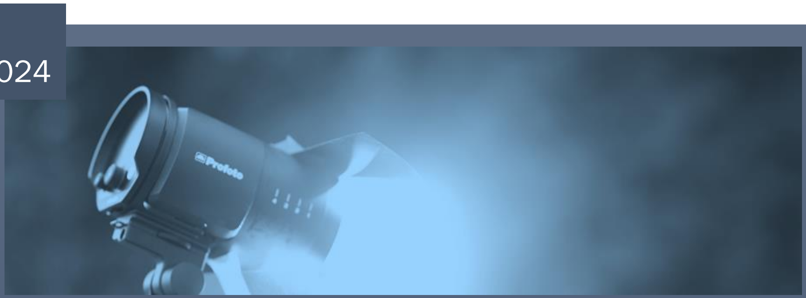
2024



A Design and Implementation for Movies Ontology

CSE488: ONTOLOGIES AND THE SEMANTIC WEB

AHMED ASHOUR	20P4222
HABIBA YASSER ABDELHALIM	20P3072
NADINE HISHAM	20P9880
SOHAYLA IHAB ABDELMAWGOUD AHMED HAMED	19P7343

REPOSITORY: PerfectionistAF/Movies_Ontology (github.com)

Contents

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:**
   a) Movie: defines basic Movie class
   b) Genre: defines basic Genre class
   c) Person: defines basic Person class
   d) Actor: basic Actor class is a subclass of Person class
   e) Writer: basic Writer class is a subclass of Person
   f) Director: basic Director class is a subclass of Person
2) **Object Property:**
   a) hasGenre: Each Movie has one or more instances of class Genre
   b) hasActor: Each Movie has at least one instance of class Actor
   c) hasWriter: Each Movie has at least one instance of class Writer
   d) hasDirector: Each Movie has at least one instance of class Director
   e) isGenreOf: Inverse of hasGenre
   f) isActorOf: Inverse of hasActor
   g) isDirectorOf: Inverse of hasDirector
   h) isWriterOf: Inverse of hasWriter
3) **Data Property:**
   a) name: Name of Person
   b) age: Age of Person
   c) nationality: Nationality(s) of Person
   d) gender: Gender of Person
   e) genre: Genre(s) of Movie
   f) title: Title of Movie
   g) country: Country(s) of Production of Movie
   h) language: Language(s) of Movie
   i) 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).
14) Only 1 type of language tags were used, the default English tags. There weren't explicitly declared.
15) Age 0 means the Person is deceased.
16) A Person (Actor, Director, Writer) can have multiple nationalities.

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

a) 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

b) 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

c) 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

d) 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

e) 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

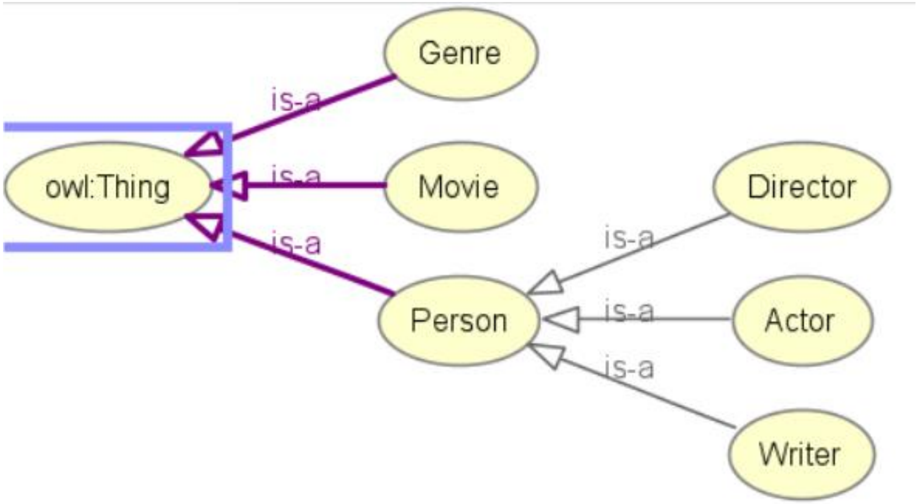
- **Disjointedness:**
- Genre
- Movie

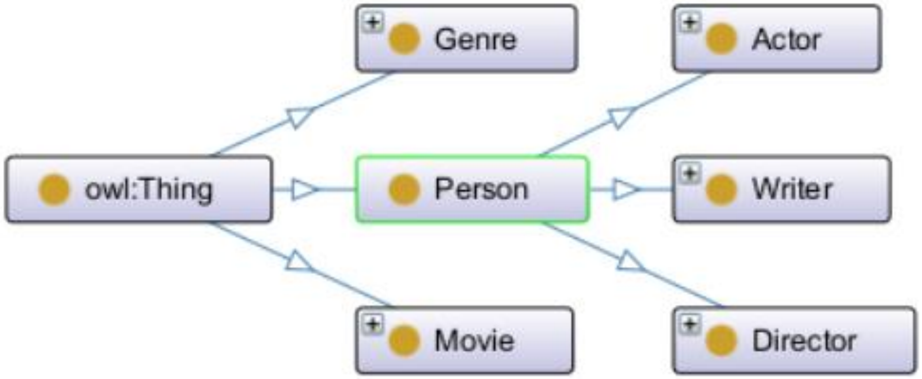
f) Writer

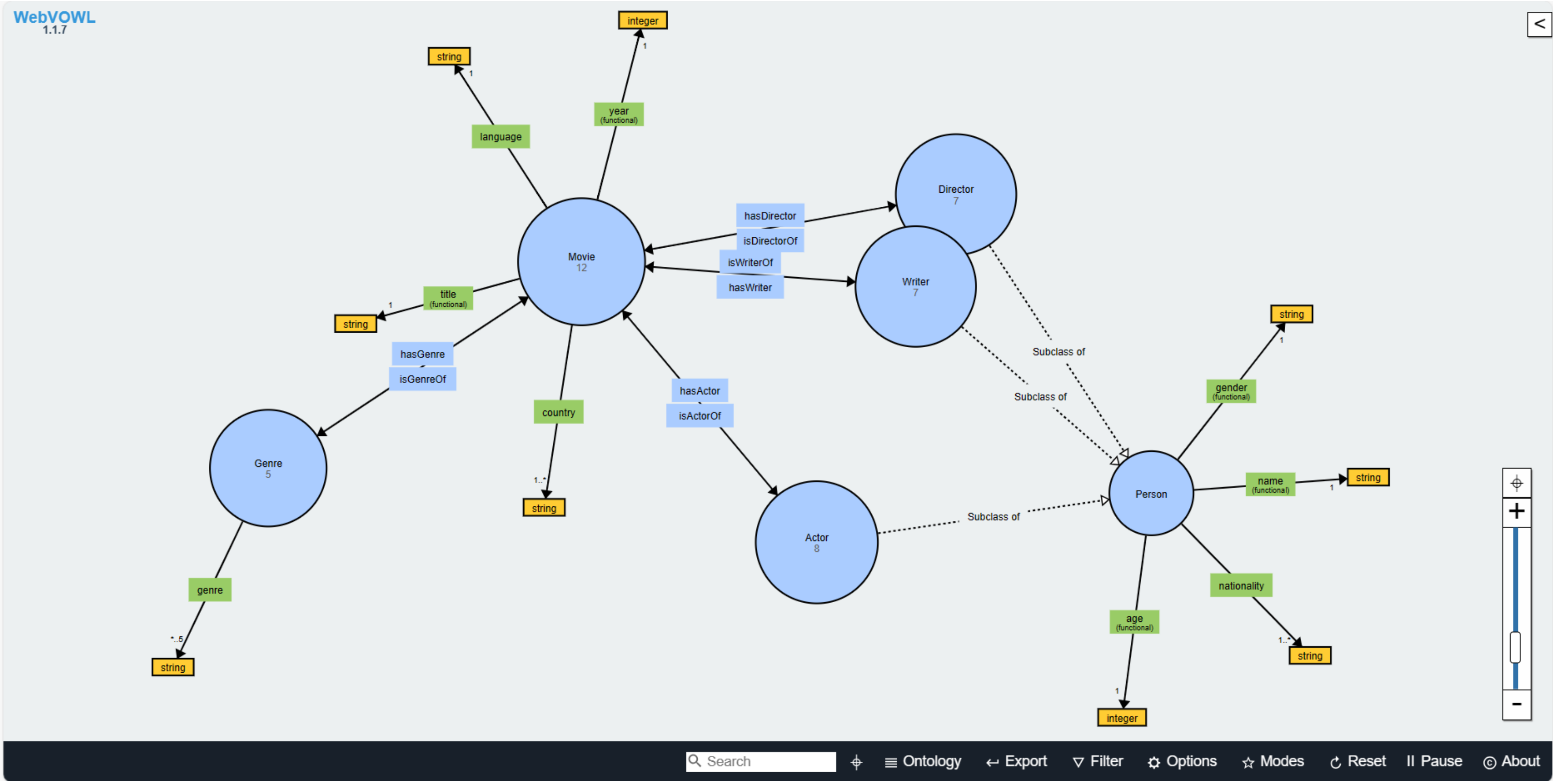
+ **Properties:**
+ **Subclass of Person**
+ isWriterOf min 1 Movie **| Disjoint:** hasGenre, isGenreOf **| Inverse:** hasWriter **| Domain: Writer | Range: Movie**
+ age exactly 1 xsd:integer **| Functional | age some xsd:integer[>= 0]**
+ gender exactly 1 xsd:string **| Functional | (gender value "Female") 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:



