Final Project Report - Art Museums in Amsterdam

Introduction

This project report contains a description of the individual steps we took to make a data analysis application for the course 'Knowledge and Data'. Our application includes data about art museums in Amsterdam that can be used as a source of information, as the data has been visualised in various ways. We also managed to include interactive features in our application, allowing it to be used as a recommendation system as well. The report will consist of descriptions of our domain together with the design of our application. Additionally, the OWL ontology that we created for this project will be explained and we will show several SPARQL queries and show their use for our project. Finally, for both the ontology and the queries, we have shown meaningful inferences.

Goal

For our project, we are aiming to answer the question 'Which art museums are related to different locations in Amsterdam?' We chose this domain because museums are an intrinsic part of the culture of Amsterdam and we want to analyze data that could help tourists as well as art lovers. Currently, you can find 89 different museums spread across the city. Because of this great variety of all types of different museums, it can be overwhelming to know which one to visit.

In our project, we want to solely focus on art museums. This way, we could also analyze or filter our data on different artists that people might be searching for. This would be a lot harder to do if we included museums that cover topics such as history and science. After eliminating the other types of museums from the list, we are left with 24 art museums spread across Amsterdam. The data that we want to analyze can be used as a recommendation system for people who do not know where to start when it comes to admiring art in Amsterdam. On the other hand, it could also be a handy tool for people who are already familiar with art. The exact stakeholders will be discussed in the subsequent section.

Stakeholders

Visiting a museum can be an enjoyable experience. That is, if you know which museum is best suited to your taste and artistic preference. Of course, one would want to plan their visit ahead to ensure the best experience. Therefore, potential users can use our application to have an enjoyable experience when picking out a museum where they can view artworks that

suit their interests. Our application helps users who are interested to visit a museum in Amsterdam and offers them various options based on the museum's location, artist of interest, etc. These options are optimizable to fit the user's taste based on their choice. The stakeholders can be divided to three main categories of:

Art lovers who are searching for a specific artist: There are 24 different art museums in Amsterdam and it is not an easy task to search for a specific one that contains artwork of your favorite artist. Therefore, by taking the user's request of a specific artist in account and optimizing the dataset to fit their demand, our application is a useful tool to benefit the users and aid them in finding the museums that display the art that they would like to see.

Researchers: Researchers who are interested in information about museums in Amsterdam and are gathering information based on artist, location, etc. can also use our application to gather data and use it in their analysis.

Tourists who want to explore the museums of Amsterdam for fun: There are many tourists that desire a delightful experience of museums in Amsterdam and visiting a museum is an entertaining and interesting choice for those who enjoy artistic activities. However, searching the web or asking around about a location you are not very familiar with can be tiring, and it may affect the enthusiasm of the tourist. Therefore, tourists can use our application to view museums in Amsterdam based on their location. In addition, a small description of the museum and their popularity will be provided to help the tourists choose the best museum and enhance their experience.

The municipality of Amsterdam: Lastly, the municipality of Amsterdam could also be a possible stakeholder for our application. In our recommendation system, numerous information about the different museums in Amsterdam will be gathered, analysed and displayed. This information could also be appealing for the municipality of Amsterdam as our analyses might be an interesting source for them to work with.

Design

Description

In our application, we want to be able to show information about art museums in Amsterdam based on the preferences of the users.

The user can check whether an artist that they like has art work in a museum. Via a drop-down menu, the user will be able to choose an artist. The notebook will then output the museums that exhibit art from that specific artist (see Figure A.2 of the Appendix).

If the user wants to visit a museum that exhibits objects from a specific art period, the notebook allows the user to choose an art style (either renaissance, 20th century, modern

art, early modern or all) and based on that it will output museums that exhibit objects from that specific style (see Figure A.3 of the Appendix).

Furthermore, the notebook shows the popularity rate for each museum. Based on this, the user can take this into account when choosing which museum to visit (see Figure A.4 of the Appendix). The popularity rates of the museums are colored. Popular museums will have a red color while less populated museums will be marked green.

Our notebook furthermore shows a map of Amsterdam. Firstly, the user is able to choose a distance range. Subsequently, the notebook outputs a map of Amsterdam, showing the user location and the museums that are in the distance range (see Figure A.5 of the Appendix). The user location is marked blue, the museums that are in the distance range are marked green and the museums that are not in the distance range are marked red (see Figure A.6 of the Appendix).

Additionally, the notebook shows a scatter chart of museums that either accept or refuse a museum card. The red plus sign indicates that a museum does not accept a museum card whereas the green plus sign indicates that a museum accepts a museum card (see Figure A.7 of the Appendix).

Lastly, the application shows a price table to the user. The prices can either be labeled as cheap, affordable or expensive. Cheap prices are marked green, affordable prices are marked yellow and expensive prices are marked red. The user can take this table into account when deciding which museum to visit (see Figure A.8 of the Appendix).

Motivation

We wanted to use a map, graphs and tables in our application to formalize our data. The map would be ideal to visualize Amsterdam in a clear way and to have the user use the map to point to their location. We believe it is the most intuitive way for the user to navigate through Amsterdam. Additionally, the graphs would allow the user to inspect the information in a human readable way that can be interpreted with a single glance. Lastly, the tables will contain all the further data about the museums that we want to show to the user to optimize our recommendations and make them choose the option that fits them the most.

Domain and Scope

Domain

The ontology for this project will cover information about Museums in Amsterdam that are specifically recognized under the category of “Art & Design”. Therefore, the domain of the ontology can be recognized as Art Museums that are located in Amsterdam and it contains the following information regarding the domain:

1. The geographical location of the museums (which is then used for display on Folium map in the application)
2. The number of visitors in a year (which is then used for calculator the popularity)
3. The most important artists of the museum
4. The art style that the museum specializes in
5. The director of the museum
6. The average price of the entrance fee
7. Whether a museum accepts a museum card

Scope

The scope of the application is to provide the user with desired information on these museums based on their selected feature. This way, after the application has given all this information to the user, they can use it as a recommendation system and compare all possible museums to pick the one they want to visit. For instance, a user may be interested to know which museum displays works from a certain artist, or if a certain museum exhibits a certain style of art. The data was gathered in the hope of helping users in their decision for visiting a museum, or answer their research questions if needed.

Methodology

In the first stages of constructing this ontology, we followed a top-down development process of first defining the general concepts of our domain; namely classes and important properties necessarily for our purpose of application. Subsequently, we incorporated specialization of the concepts in order to make our application desirable and different from similar existing datasets. There are many ontologies such as DBpedia and Wikidata that contain useful information regarding the domain and scope of our application. Hence, we decided to reuse this information by refining and extending their data for our application. We also used the ontology of one of our group members to insert additional data into our own ontology. We will extract information about the museum directors of all the different museums from their ontology.

We then prepared a list of the terms, classes, properties and similar concepts that we would like to offer to the users. On the basis of previous steps, we then defined the classes and class hierarchy which form the basis of our ontology. We then proceed to define relative properties of the classes, and subsequently the instances in compliance with the purpose of our application that we intended. The reason for deciding on a top-down approach is that

once the general idea of our concepts are formed, it is easy to expand and specialize the concept extensively since we settled the purpose of the application in the beginning.

Data reuse

Two reusable ontologies

In this application, the following ontologies were used:

1. The first ontology that we will use is an ontology called "Amsterdam Museum Linked Open Data in the Europeana Datamodel". Because this ontology is incredibly extensive, we only collected data from smaller parts of the dataset by running Queries to get the names for the three categories of "painter, photographer and sculptor" from the file "people.ttl" in the source "bitbucket.org". [1]
2. The second ontology that we will use is one of our group member's ontology. This ontology uses OWL, RDFS and RDF vocabulary. Our group member made an ontology about art museums in Amsterdam. The main difference between our ontology and theirs is that they included instances of museum directors of the art museums of Amsterdam. We will map our own ontology with theirs, to retrieve this information about the museum director.

