Free University of Bolzano
Faculty of Computer Science
Computational Data Science

Course: Data Integration

Project: HETOR - La Campania da scoprire

Professor:
Prof. Diego Calvanese

Authors:
Marco Di Panfilo
Marco Zenere

Academic Year: 2021/2022
Bolzano - 10/02/2022

Contents

1 Introduction

1.1 Motivation Scenario

Cultural heritage is a resource with a strategic value for each country's social and economic development, particularly for Italy, famous for having one of the largest heritage in the world. Considering the continent we live in, the European Union supports the protection and the promotion of the cultural heritage of the state members. In recent years, several projects were born for supporting the European initiatives, and HETOR [1] is one of them, focused on the heritage in the Campania region. HETOR project aims to exploit the Open Data power to reveal the Campania cultural heritage and rekindle people's interest in the "old things" in Campania.

1.2 Domain specification

The HETOR project includes several domains of the Campania heritage, each composed of multiple datasets. For the Data Integration project, our group focused on the itineraries domain composed of the following four datasets:

- Itineraries - Caserta [CSV]
- Itineraries - Nocera Inferiore [CSV]
- Itineraries - Nocera [CSV]
- Itineraries - Costiera Amalfitana [CSV]

Each of the files includes information about an itinerary, such as geographical coordinates, type of itinerary, bibliography, and sitography. Still, it is important to mention the inconsistency of the information among the files. Some information in one file may be missing in another, even though the files refer to the same domain. Sometimes it's not possible to extract the same type of information for each type of itinerary in the datasets (cultural, company, culinary). The list of the attributes of each dataset, together with the datasets themselves, can be found in the HETOR project's shared folder [2].

1.3 Expected Result

The expected result for this project is to have an application able to formulate SPARQL queries over an ontology representing the domain of interest, designed by ourselves. The expected application for this project elaborates the answers retrieved through SPARQL queries for better comprehension of the data to the system's users.

2 System Architecture

The developed system is composed of four layers, starting from the lowest layer of the architecture we have:

- **CSVs Files:** Represents the data sources of our domain of interest. The files are loaded into the database through the extract, transform, and load process (ETL) using a Jupyter notebook written in Python.
- **Database Layer:** Stores the datasets of the itineraries domain.
- **Ontology Layer:** Designed in Protégé, represents the ontology of the domain. The ontology objects are constructed using the mapping defined in OBDA language.
- **Query Engine Layer:** Provided by OnTop reasoner, gives the capability to access data using the SPARQL query language.

[1] http://www.hetor.it/site/en/il-progetto-2/hetor/
[2] https://drive.google.com/drive/u/0/folders/1o-IKgdpY4W7xto9fdoFjkR1Ysv_PYpxx

- **Application Layer:** Through Jupyter notebook written in Python, gives the possibility to process data retrieved from the ontology through the usage of SPARQL queries.

The figure below provides a graphical representation of the system architecture.

Figure 1: System architecture

3 Data cleaning and ETL

Before importing the datasets into the relational database, the CSV files of the domain were cleaned following the detailed instructions [3] provided by the data owner. The proposed data cleaning was mostly related to the alignment of the data to standards (ISTAT names of the municipalities), renaming of NULL values, and consistency check of the values of each attribute.

We created an appropriate method to align the data of the municipalities with the ISTAT names, taking into account also official names in foreign languages. The method checks first if there is a perfect match, then if the string is partially contained as a whole word in the ISTAT list, and finally returns the municipality with the closest Levensthein-distance. Nevertheless, in some cases, the location indicated in the files was not the municipality but of a small village in the municipality such that a manual correction was needed.

Most of the itineraries contained geographical data like latitude and longitude. Some contained GeoJson strings for more complex spatial data. We used GeoPandas to read them and transformed them to latitude and longitude coordinates of their centroids.

Empty values and other forms of empty or null values were replaced with NULL.

Detailed descriptions and the code used to clean the data can be found in the Jupyter notebooks in the Data_Cleaning folder.

The first idea of the database schema was to identically import the CSV to the relational schema. The problem of this design choice was related to very poor relations among the different datasets. Therefore we decided to design the database differently and create three different types of itinerary categories. Additionally, we created a table for *comune*, *luogo_di_interesse*, and azienda, which we referenced with foreign keys in the tables of the itineraries. In order to import the data to the new schema, we mapped the columns of the CSV to the new schema in the database. All mappings and steps needed to extract the data from the CSV can be found in the Create_database.ipynb notebook in the Database folder.

