INFO9014 - Project

Alexia Donati s200742
Lei Yang s201670
Bruce Andriamampianina s2302253

May 20, 2024

LIÈGE université
Sciences Appliquées

Abstract

The Show Catalog ontology represents information about various shows. It facilitates the description of key attributes such as show, person, genre, rating, and country.

Contents

1 Introduction

This document outlines the development of our project, chosen as part of the Knowledge Representation and Reasoning course. Our project aims to create a solution that facilitates the search for films and TV series on a streaming platform. In an era where everyone has experienced endlessly scrolling through these platforms at least once, it seems essential to provide users with tools to reduce this search time.

To address this need, we must first design a database containing all the necessary information for filtering and segmenting shows. Next, we will create the ontology associated with this database, ensuring that it is general enough to be used in other systems. To distribute our solution across different platforms, it is crucial to map our ontology to our database and deploy our solution on these platforms. Finally, as a proof of concept, we will implement several SPARQL queries.

2 Information System

Our goal with this information system is to improve and expand users' search capabilities for movies and television shows. To achieve this, we used a dataset available on Kaggle [5] and created a conceptual database schema using the information from this dataset.

2.1 Database's description

This information system is based on Bansal Shivam's dataset "Netflix Movies and TV Shows" [5], which lists TV shows and movies added to the Netflix platform from 2014 to mid-2021. This dataset is composed of 12 columns:

- Show_id: A string identifying a Show within the data set (for example, s1 or s2).
- Type: A string indicating whether it's a movie or a TV show ("Movie" or "TV Show").
- Title: A string indicating the name of the show (examples include "Blood & Water" and "Midnight Mass").
- Director: A string list of the names or aliases of the directors of the show.
- Cast: A string list of the names or aliases of the actors in the show.

- Country: A string list indicating the production locations of the work (examples include ”South Africa”, ”United States”).
- Date_added: A date column giving the date the show was added to Netflix (for example, ”September 25, 2021” or September 24, 2021”).
- Release_year: A date indicating the year of the show’s release (for example, ”1999” or ”2021”).
- Rating: A string indicating the show’s content rating category (examples include ”TV-Y” and ”TV-14”).
- Duration: An integer indicating the length of the show, in minutes for movies (for example, ”90”) or seasons for TV shows (for example, ”2 seasons”).
- Listed_in: A string list of genres in which the show can be categorized (examples include ”TV Dramas” and ”Crime TV Shows”).
- Description: A string giving a synopsis of the show.

We have developed a conceptual database schema from this dataset that supports our solution best. It comprises 8 entities and 7 relations:

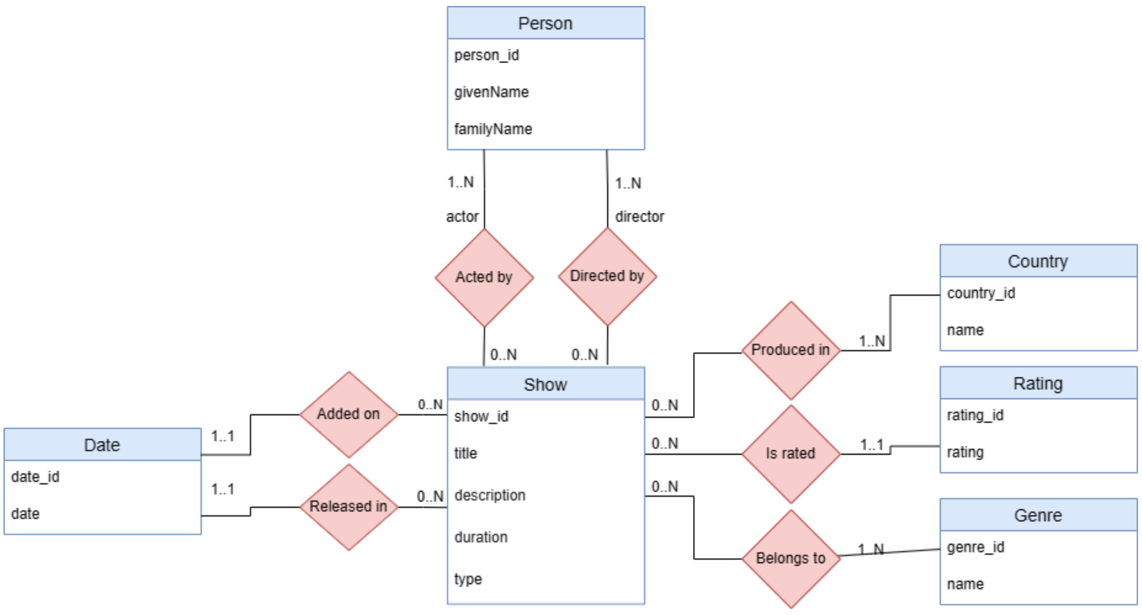


Figure 1: Relational Database conceptual schema

As you can see from our diagram, we have made several modifications to the initial dataset to obtain the relational database shown above:

- A show can be connected to multiple instances of certain entities. It seemed better to split these entities into different tables. These tables are Country, Genre, and Person. This decision helps us organize the database better by arranging the data in a way that reduces repetition and mistakes. By using separate tables for each entity, we can manage the data more effectively, preventing the same information from being repeated and ensuring that the data is more accurate.
- Many normalization operations were performed on the raw dataset to make the results easier to read, such as removing outliers (for example, removing null values) and simplifying manipulation (for example, removing spaces and changing column types).
- Artificial primary keys were introduced to ensure each record's uniqueness and streamline search and join operations within the database. This happens because integers are faster to handle compared to other data types.

2.2 Approach Description

Our database is based on several constraints and arbitrary choices, enabling us to obtain a solution that supports our application:

- The utilization of two relationships connecting the entities "person" and "show" was favored over establishing two distinct classes for actors and directors. This approach facilitated a broader implementation, thus increasing the likelihood of its applicability in various structures. Additionally, maintaining the same information in the database for these two entities rendered the creation of two separate tables unnecessary.
- We also chose to store the duration as an integer for the TV Shows and Movies by removing the attribute's textual part, to enable easy creation of statistics such as averages or maximum and minimum.

2.3 Contextual Description

The main idea of this database is to help users easily find movies by using specific filters. In our diagram, each connection linked to the "Show" entity can be seen as a different way to filter shows. This structure allows for various queries to be conducted:

- The user may want to search for movies shot in the United States suitable for 14-year-olds, whose content rating is PG-14.
- The user may want to search for a TV Show released between 2018 and 2021 (for modern image quality) featuring a specific actor.
- The user may want to search for a movie whose duration is below the average duration of all movies and is categorized as a thriller or drama.

The database is on PostgreSQL to initialize it, we can use the file Main.py

3 Ontology and Ontology Engineering

To create an ontology that fits well with our database, we first looked into which OWL 2 profile would suit our system and its uses. Then, we built the ontology, organizing the elements of our database according to the chosen OWL 2 QL profile, making sure to define each class and property clearly. This process raised some questions and required us to make compromises.

3.1 Design Decisions

The ontology development process encompasses our various design decisions and considerations, such as the OWL 2 profile [4] we chose and the choice we made surrounding the various concepts.

3.1.1 Motivation for using OWL 2 QL

Our ontology, designed to facilitate quick searches based on criteria such as genre, country of production, director, and actor, is derived directly from an existing relational database.

In this context, choosing the OWL 2 QL profile is fully justified, as it balances expressiveness and computational efficiency. However, it also has significant limitations that need to be considered. These limitations, such as the inability to specify complex rules about the relationships between different ontology elements, may restrict the modeling of very detailed ontologies.

Nevertheless, OWL 2 QL is still a great option for our project. It works well with relational databases, allowing us to seamlessly integrate our current database and ontology to ensure data consistency across different representations. Additionally, it is perfect for quickly and effectively running searches, which is exactly what we need for our search tool, where quick responses and search performance are crucial.

Furthermore, our basic ontology did not require the use of concepts such as cardinality restrictions, transitive properties, or disjunction (ObjectUnionOf, DisjointUnion, and DataUnion), which were too advanced.