Two reusable resources

In this application, the following sources were used to collect relevant data and use them to optimize our ontology:

1. Since the main domain for this application are Art Museums, we have extracted a CSV file from the website of "whichmuseum.nl" [2] to create all the instances in the class Museum. We have made the CSV file based on the table that is shown in the website. The reason we have decided to use this source is mainly due to the fact that it categorizes museums based on the scope of what the museum exhibits, and our application's domain is "Art Museums". Therefore, with the help of this source, we can easily extract the relevant museums of all the museums that exist in Amsterdam.
2. Additionally, we will use the DBpedia SPARQL endpoint [3] in our application to extract some useful features like popularity (number of visits), location (longitude and latitude) and use them for our recommendation system in order to ensure the satisfaction of the users. The reason for choosing DBpedia [3] is that this website has a broad scope of entities covering different areas and it is a rich source of structured cross-domain knowledge. This also includes the domain of "Art Museums" in Amsterdam which is the domain of our application. In addition, the DBpedia dataset is interlinked on the RDF

level with various other Open Data datasets on the Web which ensures the accuracy of our data to some extent.

Conceptualization

Our ontology about art museums located in Amsterdam consists of 19 classes (5 main classes and 14 subclasses) and 12 properties (6 ObjectProperties and 6 DataTypeProperties). All the classes and properties are shown and explained further in this section.

Classes

- Place
    - Museum
- Person
    - Artist
        - Painter
        - Photographer
        - Sculptor
    - MuseumDirector
- ArtForm
    - Painting
    - Photography
    - Film
    - Sculpting
- AveragePrice
- ArtStyle
    - 20thCentury
    - EarlyModern
    - ModernArt
    - Renaissance

With these classes, we are able to store all the different instances that we desire. Additionally, we defined object- and data properties to be able to give the individuals object and data property assertions later on. The properties are as follows:

Object properties

- exhibitsArtFrom
- hasArtStyle

- hasAveragePrice
- isDirectorOf
- hasDirector

The object properties connect instances from different classes together and create relations that are desired for our application. In addition, we have included an inverse property (isDirectorOf is inverse of hasDirector) to infer new instances when we run the reasoner. Some examples of relations of the object properties are shown in Table 1.

Museum exhibitsArtFrom *Artist*	*Artist* hasArtStyle *ArtStyle*
Museum hasAveragePrice *AveragePrice*	*MuseumDirector* isDirectorOf *Museum*
Museum hasArtStyle *ArtStyle*	*Museum* hasDirector *MuseumDirector*

Table 1: Examples of the object property relations that our ontology will include

Data properties

- acceptsMuseumCard
- hasFee
- hasLong
- hasLat
- hasName
- hasPopularity

The data properties connect instances to a certain value. In our case, they will have either a string, literal, integer or boolean data type. On the next page all the individual data properties will be described together with an example value of that property.

The data property *acceptsMuseumCard* is used to recognize if a museum accepts a museum card for payment (or discount) of their entrance fee. For instance, the entrance fee for Van Gogh Museum of Amsterdam is free if you use a museum card which means that the acceptsMuseumCard property for the Van Gogh Museum in our ontology is set as the boolean value “True”.

The data property *hasPopularity* is used to determine the popularity of a museum by storing the yearly number of visits of the museum. For instance, the Van Gogh Museum has a yearly number of visitors of around 2300000 people (this data is gathered from

"DBpedia.org") which means that the *hasPopularity* property for the Van Gogh Museum in our ontology is set as set as the integer value "2300000".

The data properties *hasLong* and *hasLat* are used to determine the geographical location of a museum by storing the latitude and longitude of the museum. For instance, the Van Gogh Museum has the latitude of 52.3583 and longitude of 4.88111 on the map (this data is gathered from "DBpedia.org") which means that the *hasLong* property for the Van Gogh Museum in our ontology is set as the float "4..8811" and the *hasLat* property for the Van Gogh Museum in our ontology is set as the float "52.3583".

The data property *hasName* is used to determine the name of an artist. For instance, the artist "?" has the name "?" (this data is gathered from "DBpedia.org") which means that the *hasName* property for the artist "?" in our ontology is set as the literal "?".

The data property *hasFee* is used to determine the range of the instances of the class AveragePrice. For instance, the instance Cheap is a price that is lower than 10€ (this is purely a modeling choice) which means that the *hasFee* property for the instance Cheap in our ontology is set as the literal "less than 10€".

Once again, we wanted to show the properties in a table that portrays what the relations will look like. This information can be observed in Table 2.

Museum acceptsMuseumCard *xsd:boolean*	*Museum* hasLat *xsd:float*
Museum hasPopularity *xsd:integer*	*Artist* hasName *rdfs:literal*
Museum hasLong *xsd:float*	*AveragePrice* hasFee *rdfs:literal*

Table 2: Examples of the data property relations that our ontology will include

RDF Schema

Finally, to be able to fully grasp the relations between these classes and properties, we have depicted them in an RDF schema that shows the relations via arrows. This RDF schema can be found in Figure A.1 of the Appendix.

In this RDF schema, the classes are shown with a red border and the properties are shown with a blue border. This is purely for readability purposes. The schema shows the classes and their subclasses as well as five properties. For clarity, we have chosen to omit the relations from 'subclass rdf:type rdfs:Class' as this would make the schema chaotic and unreadable. With this RDF schema, we hope to be able to show the relations between the classes of our ontology at a single glance.

The five properties are all data properties, except for gp:hasPopularity and gp:acceptsMuseumCard. The properties are all of rdf:type rdf:Property and have a rdfs:domain and/ or rdfs:range relation to (sub)classes. These properties will give further information about the individuals in the ontology and could hopefully cause meaningful inferences.

OWL Ontology

For our OWL ontology, we used 19 classes, as listed in the section 'Conceptualization'. We will now shortly describe every individual class. There is a class *Place*, which has a subclass *Museum*. Furthermore, the class *Painter*, *Photographer* and *Sculptor* are subclasses of the class *Artist*. We decided to use these classes because our application will show prominent artists of a museum to the user. The class *Artist* is a subclass of *Person*. Additionally, the class *MuseumDirector* was created since the application shows the museum director of an art museum. *MuseumDirector* is also a subclass of *Person*. The ontology also includes a class *ArtForm* to show the user what type of art a museum exhibits. The classes *Painting*, *Photography*, *Film* and *Sculpture* are subclasses of *ArtForm*. Furthermore, a class called *AveragePrice* was created. This class will show the user whether the entrance fee of a museum is expensive, average or cheap. The class *ArtStyle* is meant to give extra information about the art style of objects. This class has the subclasses *20thcentury*, *EarlyModern*, *ModernArt* and *Renaissance*.

After the finalization of classes and properties, we defined a few class restrictions using OWL semantics that will help us in the inferring process of our data in the ontology. For instance, as mentioned in previous sections, each instance from the class Artist has a relation of hasArtStyle with one or more instances from the class ArtStyle. This will be added manually for each individual person. Similarly, each instance from the class museum should have a relation of hasArtStyle with one or more instances from the class ArtStyle. However, it would be tedious to add these relations for all of these instances. This is where the class restrictions can help us save time.

By defining necessary and sufficient rules for the class and using OWL semantics, we can run the reasoner and have the relative data added automatically. The OWL characteristics are further explained in Table 3.