4 Relational Database

Figure 2 depicts the ER schema of the database designed for this project. The designed relational database consists of six relational schemas:

- **Comune/Municipality:** The table includes information regarding the name, zip code, province, region of each municipality.
- **Azienda/Company:** The table includes information regarding the local companies in the Campania region.
- **Luogo di interesse/Point of interest:** The table includes the point of interest, such as churches, monuments, or statues of the Campania region.

[3] https://docs.google.com/document/d/1DQEBDH4wz83fhDUaupdxuAW2YuS2tmPLsUtJNpC5-BI/edit

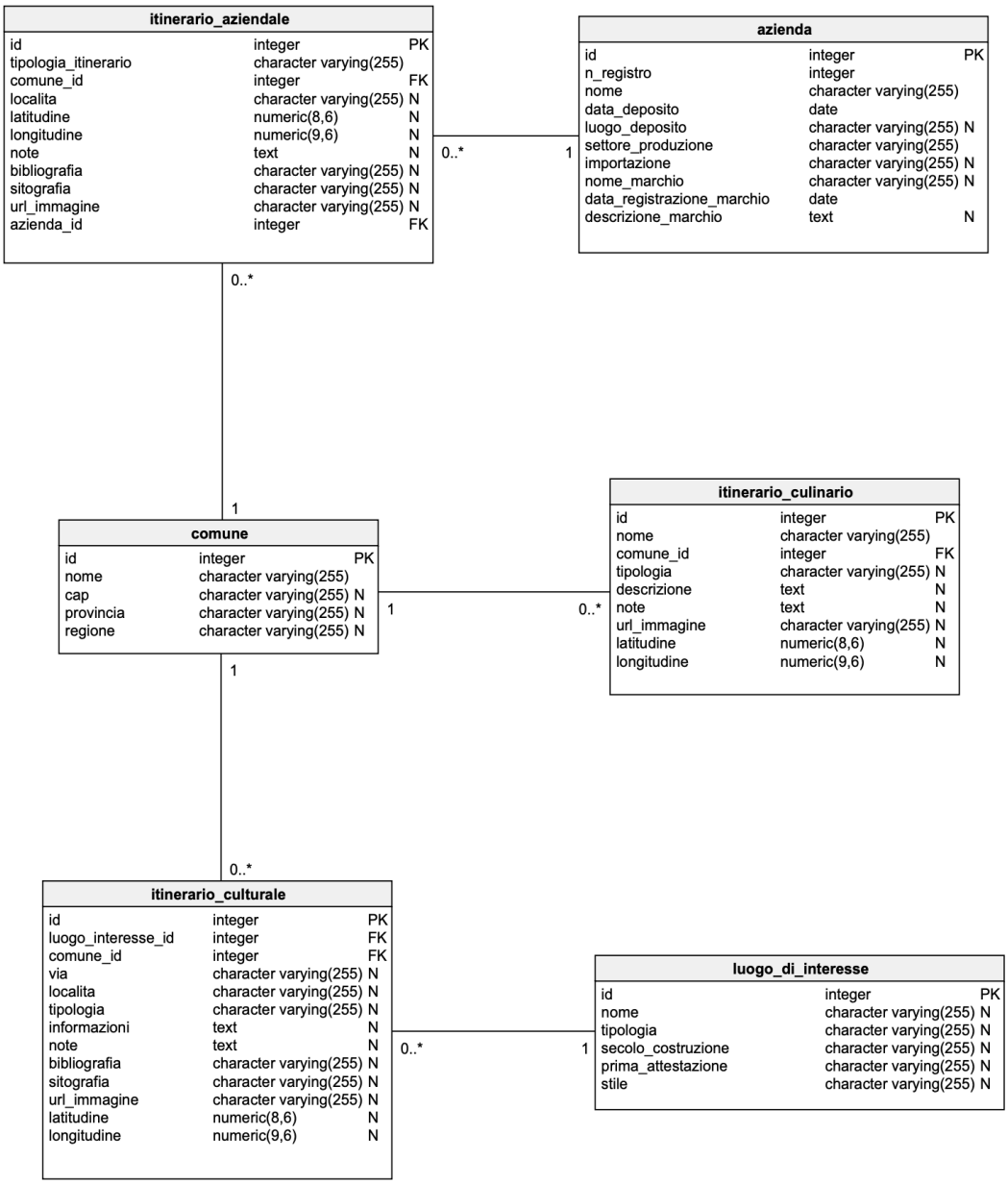

Figure 2: ER schema of the relational database

- **Itinerario aziendale/Company Itinerary:** The table includes the information of itineraries that involve visiting a local company in the Campania region.
- **Itinerario Culinario/Culinary Itinerary:** The table includes the information of itineraries that involve visiting a place where it is possible to eat a typical culinary product of the Campania region.
- **Itinerario Culturale/Cultural Itinerary:** The table includes the information of itineraries that involve visiting a point of interest of the Campania region.