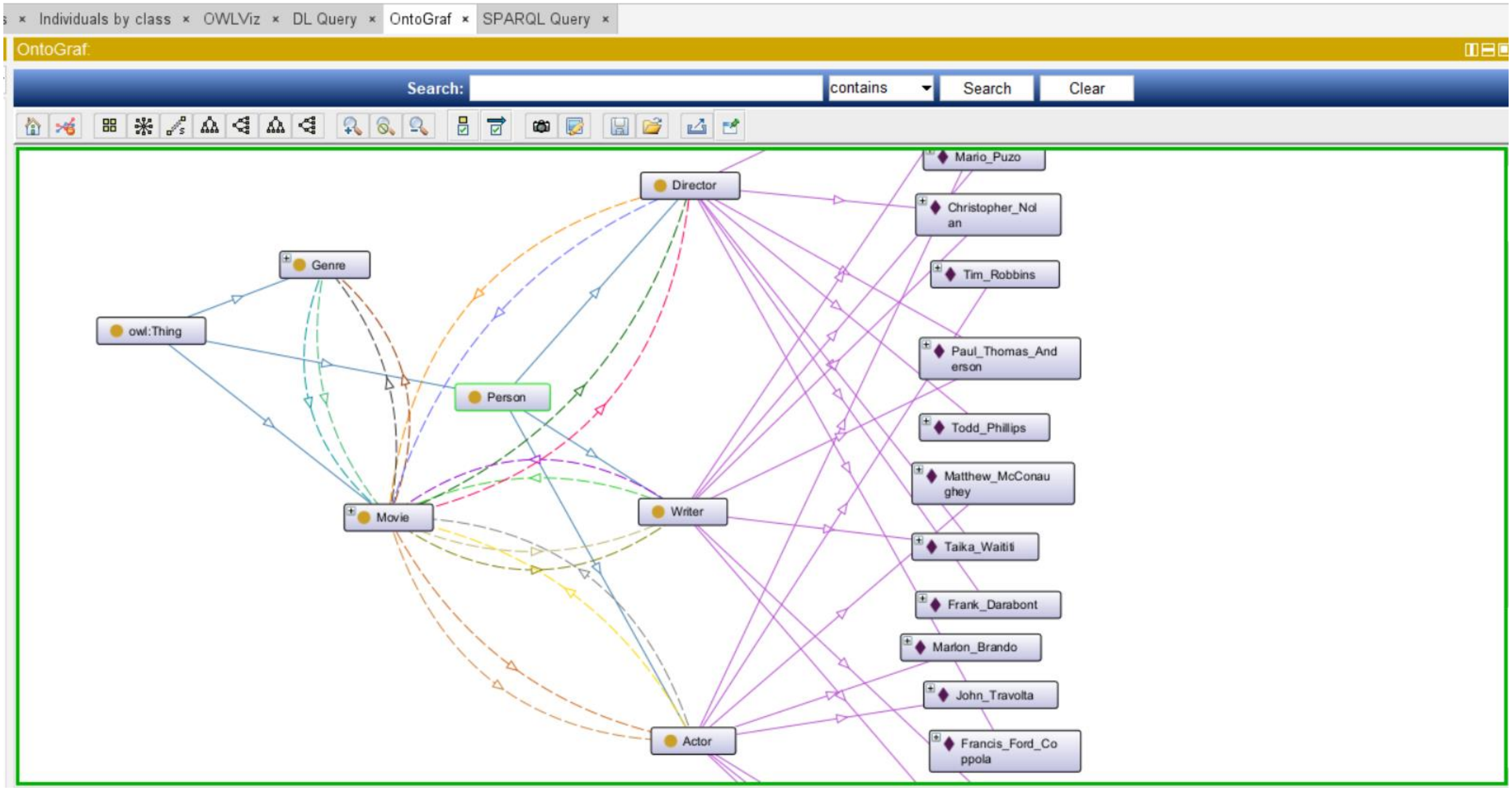


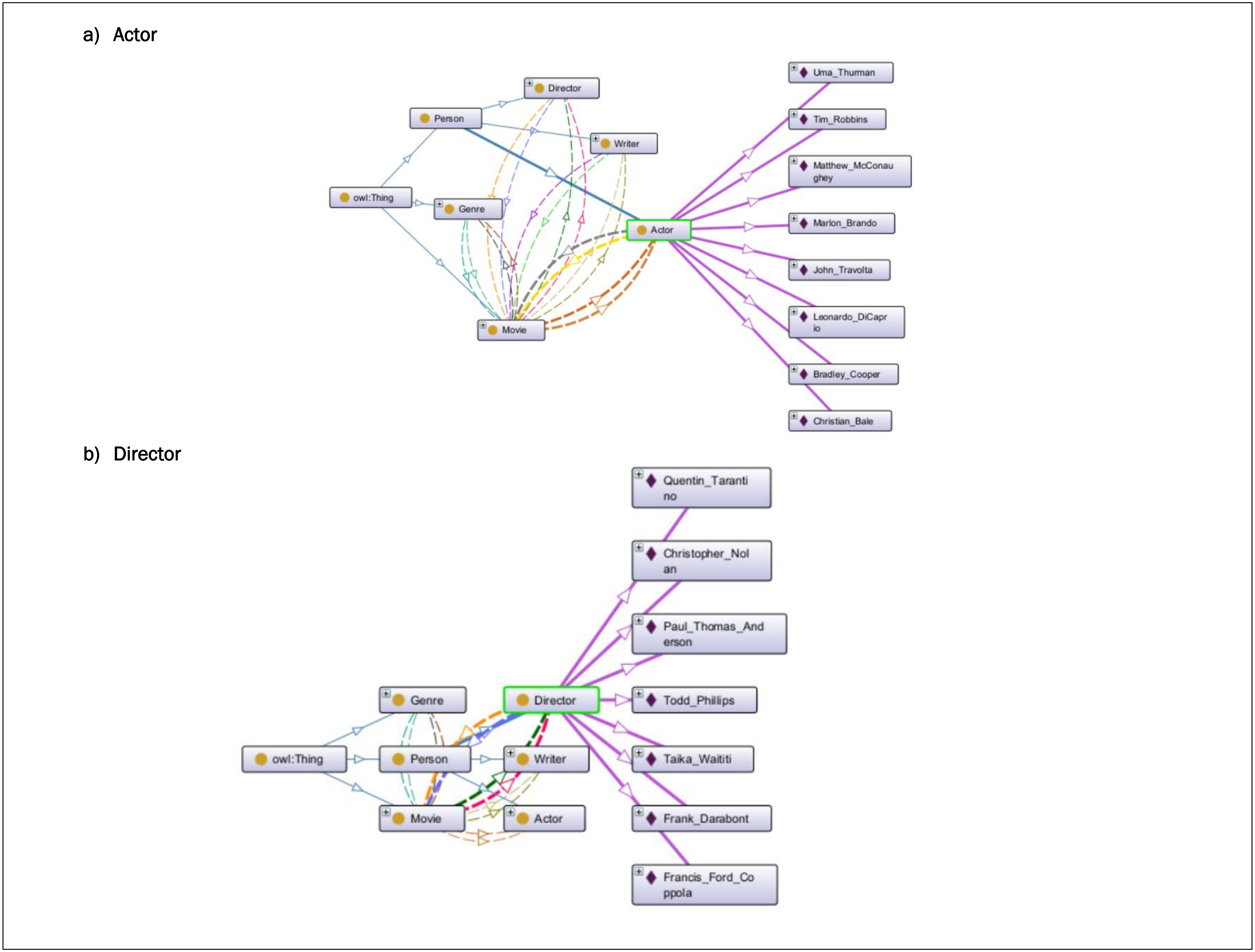

WebVOWL
1.1.7
<
integer
1
year
(functional)
string
1
language
Movie
12
Director
7
hasDirector
isDirectorOf
isWriterOf
hasWriter
Writer
7
title
(functional)
1
string
hasGenre
isGenreOf
Genre
5
country
1..*
string
hasActor
isActorOf
Actor
8
Subclass of
Subclass of
Subclass of
Person
string
1
gender
(functional)
name
(functional)
1
string
nationality
1..*
string
age
(functional)
1
integer
genre
*..5
string
Search
Ontology
Export
Filter
Options
Modes
Reset
Pause
About
+
-

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)**

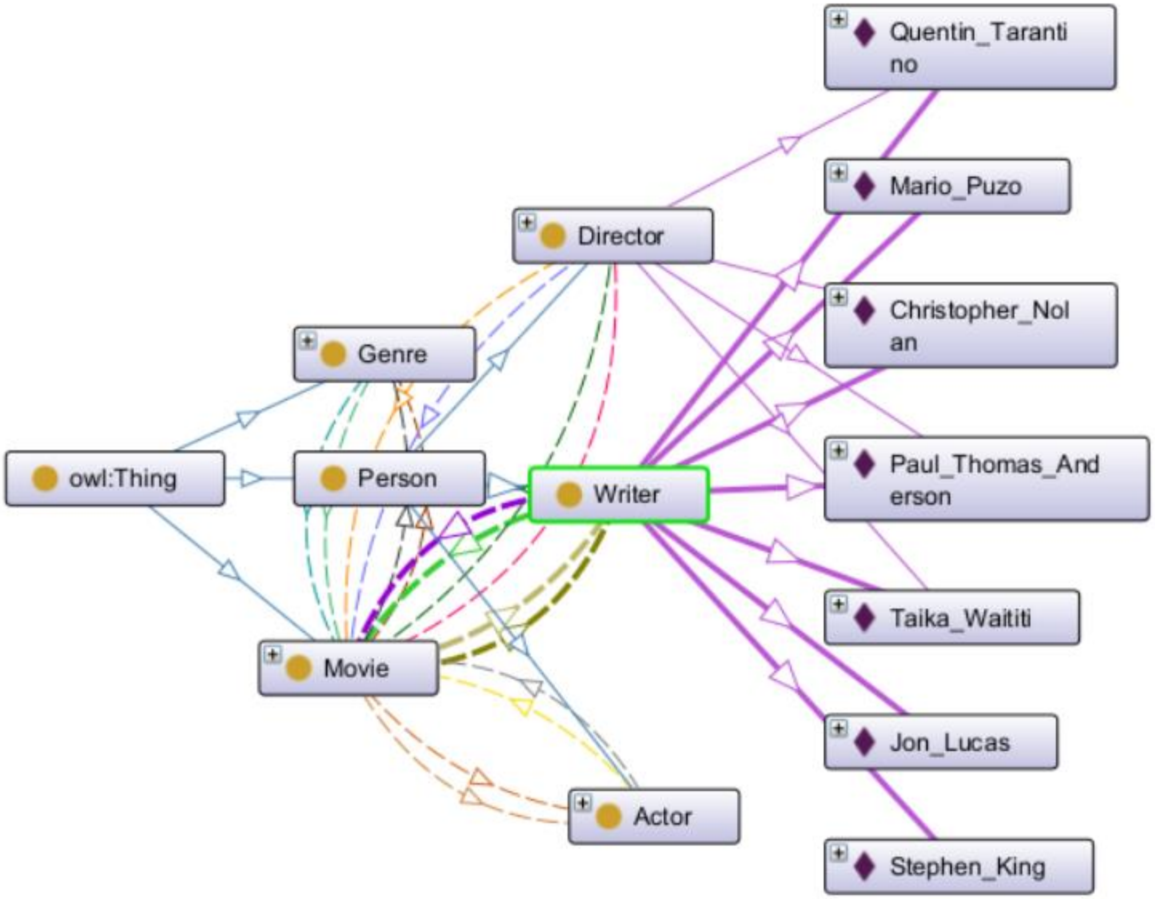




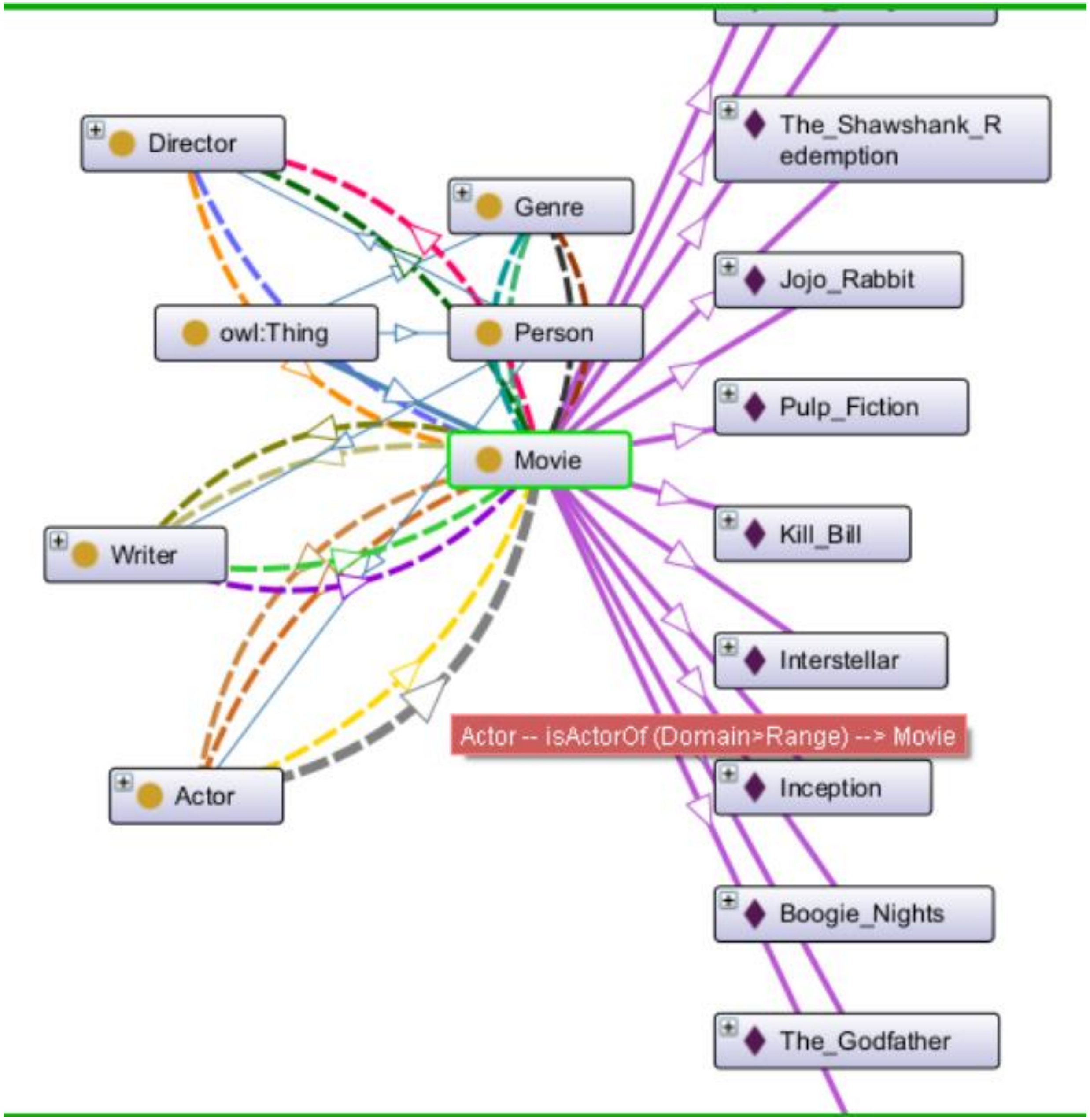
a) Actor

b) Director

c) Writer

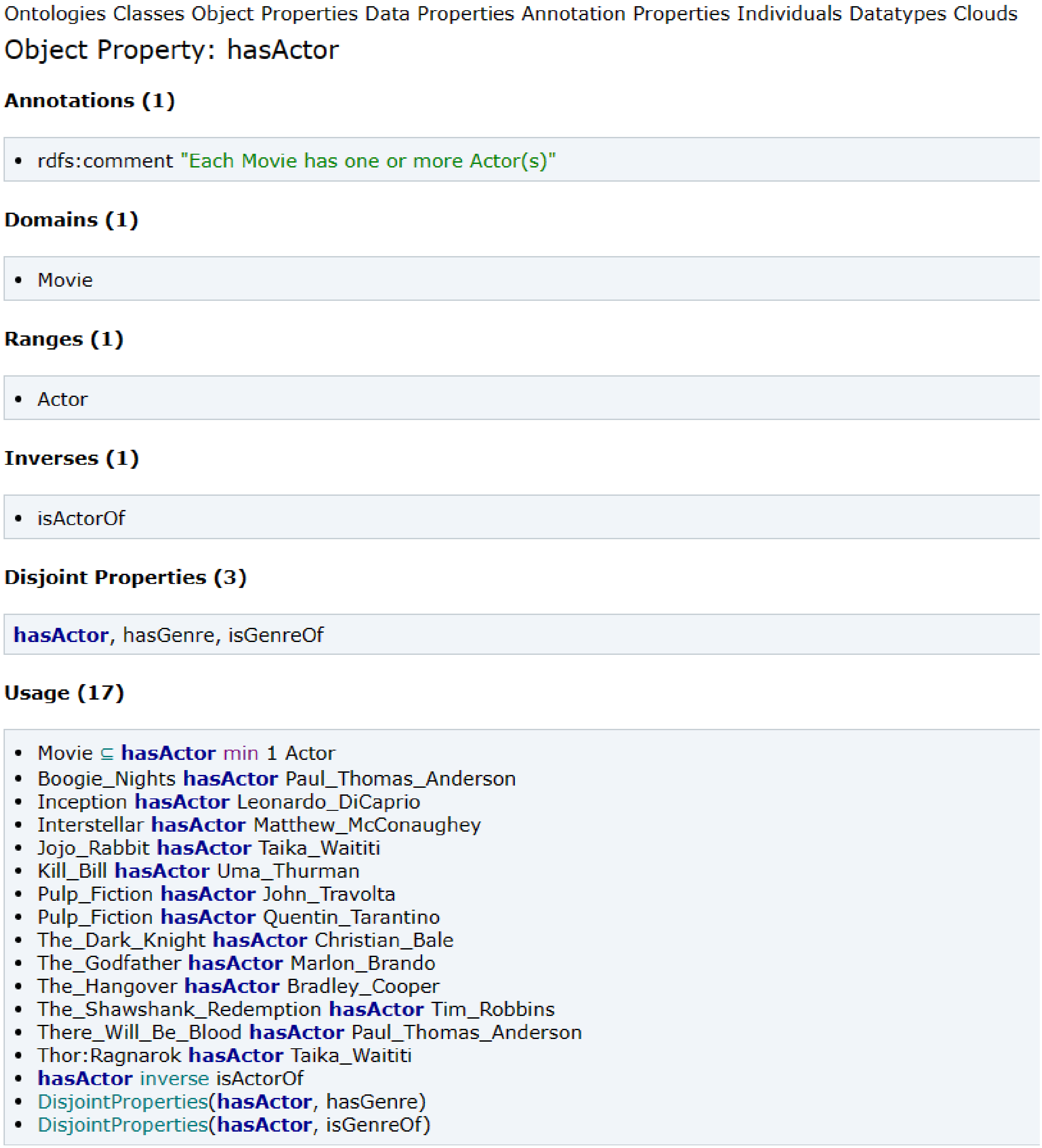

Quentin_Tarantino
Mario_Puzo
Director
Christopher_Nolan
Genre
owl:Thing
Person
Writer
Paul_Thomas_Anderson
Taika_Waititi
Movie
Jon_Lucas
Actor
Stephen_King