Class Name	**Restriction (necessary and sufficient)**	**Explanation**
Artist	owl:equivalentClass [(gp:hasArtStyle some gp:ArtStyle) and (gp:hasName some rdfs:Literal)]	All the "Artists" must have some "ArtStyle" value for the property hasArtStyle and must have some literal (the string of their name for instance) for the property hasName.
Museum	owl:equivalentClass [gp:hasDirector some gp:Director]	All the "Museums" must have some "MuseumDirector" for the property hasDirector.
AveragePrice	owl:equivalentClass [gp:hasFee some xsd:integer]	All the "AveragePrices" must have some "integer value" for the data property hasFee.

Table 3: This table displays the restrictions that were made on the classes using OWL semantics.

We have also reused data from two existing ontologies. One ontology was made by a classmate in a previous assignment which shared the same domain with our ontology, and the other one we retrieved from "bitbucket.org" [1]. These external datasets will help us complete the instances for the class MuseumDirector and create useful relations which will be useful for recommendations in the application.

It is important to mention that the "data.ttl" file from the source "bitbucket.org" [1] was too large to import in the main ontology. It was also too large to import in the GraphDB repository. For this reason, we collected data from smaller parts of the dataset by running Queries to get the names for the three categories of "painter, photographer and sculptor" from the file "people.ttl" in the source "bitbucket.org" [1] and then connect them to museums by looking through the "data.ttl". We then completed the missing data for museums by gathering data from "wikidata.org" [4] and created a CSV file containing the artist data. Finally, we used the "Tabular(OntoRefine)" option from GraphDB to add this data to our ontology.

The mappings between ontologies are explained in Table 4.

External ontology	Relation	Main ontology	Explanation
Class "MuseumDirector"	owl:equivalentClass	Class "gp:MuseumDirector"	These 2 classes have essentially the same definition in both ontologies.
Class "ArtMuseum"	owl:equivalentClass	Class "gp:Museum"	These 2 classes have essentially the same definition in both ontologies.
Every instance of the class "ArtMuseum"	owl:sameAs	Every instance of the class "gp:Museum"	These instances of "ArtMuseum" from the external ontology are also all included in our main ontology in class Museum.
Property "isDirectedBy"	owl:equivalentProperty	Property "gp:hasDirector"	These 2 properties have essentially the same definition in both ontologies.
Property "directedOf"	owl:equivalentProperty	Property "gp:isDirectorOf"	These 2 properties have essentially the same definition in both ontologies.
Class "ArtMuseum"	rdfs:subclassOf	Class "gp:Museum"	The class "ArtMuseum" from the external ontology is a subclass of "gp:Museum" since art museums are a subclass of museums in general.

Table 4: This table displays the mapping of our ontology with the external ontologies using OWL semantics.

Integration

One external dataset is the CSV table that contains all the names of museums of Amsterdam, addresses, what category they belong to, and their location (all of them have the location Amsterdam). The data we intended to extract from this CSV file were the names of all the museums that have the category "Art and Design". For this purpose, we performed the following operations to insert these names as instances in the class "Museum" in our own ontology. Below, the individual steps of the process of extracting this intended data and subsequently inserting it to our ontology are shown.

1. We created a CSV file based on the HTML table provided by "whichmuseum.nl" [2]
2. We import the CSV file in GraphDB repository using Tabular(OntoRefine)
3. We filtered the rows that had "Art & Design" as their category
4. We added them to our class "Museum" by editing the "RDF mapping" and converting it to a SPARQL query
5. We then used "INSERT"on the query to add them to our base graph in GraphDB

The other external datasets that are used are the DBpedia [3] and Wikidata [4] sparql endpoints. We intend to show the location of each museum and also recommend them based on popularity if the user asks. To do this, we extracted the longitude, latitude and the number of visits of each museum in Amsterdam from the DBpedia and Wikidata dataset and used an insert query and some owl characteristics. For instance:

1. The geo:lat property from the DBpedia is set to be owl:sameAs gp:hasLat which is an equivalent property in our ontology
2. The geo:long property from the DBpedia which is set to be owl:sameAs gp:hasLong
3. The dbp:visitors property from the DBpedia which is set to be owl:sameAs gp:hasPopularity in our ontology

Additionally, we performed the following operation to insert these values as to each of the instances in the class “Museum” in our ontology.. Lastly, it is important to mention that when looking for this information on Wikidata and DBpedia, we encountered some problems:

1. Wikidata and DBpedia were not able to provide us information about 24 museums, but solely for 11 museums. This issue could be due to the fact that the labels may have been different from our ontology, or that these two databases simply didn’t categorize the museums in “Art & Design”. Therefore, we only retrieved the data for 6 of these museums from Wikidata, and the 5 museums from DBpedia
2. Wikidata also did not offer information about the number of visits for the museums. Hence, we tried to retrieve visits from DBpedia, but it only returned this information for 5 museums

To resolve the issue of the missing data for our ontology, we decided to add the missing data manually.

Ontology Inferences

When we run the reasoner in our ontology, many notable inferences are made. These inferencesses are a result of class restrictions and mapping between the ontologies and they will aid us when we implement the recommendation system in the application. The most valuable inference that was a result of the mapping were the museum directors. Since the external ontology already consisted of museum names and their directors, by applying the mapping using OWL semantics, as was previously mentioned in Table 3, we were able to easily infer these directors to our initially empty class of “gp:MuseumDirector”.

Furthermore, by implementing mappings of properties, we were also able to connect each director to its desired museum simply by running the reasoner and having the data inferred automatically. Without the reasoner on, the class MuseumDirector produces the result that can be observed in Figure 1.

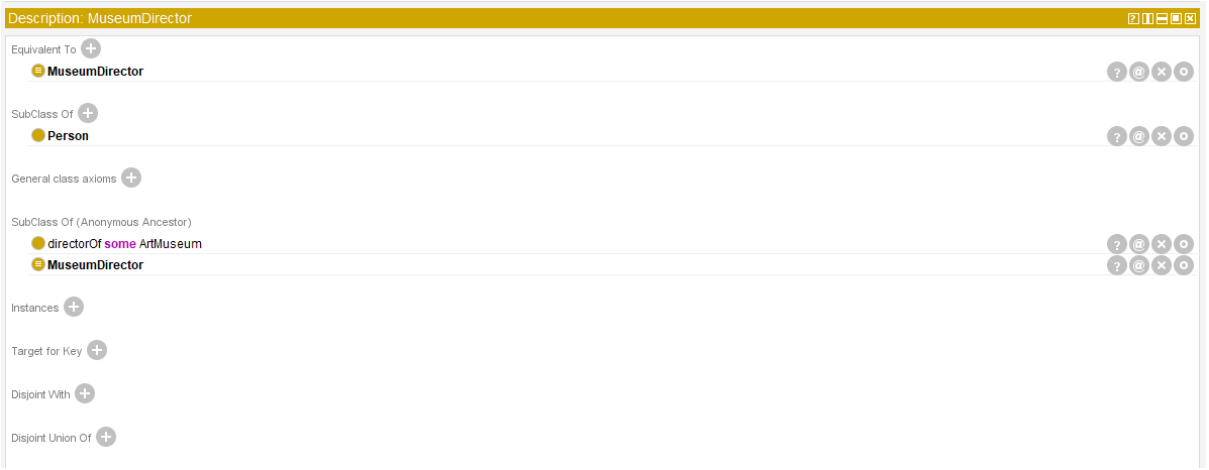

Figure 1: This figure displays the class "MuseumDirector" when the reasoner is off.

With the reasoner on, it will infer all instances of the class MuseumDirector. This can be seen in the figure below, Figure 2.