5 Ontology

This chapter describes the Ontology that we used for our project.

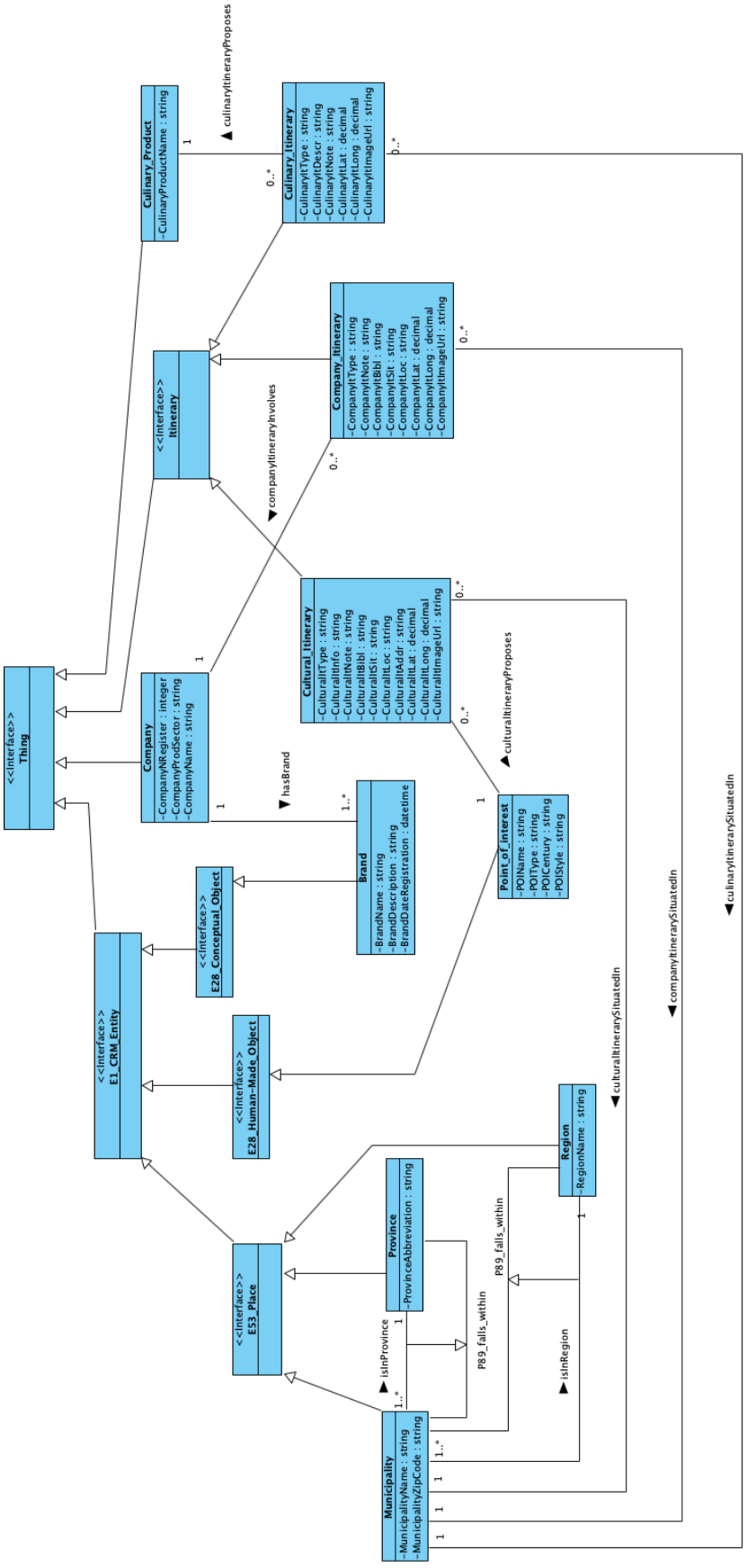

Figure 3: Ontology representation as UML class diagram

Figure 3 shows a graphical representation of the designed ontology for the itineraries domain in the form of a UML class diagram. The proposed ontology comprises sixteen entities, thirty-six data properties, and eleven object properties. The ontology design was conducted by integrating and exploiting as much as possible the ontology standards available nowadays. The designed ontology considers the usage of the CIDOC CRM [4] standard, which defines an ontology for the cultural heritage field. In the following project, the CIDOC CRM ontology was used and extended to include some of the entities and object properties in our domain of interest entities. The CIDOC CRM standard does not satisfy all our needs, and for this reason, some of the entities we wanted to design for the ontology are outside of CIDOC CRM and are defined as part of the custom ontology.

