3128047679249
74963	EU	20	14.3128047679249
75684	EU	83	14.3128047679249

Figure 3.9: Result of Assignment 3

1.6 Assignment 4: Create a dashboard that shows the geographical distribution of correct answers and incorrect answers.

The dashboard shown in Figure 3.10 was built in Power BI after connecting to the *Group_18_cube*. It involves the use of two map charts, one for the geographical distribution of correct answers (in green) and one for the distribution of incorrect answers (in red), plus the use of a stacked bar chart to have a quantitative measure of the distribution of the answers. Both types of charts allow roll-up and drill-down movements through the Geography hierarchy.

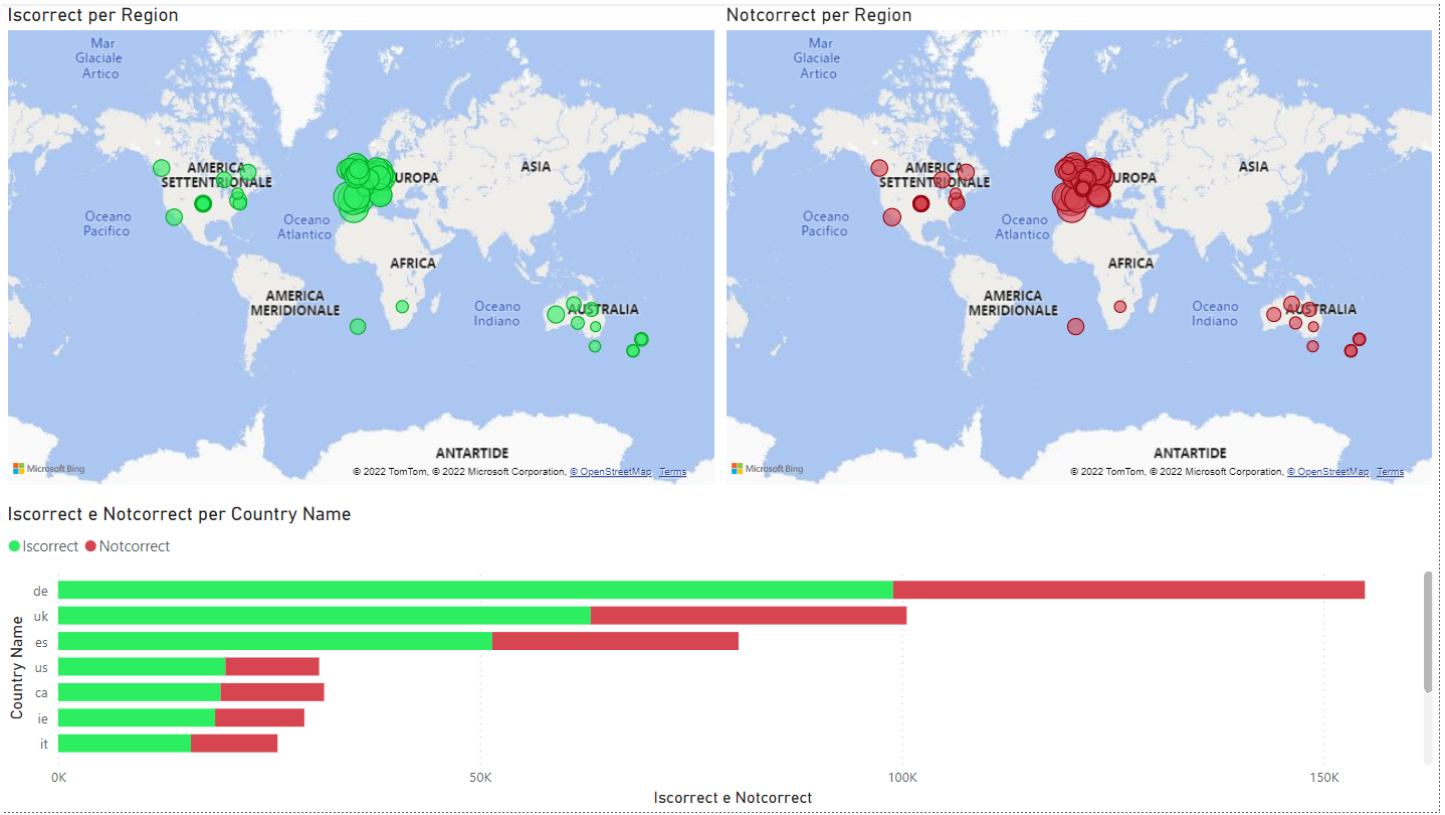

Figure 3.10: Geographical distribution of correct and incorrect answers

1.7 Assignment 5: Create a plot/dashboard of your choosing, that you deem interesting w.r.t. the data available in your cube.