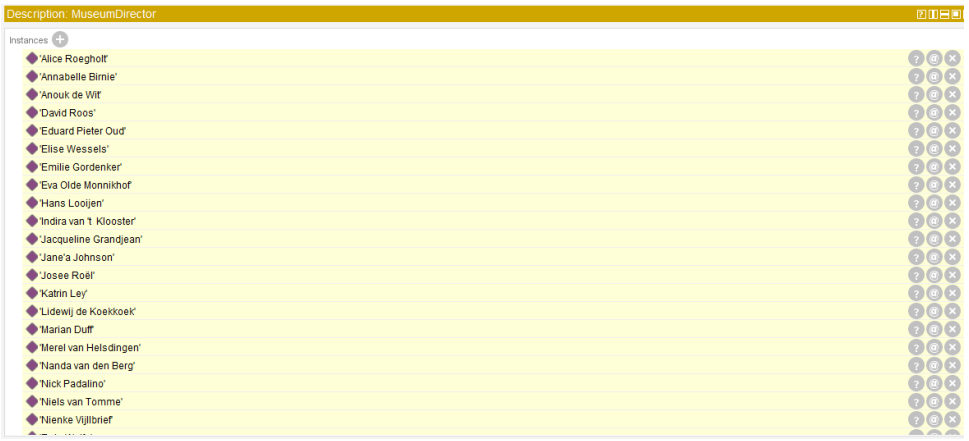

Figure 2: This figure displays the class "MuseumDirector" and its inferences when the reasoner is on.

The other significant inference was the connection between the "gp:ArtForm" of each museum to the instances of "gp:Museum". By applying the class restrictions that were previously mentioned in Table 2, we were able to categorize each museum in its rightful "ArtForm" classes that were initially empty and have the data inferred automatically.

Without the reasoner on, instances of the class “Museum” produce the results that can be observed in Figure 3.

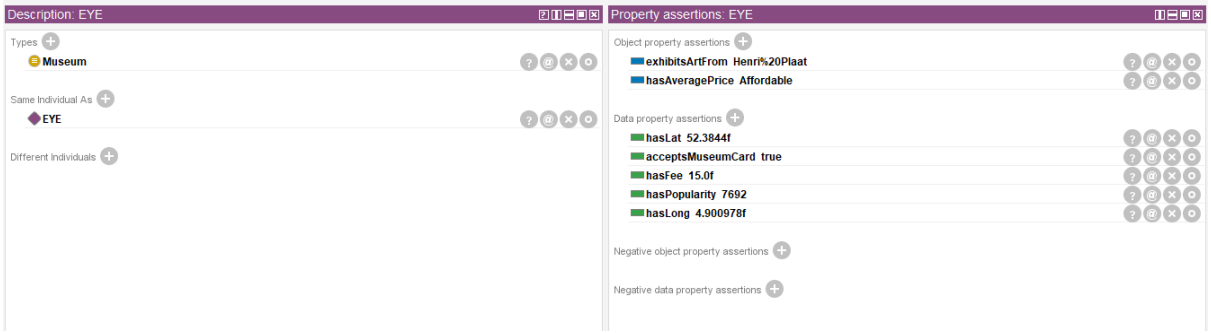

Figure 3: This figure displays an instance of the class “Museum” when the reasoner is off.

With the reasoner on, the instance of the class “Museum” will infer the artForm of the museum, as one can observe in Figure 4.

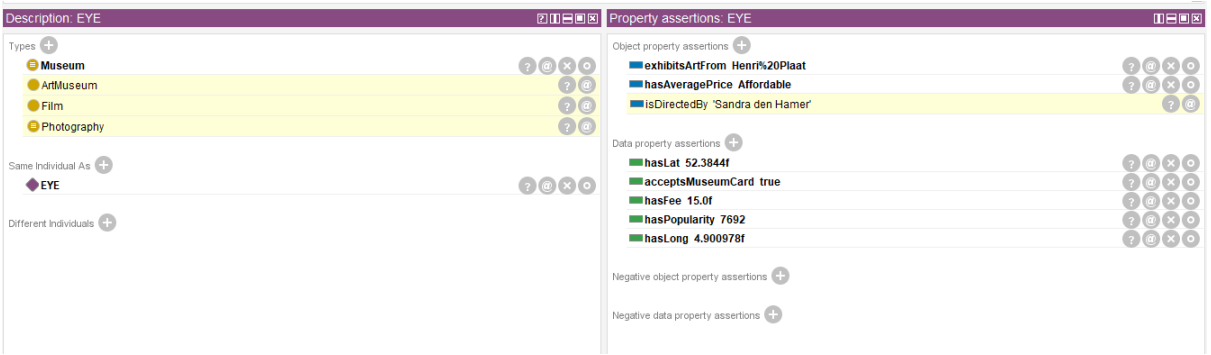

Figure 4: This figure displays an instance of the class “Museum” and its inferences when the reasoner is on.

There are few other inferences that have been made in regards to properties and instances. However, we will not elaborate them in detail since they do not provide essential and effective for the application.

SPARQL Queries

We used multiple SPARQL queries in the jupyter notebook file. The first SPARQL query (see Figure A.9 of the Appendix) is used to retrieve museums and their artists. This is later used for the drop-down menu where the user can select an artist, which will result in a table of museums that exhibit art from that specific artist. The results are ordered by the popularity rate of the museums.

The second SPARQL Query is used to return museums and their art style (see Figure A.10 of the Appendix). This is also used for the drop-down menu, where the user can select

an art style. Depending on that, the application outputs the museums that exhibit art from that specific art style. The output is ordered by the popularity rate of the museums.

The third SPARQL query (see Figure A.11 of the Appendix) retrieves a list of museums and their popularity rate. The popularity rate is determined by the number of visitors per day of a museum. To give the popularity a range of 5, the line "BIND(((?popularity*5)/(?popularity+5)) AS ?PopularityRate)" was written. The museums are ordered by their popularity rate.

The fourth SPARQL query (see Figure A.12 of the Appendix) retrieves a list of all museums with their longitude and latitude, and is ordered by their popularity rate.

SPARQL Inferences

In order to write the SPARQL queries that produce inferences we first defined what data we wanted the SPARQL queries to return when running the query. The first query that we thought of is supposed to show an artist with a specific artstyle. Subsequently, it should also return in which museums this work can be found and the museum should have a popularity rate that is greater than 4, indicating that it is a popular museum. We defined the popularity rate as a score between 0 and 5, the formula to transform it into this rating can also be found in the query.

For this query, we chose photography to be the artstyle that we are looking for. To get the info we desired, we first retrieved all museums that are of type photography, indicating that they exhibit art from artists that have photography as their artstyle. Subsequently, the artists will be retrieved, together with their name. The museums will be filtered on their popularity and this all together shows the final output. The SPARQL query with the reasoner turned on and off can be found in Figures 13 and 14 of the Appendix.

The second query that we thought of is supposed to show the directors of museums that have a cheap entrance fee. To do this, we retrieved the information about the museum directors and filtered the query on which museums have an average price that has the property 'cheap'. The final SPARQL query with the reasoner turned on and off can be found in Figures 15 and 16 of the Appendix.

