# Integrating world data in a virtual knowledge graph

Candidates: 131, 112

Spring 2024

## 1 Introduction

In this paper, we will explore the possibilities of integrating world data into a Virtual Knowledge Graph (VKG). VKGs are rapidly becoming a compelling alternative to relational databases due to their ability to integrate heterogeneous data, perform reasoning based on axioms, and scale projects without the need to manifest data. We will take a closer look at the data and tools we have used, how we created our ontology and mappings, how we implemented our application, and lastly, we will discuss our findings and conclude the paper.

## 2 Motivation

Our aim was to construct a VKG from a variety of data in order to integrate it using ontology based access (OBDA) and ontology based integration (OBDI) tools. Interconnecting such data can make it possible to learn from it and thus pursue scientific endeavors. We were curious about correlations withing world data such as demographics and decided to make a VKG out of several datasets containing information about countries in the world.

## 3 Tools

To realize our project, we needed to use several tools in conjunction with each other. We downloaded several datasets and proceeded to put them into database tables in the database program *H2*. We decided to use H2 because of its simplicity and ease of use. We considered other tools such as *Denodo* and *PostgreSQL*, but settled on H2 for the reasons previously mentioned.

We wrote the ontology by hand in turtle (.ttl) format, carefully defining each class, subclass, and the relationships between them. We made multiple modifications to the ontology as we worked on the project, resulting in a detailed description of each class that aligned well with our data.

For the VKG, we utilized the ontology editor *Protégé* with the Ontop plugin to create mappings from the H2 database to our VKG. We created several mappings to fully utilize our ontology, ensuring the data was interconnected with numerous relationships between them.

Once we had our VKG, we needed to develop an application for querying the VKG. We decided to use Python with the Flask module to create a simple web page where users can send queries in SPARQL format to the VKG. To achieve this, we utilized Docker by using the Ontop image, allowing us to create an endpoint accessible across applications.

# 4 Domain

We started by gathering data from the domain of choice. Since we wanted to make a knowledge graph about statistical information about countries, we began by searching for a suitable dataset on *kaggle.com*, a popular website for sharing datasets and other machine learning tools. One of the requirements of the assignment was that we needed multiple classes. Initially, none of the datasets we found appeared to meet our criteria in terms of size. To overcome this limitation, we ended up combining multiple datasets from various sources. The datasets we ended up using included data about the following:

- Government type (democracy, dictatorship) [1]
- WHO Obesity data [2]
- Country data (Capital, region, continent) [3]
- Life expectancy [4]
- World happiness report [5]
- GDP [6]

Together, these datasets capture important information about countries, and can be used to create some interesting queries. These datasets shared a common domain and complemented each other as they were all in the same file format (CSV). Due to the consistency and coherence of the data, we decided they were good candidates for our VKG.

### 4.1 Data preprocessing

Some of the datasets needed to be processed in order to be used in our VKG. For example, we were unable to create mappings with the government type dataset since the key for each country was the country code rather than the name of the country itself. Therefore, we had to convert these country codes to the full names of the countries to be able to create appropriate mappings. We also had to modify the dataset slightly by removing spaces between words so it would fit into our ontology. This was done by several Python scripts and SQL queries.

After the data had been preprocessed, we proceeded to import each CSV file into their respective database tables in the H2 database.

## 5 Ontology and mappings

After having written our ontology, it turned out to be quite comprehensive for our datasets. We made an appropriate number of classes and sub-classes, and confirmed that the correct data types were set to ensure smooth querying. In total, we ended up with 20 classes, 15 of which were subclasses. Figure 1 illustrates the class hierarchy of our ontology. Figure 2 shows a visualization of the ontology.

