

2024

 **REPOSITORY:** [**PerfectionistAF/Movies\_Ontology (github.com)**](https://github.com/PerfectionistAF/Movies_Ontology)

CSE488: ontologies and the semantic web

AHMED ASHOUR 20P4222

Habiba yasser abdelhalim 20P3072

nadine hisham 20P9880

Sohayla Ihab Abdelmawgoud Ahmed Hamed 19P7343

A Design and Implementation for Movies Ontology

Contents

[Problem Description: 2](#_Toc166534884)

[Part I: Modeling the Ontology 2](#_Toc166534885)

[Part II: Populating the Ontology 8](#_Toc166534886)

[Part III: SPARQL Queries on the Ontology 14](#_Toc166534887)

[Additional queries tested on Jena 24](#_Toc166534888)

[Part IV: Manipulating the ontology using Jena 26](#_Toc166534889)

[Part V: Python application 32](#_Toc166534890)

[Data Flow Diagram Level 0 32](#_Toc166534891)

[Data Flow Diagram Level 1 33](#_Toc166534892)

[Data Flow Diagram Level 2 34](#_Toc166534893)

[Environment Dependencies and Libraries 35](#_Toc166534894)

[Setting Up a SPAQRL Endpoint 35](#_Toc166534895)

[Desktop Application Interface 36](#_Toc166534896)

[Test Cases 37](#_Toc166534897)

# Problem Description:

This project models an ontology of movies. Protégé Editor was used to create this ontology of movies. Basically, we aim to create a blueprint for the online referencing and storage of movie datasets. This ontology will be used to model a python desktop application to reason querying and selecting movies according to the required included and excluded individuals. This work is divided into sections corresponding to the given project description. Part I models the ontology, Part II populates the ontology and checks for consistency, Part III performs SPARQL queries, Part IV uses Jena and rdflib to test SPARQL queries and finally, Part V focuses on a user-friendly interface to query the ontology.

# Part I: Modeling the Ontology

First, the following classes and properties are required to be set:

1. **Class:**
2. Movie: defines basic Movie class
3. Genre: defines basic Genre class
4. Person: defines basic Person class
5. Actor: basic Actor class is a subclass of Person class
6. Writer: basic Writer class is a subclass of Person
7. Director: basic Director class is a subclass of Person
8. **Object Property:**
9. hasGenre: Each Movie has one or more instances of class Genre
10. hasActor: Each Movie has at least one instance of class Actor
11. hasWriter: Each Movie has at least one instance of class Writer
12. hasDirector: Each Movie has at least one instance of class Director
13. isGenreOf: Inverse of hasGenre
14. isActorOf: Inverse of hasActor
15. isDirectorOf: Inverse of hasDirector
16. isWriterOf: Inverse of hasWriter
17. **Data Property:**
18. name: Name of Person
19. age: Age of Person
20. nationality: Nationality(s) of Person
21. gender: Gender of Person
22. genre: Genre(s) of Movie
23. title: Title of Movie
24. country: Country(s) of Production of Movie
25. language: Language(s) of Movie
26. year: Year of Copyright of Movie

In order to create restrictions and proper relationships between classes and properties, the following assumptions have been observed:

1. Datasets all have films dating from the early 1900s till the current documentation
2. An individual can be an actor or a director or a writer. They can also play multiple roles (ie: an actor can be a director or a writer as well)
3. Disjointedness with Persons is also defined between Actors, Directors, Writers. It is assumed that the data is apt to change with time.
4. A Movie cannot have the same values of the Genre.
5. A Movie can have the same name or title as an Actor, Director, Movie (a Movie called Maya Angelou should not be shown in a query looking for a Director named Maya Angelou).
6. A Director with a minor role in a Movie is assumed to be also an Actor.
7. A Genre cannot have name of Person (Actor, Writer, Director) or Movie
8. hasGenre cannot be equivalent to hasActor, hasDirector, hasWriter between the same individuals.
9. Disjointedness between data properties is always between properties belonging to the same class.
10. Language and Country of a Movie can be equivalent values, but not necessarily (i.e.: A Movie whose country is New Zealand can have the language English, Māori (in English tags only). Language can also be entered as New Zealand, so New Zealand should be filtered as a value among others in a query searching for country == “New Zealand”). The data should be understandable if France or French is used for language. However, nationality of a person and country of a movie are not disjoint, to provide more search results.
11. Only one name and one title for a Person or a Movie should be registered.
12. There are only 2 genders, male or female.
13. A Movie in a different language is considered a different movie (ie: Sleeping Beauty exists in languages: English, Arabic and

Al-Jameela Al-Na’ema also exists in the languages: Arabic, English).

1. Only 1 type of language tags were used, the default English tags. There weren’t explicitly declared.
2. Age 0 means the Person is deceased.
3. A Person (Actor, Director, Writer) can have multiple nationalities.

To define disjointedness and other characteristics of defined properties, the following was included:

1. Genre:

**+** **Properties:**

+ genre max 5 xsd:string

+ isGenreOf min 1 Movie **| Disjoint:** isActorOf, hasDirector, hasWriter, hasActor, isWriterOf, isDirectorOf **| Inverse:** hasGenre **| Domain: Genre | Range: Movie**

* **Disjointedness: Disjoint with a Person ≡ Disjoint with an Actor; Disjoint with an Actor ≠ Disjoint with a Person**
* Actor
* Writer
* Director
* Movie

1. Movie

**+** **Properties:**

+ country min 1 xsd:string

+ language exactly 1 xsd:string

+ title exactly 1 xsd:string **| Functional**

+ year exactly 1 xsd:integer **| Functional | year some xsd:integer[>= 1900]**

+ hasActor min 1 Actor **| Disjoint:** hasGenre, isGenreOf **| Inverse:** isActorOf **| Domain: Movie | Range: Actor**

+ hasDirector min 1 Director **| Disjoint:** hasGenre, isGenreOf **| Inverse:** isDirectorOf **| Domain: Movie | Range: Director**

+ hasGenre min 1 Genre **| Disjoint:** isActorOf, hasDirector, hasWriter, hasActor, isWriterOf, isDirectorOf **| Inverse:** isGenreOf **| Domain: Movie | Range: Genre**

+ hasWriter min 1 Writer **| Disjoint:** hasGenre, isGenreOf **| Inverse:** isWriterOf **| Domain: Movie | Range: Writer**

* **Disjointedness:**
* Genre
* Actor
* Writer
* Director

1. Person

**+** **Properties:**

+ age exactly 1 xsd:integer**| Functional | age some xsd:integer[>= 0]**

+ gender exactly 1 xsd:string **| Functional | (gender value "Female") or (gender value "Male") or (gender value "female") or (gender value "male")**

+ name exactly 1 xsd:string **| Functional**

+ nationality min 1 xsd:string

* **Disjointedness:**
* Genre

1. Actor

**+** **Properties:**

**+ Subclass of Person**

+ isActorOf min 1 Movie **| Disjoint:** hasGenre, isGenreOf **| Inverse:** hasActor **| Domain: Actor | Range: Movie**

+ age exactly 1 xsd:integer**| Functional | age some xsd:integer[>= 0]**

+ gender exactly 1 xsd:string **| Functional | (gender value "Female") or (gender value "Male") or (gender value "female") or (gender value "male")**

+ name exactly 1 xsd:string **| Functional**

+ nationality min 1 xsd:string

* **Disjointedness:**
* Genre
* Movie

1. Director

**+** **Properties:**

**+ Subclass of Person**

+ isDirectorOf min 1 Movie **| Disjoint:** hasGenre, isGenreOf **| Inverse:** hasDirector **| Domain: Director | Range: Movie**

+ age exactly 1 xsd:integer**| Functional | age some xsd:integer[>= 0]**

+ gender exactly 1 xsd:string **| Functional | (gender value "Female") or (gender value "Male") or (gender value "female") or (gender value "male")**

+ name exactly 1 xsd:string **| Functional**

+ nationality min 1 xsd:string

* **Disjointedness:**
* Genre
* Movie

1. Writer

**+** **Properties:**

**+ Subclass of Person**

+ isWriterOf min 1 Movie **| Disjoint:** hasGenre, isGenreOf **| Inverse:** hasWriter **| Domain: Writer | Range: Movie**