Honorary Title - Visualization Guru

For the final part of our project, we decided to apply for the honorary title of the visualization guru. In our application, we used numerous visualization methods to make the data as clear and visually appealing as possible. These visualizations include:

1) *Drop down menus* for the interactive section of our application. In this part of the notebook the user can choose between different artists and art periods and the application will return a table including the information about in which museums these artists or art styles can be seen.
2) A *horizontal bar chart* representing the overall popularity of all the museums. This bar chart shows the popularity of the museums on a scale of 1 to 5. With a high popularity rate, the bar will be green and a decreasing popularity will cause the bars to become a shade of red. This will allow the user to be able to decide which museum they want to visit based on popularity, as opposed to their location or certain artists/ art styles.
3) We also included a *slider icon* where the user can determine their desired distance from the museums they would want to visit. The slider consists of a circle which the user can drag to the left and right where it would decrease and increase respectively.
4) With the information gathered about the desired distance, we visualized a *map* that shows the user's location together with the locations of all the museums. The recommended museums will show up with a green circle around them. All the museums out of reach are shown with a red circle around them. Additionally, the user can hover over the circles and a text bubble will pop up that shows the name of the museum. The user can zoom in and out on the map and navigate through it however they like.
5) For more additional information, we added a *figure* that shows all museums and if they accept a museum card or not. Museum cards can be used for discounts on the entrance fee or even free entrance. Intuitively, we visualized this using red and blue plus signs. The red signs indicate that they do not accept a museum card and the blue ones indicate that the museum does accept a museum card. This can also be derived by the x-axis that shows 'no' and 'yes' at the bottom of the figure.
6) Lastly, we included a *table* that showcases all the prices of the museums, together with an average range of either 'cheap', 'affordable' or 'expensive'. The exact range is also shown in the rightmost column where it says what exact price one can expect when visiting the museum. To make it even more visually clear, all cheap ranges are surrounded by green borders, all average ranges are surrounded by a yellow border and the expensive ranges are surrounded by red borders.

With all of this, we hope to have supported our choices and explained our visualizations enough for the honorary title of 'visualization guru'. We added screenshots of these visualizations that can be found in the Appendix figure A.2 until A.8. The visualization of the entire notebook can also be found in the mp4 file called 'Final notebook', where a screencast has been taken of the full application and its functionalities.

References

[1] de Boer, V., 2017. *Amsterdam Museum Linked Open Data in the Europeana Datamodel.* [online] Available at: https://bitbucket.org/biktorrr/amlod/src/master/.

[2] Whichmuseum.nl. 2021. *Musea in amsterdam*. [online] Available at: https://whichmuseum.nl/nederland/amsterdam/musea [Accessed 28 October 2021].

[3] DBpedia Association. n.d. *Home - DBpedia Association*. [online] Available at: https://www.dbpedia.org/ [Accessed 28 October 2021].

[4] Wikidata.org. n.d. *Wikidata*. [online] Available at: https://www.wikidata.org/wiki/Wikidata:Main_Page [Accessed 28 October 2021].

Appendix

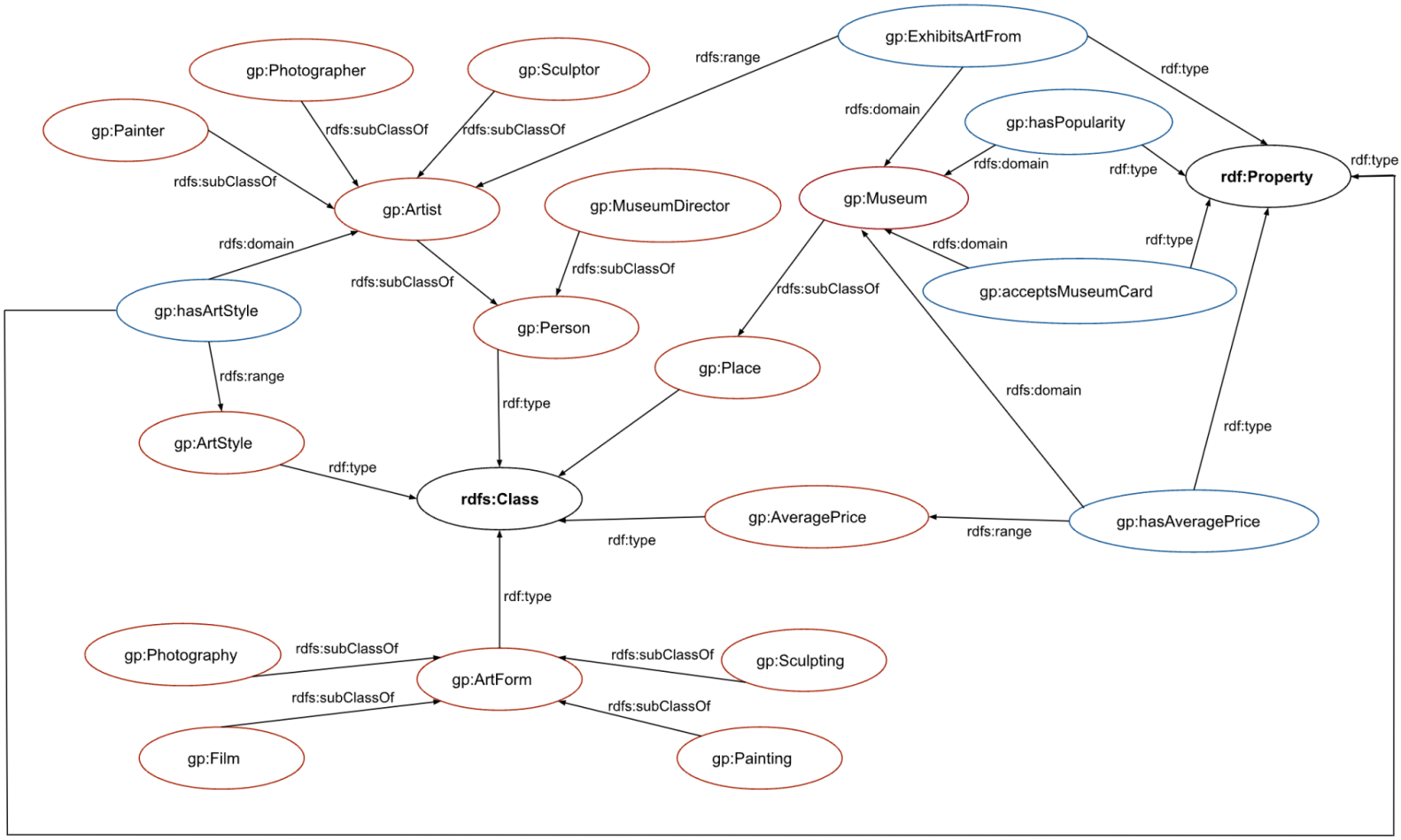

Figure A.1: This figure displays the RDF schema that portrays relations between every class of our ontology, as well as five properties

Please choose an artist:

style Alexej von Jawlensky

Museum	name
Stedelijk Museum Amsterdam	Alexej von Jawlensky

Figure A.2: This figure displays the museum that exhibits art from a specific artist

Please choose an art period:

style Renaissance

Museum	styleName
Rijksmuseum	Renaissance
Hermitage Amsterdam	Renaissance
Rembrandt House Museum	Renaissance
Old Church	Renaissance

Figure A.3: This figure shows museums that exhibit a specific art style that exhibits art style