OWL 2 EL could have been a good fit, as it is designed for simple ontologies and provides efficient performance for ontological reasoning. However, since our ontology does not contain many entities and properties, it was not the optimal choice.

Similarly, OWL 2 RL was not the right fit either, as we do not have a highly complex ontology that would require manipulating data in triple form. Additionally, we do not need to deduce new information from existing data since most of the information is explicit.

3.1.2 Ontology description

For our ontology, we did not strictly follow our database schema and adapted some of the concepts to fit a broader range of use cases.

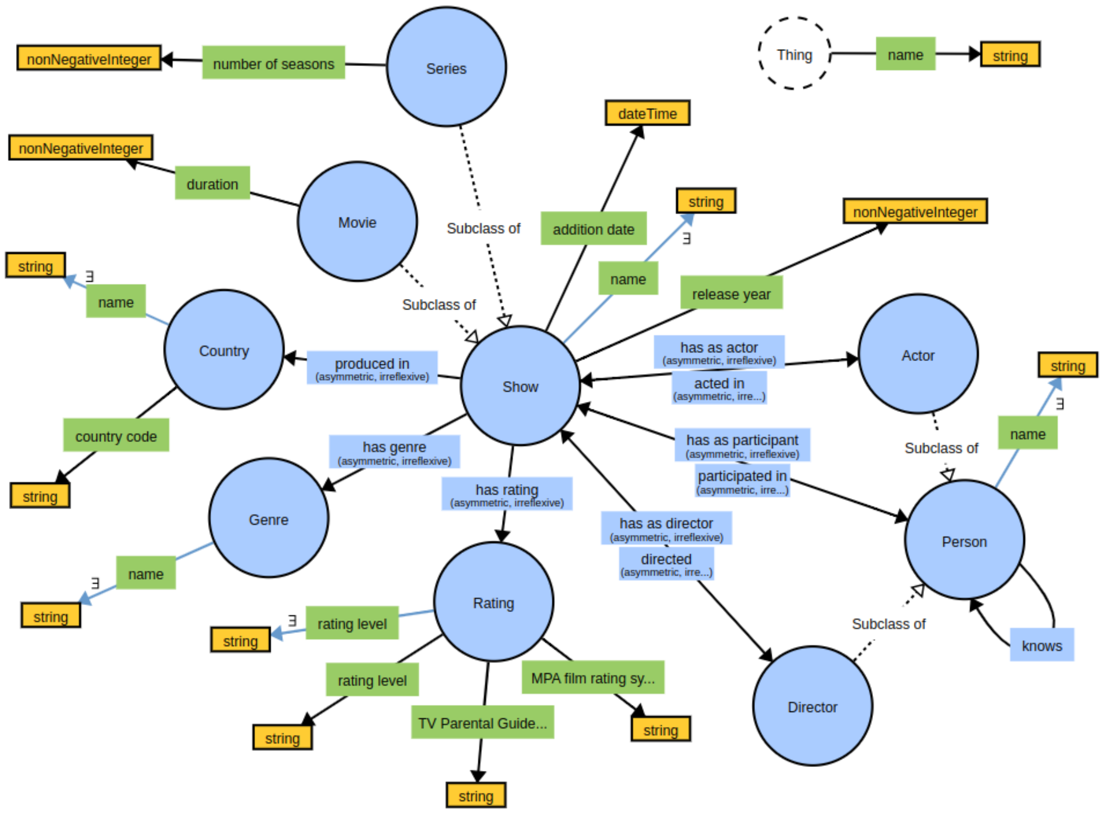


Figure 2: Ontology schema

A notable change from the original database schema is that our ontology adds specific Actor and Director sub-classes to the existing Person class. Following this change, the range of the object property ”acted in” is now Actor, while the range of ”directed” is now Director. This allows for a person to be inferred as either an Actor or Director if they are part of one of these relationships. Additionally, a person is considered to have ”participated in” a show if they have either acted in or directed it.