2) Movie


Director
Genre
owl:Thing
Person
Movie
Writer
Actor
The_Shawshank_R
edemption
Jojo_Rabbit
Pulp_Fiction
Kill_Bill
Interstellar
Actor -- isActorOf (Domain>Range) --> Movie
Inception
Boogie_Nights
The_Godfather

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:



Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Object Property: hasActor

Annotations (1)

- rdfs:comment "Each Movie has one or more Actor(s)"

Domains (1)

- Movie

Ranges (1)

- Actor

Inverses (1)

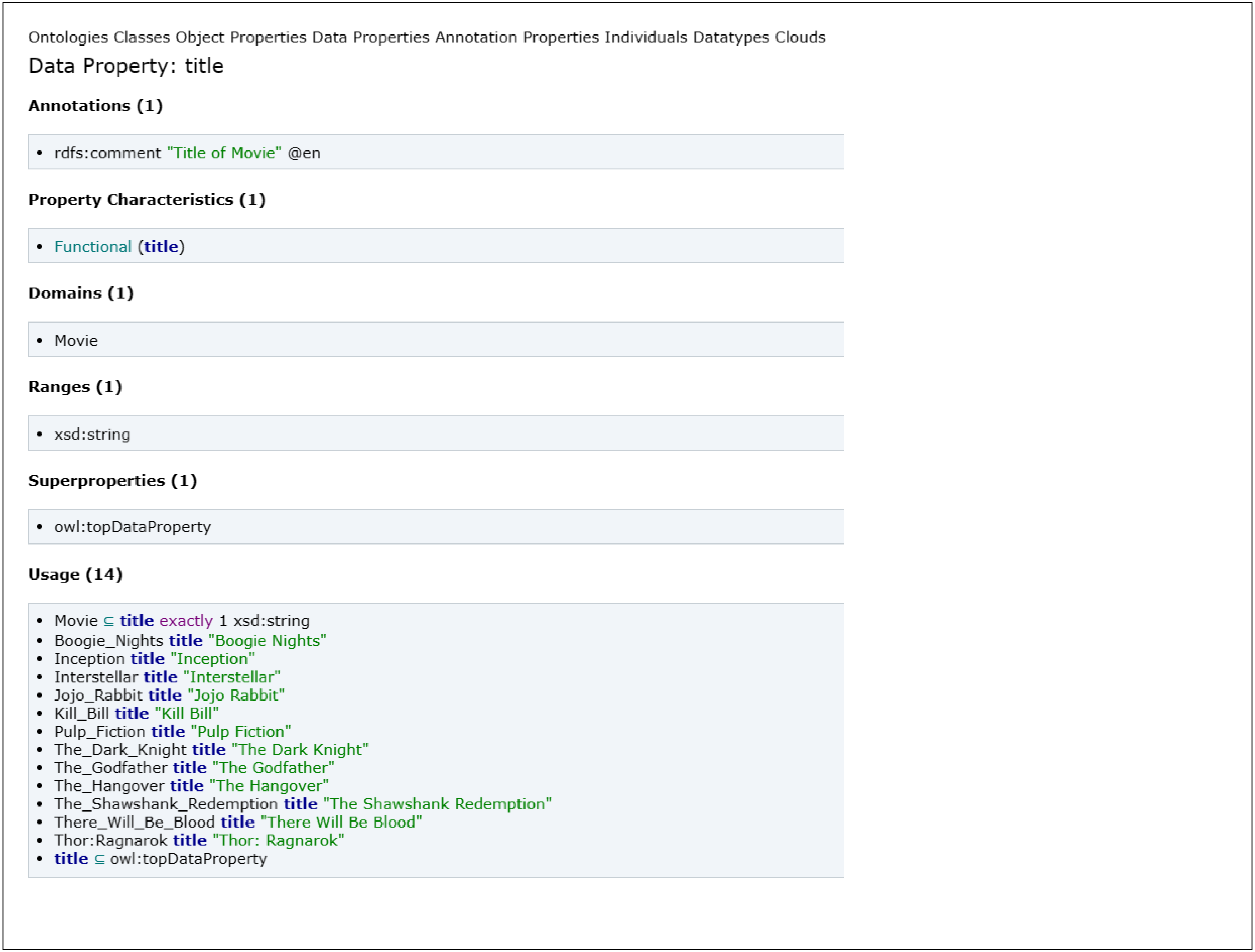
- isActorOf

Disjoint Properties (3)

hasActor, hasGenre, isGenreOf

Usage (17)

- Movie ⊆ **hasActor** min 1 Actor
- Boogie_Nights **hasActor** Paul_Thomas_Anderson
- Inception **hasActor** Leonardo_DiCaprio
- Interstellar **hasActor** Matthew_McConaughey
- Jojo_Rabbit **hasActor** Taika_Waititi
- Kill_Bill **hasActor** Uma_Thurman
- Pulp_Fiction **hasActor** John_Travolta
- Pulp_Fiction **hasActor** Quentin_Tarantino
- The_Dark_Knight **hasActor** Christian_Bale
- The_Godfather **hasActor** Marlon_Brando
- The_Hangover **hasActor** Bradley_Cooper
- The_Shawshank_Redemption **hasActor** Tim_Robbins
- There_Will_Be_Blood **hasActor** Paul_Thomas_Anderson
- Thor:Ragnarok **hasActor** Taika_Waititi
- **hasActor** inverse isActorOf
- DisjointProperties(**hasActor**, hasGenre)
- DisjointProperties(**hasActor**, isGenreOf)



Ontologies Classes Object Properties Data Properties Annotation Properties Individuals Datatypes Clouds

Data Property: title

Annotations (1)

- rdfs:comment "Title of Movie" @en

Property Characteristics (1)

- Functional (**title**)

Domains (1)

- Movie

Ranges (1)

- xsd:string

Superproperties (1)

- owl:topDataProperty

Usage (14)

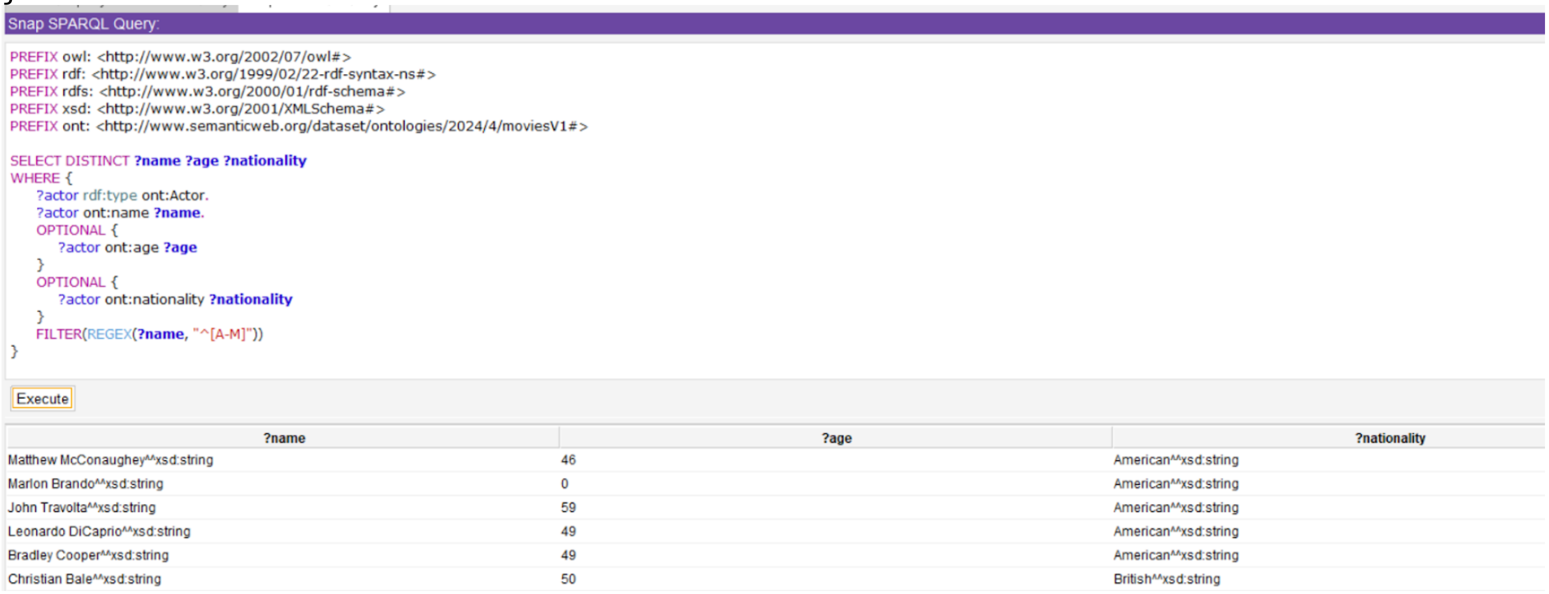
- Movie ⊆ **title** exactly 1 xsd:string
- Boogie_Nights **title** "Boogie Nights"
- Inception **title** "Inception"
- Interstellar **title** "Interstellar"
- Jojo_Rabbit **title** "Jojo Rabbit"
- Kill_Bill **title** "Kill Bill"
- Pulp_Fiction **title** "Pulp Fiction"
- The_Dark_Knight **title** "The Dark Knight"
- The_Godfather **title** "The Godfather"
- The_Hangover **title** "The Hangover"
- The_Shawshank_Redemption **title** "The Shawshank Redemption"
- There_Will_Be_Blood **title** "There Will Be Blood"
- Thor:Ragnarok **title** "Thor: Ragnarok"
- **title** ⊆ owl:topDataProperty

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]"))
}
```



Snap SPARQL Query:

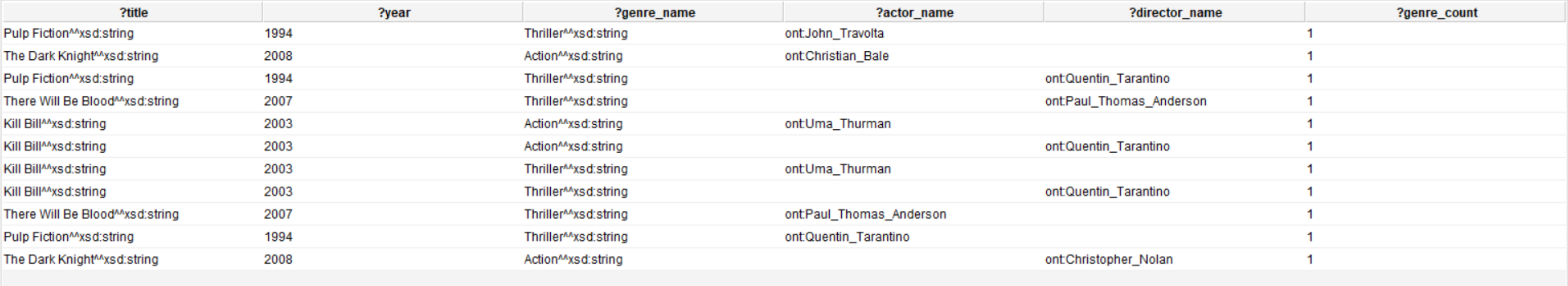
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

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]"))
}
```

?name	?age	?nationality
Matthew McConaughey^^xsd:string	46	American^^xsd:string
Marlon Brando^^xsd:string	0	American^^xsd:string
John Travolta^^xsd:string	59	American^^xsd:string
Leonardo DiCaprio^^xsd:string	49	American^^xsd:string
Bradley Cooper^^xsd:string	49	American^^xsd:string
Christian Bale^^xsd:string	50	British^^xsd:string

2) **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
```

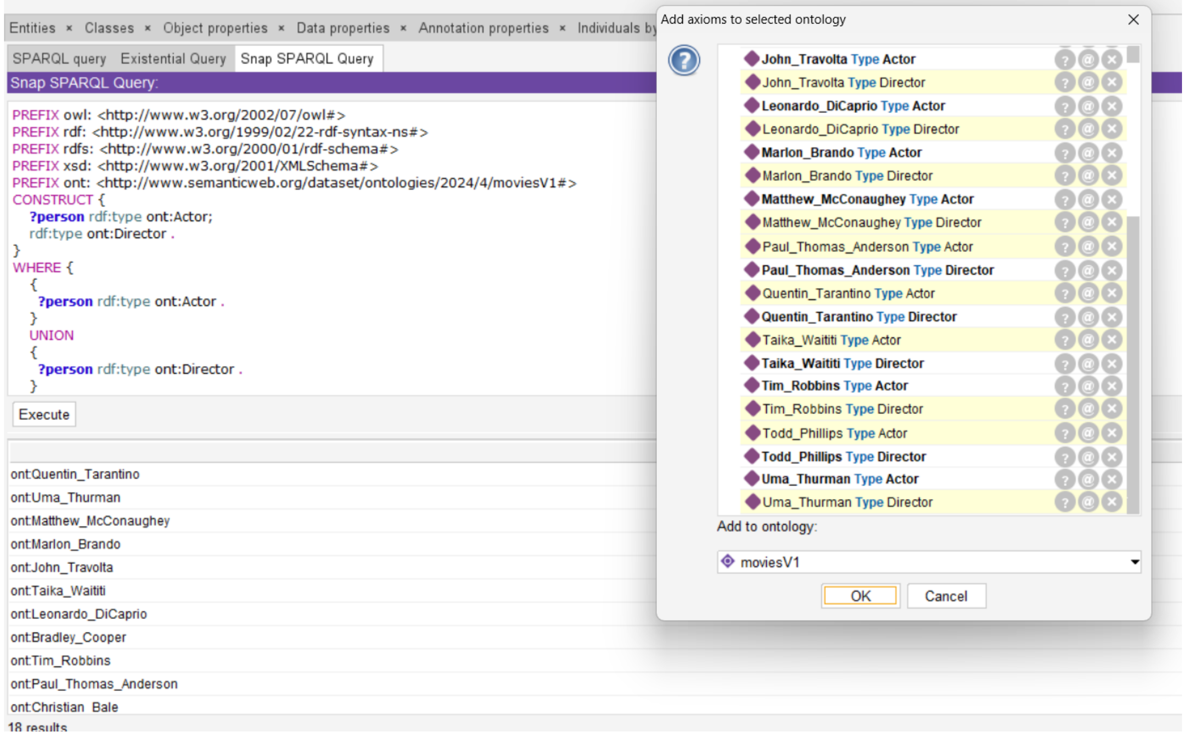


?title	?year	?genre_name	?actor_name	?director_name	?genre_count
Pulp Fiction^^xsd:string	1994	Thriller^^xsd:string	ont:John_Travolta		1
The Dark Knight^^xsd:string	2008	Action^^xsd:string	ont:Christian_Bale		1
Pulp Fiction^^xsd:string	1994	Thriller^^xsd:string		ont:Quentin_Tarantino	1
There Will Be Blood^^xsd:string	2007	Thriller^^xsd:string		ont:Paul_Thomas_Anderson	1
Kill Bill^^xsd:string	2003	Action^^xsd:string	ont:Uma_Thurman		1
Kill Bill^^xsd:string	2003	Action^^xsd:string		ont:Quentin_Tarantino	1
Kill Bill^^xsd:string	2003	Thriller^^xsd:string	ont:Uma_Thurman		1
Kill Bill^^xsd:string	2003	Thriller^^xsd:string		ont:Quentin_Tarantino	1
There Will Be Blood^^xsd:string	2007	Thriller^^xsd:string	ont:Paul_Thomas_Anderson		1
Pulp Fiction^^xsd:string	1994	Thriller^^xsd:string	ont:Quentin_Tarantino		1
The Dark Knight^^xsd:string	2008	Action^^xsd:string		ont:Christopher_Nolan	1

3) 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 .
   }
}
```



```
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 .
   }
```

```
ont:Quentin_Tarantino
ont:Uma_Thurman
ont:Matthew_McConaughey
ont:Marlon_Brando
ont:John_Travolta
ont:Taika_Waititi
ont:Leonardo_DiCaprio
ont:Bradley_Cooper
ont:Tim_Robbins
ont:Paul_Thomas_Anderson
ont:Christian_Bale
18 results
```