The designed ontology is not a 1:1 representation of the relational schemas in the relational database, but there are more entities than relational schema. Some of the entities are constructed considering the same relational schema, and the goal of this choice is to give more expressivity to the ontology. An example is the entity Brand, which comes from the same relational schema of the Company entity and constructed considering some columns of Azienda/Company relational schema.

6 Mapping

Mapping is the technology in data integration used to connect the lowest layers of our system, the ontology, and the relational database. Instead of using the R2RML mapping language, we chose to use the more intuitive and straightforward OBDA mapping language. The design of the set of mappings assertions was done using the mapping manager of Protégé rather than using the ONTOPIC tool. The design of the mapping assertion follows as much as possible the mapping patterns explained in the data integration course. Still, those patterns are not always applicable since they do not cover all the cases. For this reason, some mapping is a small variation of them. The following chapter will highlight the main mapping patterns used for mapping entities and object properties of the designed ontology.

Figure 4: Mapping pattern: Entity(MpE)

Figure 4 shows the typical mapping pattern for the entities suggested in the data integration slides. Most of the designed ontology entities were mapped by following the above design pattern since there is a 1:1 correspondence between such entities and the relational tables in the relational database. However, as mentioned in chapter 5, there is not a 1:1 correspondence between relational tables and all the entities, and some of the entities are constructed considering some attributes of a specific relational schema. The mapping pattern applied for those entities is the same as depicted in figure 4, but the "primary key" used for those entities corresponds to a column of the relational table that ensures the primary key functionality, the possibility to identify the entity uniquely. An example is the entity Province, which is mapped considering some of the comune/municipality relational schema columns.

Figure 6 shows the typical pattern suggested in the data integration slides for mapping relationship with merging. For most of the object properties of the ontology, it was followed the mapping pattern depicted in figure 6. Still, as mentioned previously, the ontology also includes entities that

[4] https://cidoc-crm.org

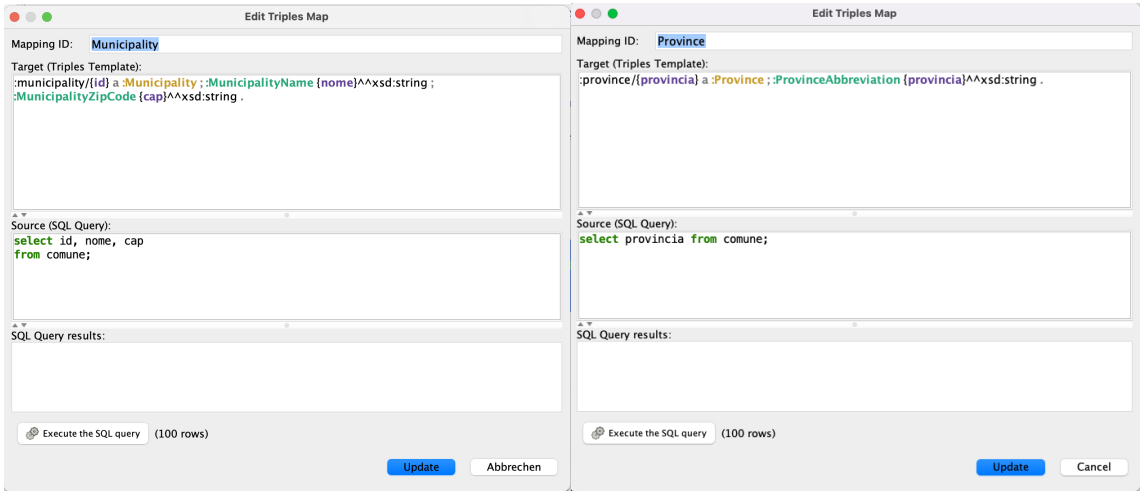

Figure 5: On the left the mapping done by following the pattern, on the right the adaptation when there is not a 1:1 correlation between entity and relational schema

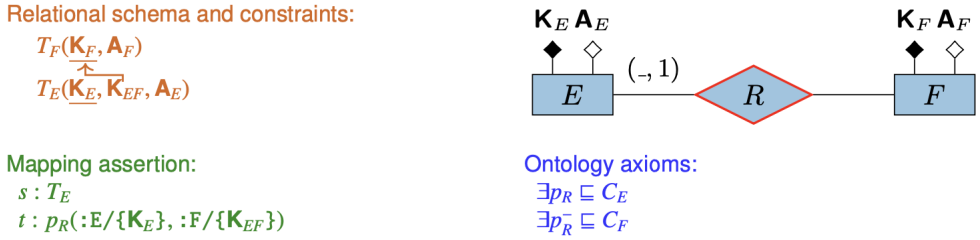

Figure 6: Mapping pattern: Relationship with Merging(MpRm)

do not correspond to a specific table on the relational database. So, considering our case, in the designed ontology, there are also object properties that link entities of the ontology related to the same tables. Since there is no specific mapping pattern covering the following case, we decided to apply the same mapping pattern we used for mapping the other object properties of the ontology.