+ age exactly 1 xsd:integer **| Functional | age some xsd:integer[>= 0]**

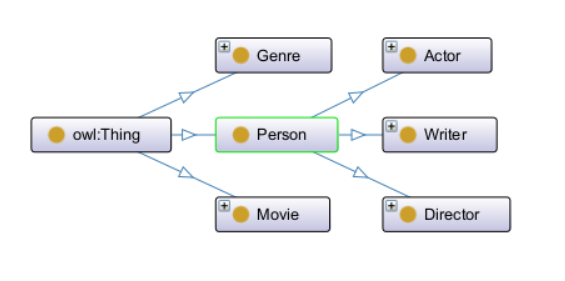
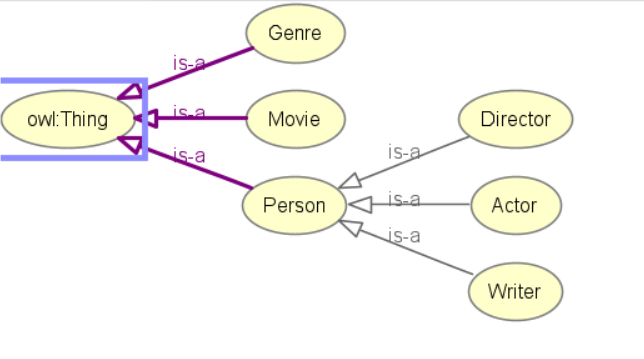
+ gender exactly 1 xsd:string **| Functional | (gender value "Female") or (gender value "Male") or (gender value "female") or (gender value "male")**

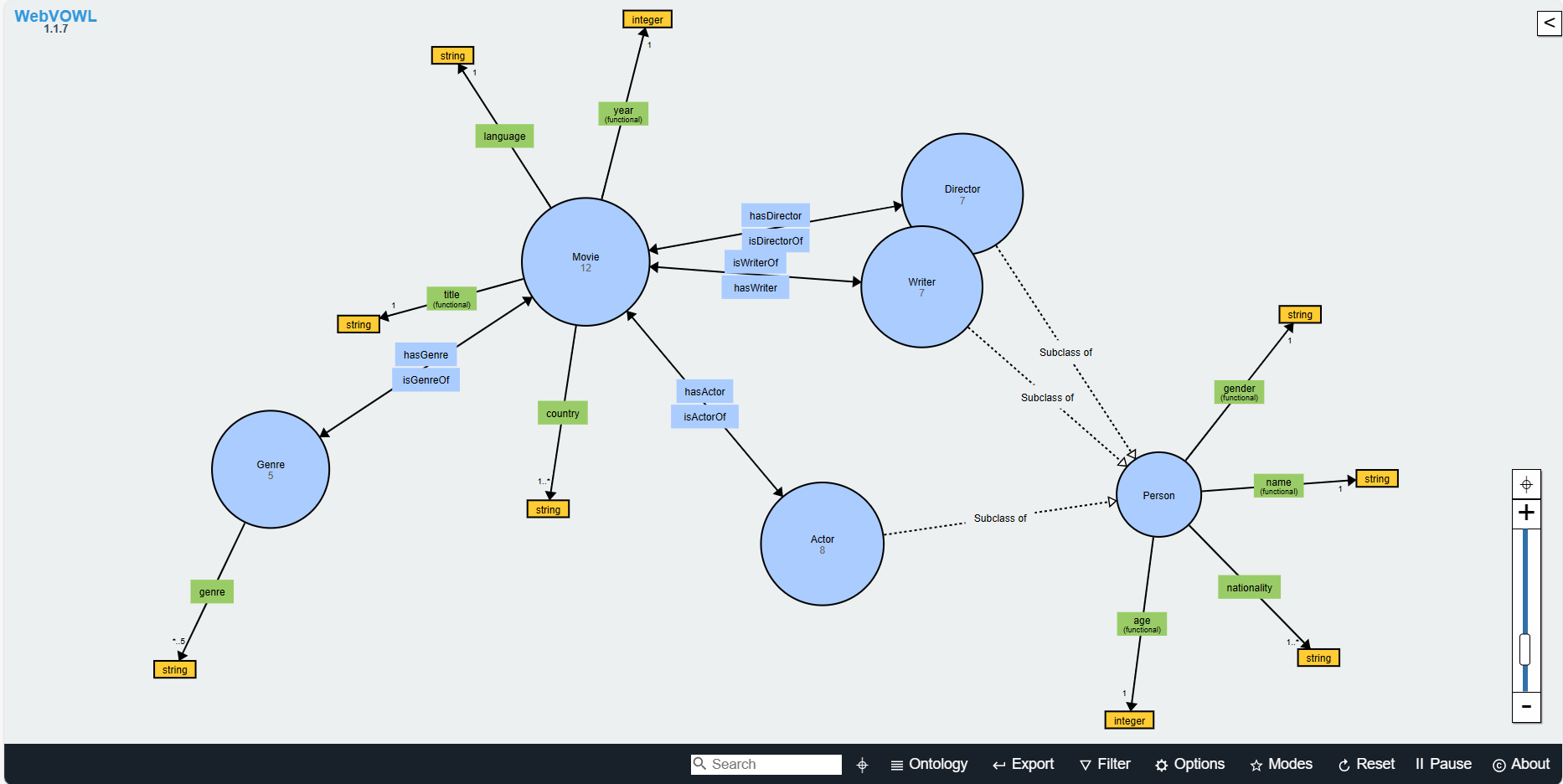
+ name exactly 1 xsd:string **| Functional**

+ nationality min 1 xsd:string

* **Disjointedness:**
* Genre
* Movie

Finally, view the ontology:

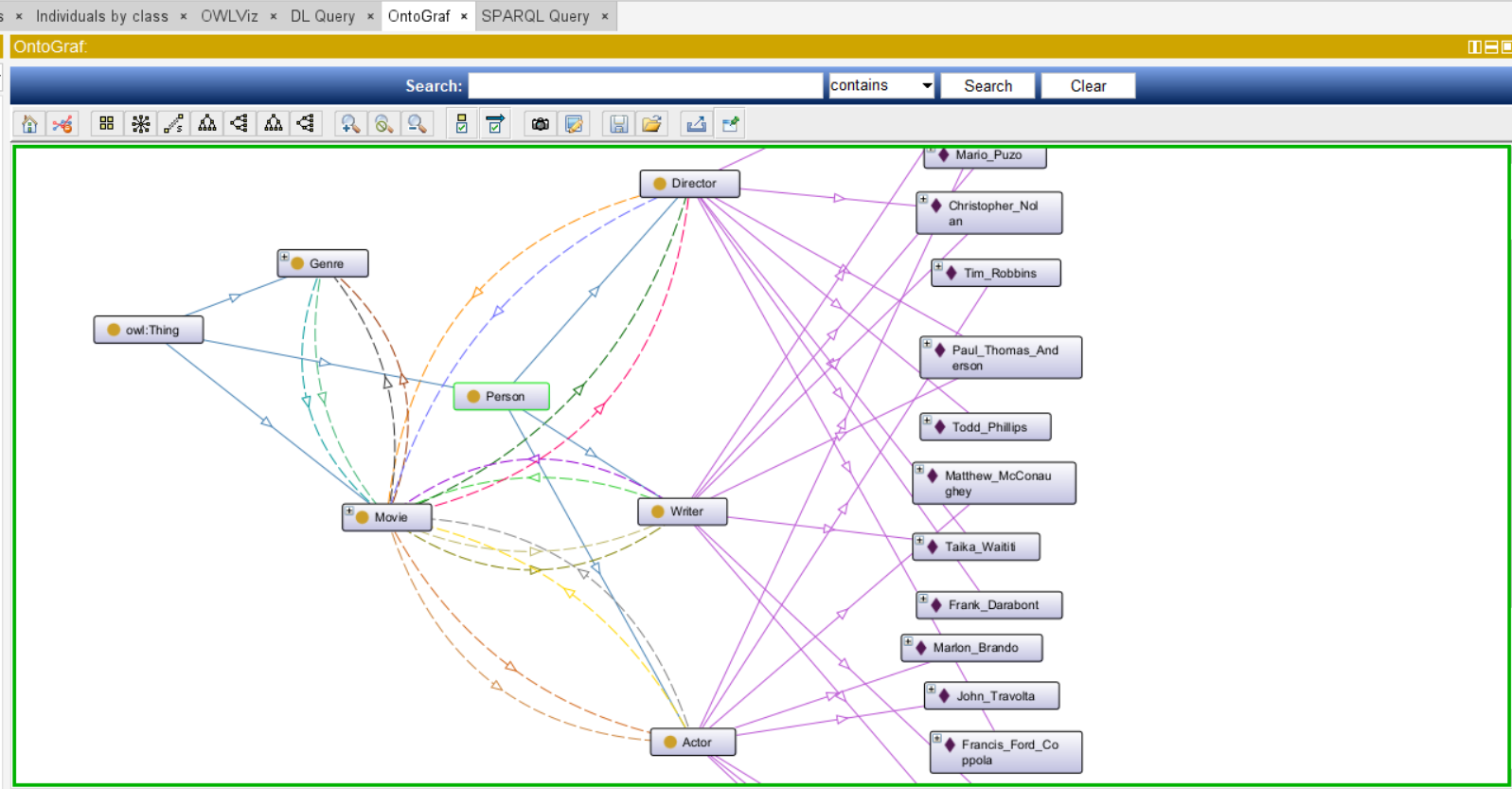




# Part II: Populating the Ontology

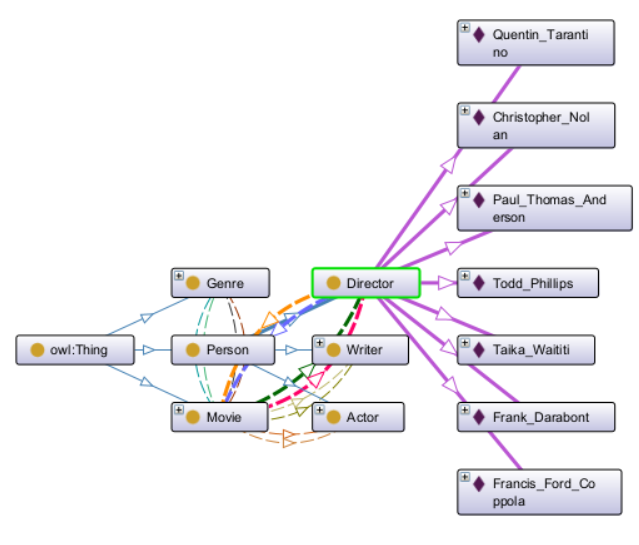
Now, having designed the ontology, it is time to populate it. Here are some individuals:

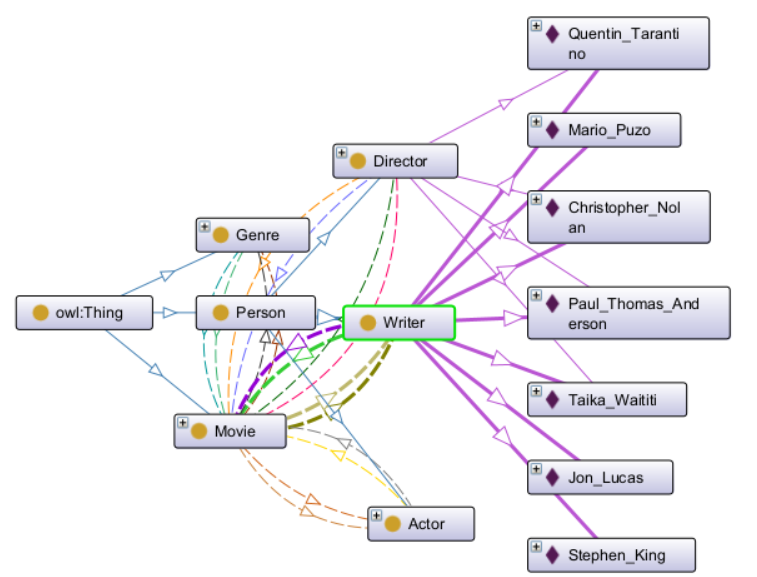
1. **Person (Actor, Director, Writer)**



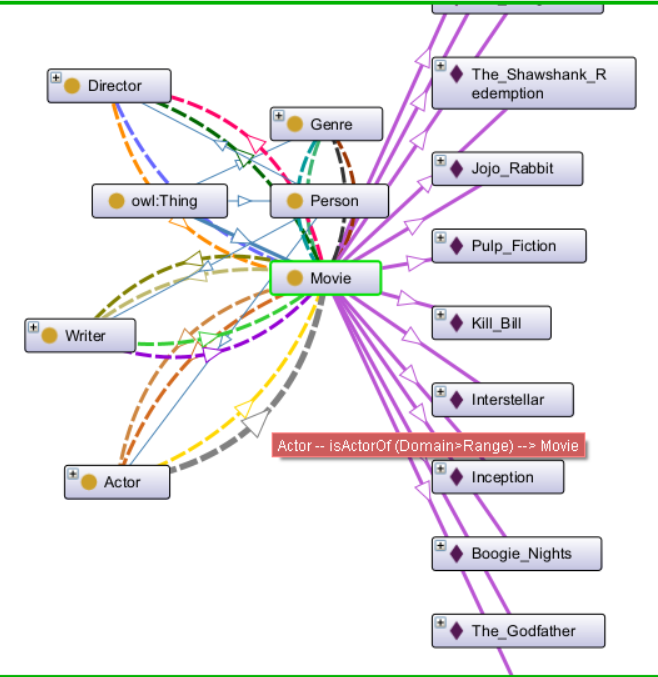
1. **Actor**



1. **Director**
2. **Writer**

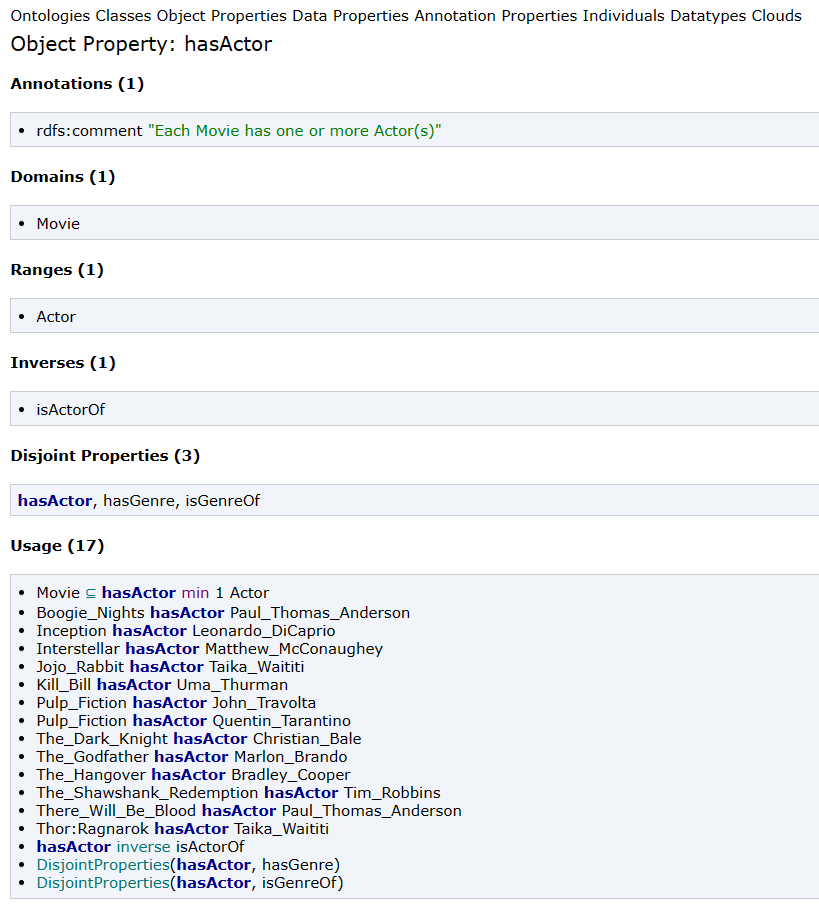


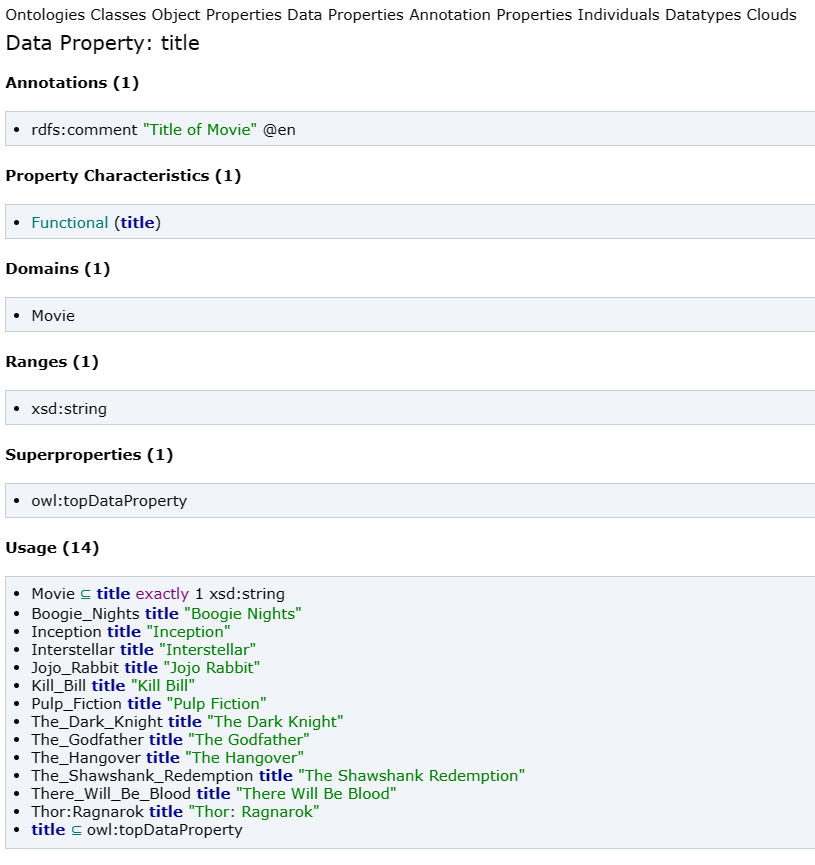
1. **Movie**



Finally, test the consistency of the ontology with PELLET reasoner (Turn on Pellet and press CTRL+R)

Below, the restrictions of some object properties and datatype properties are summarized:





# Part III: SPARQL Queries on the Ontology

Start querying your ontology with sparql, use different **types** and nests of queries**.** Each **type of query** is listed with its output:

1. **Query 1: A query that contains at least 2 Optional Graph Patterns and uses a FILTER with regular expressions**

Query 1: Extracts the names, ages, and nationalities of actors whose names start with letters A-M.

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT DISTINCT **?name** **?age** **?nationality**

WHERE {

?actor rdf:type ont:Actor.

?actor ont:name **?name**.

OPTIONAL {

?actor ont:age **?age**

}

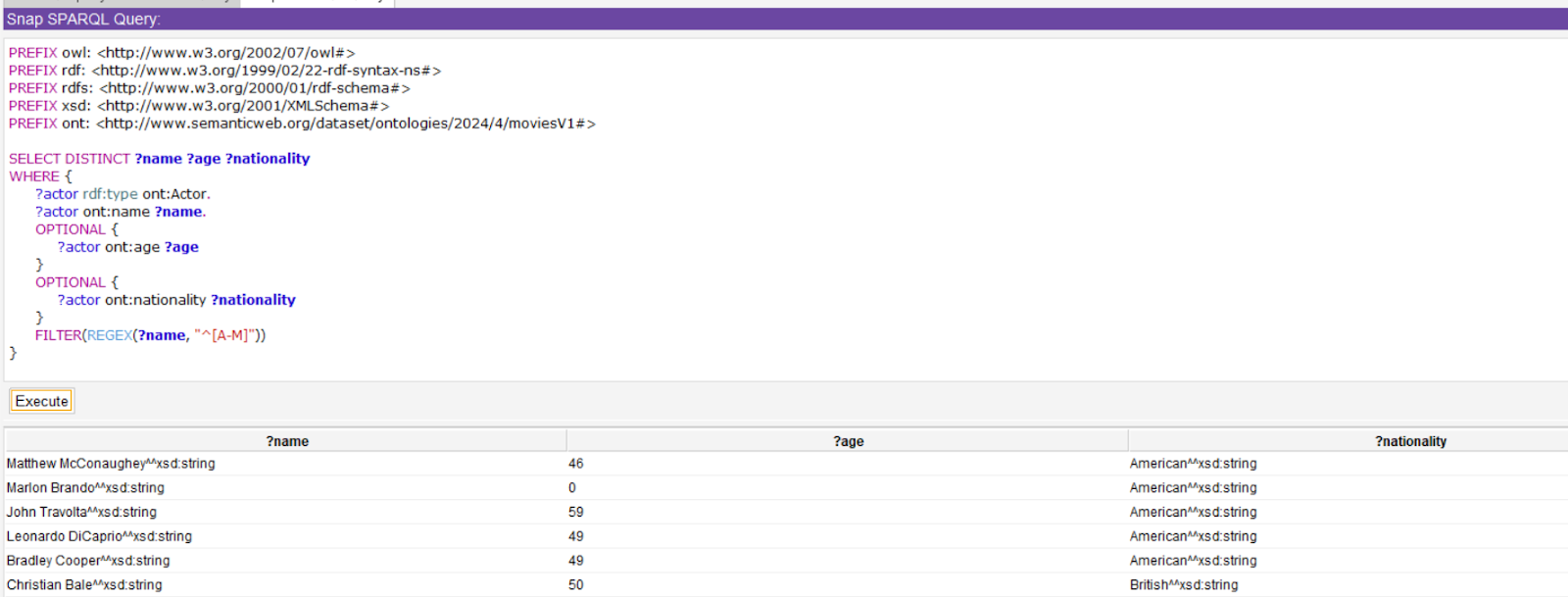
OPTIONAL {

?actor ont:nationality **?nationality**

}

FILTER(REGEX(**?name**, "^[A-M]"))

}



1. **Query 2: A query that contains at least 2 alternatives and conjunctions and uses aggregate functions (COUNT)**

Query 2: Retrieves titles, years, genre names, actor names, and director names for movies released before 2010 and categorized as Action or Thriller, along with the count of genres for each movie.

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT **?title** **?year** **?genre\_name** **?actor\_name** **?director\_name** (COUNT(?genre) AS **?genre\_count**)

WHERE {

?movie rdf:type ont:Movie.

?movie ont:title **?title**.

?movie ont:year **?year**.

?movie ont:hasGenre ?genre.

?genre ont:genre **?genre\_name**.

{

?movie ont:hasActor **?actor\_name**.

}

UNION

{

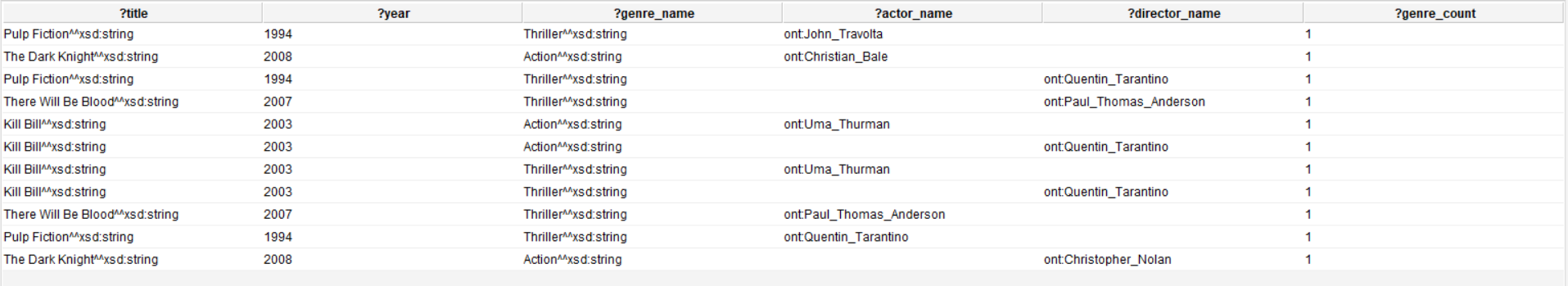
?movie ont:hasDirector **?director\_name**.

}

FILTER(**?year** < 2010 && (**?genre\_name** = "Action" || **?genre\_name** = "Thriller"))

}

GROUP BY **?title** **?year** **?genre\_name** **?actor\_name** **?director\_name**



1. **Query 3: A query that contains a CONSTRUCT query form with nested patterns**

Query 3: Constructs a new RDF graph containing individuals who are both actors and directors.