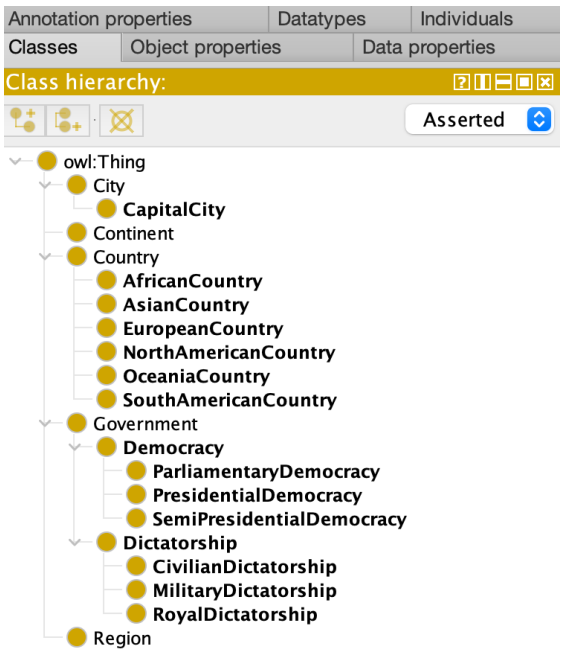



Figure 1: Ontology

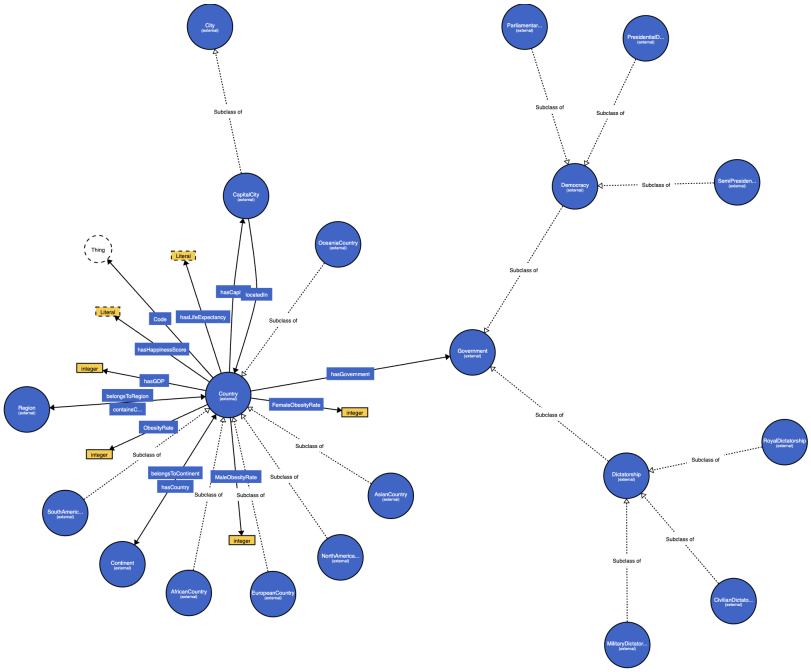



Figure 2: Visualization of ontology

We created in total 19 mappings from the H2 database to our VKG. Figure 3 shows an example of a mapping.

Mapping ID: country-obesity

Target (Triples Template):

```
ex:country/{COUNTRY} ex:ObesityRate {BOTHSEXES}^^xsd:integer ;
    ex:MaleObesityRate {MALE}^^xsd:integer ;
    ex:FemaleObesityRate {FEMALE}^^xsd:integer .
```

Source (SQL Query):

```
select country, bothsexes, male, female from obesity
```

SQL Query results:

| COUNTRY | BOTHSEXES | MALE | FEMALE |
|---|---|---|---|
| Afghanistan | 4.5 | 2.7 | 6.2 |
| Albania | 22.3 | 21.9 | 22.8 |
| Algeria | 26.6 | 19.4 | 34.0 |
| Andorra | 28.0 | 27.9 | 28.1 |