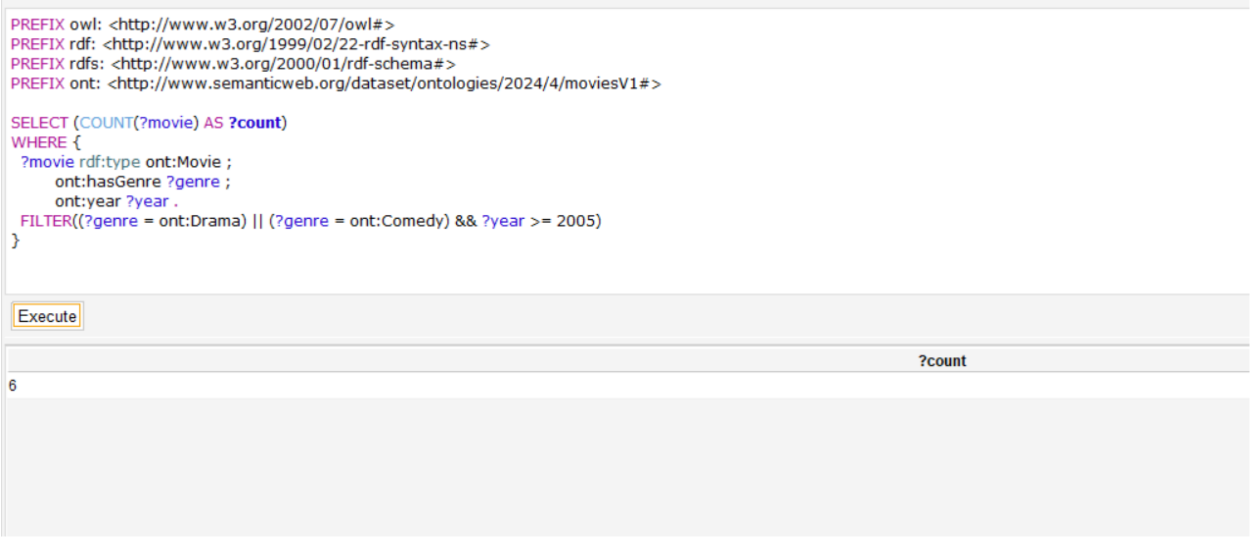
Add axioms to selected ontology

```
John_Travolta Type Actor
John_Travolta Type Director
Leonardo_DiCaprio Type Actor
Leonardo_DiCaprio Type Director
Marlon_Brando Type Actor
Marlon_Brando Type Director
Matthew_McConaughey Type Actor
Matthew_McConaughey Type Director
Paul_Thomas_Anderson Type Actor
Paul_Thomas_Anderson Type Director
Quentin_Tarantino Type Actor
Quentin_Tarantino Type Director
Taika_Waititi Type Actor
Taika_Waititi Type Director
Tim_Robbins Type Actor
Tim_Robbins Type Director
Todd_Phillips Type Actor
Todd_Phillips Type Director
Uma_Thurman Type Actor
Uma_Thurman Type Director
```

Add to ontology: moviesV1

4) 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)
}
```



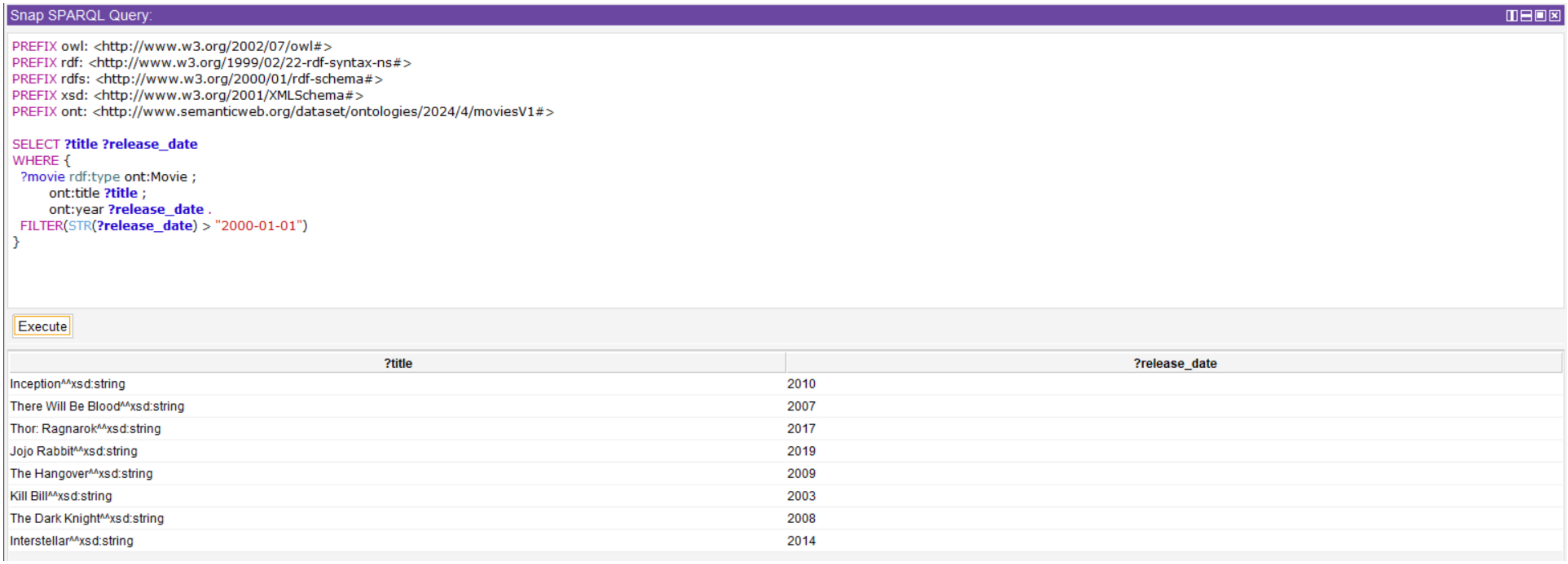
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT (COUNT(?movie) AS ?count)
WHERE {
 ?movie rdf:type ont:Movie ;
      ont:hasGenre ?genre ;
      ont:year ?year .
 FILTER((?genre = ont:Drama) || (?genre = ont:Comedy) && ?year >= 2005)
}
```

?count
6

5) **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")
}
```

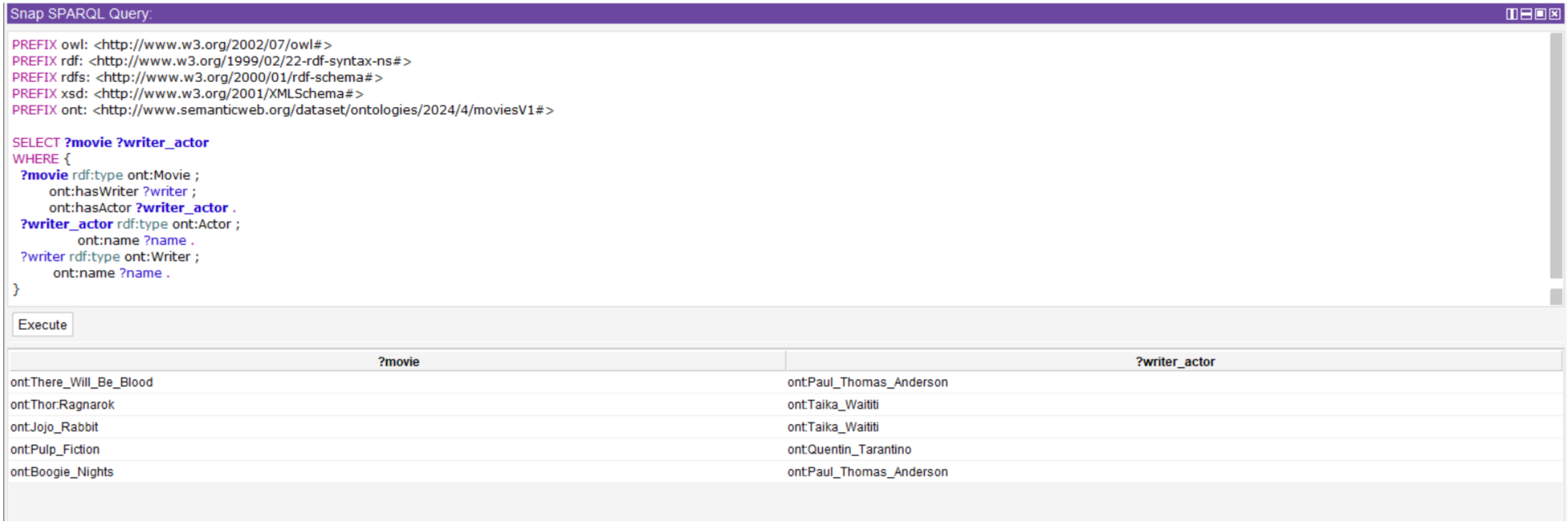


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT ?title ?release_date
WHERE {
 ?movie rdf:type ont:Movie ;
       ont:title ?title ;
       ont:year ?release_date .
 FILTER(STR(?release_date) > "2000-01-01")
}
```

?title	?release_date
Inception^^xsd:string	2010
There Will Be Blood^^xsd:string	2007
Thor: Ragnarok^^xsd:string	2017
Jojo Rabbit^^xsd:string	2019
The Hangover^^xsd:string	2009
Kill Bill^^xsd:string	2003
The Dark Knight^^xsd:string	2008
Interstellar^^xsd:string	2014

6) 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 .
}
```

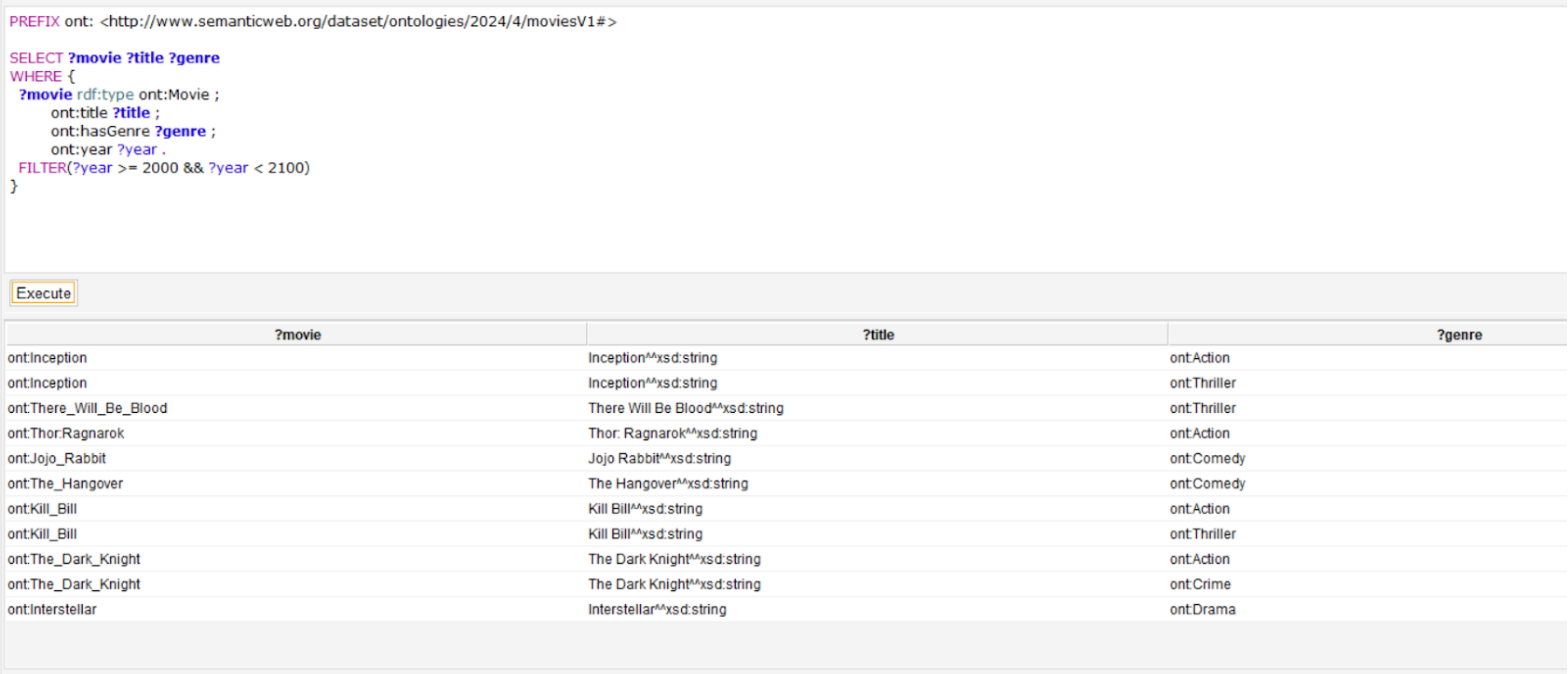


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

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 .
}
```