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

CONSTRUCT {

**?person** rdf:type ont:Actor;

rdf:type ont:Director .

}

WHERE {

{

**?person** rdf:type ont:Actor .

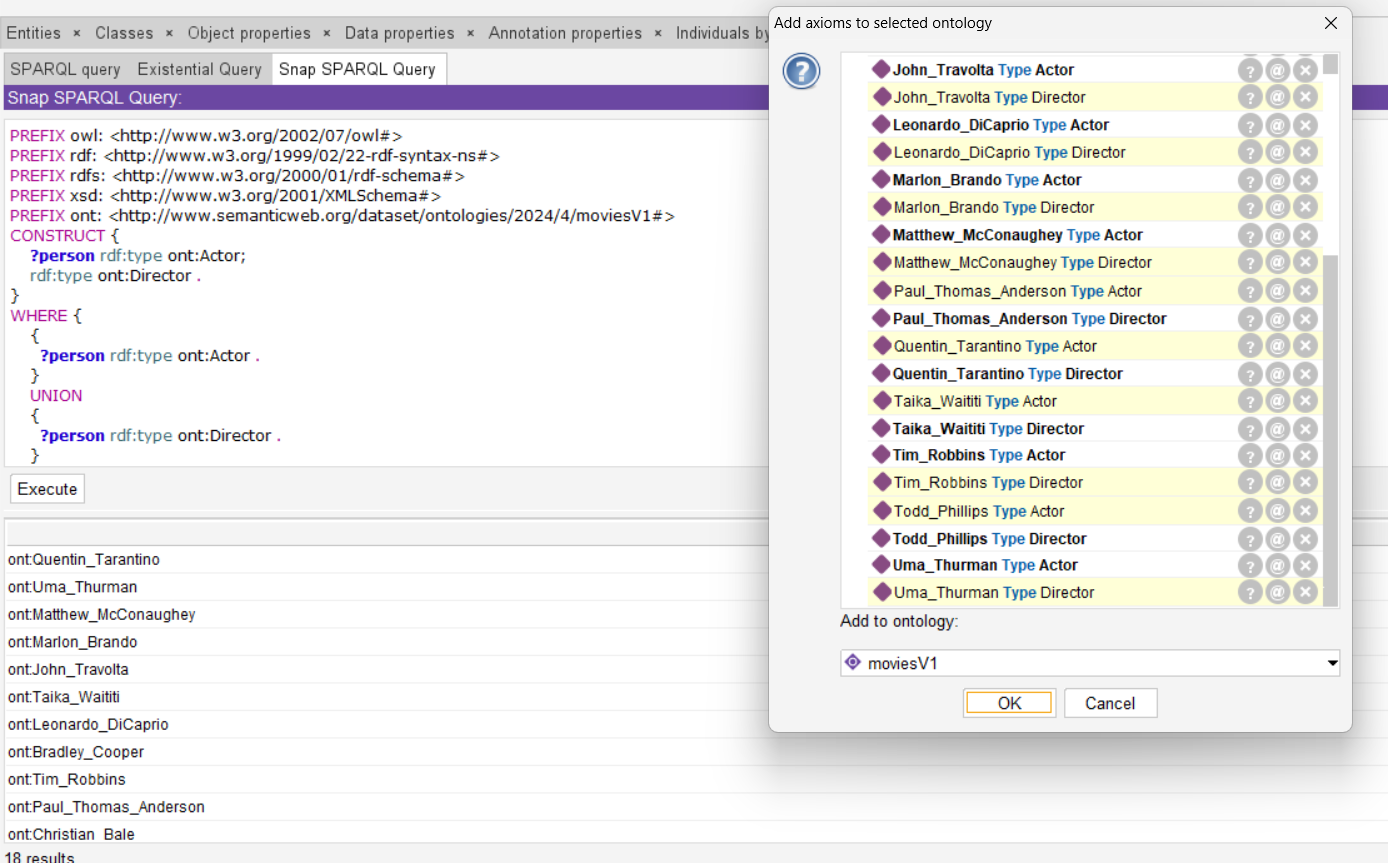
}

UNION

{

**?person** rdf:type ont:Director .

}

}

1. Query 4: Count movies with both "Comedy" and "Drama" genres, excluding those released before 2005.

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT (COUNT(?movie) AS **?count**)

WHERE {

?movie rdf:type ont:Movie ;

ont:hasGenre ?genre ;

ont:year ?year .

FILTER((?genre = ont:Comedy) || (?genre = ont:Drama) && ?year >= 2005)

}

A screenshot of a computer

Description automatically generated

1. **Query 5: A query that contains a FILTER with date comparison**

Query 5: Fetches titles and release dates of movies released after January 1, 2000.

Query 7: Fetches titles and release dates of movies released after January 1, 2000.

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT **?title** **?release\_date**

WHERE {

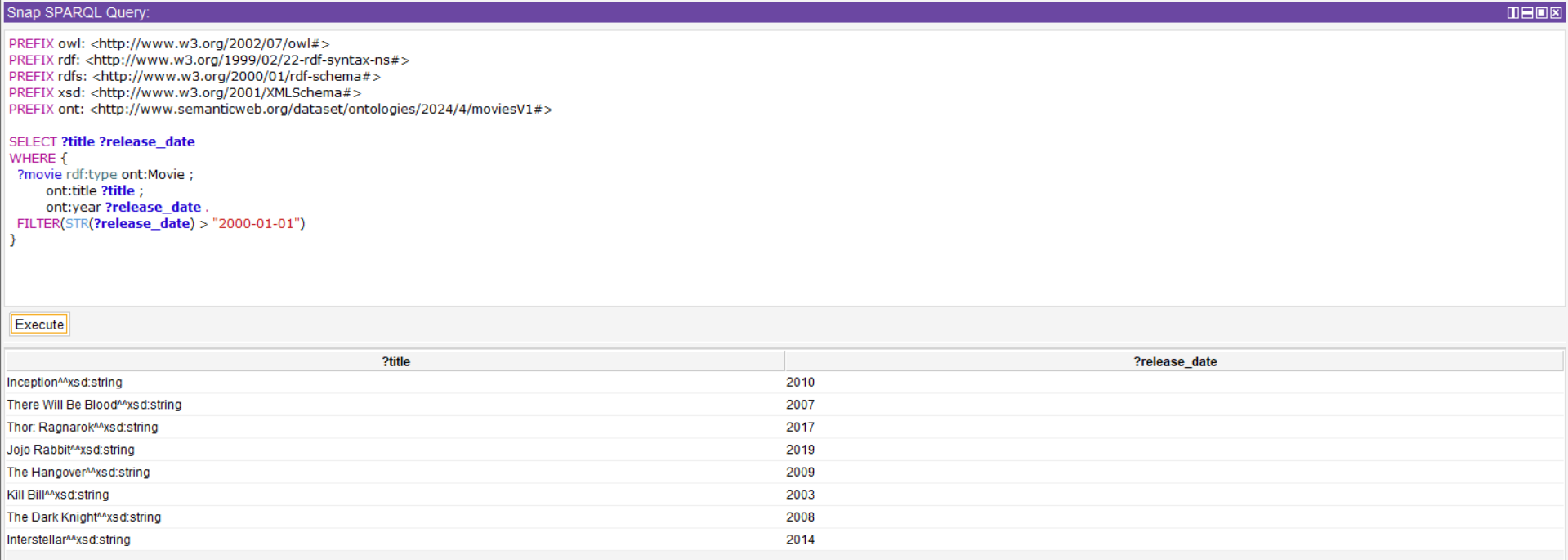
?movie rdf:type ont:Movie ;

ont:title **?title** ;

ont:year **?release\_date** .

FILTER(STR(**?release\_date**) > "2000-01-01")

}



1. **Query 6: Retrieve all movies written by writers who are also actors.**

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT **?movie** **?writer\_actor**

WHERE {

**?movie** rdf:type ont:Movie ;

ont:hasWriter ?writer ;

ont:hasActor **?writer\_actor** .