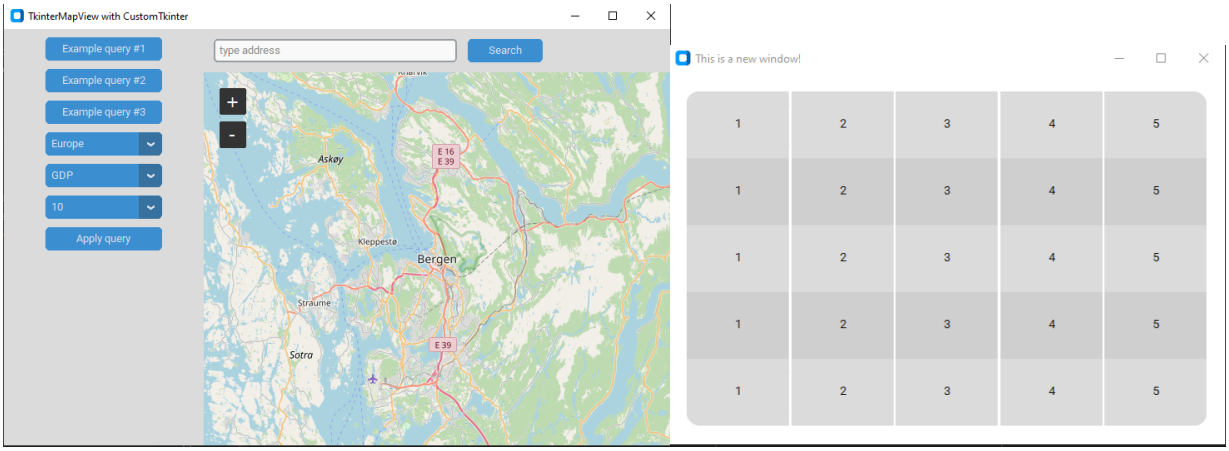
Figure 3: Mapping of obesity

# 6 App

After completing the mappings, we began working on how to visually represent our knowledge graph within the application. We wanted to have some buttons with predefined queries, but also make room for the user to experiment with queries of their own. Additionally, we sought to enhance the GUI with an extra feature for added interest. We discussed adding a map which would display the selected country. We tested two different GUI approaches: Tkinter and Flask.

## 6.1 Tkinter GUI

In our initial approach, we wanted to utilize the Tkinter package in Python, which is commonly used for creating graphical user interfaces. During our research, we came across CustomTkinter, an extension for Tkinter offering more customizable widgets. It also included a template with a built in map which we could expand upon. We created some buttons with example queries and other optional parameters for more specific queries. The parameters enabled users to extract a specified number of countries from specific continents with desired data. For instance, top 20 countries in Europe with the highest GDP. However, we encountered a challenge: integrating the data into Python proved to be more difficult than anticipated. We wanted to use the SPARQL endpoint, but at the time we had not figured it out yet. We had to materialize the data in a turtle file to continue working on the GUI. Clicking any of the buttons would trigger a new window pop-up, with the queried results. The main window and an example of what happens when a button is clicked is shown in figure 4. At this point, we decided to try out Flask as an alternative. Therefore, the data was never integrated into the pop-up window.




(a) Main window (b) Pop-up window without data

Figure 4: CustomTkinter GUI

## 6.2 Flask app

After experimenting with the Python program, we decided to create a simple web page instead. We decided upon using the Flask module for Python because of its simplicity and allowing us to have all the code for the web page in just one Python script. We considered using Django, but because of its complexity and our project being relatively low in scope, we decided it was redundant to use such a large library for our simple application.

The Flask app serves the user with an HTML page with several pre-made SPARQL queries and a SRARQL input field in which the user can write custom queries as shown in figure 5.