?movie	?writer_actor
ont:There_Will_Be_Blood	ont:Paul_Thomas_Anderson
ont:Thor:Ragnarok	ont:Taika_Waititi
ont:Jojo_Rabbit	ont:Taika_Waititi
ont:Pulp_Fiction	ont:Quentin_Tarantino
ont:Boogie_Nights	ont:Paul_Thomas_Anderson

7) 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)
}
```

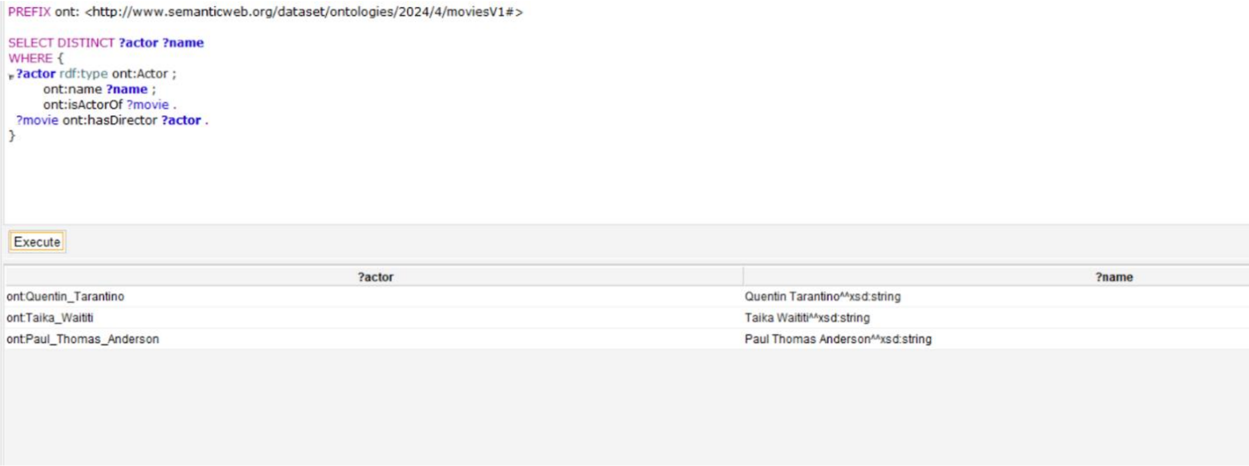


```
PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT ?movie ?title ?genre
WHERE {
 ?movie rdf:type ont:Movie ;
       ont:title ?title ;
       ont:hasGenre ?genre ;
       ont:year ?year .
 FILTER(?year >= 2000 && ?year < 2100)
}
```

?movie	?title	?genre
ont:Inception	Inception^^xsd:string	ont:Action
ont:Inception	Inception^^xsd:string	ont:Thriller
ont:There_Will_Be_Blood	There Will Be Blood^^xsd:string	ont:Thriller
ont:Thor:Ragnarok	Thor: Ragnarok^^xsd:string	ont:Action
ont:Jojo_Rabbit	Jojo Rabbit^^xsd:string	ont:Comedy
ont:The_Hangover	The Hangover^^xsd:string	ont:Comedy
ont:Kill_Bill	Kill Bill^^xsd:string	ont:Action
ont:Kill_Bill	Kill Bill^^xsd:string	ont:Thriller
ont:The_Dark_Knight	The Dark Knight^^xsd:string	ont:Action
ont:The_Dark_Knight	The Dark Knight^^xsd:string	ont:Crime
ont:Interstellar	Interstellar^^xsd:string	ont:Drama

8) 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 .
}
```

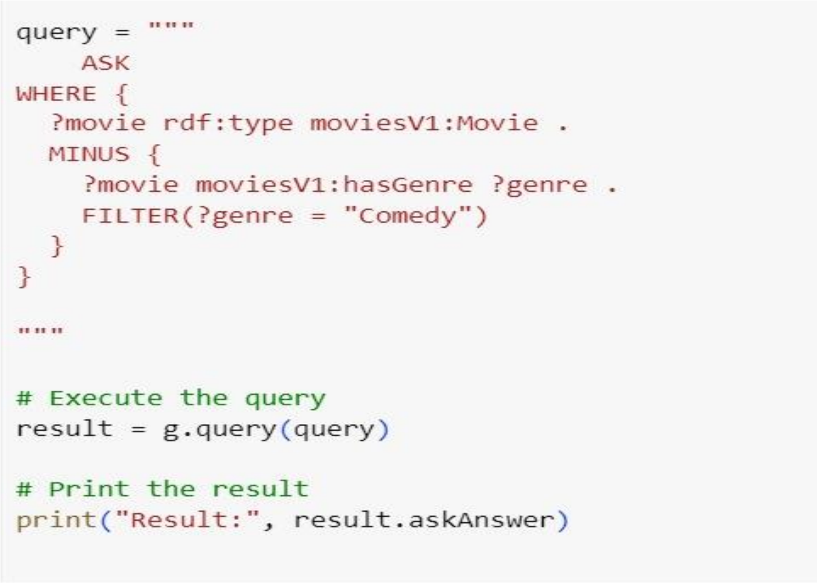


```
PREFIX ont: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#>

SELECT DISTINCT ?actor ?name
WHERE {
 ?actor rdf:type ont:Actor ;
      ont:name ?name ;
      ont:isActorOf ?movie .
 ?movie ont:hasDirector ?actor .
}
```

?actor	?name
ont:Quentin_Tarantino	Quentin Tarantino^^xsd:string
ont:Taika_Waititi	Taika Waititi^^xsd:string
ont:Paul_Thomas_Anderson	Paul Thomas Anderson^^xsd:string

9) **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")
  }
}
```



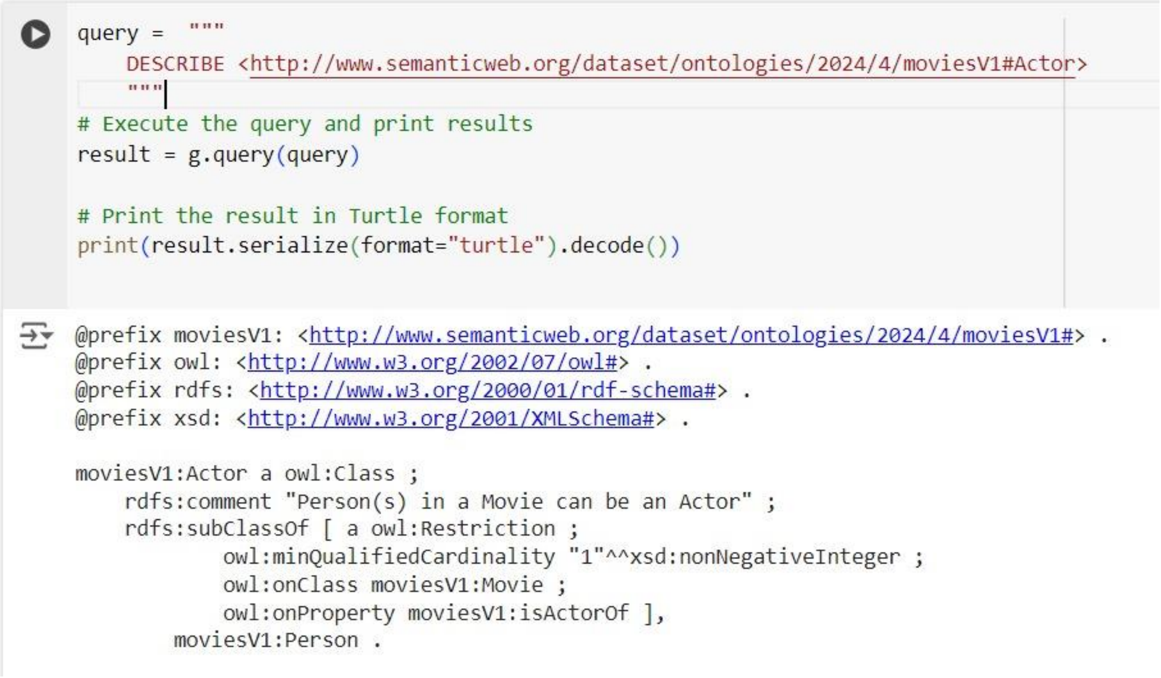
```
query = """
    ASK
WHERE {
  ?movie rdf:type moviesV1:Movie .
  MINUS {
    ?movie moviesV1:hasGenre ?genre .
    FILTER(?genre = "Comedy")
  }
}

"""

# Execute the query
result = g.query(query)

# Print the result
print("Result:", result.askAnswer)
```

```
Result: True
```

10)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>
```



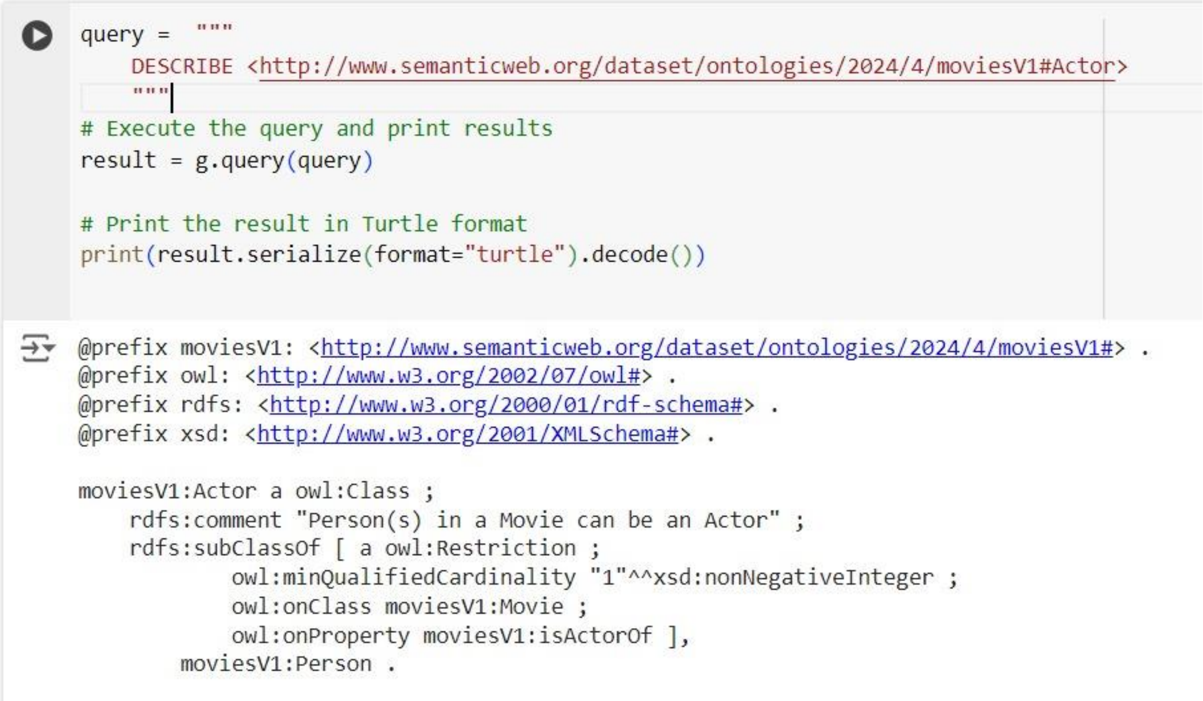
```
query =  """
    DESCRIBE <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#Actor>
    """
# Execute the query and print results
result = g.query(query)

# Print the result in Turtle format
print(result.serialize(format="turtle").decode())
```

```
@prefix moviesV1: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

moviesV1:Actor a owl:Class ;
    rdfs:comment "Person(s) in a Movie can be an Actor" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass moviesV1:Movie ;
            owl:onProperty moviesV1:isActorOf ],
        moviesV1:Person .
```

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())
```

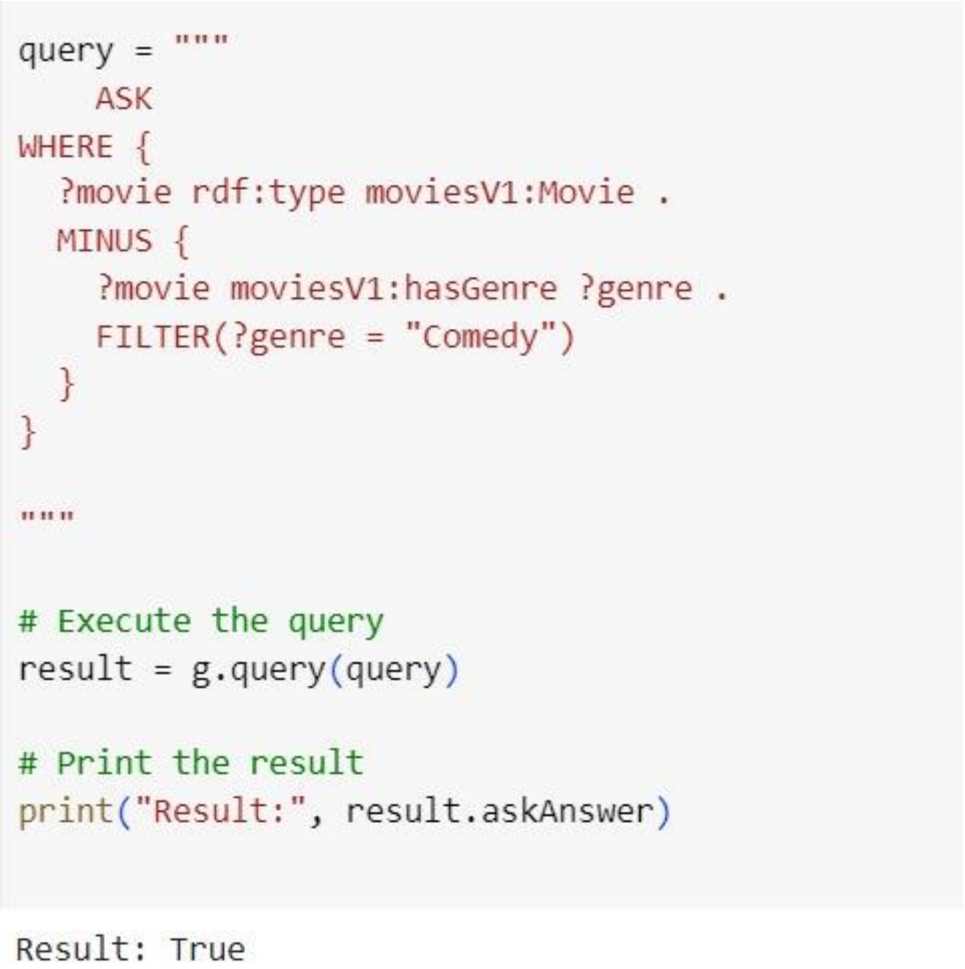