The Show class is at the center of our ontology, and it is linked by an object property to every other class. Therefore, most decisions we have made relate to it in one way or another. We decided on a list of existential quantifications that applies to it. Regarding object properties, every show has some director, genre, and country of production. As for data properties, every show has a name. Although in our original database every show was assigned a rating, this does not hold in the real world, as a show could be missing one if it hasn't yet been submitted to a rating association. As opposed to a relational database, because of the open world assumption, these existential quantified element can denote unknown individuals which are not yet part of the ontology individuals.

Another change we made from the original database schema was to generalize the different properties relating to the designation of a thing into a unique data property called ”name.”

3.2 Ontology Engineering Process

The process of creating our ontology involved using various tools and gathering information. It also led to discussions about how to adhere to the rules of the OWL 2 QL profile.

3.2.1 Ontology Development Tools

To construct our ontology, we employed specific tools and resources.

We used Protégé [7] as our primary ontology editor. To understand how to use the program, we relied on the Protégé documentation [8] and followed the tutorial authored by Michael DeBellis [1] that it links to.

We also relied on the OWL 2 documentation, available online [4], to help us better understand the constraints and advantages associated with our choice of OWL 2 profile. To verify that our ontology did follow our selected OWL profile we also used a profile checker [6].

Lastly, we used WebVOWL [3] to generate a graph of our ontology to facilitate its understanding.

3.2.2 Encountered Problems and Solutions

During the development of our ontology, we were faced with several challenges related to the choice of our OWL profile:

- In our original relational database schema, classes have an identifying key. However, the OWL 2 QL profile does not allow the use of keys. This has led to the deletion of the key properties we initially planned on adding.

  This modification could raise some issues in the long term since, in practice, the database is not always correctly cleaned up, which increases the chances of having duplicates.

- The OWL 2 QL profile does not support cardinality restrictions. Luckily, we mostly used existential quantification, which is allowed by the profile. One such example is that each show has some title, director, genre, and country of production.

- In our initial ontology version, we implemented the ”knows” object property, which is defined by two people having participated in the same show, with a property chain: ”participatedIn o hasParticipant.”

  However, upon reviewing the OWL 2 QL profile documentation [4], we found out that it does not support this feature. For this reason, we had to implement it in our mapping instead.

4 Mapping Relational Databases to RDF

4.1 Mapping development process

We employed the principles outlined in lecture 8 of the course on ”Knowledge Graph Generation and Virtual Knowledge Graphs” to produce various files concerning the mapping between our SQL relational database and ontology.

4.2 Knowledge Organization and URI Strategies

For our URI strategy we decided on using the name of the class and the key identifier of the original SQL database element.

We designed our mappings to closely align the relational database data with the ontology, ensuring accurate mapping and transformation into RDF triples. This approach involved several key steps.

Firstly, we created triples map for each table of the database that we want to map onto our ontology. Each of these triples map is made of multiple elements.

Firstly, it specifies the logical table from which the data is retrieved. Then, it specifies a subject map that defines how to generate the subject of the RDF triples from the rows of the SQL database. Lastly, it specifies one or more predicate object map to define how to generate the predicate and object pairs for RDF triples.

Furthermore, the #ActorMapping and #DirectorMapping demonstrate our approach to handling multiple roles of individuals. Each role (actor or director) is associated with its own class (show:Actor and show:Director, respectively), with relationships to shows managed through predicate-object maps. This separation ensures clarity and precision in role-specific data representation, while the use of a common URI template for persons maintains consistency across different mappings.

We also had to implement some complex join queries to establish relationships between certain entities. For instance, #KnowsMapping merges data from both the acted_in and directed_by tables to determine relationships between individuals who participated in the same shows. The SQL query within this mapping performs a self-join on the combined dataset to identify this relationship.

Additionally, #ShowMapping uses multiple attributes from various of our database tables. This mapping includes joins with rated, genresof, and filmed_in tables to comprehensively capture show details like genres, ratings, and production countries. This mapping ensures that all relevant information about a show is accurately linked and represented in the RDF model.

We also used the mapping to combine the first and last names of a Person, which is represented by two separate properties in our database, into a single literal value in order to fit our ontology definition.