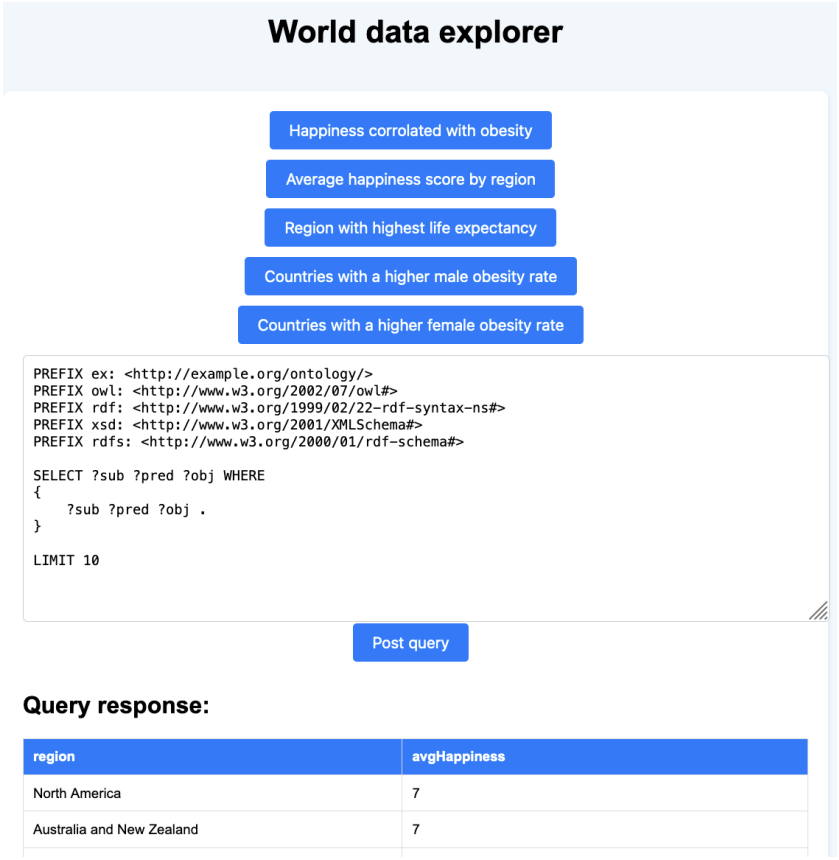



Figure 5: Flask app

The Flask app listens for POST requests at the endpoint "/query" and and then sends this POST request to the SPARQL endpoint, processes the response and sends it back to the end user. We had some trouble displaying the output in a structured manner and decided therefore to create a simple script that converts the SPARQL endpoint response into an HTML table, which is then sent to the client and displayed on the web page.

In the "/gui" endpoint there is a JSON request that gathers the value from a cell if it is clicked on. This value is then processed by our GUI function which opens a new tab in *openstreetmap.org*. The value is inserted into the URL which specifies what country is shown on the map. We also experimented with google maps, but found it easier to insert values into OpenStreetMap. Figure 6 shows the pop-up that appears when a user clicks on a country.

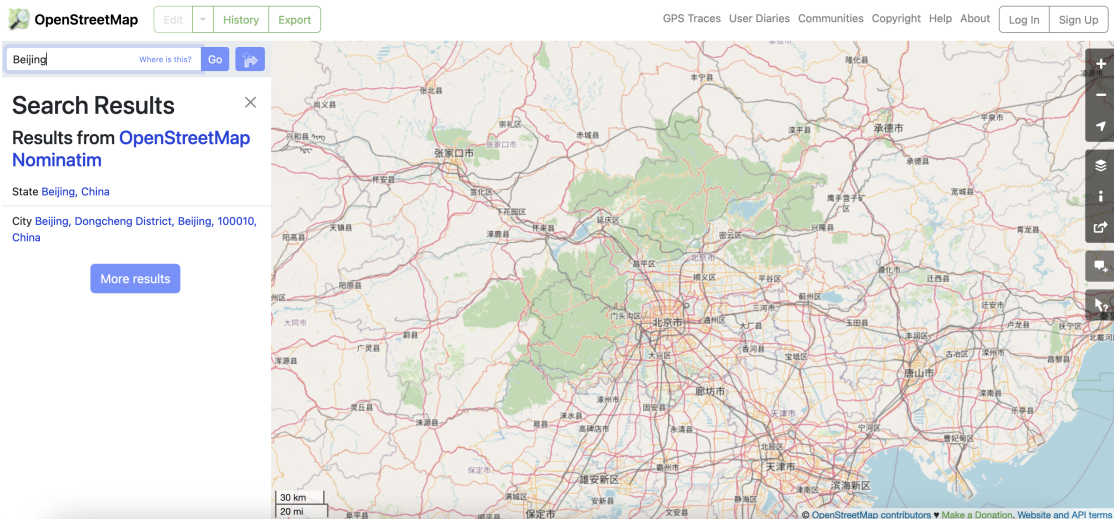



Figure 6: Open street map

# 7 Discussion

After constructing our ontology and integrating data into it, running our app, and querying our VKG, we were able to learn from all the disconnected datasets that now are interconnected. We decided to make a study of the relationships between the data. Since two of our datasets were about happiness and obesity, we decided to further explore this to identify any correlation. To find a correlation between these two variables, we needed to write a SPARQL query to retrieve the relevant information as seen below.

```
PREFIX ex: <http://example.org/ontology/>

SELECT ?country ?happinessScore ?obesityRate
WHERE {
    ?country ?p ex:Country .
    ?country ?p ?countryName .

    FILTER (bound(?happinessScore))
    FILTER (bound(?obesityRate))
}

ORDER BY DESC(?happinessScore)
```

After retrieving the data from the query, we exported it and used a Python script to visualize the data in a graph. Figure 7 shows the relationship between obesity and happiness . After analyzing the graphical representation of the SPARQL

query, we concluded that there is a slight linear relationship between happiness and obesity. However, it is important to note that correlation does not equal causation. There might be one or several other factors that affect the outcome of this correlation.