```
query =  """
    DESCRIBE <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#Actor>
    """
# Execute the query and print results
result = g.query(query)

# Print the result in Turtle format
print(result.serialize(format="turtle").decode())
```

```
@prefix moviesV1: <http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

moviesV1:Actor a owl:Class ;
    rdfs:comment "Person(s) in a Movie can be an Actor" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass moviesV1:Movie ;
            owl:onProperty moviesV1:isActorOf ],
        moviesV1:Person .
```

2) *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)
```

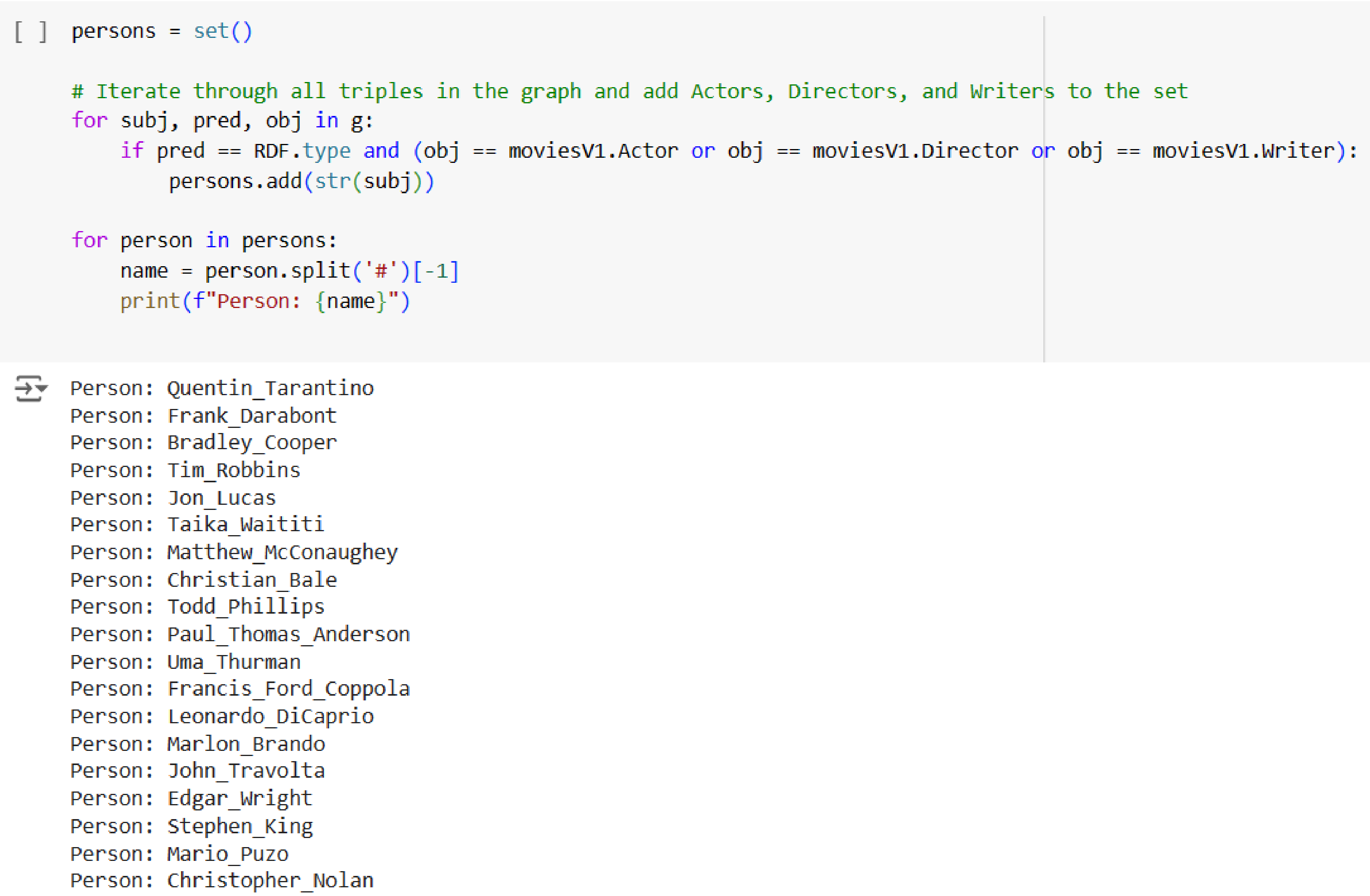


```
query = """
    ASK
WHERE {
  ?movie rdf:type moviesV1:Movie .
  MINUS {
    ?movie moviesV1:hasGenre ?genre .
    FILTER(?genre = "Comedy")
  }
}

"""


# Execute the query
result = g.query(query)

# Print the result
print("Result:", result.askAnswer)
```

```
Result: True
```

Part IV: Manipulating the ontology using Jena

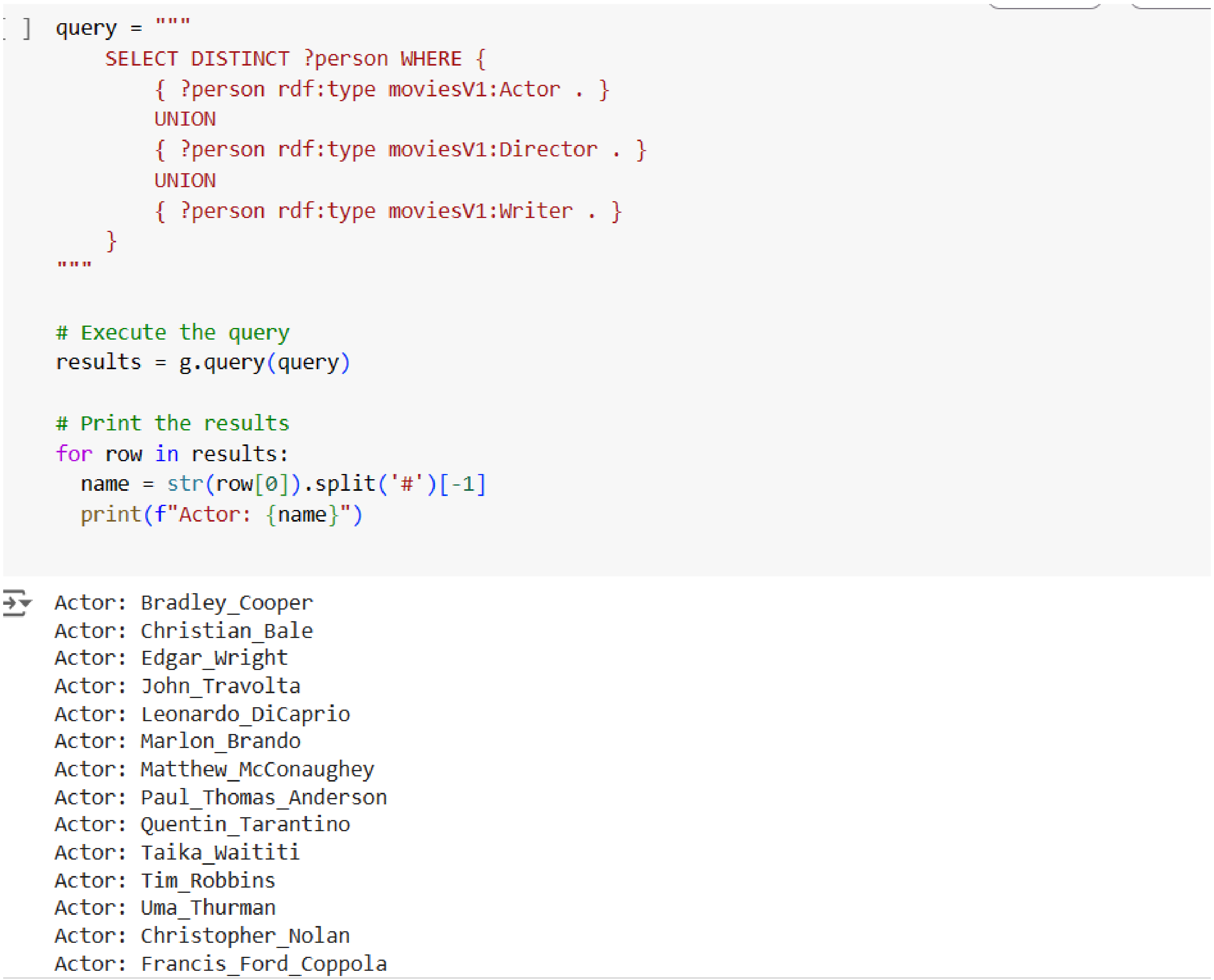
1) Display all Persons without Query or Inference



```python
persons = set()

# Iterate through all triples in the graph and add Actors, Directors, and Writers to the set
for subj, pred, obj in g:
    if pred == RDF.type and (obj == moviesV1.Actor or obj == moviesV1.Director or obj == moviesV1.Writer):
        persons.add(str(subj))

for person in persons:
    name = person.split('#')[-1]
    print(f"Person: {name}")
```

```
Person: Quentin_Tarantino
Person: Frank_Darabont
Person: Bradley_Cooper
Person: Tim_Robbins
Person: Jon_Lucas
Person: Taika_Waititi
Person: Matthew_McConaughey
Person: Christian_Bale
Person: Todd_Phillips
Person: Paul_Thomas_Anderson
Person: Uma_Thurman
Person: Francis_Ford_Coppola
Person: Leonardo_DiCaprio
Person: Marlon_Brando
Person: John_Travolta
Person: Edgar_Wright
Person: Stephen_King
Person: Mario_Puzo
Person: Christopher_Nolan
```

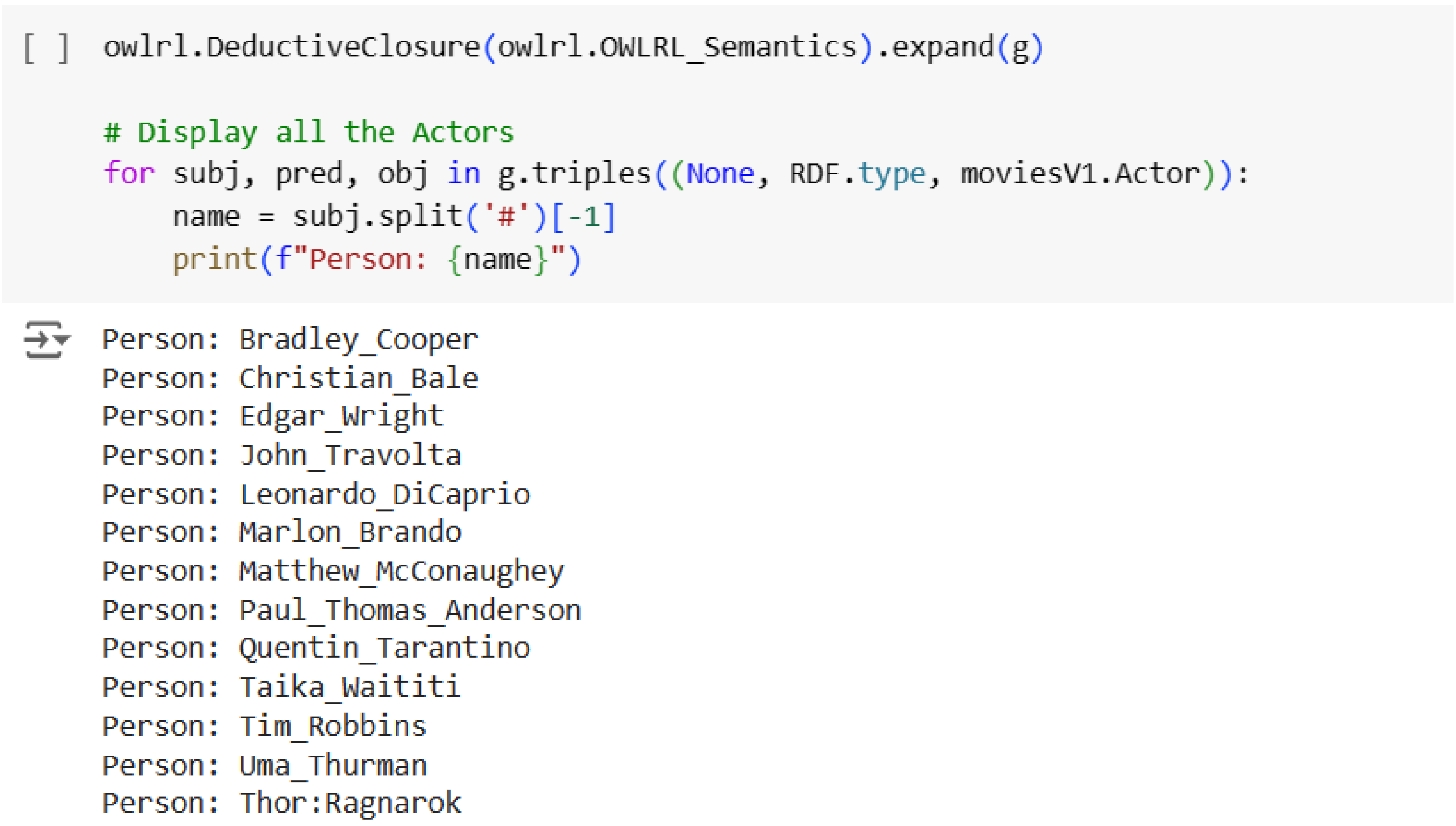
2) Display all Persons by Query only



```
query = """
    SELECT DISTINCT ?person WHERE {
        { ?person rdf:type moviesV1:Actor . }
        UNION
        { ?person rdf:type moviesV1:Director . }
        UNION
        { ?person rdf:type moviesV1:Writer . }
    }
"""

# Execute the query
results = g.query(query)

# Print the results
for row in results:
  name = str(row[0]).split('#')[-1]
  print(f"Actor: {name}")
```

```
Actor: Bradley_Cooper
Actor: Christian_Bale
Actor: Edgar_Wright
Actor: John_Travolta
Actor: Leonardo_DiCaprio
Actor: Marlon_Brando
Actor: Matthew_McConaughey
Actor: Paul_Thomas_Anderson
Actor: Quentin_Tarantino
Actor: Taika_Waititi
Actor: Tim_Robbins
Actor: Uma_Thurman
Actor: Christopher_Nolan
Actor: Francis_Ford_Coppola
```