4.3 Critics of our mapping

In our mapping process, we had to make certain compromises due to technical constraints and time limitations. Even tho our mapping fully links our relational database to our ontology, we noticed specific areas where our approach could have been better.

Our mapping relies on some complex SQL queries to extract and transform data into RDF triples. For instance, as mentioned above, #KnowsMapping uses an intricate queries to obtain the information it needs from the SQL database. While this approach allowed us to represent the knows property as we wanted, it also introduced significant dependencies on the underlying database structure. Simplifying the SQL queries, where possible, would reduce complexity and improve maintainability.

Overall, our mapping process achieved an acceptable alignment between the ontology and the relational database data. However, addressing the identified areas for improvement, such as simplifying complex queries and reducing redundancy, would enhance the robustness and efficiency of our current mapping solution. They would ensure that the mapping process remains resilient and adaptable to future changes in the ontology, ultimately leading to more reliable and accurate RDF representations.

5 Deployment and Demonstration

5.1 How do you use or deploy the ontology and RDF ?

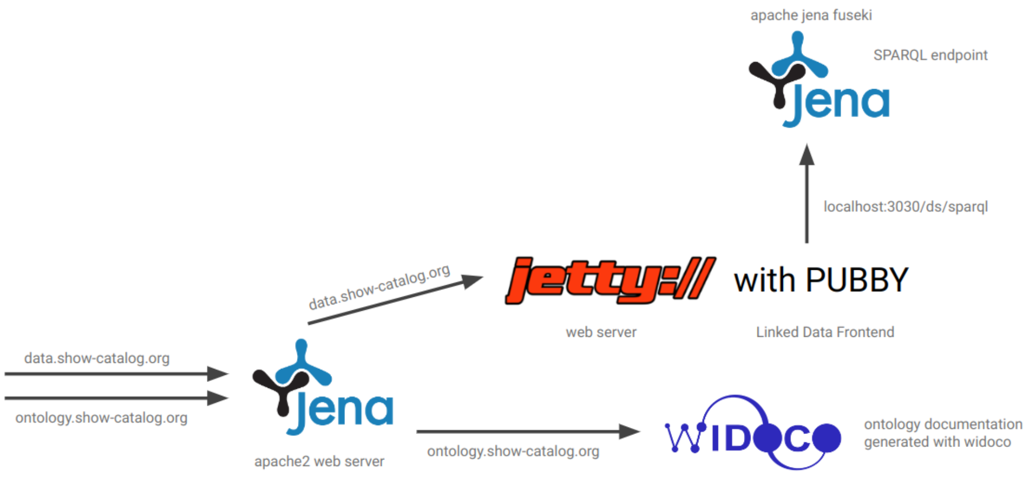
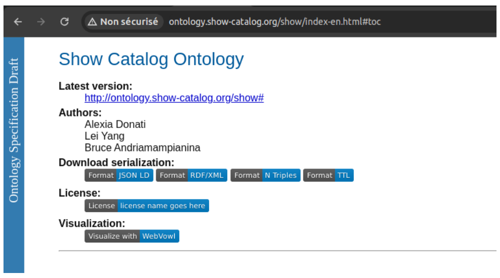


Figure 3: Deployment schema

Here are the steps needed to deploy our ontology on a machine equipped with Ubuntu 22.04.4 and OpenJDK 17.0.10.

- SPARQL Endpoint: Apache Jena Fuseki (version 5.0.0)
  - Add the following line to `./etc/hosts`: `127.0.0.1 data.show-catalog.org`
  - Run `./fuseki-server`
  - Create A new dataset name ds.
  - Upload data into ds from the `output.ttl` file generated by the mapping.
- Linked Data frontend: pubby
  - Build pubby: `mvn package -DskipTests -Dmaven.javadoc.skip=true`