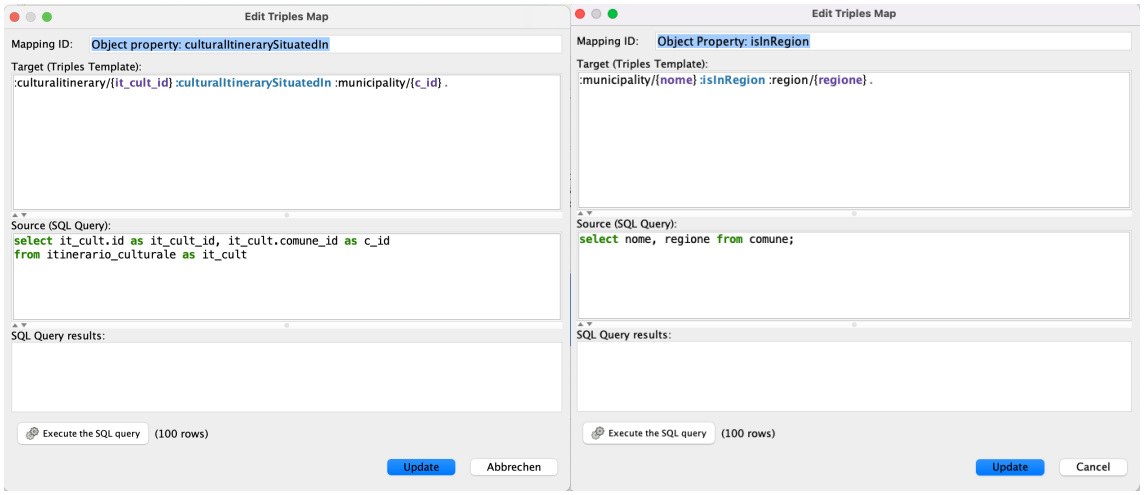

Figure 7: On the the left the mapping done by following the pattern, on the right the adaptation when there is not a 1:1 correlation between entity and relational schema

7 How to run the project

7.1 Folder structure

The following list describes all the folders and files present in the attached ZIP file of the project.

- **environment.yml:** contains the libraries and versions used to create the Python kernel to run the notebooks.
- **Dataset:** contains the original datasets.
- **Dataset_cleaning:** contains the notebooks that were used to clean the datasets.
- **Dataset_clean:** contains the cleaned version of the datasets.
- **Database:** contains the dump file of the database and the notebooks to load the cleaned datasets into the database.
- **JDBC_PostgreSQL_Driver:** contains the drivers used by Protégé to access the database.
- **Ontology:** contains the files that describe the Ontology used with Ontop and Protégé.
- **Application:** contains the files to run the web-application.

7.2 Create Python Kernel

First, install Anaconda by following instructions at https://docs.anaconda.com/anaconda/install/. To run the code, you have to create an anaconda environment with the configuration file environment.yml (step 1), activate it (step 2) and start jupyter notebook (step 3).

```
1. conda env create -f environment.yml
2. conda activate data_integration_project
3. jupyter notebook
4. conda deactivate
5. conda env remove --name data_integration_project
```

To run the Jupyter-notebooks, you have to run the code cells in the ipynb-files after step 3. If the conda environment is not needed anymore, it can be deactivated (step 4) and removed (step 5).

7.3 Data cleaning

In the folder Dataset_cleaning there is a Jupyter notebook for each dataset. The notebooks can be executed following the procedure in 7.2. The notebooks follow the proposed cleaning steps that can be found in HETOR_dataset_correction_proposals.pdf.The file data_cleaning_util.py contains methods that are shared amongst all data cleaning notebooks. The cleaned datasets are saved in Dataset_clean as CSV and pickle.

7.4 Import datasets to PostgreSQL database

If PostgreSQL and pgAdmin are not yet installed, they can be downloaded from [5] and installed.