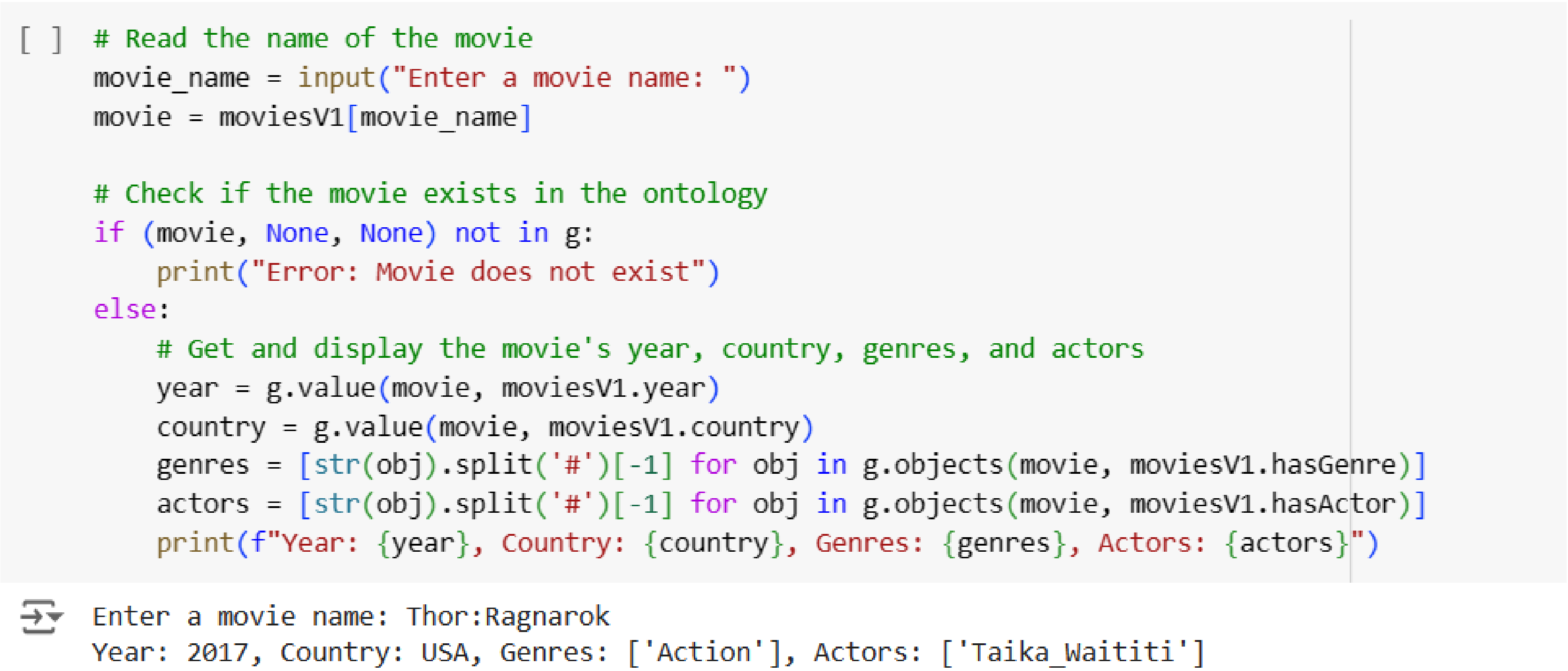
3) Display all Persons by Inference only



```
[ ]  owlrl.DeductiveClosure(owlrl.OWLRL_Semantics).expand(g)

     # Display all the Actors
     for subj, pred, obj in g.triples((None, RDF.type, moviesV1.Actor)):
         name = subj.split('#')[-1]
         print(f"Person: {name}")
```

```
Person: Bradley_Cooper
Person: Christian_Bale
Person: Edgar_Wright
Person: John_Travolta
Person: Leonardo_DiCaprio
Person: Marlon_Brando
Person: Matthew_McConaughey
Person: Paul_Thomas_Anderson
Person: Quentin_Tarantino
Person: Taika_Waititi
Person: Tim_Robbins
Person: Uma_Thurman
Person: Thor:Ragnarok
```

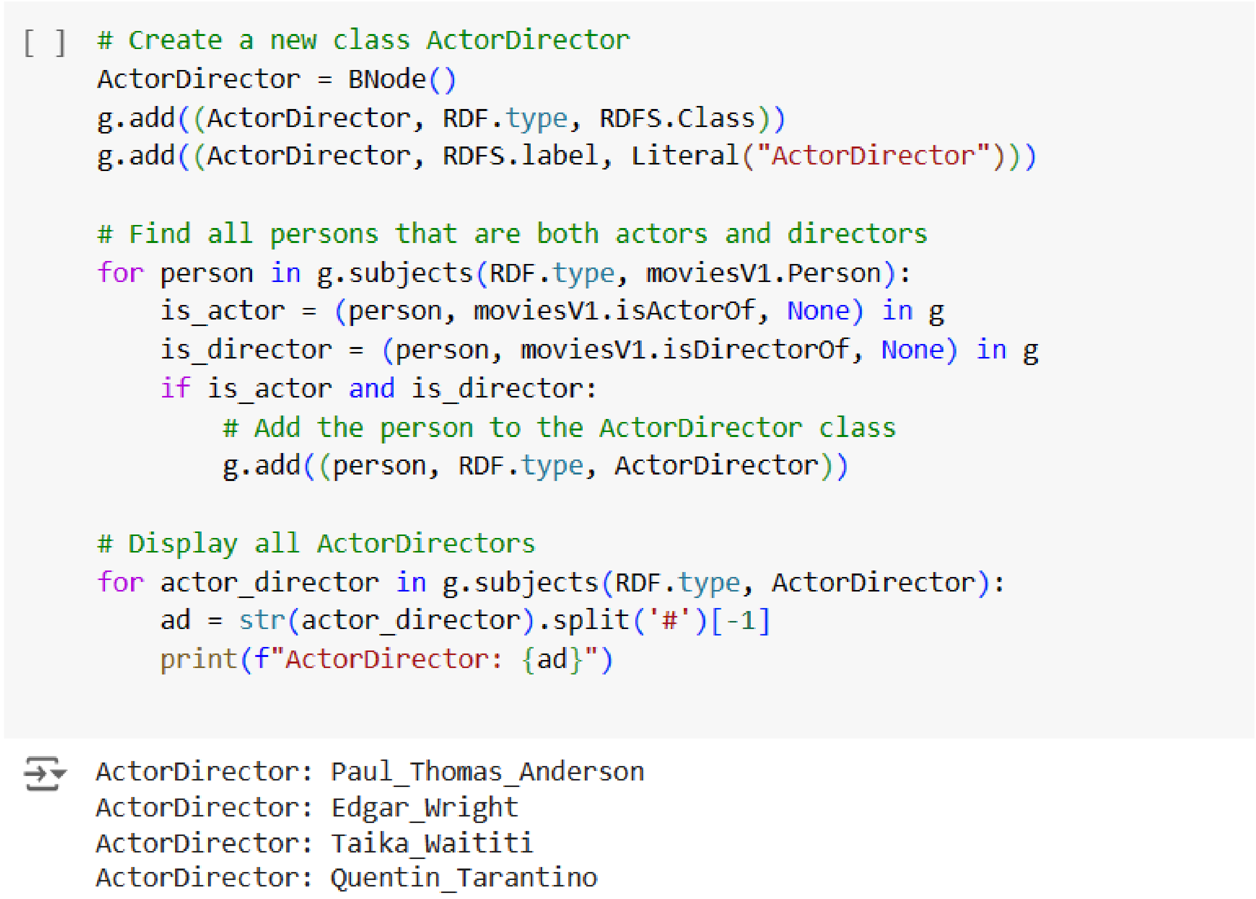
4) Display Movie properties if Movie Exists



```python
# Read the name of the movie
movie_name = input("Enter a movie name: ")
movie = moviesV1[movie_name]

# Check if the movie exists in the ontology
if (movie, None, None) not in g:
    print("Error: Movie does not exist")
else:
    # Get and display the movie's year, country, genres, and actors
    year = g.value(movie, moviesV1.year)
    country = g.value(movie, moviesV1.country)
    genres = [str(obj).split('#')[-1] for obj in g.objects(movie, moviesV1.hasGenre)]
    actors = [str(obj).split('#')[-1] for obj in g.objects(movie, moviesV1.hasActor)]
    print(f"Year: {year}, Country: {country}, Genres: {genres}, Actors: {actors}")
```

```
Enter a movie name: Thor:Ragnarok
Year: 2017, Country: USA, Genres: ['Action'], Actors: ['Taika_Waititi']
```

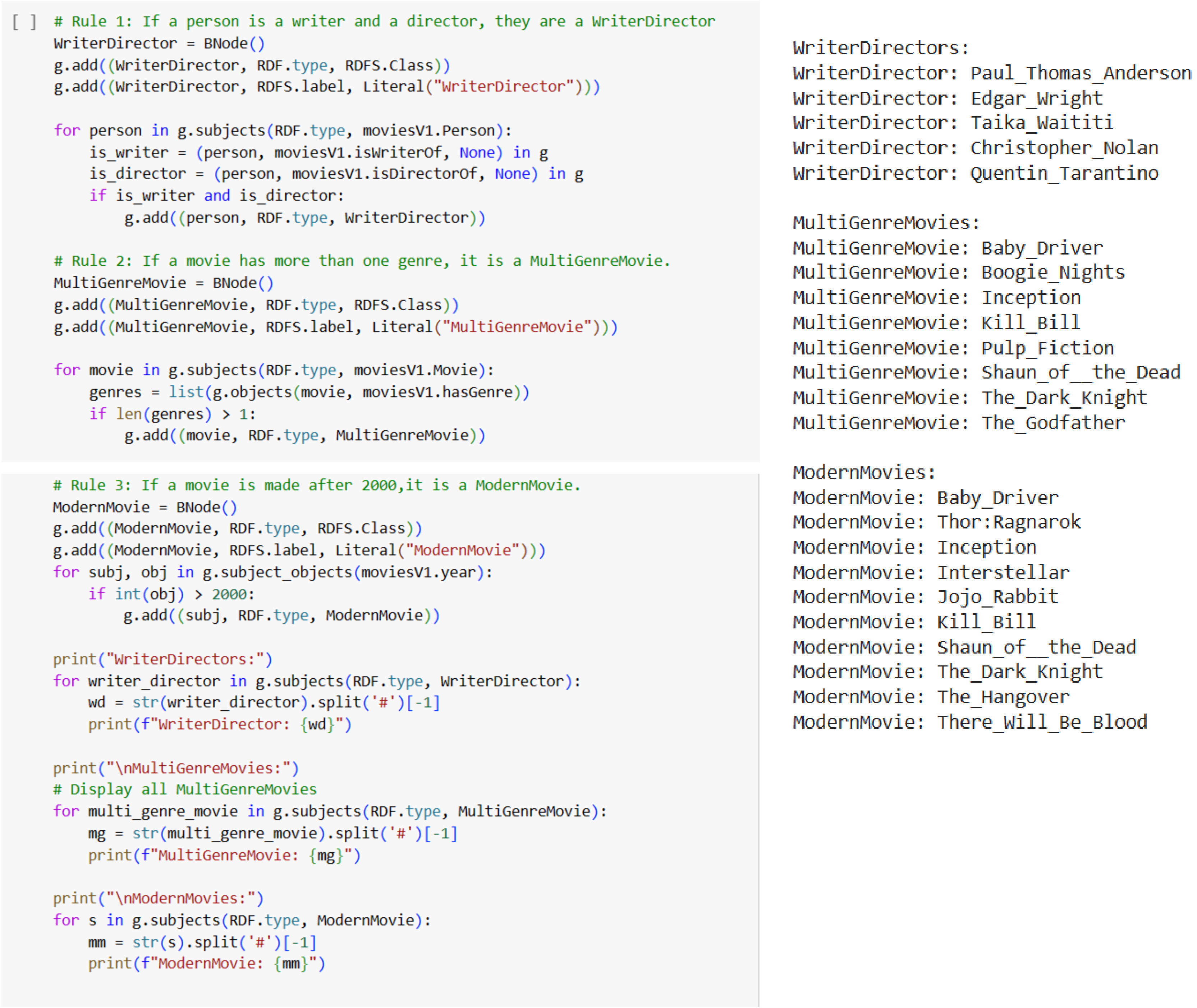
5) Add a Rule for a New Class, ActorDirector



```python
# Create a new class ActorDirector
ActorDirector = BNode()
g.add((ActorDirector, RDF.type, RDFS.Class))
g.add((ActorDirector, RDFS.label, Literal("ActorDirector")))

# Find all persons that are both actors and directors
for person in g.subjects(RDF.type, moviesV1.Person):
    is_actor = (person, moviesV1.isActorOf, None) in g
    is_director = (person, moviesV1.isDirectorOf, None) in g
    if is_actor and is_director:
        # Add the person to the ActorDirector class
        g.add((person, RDF.type, ActorDirector))

# Display all ActorDirectors
for actor_director in g.subjects(RDF.type, ActorDirector):
    ad = str(actor_director).split('#')[-1]
    print(f"ActorDirector: {ad}")
```

```
ActorDirector: Paul_Thomas_Anderson
ActorDirector: Edgar_Wright
ActorDirector: Taika_Waititi
ActorDirector: Quentin_Tarantino
```

6) Add New Rules



```
[ ] # Rule 1: If a person is a writer and a director, they are a WriterDirector
    WriterDirector = BNode()
    g.add((WriterDirector, RDF.type, RDFS.Class))
    g.add((WriterDirector, RDFS.label, Literal("WriterDirector")))

    for person in g.subjects(RDF.type, moviesV1.Person):
        is_writer = (person, moviesV1.isWriterOf, None) in g
        is_director = (person, moviesV1.isDirectorOf, None) in g
        if is_writer and is_director:
            g.add((person, RDF.type, WriterDirector))

    # Rule 2: If a movie has more than one genre, it is a MultiGenreMovie.
    MultiGenreMovie = BNode()
    g.add((MultiGenreMovie, RDF.type, RDFS.Class))
    g.add((MultiGenreMovie, RDFS.label, Literal("MultiGenreMovie")))

    for movie in g.subjects(RDF.type, moviesV1.Movie):
        genres = list(g.objects(movie, moviesV1.hasGenre))
        if len(genres) > 1:
            g.add((movie, RDF.type, MultiGenreMovie))
```