The dashboard shown in Figure 3.11 shows the distribution of correct answers with regards to the gender and the time of the year. The first pie chart gives an interactive view on the distribution of answers given by each gender, and it serves as a reference for the second pie chart, which focuses just on correct answers. The line chart at the bottom shows the distribution of correct answers by males and females through time, and it allows roll-up and drill-down movements through the Date hierarchy.

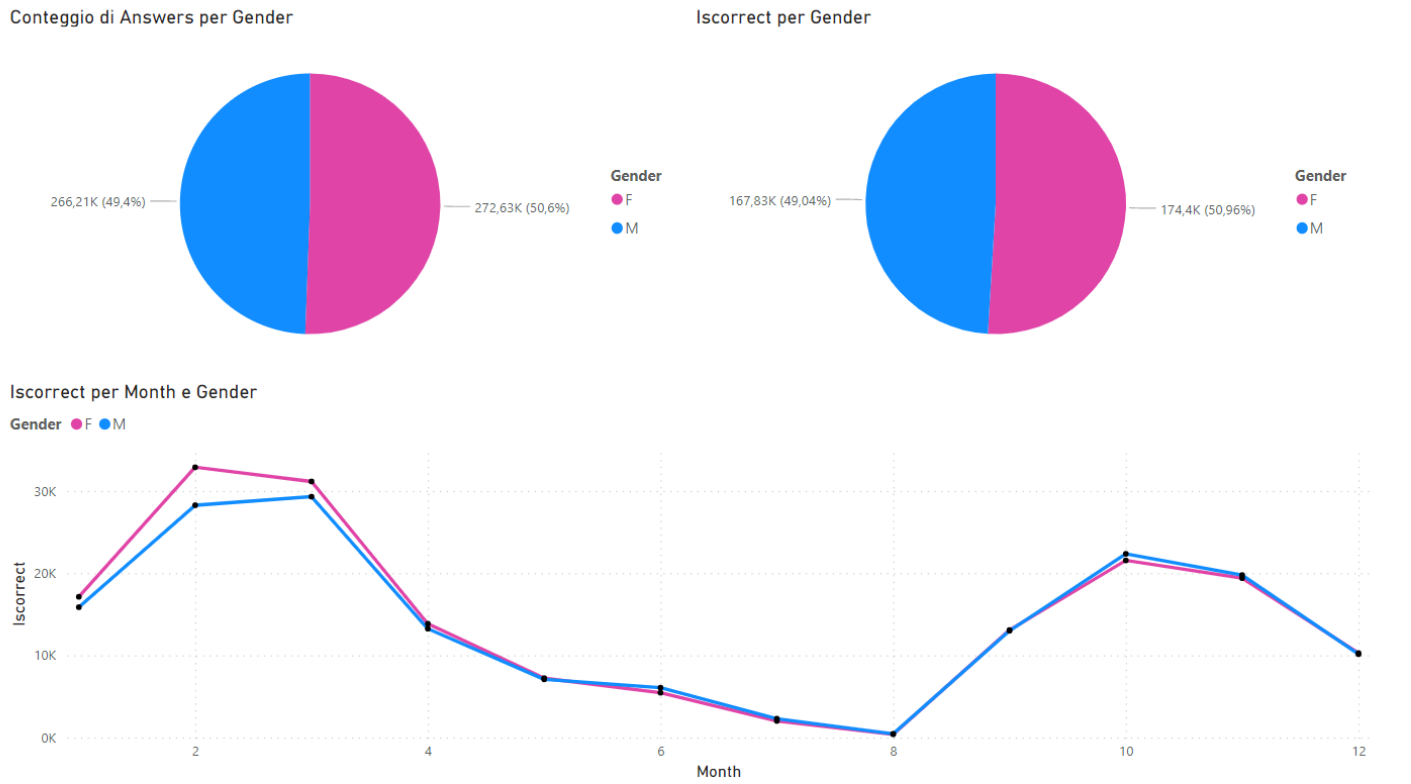

Figure 3.11: Gender distribution of correct and incorrect answers