There are 2 options to import the data into the PostgreSQL database:

- Run Create_database.ipynb. In order to access the database from the Jupyter notebook, it is necessary to set the configurations for the database connection in the database_configuration.txt file. We provide a template in database_configuration_template.txt with default values except for the password. After the configurations have been set in the template, you should rename it to database_configuration.txt. The code in the notebook replaces or creates a new schema named "public" and extracts the data from the cleaned CSV files to the database tables.
- Restore the PostgreSQL dump saved in Data_Integration_Database.sql. Create a new database with pgAdmin and restore the database by right-clicking the database, selecting Restore, entering the file path of the dump file, and pressing on Restore.

[5] https://www.postgresql.org/download/

7.5 Create Ontology on top of PostgresSQL database

7.5.1 Protégé with Ontop Plugin

We used Protégé 5.5.0[6] with the Ontop-Plugin 4.2.0 to create the Ontology. To connect Protégé with the database, it is necessary to add the JDBC-driver and set the parameters for the database.

- To add the driver go to Protégé ⇒ Preferences ⇒ JDBC Driver ⇒ Add. Select org.postgresql.Driver for the Class Name and add the Path to the driver that is stored in the folder JDBC_PostgreSQL_Driver.
- The parameters for the database connections can be set in Ontop Mappings Tab. Go to sub-tab Connection parameters and set the parameters like in Figure 8. Alternatively, you can set the parameters in the file itineraries_ontology_template.properties and rename it to itineraries_ontology.properties.

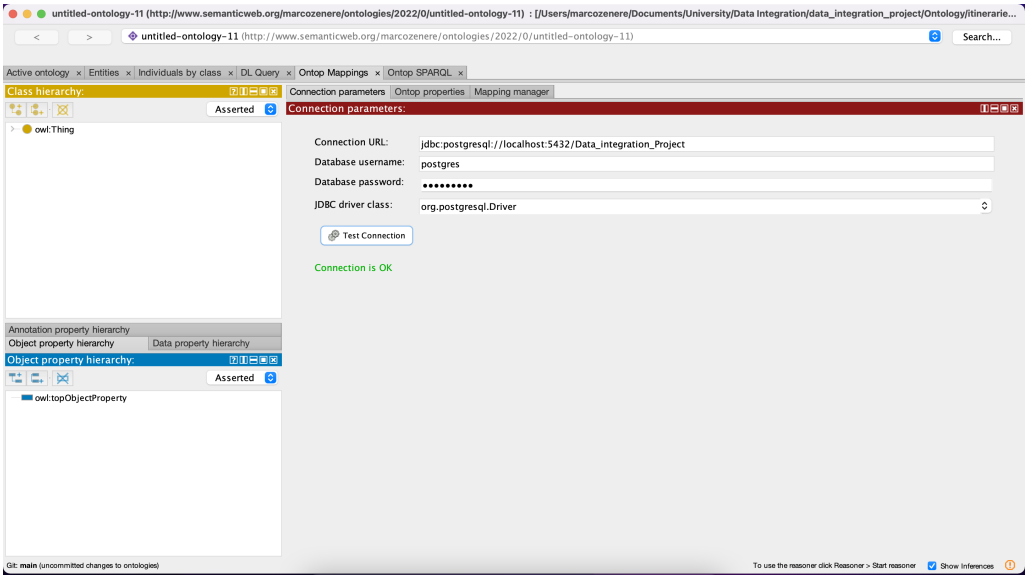

Figure 8: Settings for Protégé to connect to PostgreSQL database

7.5.2 Ontop CLI

To access the Ontology from our application and run queries on it we installed Ontop CLI 4.2.0 following the steps described in Ontop tutorial [7]. After a successful start, the Ontop endpoint will be running at http://localhost:8080/sparql.

In our case, we downloaded Ontop CLI and created a folder input within the Ontop CLI folder, and copied the following files from the Ontology folder:

- itineraries_ontology.owl
- itineraries_ontology.obda
- itineraries_ontology.properties

After that we copied the database driver postgresql-42.3.1.jar from the folder JDBC_PostgreSQL_Driver to the folder jdbc of the Ontop CLI folder. Finally, we can open a shell in the main folder of Ontop CLI and run the Ontop CLI with the following command on a Mac:

```
./ontop endpoint \
    --ontology=input/itineraries_ontology.owl \
    --mapping=input/itineraries_ontology.obda \
    --properties=input/itineraries_ontology.properties
```

[6] https://protege.stanford.edu
[7] https://ontop-vkg.org/tutorial/endpoint/endpoint-cli.html

7.6 Run the web appplication

We developed a web application that allows the user to search for itineraries. The application can be started from the command line. Open a shell in the terminal and go to the folder Application, and activate the conda environment with:

```
conda activate data_integration_project
```

Now you are ready to start the application with the command:

```
python3 application.py [-h] [-r REMOTE_ACCESS] [-p PORT] [-d DEBUG]
                       [-a AUTO_OPEN_WEBBROWSER]
```

By default, the arguments are set to –remote_access False, –port 8080, –debug False, and auto_open_webbrowser True. When setting –remote_access to True, you will get a public, shareable address for the current application. Others can access your application in their browser via this address. Because the processing happens on your device, the app will be running as long as your device stays on.