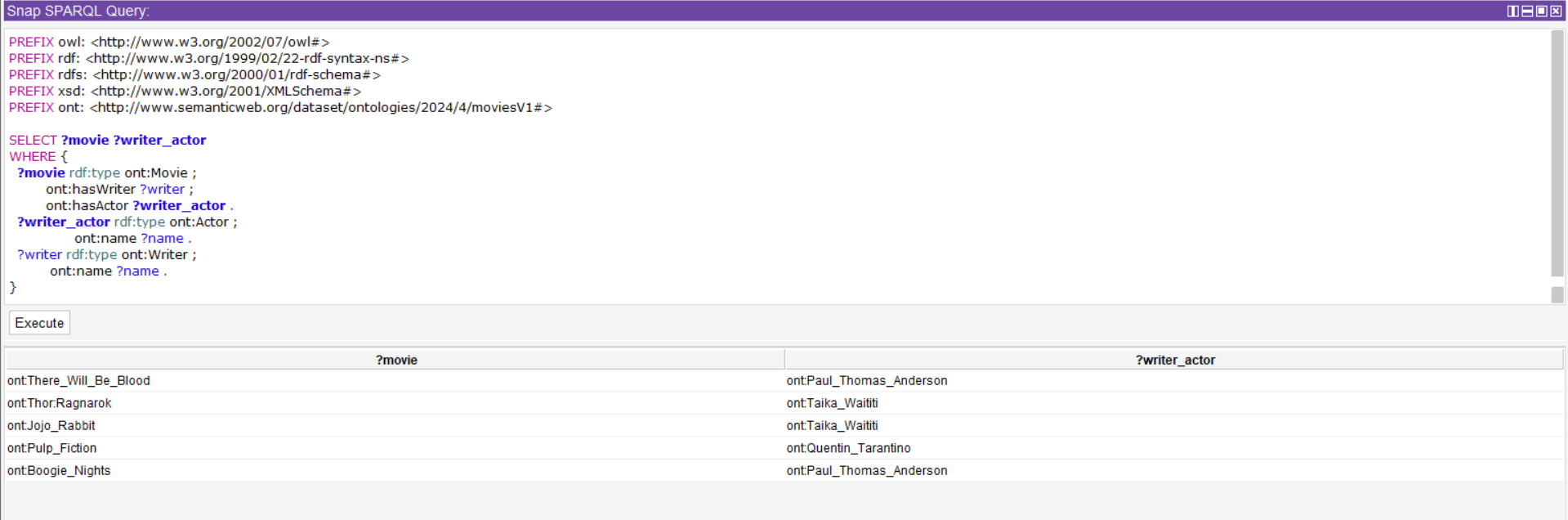
**?writer\_actor** rdf:type ont:Actor ;

ont:name ?name .

?writer rdf:type ont:Writer ;

ont:name ?name .

}



1. **Query 7: Retrieve all movies released in the 21st century along with their titles and genres.**

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT **?movie** **?title** **?genre**

WHERE {

**?movie** rdf:type ont:Movie ;

ont:title **?title** ;

ont:hasGenre **?genre** ;

ont:year ?year .

FILTER(?year >= 2000 && ?year < 2100)

}

A screenshot of a computer

Description automatically generated

1. **Query 8: List all actors who have appeared in movies directed by themselves, along with their names.**

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT DISTINCT **?actor ?name**

WHERE {

**?actor** rdf:type ont:Actor ;

         ont:name **?name** ;

         ont:isActorOf **?movie** .

**?movie** ont:hasDirector **?actor** .

}

A screenshot of a computer

Description automatically generated

1. **Query 9: A query that contains an ASK query form with negation (MINUS)**

Query 9: Checks if there are movies without the genre "Comedy."

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

ASK

WHERE {

**?movie** rdf:type ont:Movie .

  MINUS {

**?movie** ont:hasGenre **?genre** .

    FILTER(**?genre** = "Comedy")

  }

}

A screenshot of a computer code

Description automatically generated

1. **Query 10: *A query that contains a DESCRIBE query form with nested properties***

Query 10: Describes information about the Actor class in the ontology.

DESCRIBE <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#Actor>

A screenshot of a computer code

Description automatically generated

## Additional queries tested on Jena

1. ***Test DESCRIBE query.***

query = """

DESCRIBE <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#Actor>

"""

Execute the query and print results

result = g.query(query)

# Print the result in Turtle format

print(result.serialize(format="turtle").decode())

**A screenshot of a computer code

Description automatically generated**

1. ***Test ASK query***

query = """

ASK

WHERE {

?movie rdf:type ont:Movie .

MINUS {

?movie ont:hasGenre ?genre .

FILTER(?genre = "Comedy")

}

}

"""

#Execute the query and print results

result = g.query(query)

#Print the result in Turtle format

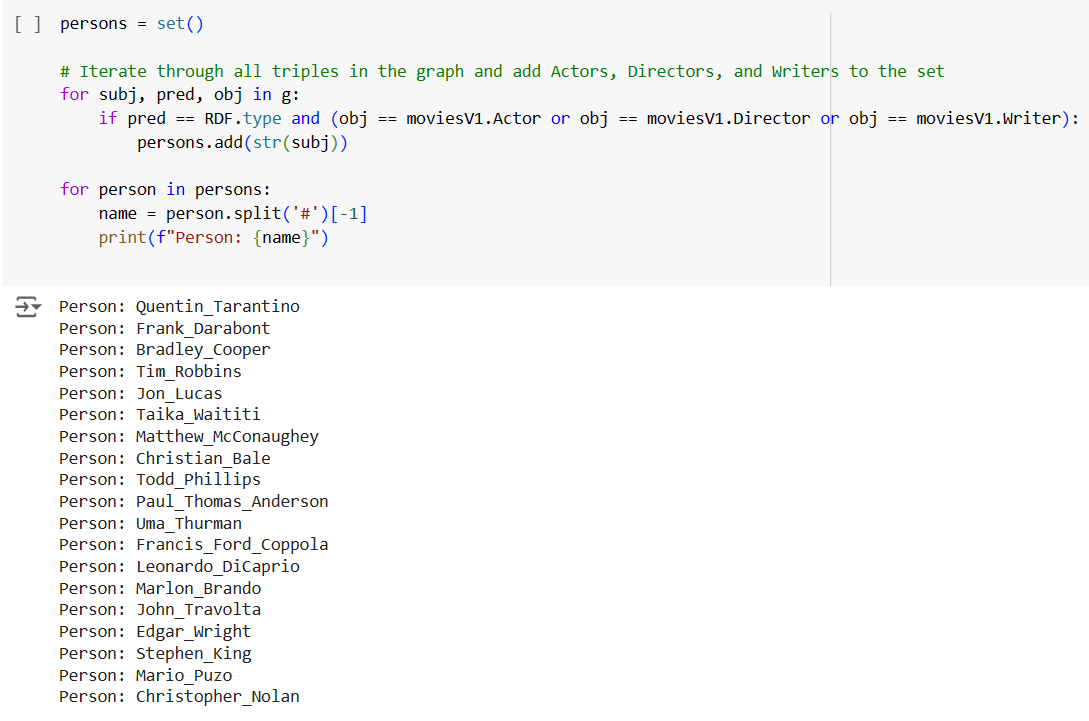
print("Result:",result.askAnswer)

A screenshot of a computer code

Description automatically generated

# Part IV: Manipulating the ontology using Jena

1. **Display all Persons without Query or Inference**

****

1. **Display all Persons by Query only**

**A screenshot of a computer

Description automatically generated**

1. **Display all Persons by Inference only**

**A screenshot of a computer

Description automatically generated**

1. **Display Movie properties if Movie Exists**

**A screen shot of a computer program

Description automatically generated**

1. **Add a Rule for a New Class, ActorDirector**

**A screenshot of a computer code

Description automatically generated**

1. **Add New Rules**

**A screen shot of a computer code

Description automatically generated**

**A screenshot of a computer program

Description automatically generatedA screen shot of a computer

Description automatically generated**

# Part V: Python application

Mainly, this application allows the user to enter included and excluded actor(s), director(s) or genre(s) to display specific movies.