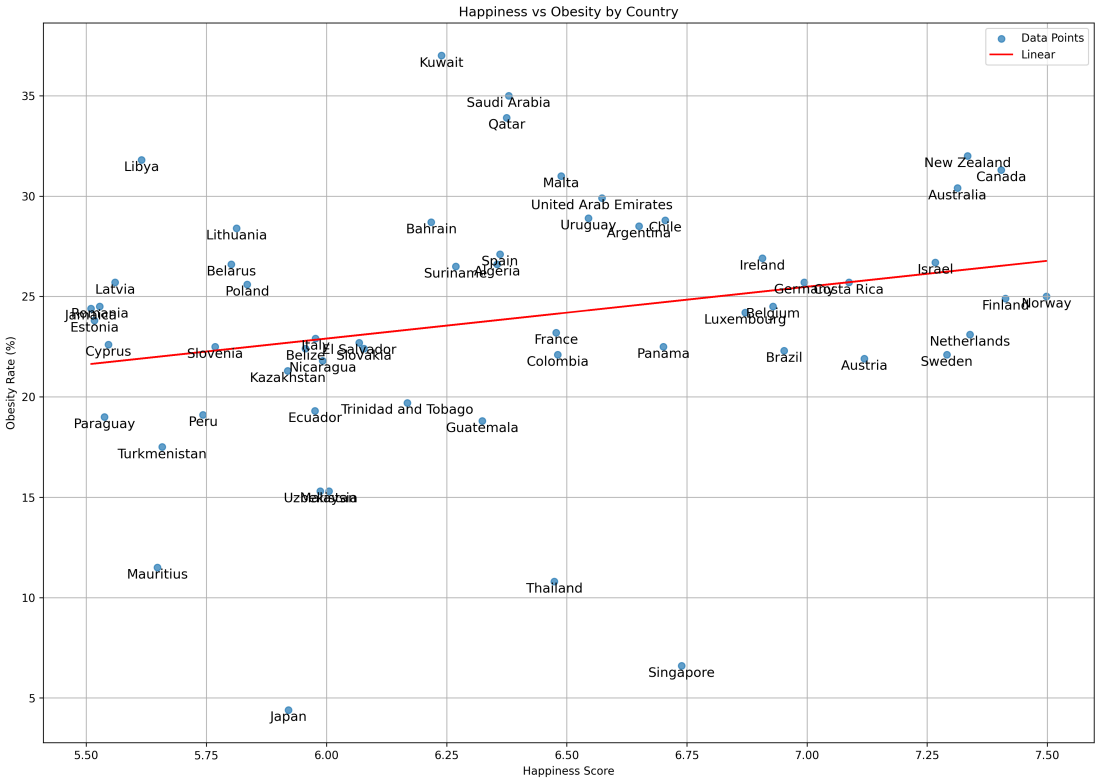



Figure 7: Correlation between happiness and obesity

# 8 Conclusion

The goal of this project was to construct a virtual knowledge graph and integrate a wide variety of data into it to perform analysis tasks. By utilizing tools such as H2, Ontop, and Docker to develop an application, we were able to complete our task and create a simple application for exploring all the datasets simultaneously. We believe we have demonstrated a use case for ontological data by performing analysis tasks and drawing conclusions about relationships between data that were previously inaccessible.

# References


[1] Aché, M. Democracy-dictatorship index, 2020. Accessed: 2024-05-20.

[2] Autio, A. Who obesity by country 2016, 2019. Accessed: 2024-05-20.

[3] Banerjee, S. World population dataset, 2022. Updated 2 years ago.

[4] Kumar, R. Life expectancy (who), 2018. Accessed: 2024-05-20.

[5] Sustainable Development Solutions Network and 1 collaborator. World happiness report, 2020. Accessed: 2024-05-20.

[6] Tas, O. C. World gdp (gdp, gdp per capita, and annual growths), 2022. Accessed: 2024-05-20.