After running the command, the web application should open automatically in your web browser. If this is not supported or blocked by the system, you can access the app at the address provided in the shell.

8 Application

To demonstrate how an ontology can be used to query data, we developed an interactive web application.

The application was created using the library PyWebIO [8]. The library provides predefined input tools, like check-boxes, selection, text-input, and many more. It provides functionalities to split the web page in different sections (scopes) and to update them when an input field changes. This is also true for text input. In our case, the app will update the results also meanwhile the user types his search keywords. If the system is slow and the typing is fast, not every keystroke will be captured and, therefore, additional checks or a button to submit the query could be added to make the web application more reliable. For the goal of this project, this was not necessary, and it was more interesting to see how the app adjusted the results at each "slow" keystroke by performing new queries to the Ontop endpoint. Since our data contained geographical data, we added a map to show the location of the itineraries. Finally, the application shows a short summary of how many itineraries have been retrieved as well as a longer table containing detailed data and images.

Figure 9 highlights all the sections of the web-applications.

8.1 Filters

In red, we can see the areas of the filters. We implemented the following filters:

- **location:** There are three selection options for region, province and municipality. The possible options are retrieved from the database with SPARQL-queries. It is not necessary to fill all the fields, but the application follows the principle that $region \supseteq province \supseteq municipality$ the web application retrieves all the possible options for the subset after each input. If a value of a subset was set and the value of the superset is changed, the value of the subset will be deleted.
- **category:** There are three check-boxes that can be selected to filter the result based on the category. Multi-selection is possible, and no selection is the same as selecting all.
- **title and description:** Two text input fields allow the user to perform a text search on the datasets. To avoid bad input from the user, only text with either single quotes or double quotes is allowed. The app will alert the user in case of bad input.

The input from the filter is stored in a dictionary filter_values which is used to create parametric queries to the Ontop Endpoint. The following code represents a parametric SPARQL query used to retrieve culinary itineraries.