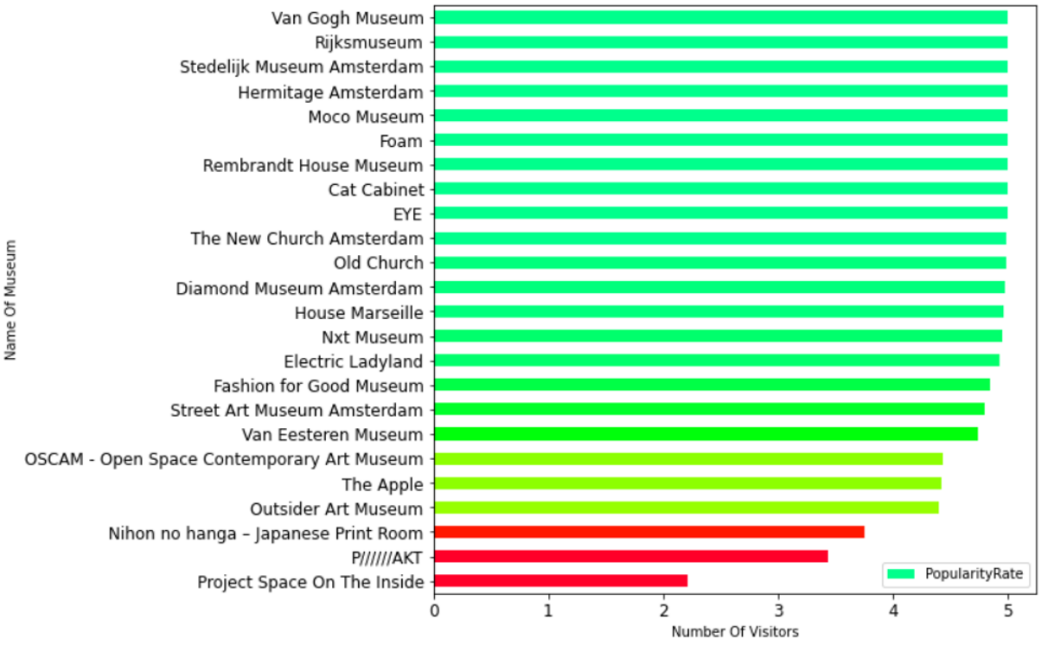

Figure A.4: This figure shows the popularity rate per museum

Please choose a distance range (km)

distance 12

Figure A.5: The user can choose a distance range for the maximum distance from the museum

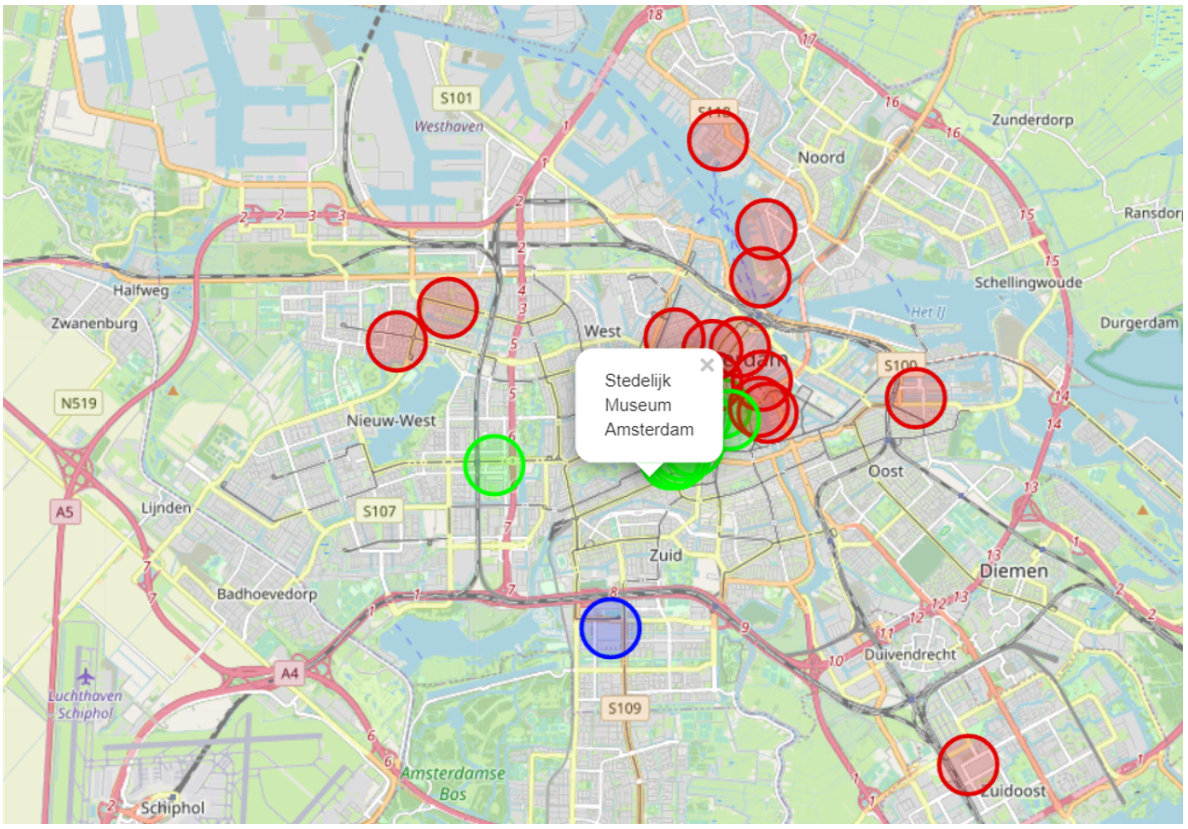

Figure A.6: This figures shows a map of amsterdam and the museums that are in the user's preferred distance range are marked green

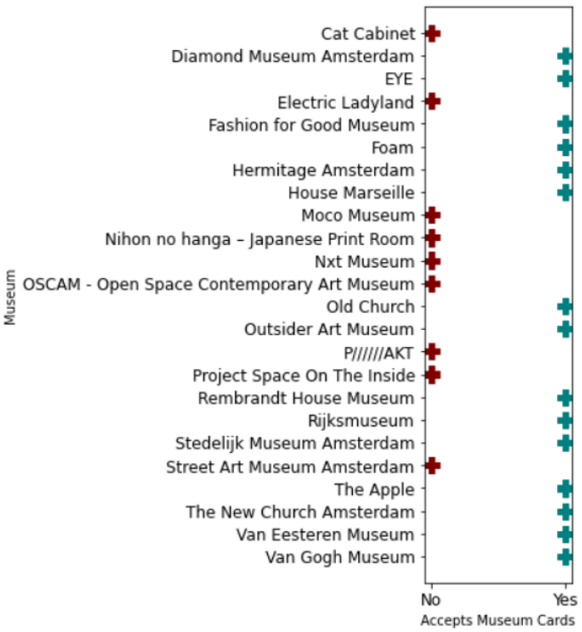

figure A.7: This figures shows a scatter chart of museums that either accept or refuse a museum card

List of price ranges of all the art museums in Amsterdam

Museum	price	fee_range
Cat Cabinet	Cheap	less than 10€
Diamond Museum Amsterdam	Cheap	less than 10€
EYE	Affordable	between 10€ and 15€
Electric Ladyland	Cheap	less than 10€
Fashion for Good Museum	Cheap	less than 10€
Foam	Affordable	between 10€ and 15€
Hermitage Amsterdam	Expensive	more than 15€
House Marseille	Cheap	less than 10€
Moco Museum	Expensive	more than 15€
Nihon no hanga – Japanese Print Room	Cheap	less than 10€
Nxt Museum	Expensive	more than 15€
OSCAM - Open Space Contemporary Art Museum	Cheap	less than 10€
Old Church	Affordable	between 10€ and 15€
Outsider Art Museum	Affordable	between 10€ and 15€
P//////AKT	Cheap	less than 10€
Project Space On The Inside	Cheap	less than 10€
Rembrandt House Museum	Affordable	between 10€ and 15€
Rijksmuseum	Expensive	more than 15€
Stedelijk Museum Amsterdam	Expensive	more than 15€
Street Art Museum Amsterdam	Expensive	more than 15€
The Apple	Cheap	less than 10€
The New Church Amsterdam	Expensive	more than 15€
Van Eesteren Museum	Affordable	between 10€ and 15€
Van Gogh Museum	Expensive	more than 15€