Non sécurisé ontology.show-catalog.org/show/index-en.html#toc

Ontology Specification Draft

Show Catalog Ontology

Latest version:
http://ontology.show-catalog.org/show#

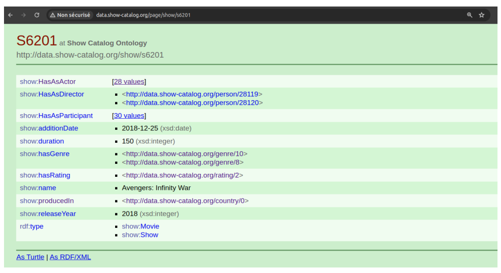
Authors:
Alexia Donati
Lei Yang
Bruce Andriamampianina

Download serialization:
Format JSON LD | Format RDF/XML | Format N Triples | Format TTL

License:
License license name goes here

Visualization:
Visualize with WebVowl

Figure 4: ontology.show-catalog.org


Non sécurisé data.show-catalog.org/page/show/s6201

S6201 at **Show Catalog Ontology**
http://data.show-catalog.org/show/s6201

show:HasAsActor	[28 values]
show:HasAsDirector	• <http://data.show-catalog.org/person/28119> • <http://data.show-catalog.org/person/28120>
show:HasAsParticipant	[30 values]
show:additionDate	• 2018-12-25 (xsd:date)
show:duration	• 150 (xsd:integer)
show:hasGenre	• <http://data.show-catalog.org/genre/10> • <http://data.show-catalog.org/genre/8>
show:hasRating	• <http://data.show-catalog.org/rating/2>
show:name	• Avengers: Infinity War
show:producedIn	• <http://data.show-catalog.org/country/0>
show:releaseYear	• 2018 (xsd:integer)
rdf:type	• show:Movie • show:Show

As Turtle | As RDF/XML

Figure 5: data.show-catalog.org

  - Replace the current file `./target/pubby/WEB-INF/config.ttl` by the `config.ttl` file attached to this report.

- Web server: jetty (version 9.4.54)
  - Rename the `./target/pubby` folder into ROOT and copy it into `./jetty/webapps/`
  - Run `java -jar start.jar`

- Generating documentation: Widoco [2]

- Apache2 Web server: LAMP stack
  - Add the following line to `./etc/hosts`: `127.0.0.1 ontology.show-catalog.org`
  - Add the `./show/` folder attached to this report to the folder `./var/www/`
  - Add the `ontology.conf` file to the folder `./etc/apache2/sites-available/`
  - Run `sudo a2ensite ontology.conf` to enable the site.
  - Run `sudo systemctl reload apache2` to reload the web server.

After that, as seen in figure 4, accessing the link ontology.show-catalog.org/show should serve the ontology and, as seen in figure 5, accessing the link data.show-catalog.org should serve the linked data.

5.2 SPARQL queries

To be able to use SPARQL, we need two files, the data file and the ontology file to initialize the graphs to do some queries. We will start with simple queries and increase the difficulty of this to show the different possibilities to use SPARQL. All the queries used are in the file SPARQL.ipynb to show the global output.

5.2.1 Simple Query

This query will give us movies with a duration of more than 90 minutes.

```
PREFIX ntf: <http://ontology.show-catalog.org/show#>
SELECT ?name ?duration
WHERE
  { ?x ntf:name ?name .
    ?x ntf:duration ?duration FILTER (?duration > 90)}
LIMIT 100
```

5.2.2 Query with a Join

This query will give us the different movies which Tom Holland has played. This will need a join to do this.

```
PREFIX ntf: <http://ontology.show-catalog.org/show#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?movieName
WHERE {
  ?show rdf:type ntf:Movie ;
        ntf:name ?movieName ;
        ntf:HasAsActor ?actor .
  ?actor rdf:type ntf:Person ;
         ntf:name "Tom Holland" .
}
LIMIT 100
```

5.2.3 Query with Aggregates

This one will give us the number of movies produced in different countries.

```
PREFIX ntf: <http://ontology.show-catalog.org/show#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?countryName (COUNT(*) AS ?count)
WHERE {
  ?show rdf:type ntf:Movie ;
        ntf:name ?name ;
        ntf:producedIn ?country .
  ?country ntf:name ?countryName .
}
```

```
GROUP BY ?countryName
ORDER BY DESC(?count)
```