## Data Flow Diagram Level 0

A yellow rectangular sign with black text

Description automatically generated

## Data Flow Diagram Level 1

A screenshot of a computer screen

Description automatically generated

## Data Flow Diagram Level 2

A screenshot of a phone

Description automatically generated

## Environment Dependencies and Libraries

* Node.js >= 10.4
* Python >= 3.9
* Tkinter
* rdflib == 7.0.0

### Setting Up a SPAQRL Endpoint

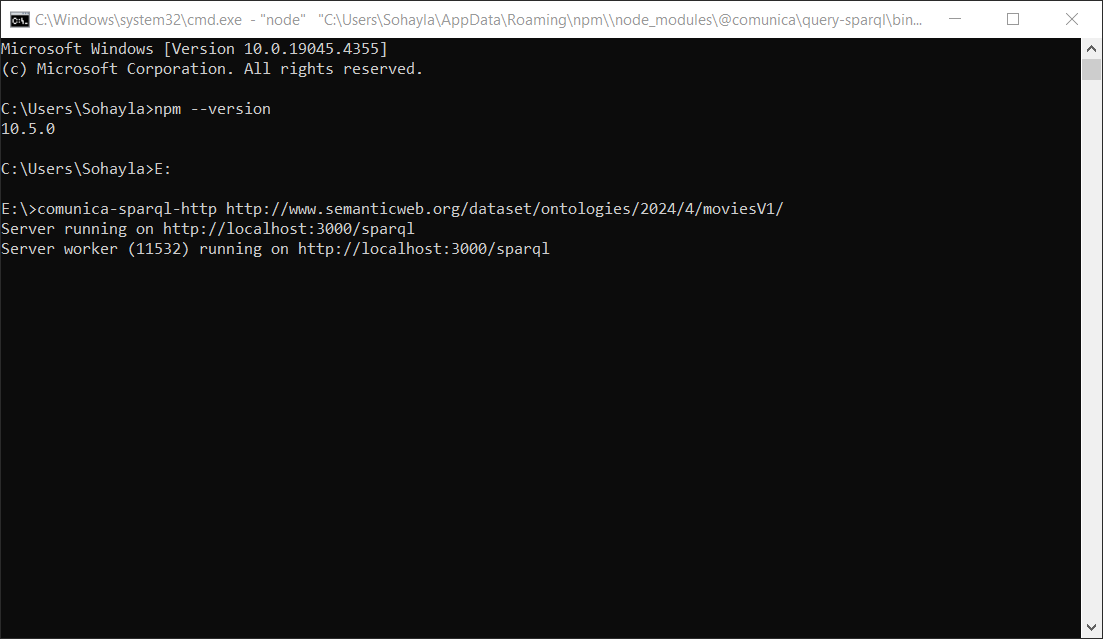
Http requests will be used in the process of parsing resource description files (rdf) and returning results of sparql query. To do that, a sparql endpoint will have to be set up. Comunica SPARQL is used to set up <http://localhost:3000/sparql> on your device. If not set up properly, a connection error will be shown.

In your cmd:

$ npm install -g @comunica/query-sparql

After the installation:

$ comunica-sparql-http <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1/>



For more information:

[Comunica – Setting up a SPARQL endpoint](https://comunica.dev/docs/query/getting_started/setup_endpoint/)

## Desktop Application Interface

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

## Test Cases

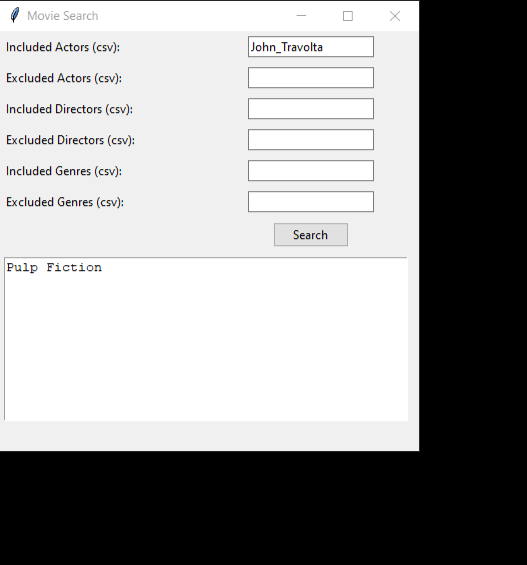
|  |  |  |
| --- | --- | --- |
| **Test Case** | **Description** | **Result** |
| Test Null value | Check null filled values | Success |
| Test Single value | Test include and exclude actor | Success |
| Test Single value | Test include and exclude director | Success |
| Test Single value | Test include and exclude genre | Success |
| Test Multiple values | Test same value for include and exclude actor | Success |
| Test Multiple values | Test 2 included values | Success |
| Test Multiple values | Test 1 included and 1 excluded values | Success |
| Test Multiple values | Test comma separated values(w/o space) | Success |
| Test Multiple values | Test comma separated values(w/ space) | Fail |
| Test All values | Test all array separated values | Success |
| Test Exception Cases | Test Edgar\_Wright case | Success |

1. **Test Null value:**

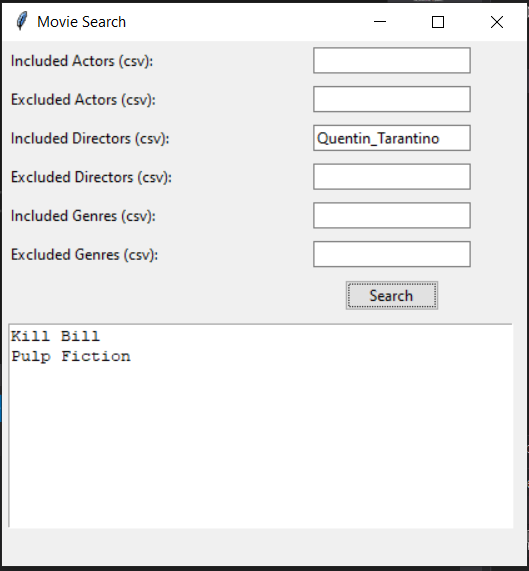
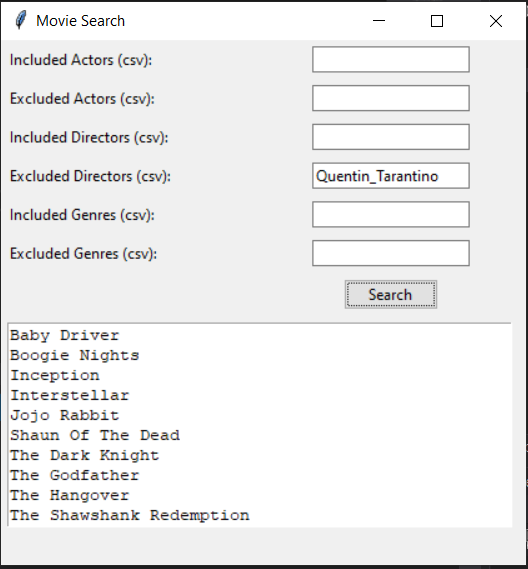
**A screenshot of a computer

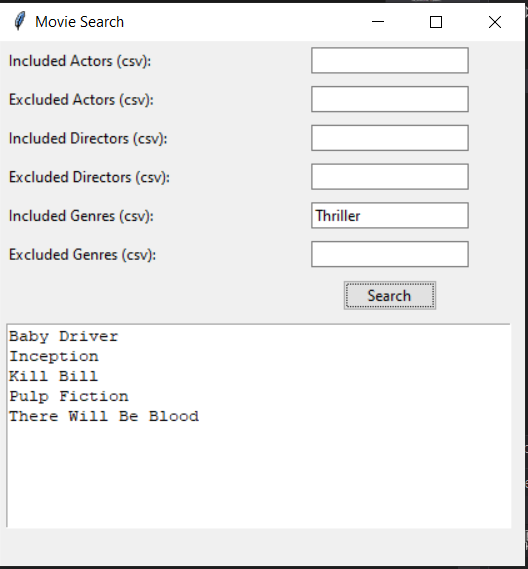
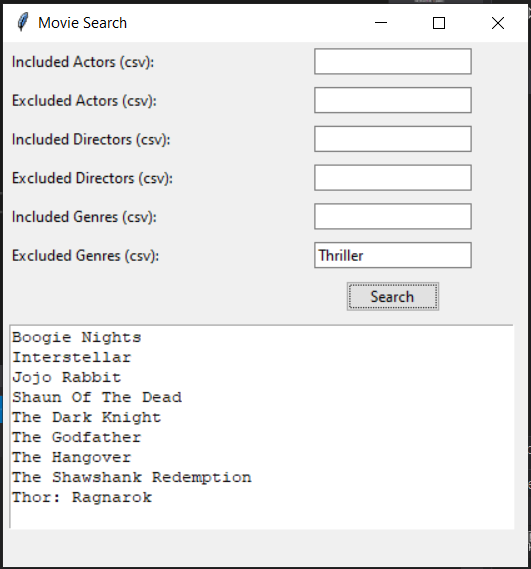
Description automatically generated**