Figure A.8: This figure shows a table of museums and the average price of the entrance fee

```
%%capture cell1
%%sparql http://LAPTOP-VVBII7UP:7200/repositories/Final -s museumartist
PREFIX gp: <http://www.semanticweb.org/kandd/group76/final_project#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?Museum ?name{
    ?museum rdf:type gp:Museum ;
    ----rdfs:label ?label;
    ----gp:exhibitsArtFrom ?artist.
    ?artist gp:hasName ?name .
    BIND(?label AS ?Museum)
    FILTER ( !strstarts(str(?museum), "http://www.semanticweb.org/kandd/group54/ontology2") )
}ORDER BY ASC(str(?name))
```

Figure A.9: This figure shows a SPARQL Query that retrieves museums and their artists

```
%%capture cell2
%%sparql http://LAPTOP-VVBII7UP:7200/repositories/Final -s museums
PREFIX gp: <http://www.semanticweb.org/kandd/group76/final_project#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?Museum ?PopularityRate ?styleName{
    ?museum rdf:type gp:Museum ;
    ----rdfs:label ?label;
    ----gp:hasPopularity ?popularity ;
    ----gp:exhibitsArtFrom ?artist.
    ?artist gp:hasArtStyle ?artstyle .

    ?artstyle rdf:type ?artgenre .
    ?artgenre rdfs:subClassOf gp:ArtStyle.
    ?artgenre rdfs:label ?styleName.
    BIND(  ( (?popularity*5)/(?popularity+5) ) AS ?PopularityRate)
    BIND(?label AS ?Museum)
    FILTER ( !strstarts(str(?museum), "http://www.semanticweb.org/kandd/group54/ontology2") )
} ORDER BY DESC (?PopularityRate)
```

Figure A.10: This figure shows a SPARQL Query that retrieves museums and their art style

```
%%capture cell3
%%sparql http://LAPTOP-VVBII7UP:7200/repositories/Final -s museumspopularity
PREFIX gp: <http://www.semanticweb.org/kandd/group76/final_project#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?Museum ?PopularityRate {
    ?museum rdf:type gp:Museum ;
    ----rdfs:label ?label;
    ----gp:hasPopularity ?popularity ;

    BIND(  ( (?popularity*5)/(?popularity+5) ) AS ?PopularityRate)
    BIND(?label AS ?Museum)
    FILTER ( !strstarts(str(?museum), "http://www.semanticweb.org/kandd/group54/ontology2") )
} ORDER BY DESC (?PopularityRate)
```

Figure A.11: This figure displays a SPARQL Query that retrieves the museums and their popularity

```
%%capture cell4
%%sparql http://LAPTOP-VVBII7UP:7200/repositories/Final -s museumlocation
PREFIX gp: <http://www.semanticweb.org/kandd/group76/final_project#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT distinct?Museum ?lat ?lon {
    ?museum rdf:type gp:Museum ;
    rdfs:label ?label;
    gp:hasLong ?lon ;
    gp:hasLat ?lat .

    BIND(?label AS ?Museum)
} ORDER BY DESC (?PopularityRate)
```

Figure A.12: This figure shows a SPARQL Query that returns the museums and their longitude and latitude

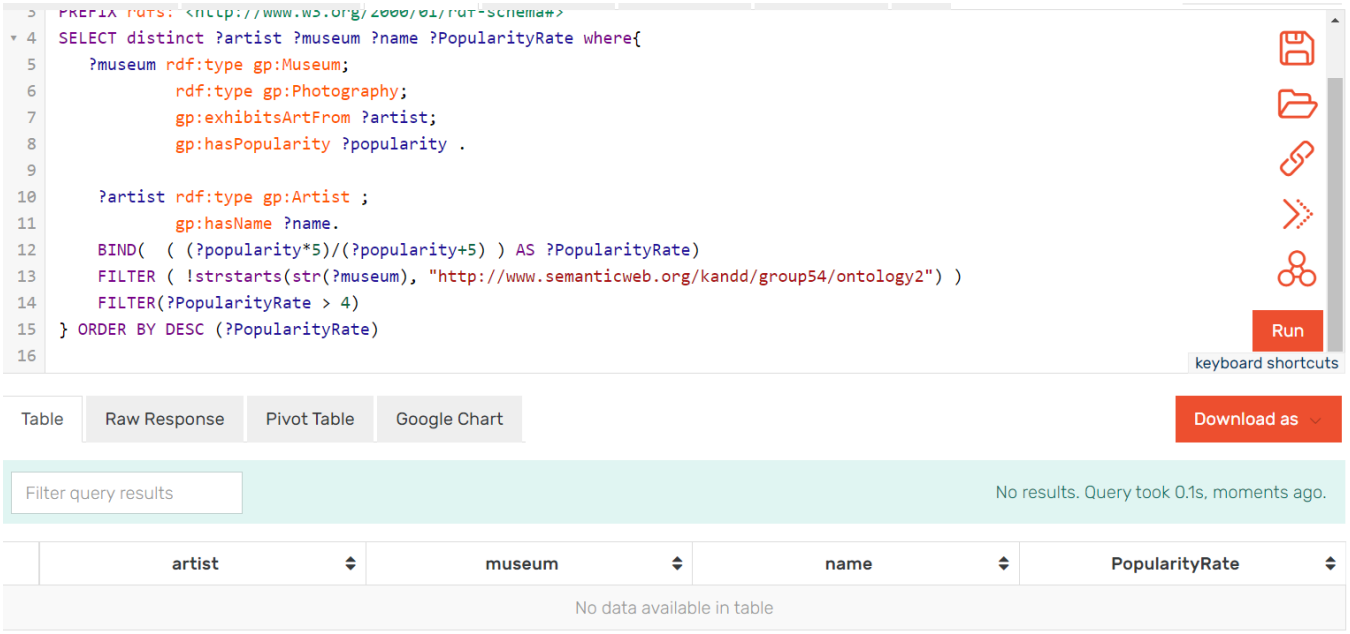

Figure A.13: This figure displays a screenshot of the SPARQL query in graphDB when the reasoner is off