[8] https://www.pyweb.io

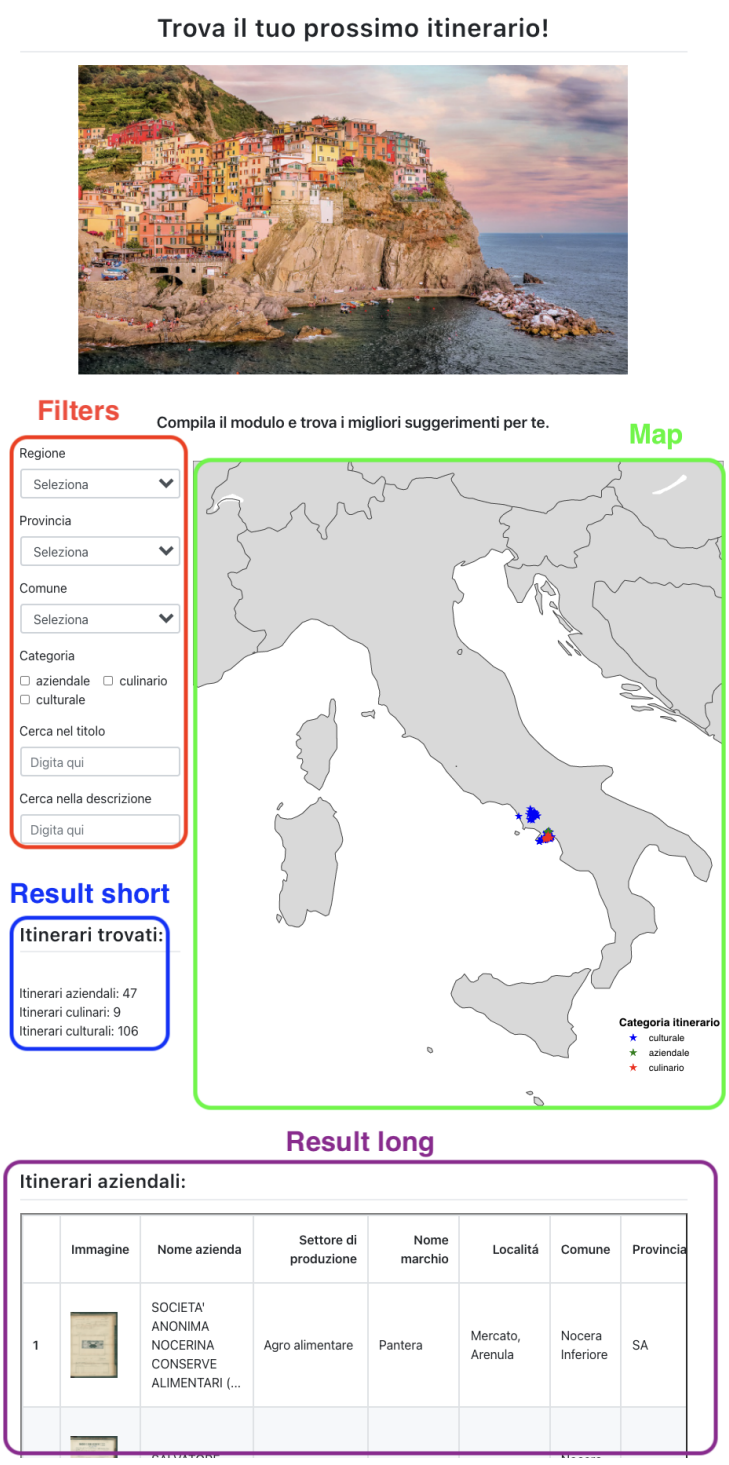

Figure 9: Screenshot of the app with highlighted areas: filter (red), map (green), summary (blue), result table (violet)

```
query_culinary_itineraries = f"""
PREFIX : <http://www.semanticweb.org/marcozenere/ontologies/2022/0/\
          untitled-ontology-11#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

select ?image_url ?name ?municipality ?province ?region ?type
       ?description ?notes ?latitude ?longitude
```

```
where{{
?x rdf:type :Culinary_itinerary;
    :CulinaryItLat ?latitude;
    :CulinaryItLong ?longitude;
    :culinaryItineraryProposes ?culinaryProduct;
    :culinaryItinerarySituatedIn ?municipality_id.

?municipality_id :MunicipalityName {'?municipality' \
                                if filter_values['municipality'] is None \
                                else surround_with_quotes(municipality)};
    :isInProvince ?province_id;
    :isInRegion ?region_id.
?province_id :ProvinceAbbreviation {'?province' \
                        if filter_values['province'] is None \
                        else surround_with_quotes(filter_values['province'])}.
?region_id :RegionName {'?region' \
                        if filter_values['region'] is None \
                        else surround_with_quotes(filter_values['region'])}.

?culinaryProduct :CulinaryProductName ?name.

optional {{ ?x :CulinaryItType ?type }}
optional {{ ?x :CulinaryItDescr ?description}}
optional {{ ?x :CulinaryItNote ?notes }}
optional {{ ?x :CulinaryItImageUrl ?image_url }}

{"" if filter_values['title_search_keyword'] is None \
    else f"filter (contains(?name, \
        {surround_with_quotes(filter_values['title_search_keyword'])}))"}
{"" if filter_values['description_search_keyword'] is None \
    else f"filter (contains(?description, \
        {surround_with_quotes(filter_values['description_search_keyword'])}))"}
}}
"""
culinary_itineraries_result_df = \
                    sparql_dataframe.get(endpoint, query_culinary_itineraries)
```

8.2 Results table and map

After retrieving the itineraries from the datasets, the web application shows the results in 3 different areas:

- **map:** An interactive map from the plotly-library allows the user to zoom in and out and read additional information about an itinerary by simply hovering over the mark of the itinerary. Green area in Figure 9
- **short summary:** This short summary retrieves only the total number of itineraries found by the web application for each category. Blue area in Figure 9
- **tables:** In this section, all the information regarding the itineraries is shown to the user in a table form. There is a table for each type of itinerary - company, culinary, cultural. Due to limited space in the table, the long text has been shortened. If available, the table also shows an image of the itinerary. Violet area in Figure 9

9 Conclusion

The present project was a great opportunity to see and test the concepts we learned from the data integration course during this first semester of the 2021/2022 academic year. An important aspect of this project was the possibility of working with real data instead of using data already clean and

ready for analysis (toy data), as done for other projects of this master. This opportunity allowed us to understand better what it means to be a data scientist, from understanding difficulties in working with real data (which could contain errors and inconsistency) to working with the data owners. An interesting future implementation would be connecting our ontology with the ontologies developed by the other groups who worked for the data integration project to connect the different domains of interest under the same system. This will allow us to retrieve additional data on itineraries in Campania that unfortunately are not present in our data, consequently enriching the user's experience.