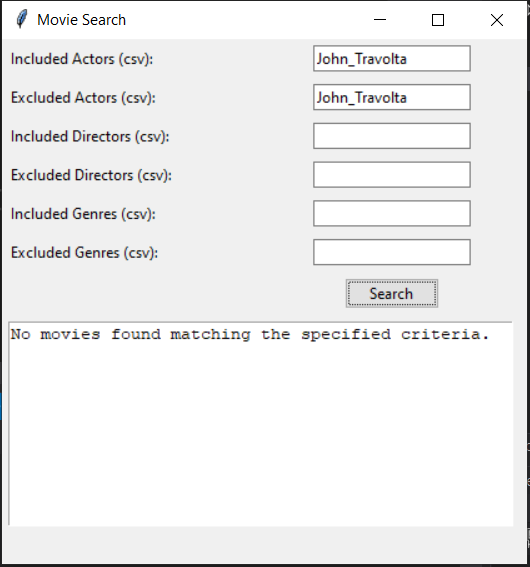
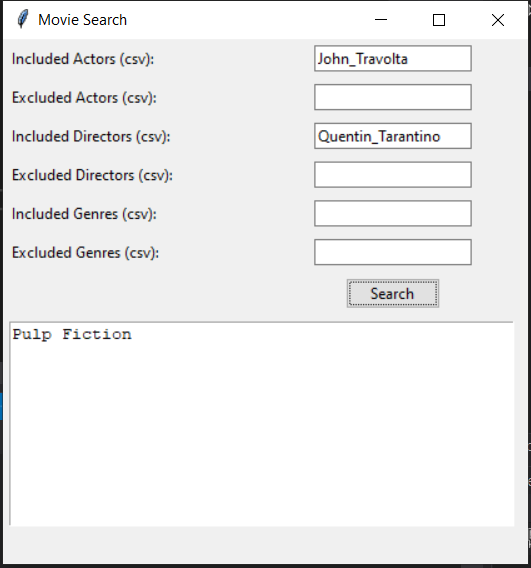
1. **Test Single value:**

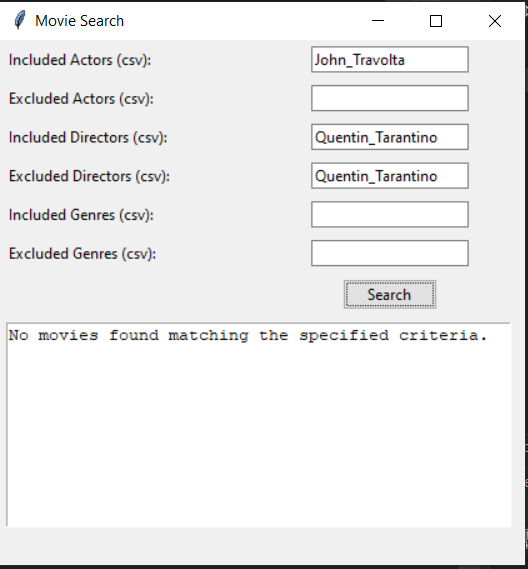
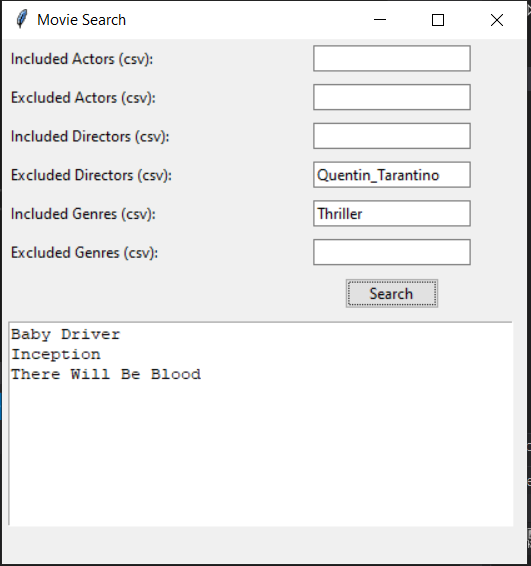
** A screenshot of a computer

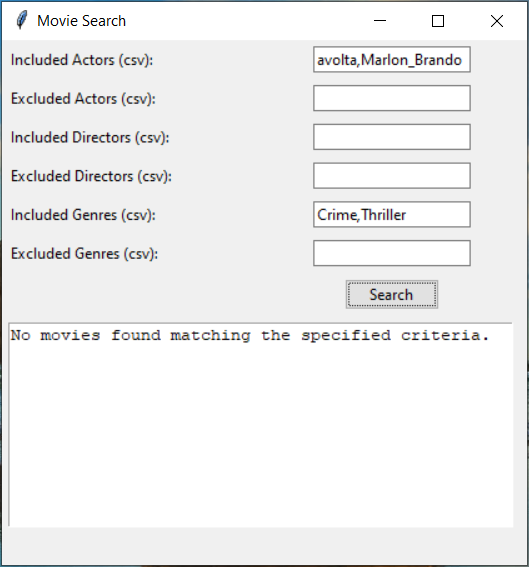
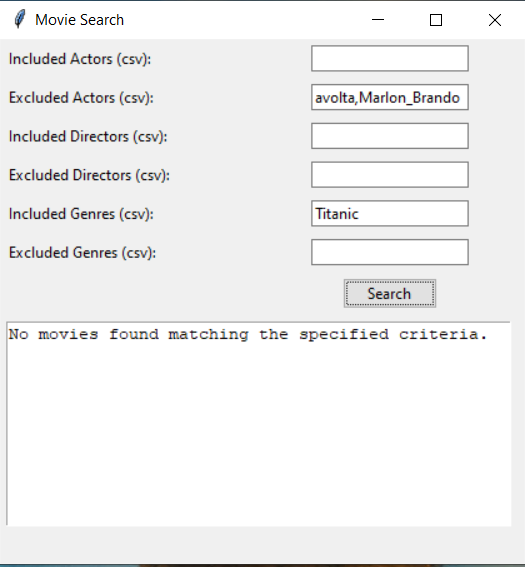
Description automatically generated**

** **

** **

1. **Test Multiple values:**

** **

** **

1. **A screenshot of a computer

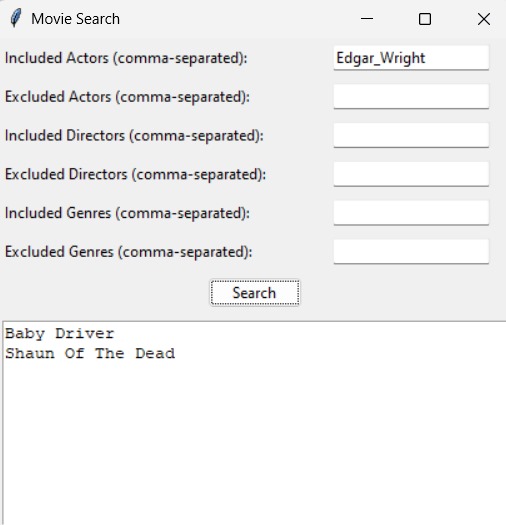
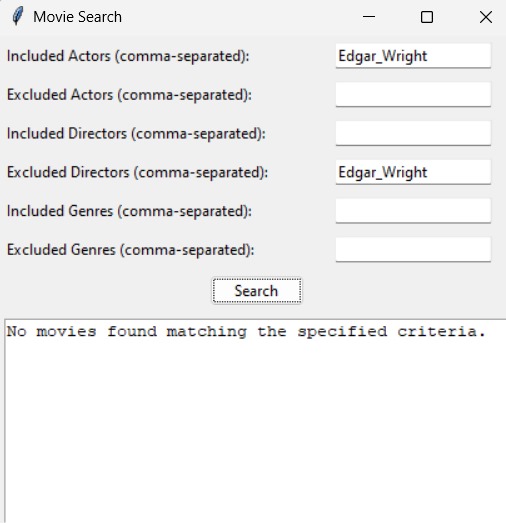
   Description automatically generatedA screenshot of a computer

   Description automatically generatedTest All values:**

**A screenshot of a computer

Description automatically generated**

1. **Test Exception Cases: Edgar Wright is an Actor and Director. Test inclusion and exclusion between classes.**

** **