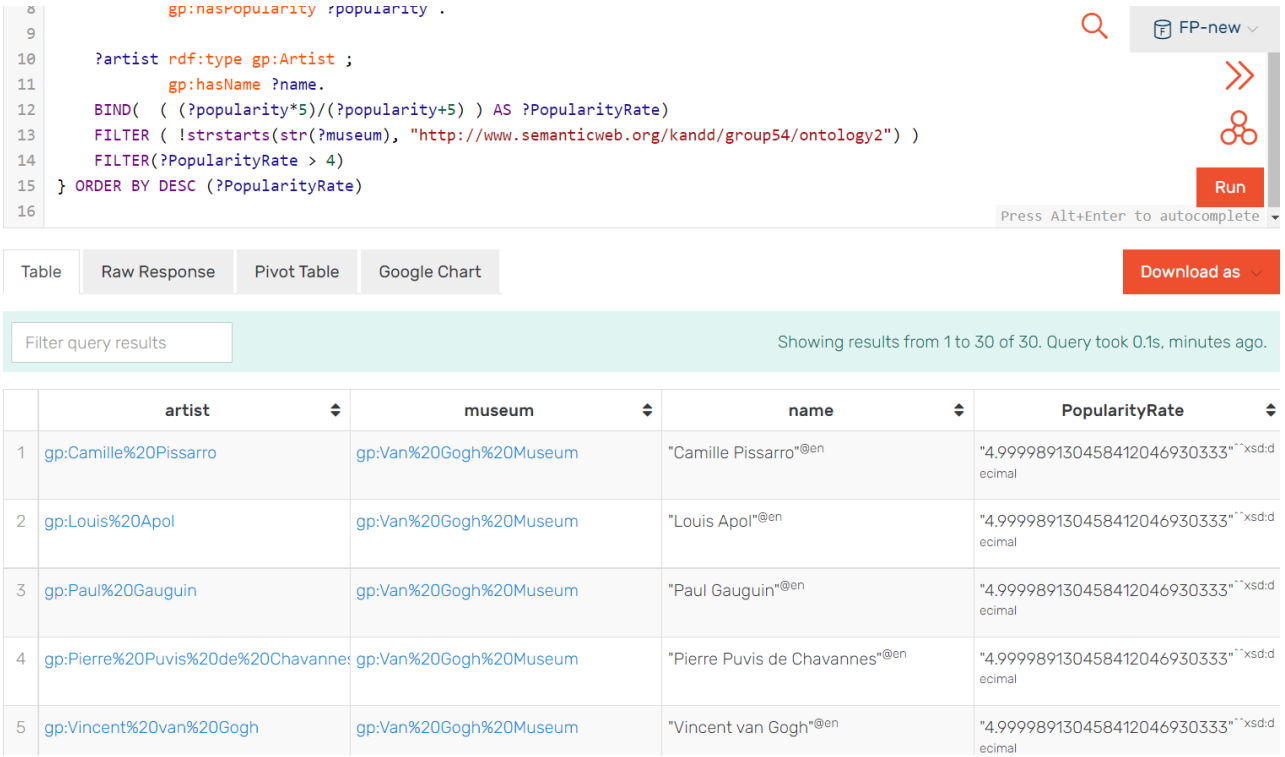

Figure A.14: This figure displays a screenshot of the SPARQL query in graphDB when the reasoner is on

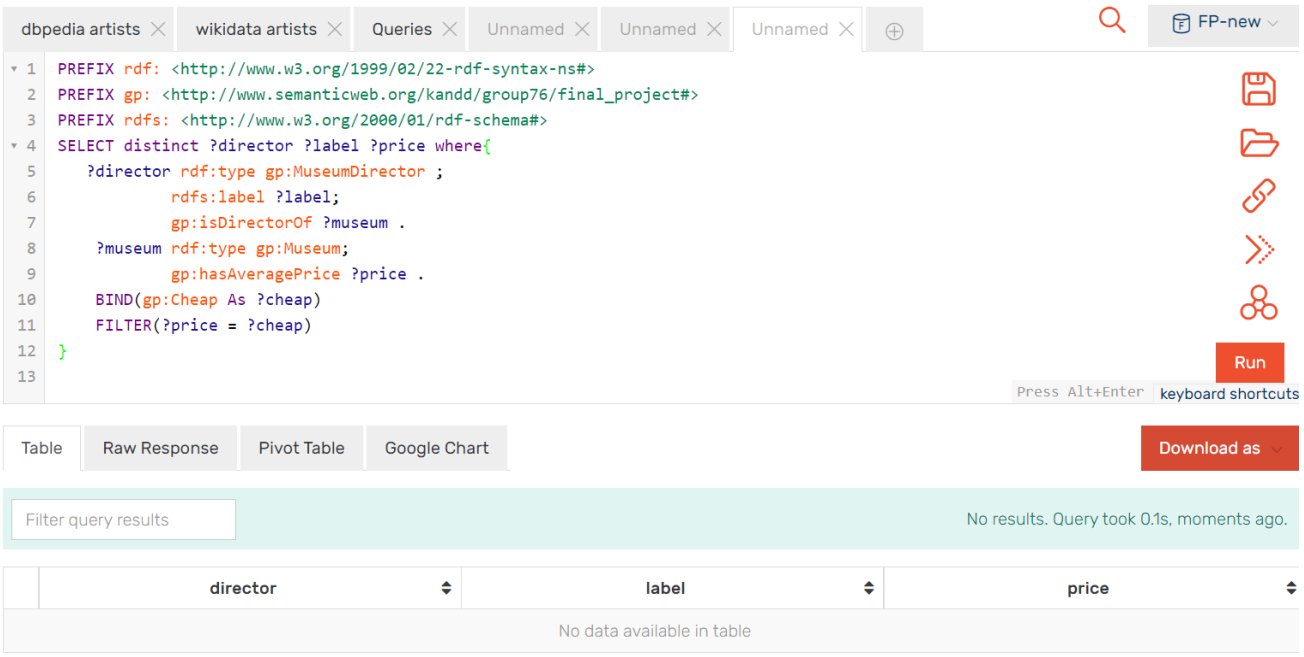

Figure A.15: This figure displays a screenshot of the second SPARQL query in graphDB when the reasoner is off

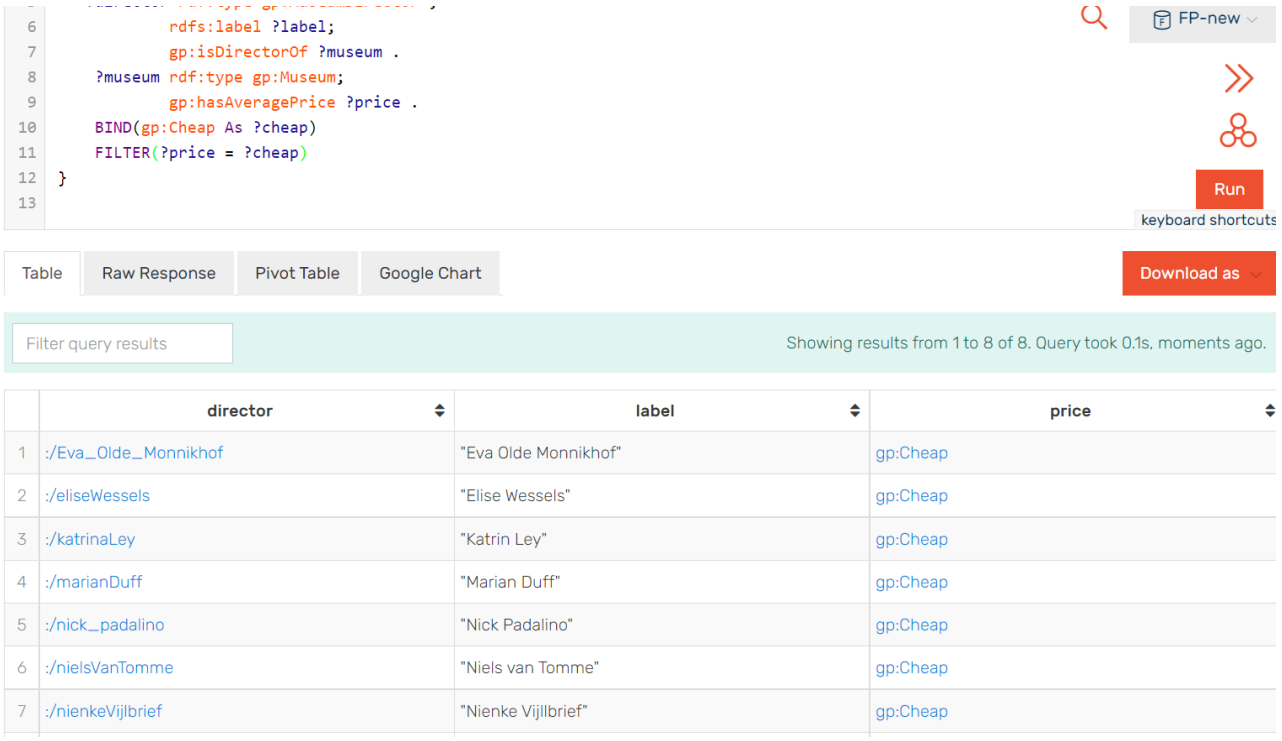

Figure A.16: This figure displays a screenshot of the second SPARQL query in graphDB when the reasoner is on