The same output but used in SQL

```
SELECT c.country_name, COUNT(f.country_id) AS count
FROM filmed_in AS f
JOIN country AS c ON f.country_id = c.country_id
JOIN show AS s ON f.show_id = s.show_id
WHERE s.type_show = 'Movie'
GROUP BY f.country_id, c.country_name
ORDER BY count DESC;
```

5.2.4 New Query possible

This query will show by using the ontology and the data, we can create new relations that we can do some queries about it. For example, in this context, we will use "know" to find all the movies with an actor who knows Tom.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ntf: <http://ontology.show-catalog.org/show#>

SELECT DISTINCT ?movie
WHERE {
  ?show rdf:type ntf:Movie;
        ntf:name ?movie ;
        ntf:producedIn ?country;
        ntf:HasAsActor ?actor .
  ?country ntf:name ?countryName .
  FILTER (?countryName = 'United States' || ?countryName = 'India')
      .
  ?actor ntf:name ?actorName .
  ?actor ntf:knows ?knownPerson .
  ?knownPerson ntf:name ?knownPersonName .
  FILTER regex(?knownPersonName, "Tom","i")
}
```

There are some advantages to using SPARQL instead of SQL query. Firstly, the join in SPARQL is simpler than in SQL. Secondly, we can also see that we do not redefine, in SPARQL there is no ambiguity. There is also no need to use the select function. By using SPARQL, we do not need the relation tables, and it will give us a much more clear query. Also, we can go beyond the initialized database design to do some more complex queries. Finally, queries in SPARQL are faster than the equivalent in SQL.

6 Non-trivial Demonstrator(s)

Because of time constraint, we weren't able to implement Non-trivial demonstrators for this project.

7 Conclusions

In conclusion, to develop our solution, we went through several steps. First, we created a database to help us manage all the information about movies and TV shows efficiently. Then, we devised a more general way to organize this information by creating an ontology that standardizes our information system. After that, we mapped our original SQL database onto that ontology. Afterwards, with the deployment, we ensured that we could distribute our new search tools across different platforms. Finally, we created SPARQL queries for specific searches to allow users to find the best recommendations based on their own criteria.

However, our solution also has weaknesses, particularly in terms of the variety of data available, which may seem insufficient for an effective long-term solution. One way to address this problem could be to integrate data from other sources to enrich our database and offer our users greater flexibility.

Finally, although we didn't have time to certain aspects of the project, we are nonetheless satisfied with the final result.

References

[1] Michael DeBellis. *A Practical Guide to Building OWL Ontologies using Protégé 5.5 and Plugins*. 2021. URL: https://drive.google.com/file/d/1A3Y8T6nIfXQ_UQOpCAr_HFSCwpTqELeP/view (visited on 03/19/2024).

[2] Daniel Garijo. "WIDOCO: a wizard for documenting ontologies". In: *International Semantic Web Conference*. Springer, Cham. 2017, pp. 94–102. DOI: 10.1007/978-3-319-68204-4_9. URL: http://dgarijo.com/papers/widoco-iswc2017.pdf.

[3] Steffen Lohmann et al. "WebVOWL: Web-based Visualization of Ontologies". In: *Knowledge Engineering and Knowledge Management*. Ed. by Patrick Lambrix et al. Cham: Springer International Publishing, 2015, pp. 154–158. ISBN: 978-3-319-17966-7.

[4] OWL Working Group. *OWL*. 2009. URL: https://www.w3.org/TR/2009/WD-owl2-new-features-20090421/#OWL_2_QL (visited on 03/27/2024).

[5] Bansal Shivam. *Netflix Movies and TV Shows*. 2022. URL: https://www.kaggle.com/datasets/shivamb/netflix-shows/data (visited on 03/02/2024).

[6] Soiland-Reyes Stian. *OWL API profile checker*. 2017. URL: https://github.com/stain/profilechecker.

[7] Stanford University. *Protégé 5.5.0*. 2019. URL: https://protege.stanford.edu (visited on 03/18/2024).

[8] Stanford University. *Protégé wiki*. 2022. URL: https://protegewiki.stanford.edu/wiki/Main_Page (visited on 03/18/2024).