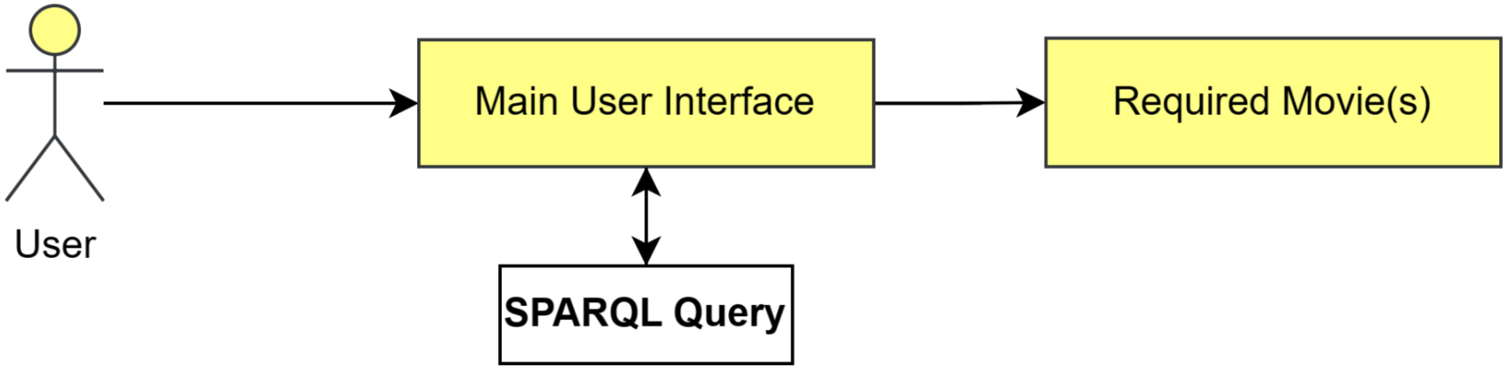
```
    # Rule 3: If a movie is made after 2000,it is a ModernMovie.
    ModernMovie = BNode()
    g.add((ModernMovie, RDF.type, RDFS.Class))
    g.add((ModernMovie, RDFS.label, Literal("ModernMovie")))
    for subj, obj in g.subject_objects(moviesV1.year):
        if int(obj) > 2000:
            g.add((subj, RDF.type, ModernMovie))

    print("WriterDirectors:")
    for writer_director in g.subjects(RDF.type, WriterDirector):
        wd = str(writer_director).split('#')[-1]
        print(f"WriterDirector: {wd}")

    print("\nMultiGenreMovies:")
    # Display all MultiGenreMovies
    for multi_genre_movie in g.subjects(RDF.type, MultiGenreMovie):
        mg = str(multi_genre_movie).split('#')[-1]
        print(f"MultiGenreMovie: {mg}")

    print("\nModernMovies:")
    for s in g.subjects(RDF.type, ModernMovie):
        mm = str(s).split('#')[-1]
        print(f"ModernMovie: {mm}")
```

```
WriterDirectors:
WriterDirector: Paul_Thomas_Anderson
WriterDirector: Edgar_Wright
WriterDirector: Taika_Waititi
WriterDirector: Christopher_Nolan
WriterDirector: Quentin_Tarantino

MultiGenreMovies:
MultiGenreMovie: Baby_Driver
MultiGenreMovie: Boogie_Nights
MultiGenreMovie: Inception
MultiGenreMovie: Kill_Bill
MultiGenreMovie: Pulp_Fiction
MultiGenreMovie: Shaun_of__the_Dead
MultiGenreMovie: The_Dark_Knight
MultiGenreMovie: The_Godfather

ModernMovies:
ModernMovie: Baby_Driver
ModernMovie: Thor:Ragnarok
ModernMovie: Inception
ModernMovie: Interstellar
ModernMovie: Jojo_Rabbit
ModernMovie: Kill_Bill
ModernMovie: Shaun_of__the_Dead
ModernMovie: The_Dark_Knight
ModernMovie: The_Hangover
ModernMovie: There_Will_Be_Blood
```

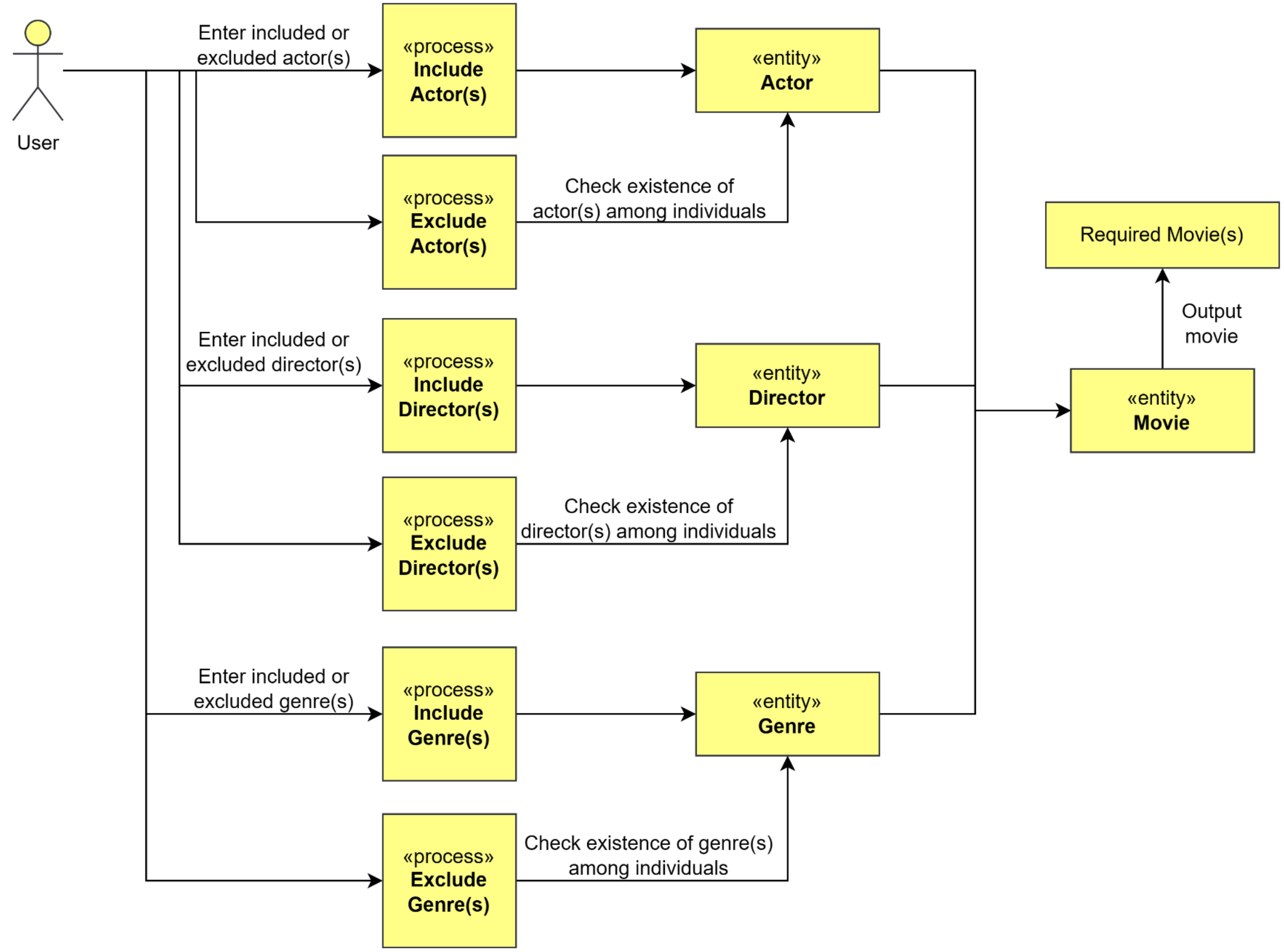
Part V: Python application

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

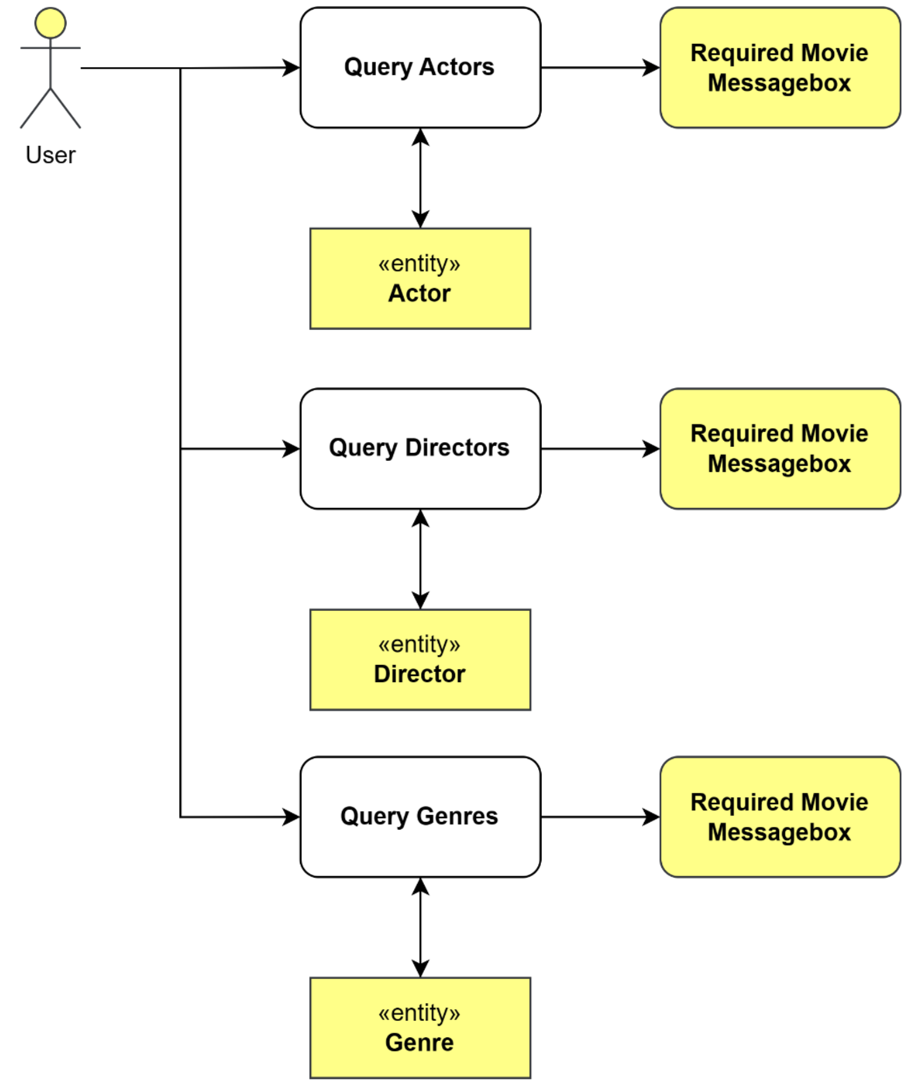
Data Flow Diagram Level 0



Data Flow Diagram Level 1


User
Enter included or excluded actor(s)
«process» Include Actor(s)
«process» Exclude Actor(s)
«entity» Actor
Check existence of actor(s) among individuals
Enter included or excluded director(s)
«process» Include Director(s)
«process» Exclude Director(s)
«entity» Director
Check existence of director(s) among individuals
Enter included or excluded genre(s)
«process» Include Genre(s)
«process» Exclude Genre(s)
«entity» Genre
Check existence of genre(s) among individuals
«entity» Movie
Output movie
Required Movie(s)

Data Flow Diagram Level 2



Environment Dependencies and Libraries

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

Setting Up a SPAQRL Endpoint

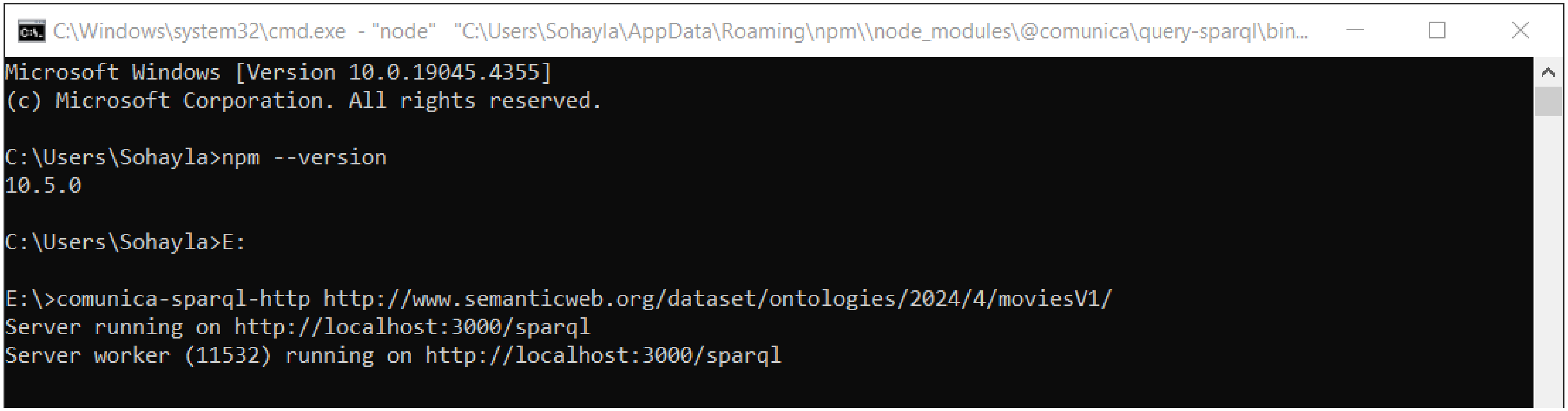
Http requests will be used in the process of parsing resource description files (rdf) and returning results of sparql query. To do that, a sparql endpoint will have to be set up. Comunica SPARQL is used to set up http://localhost:3000/sparql on your device. If not set up properly, a connection error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

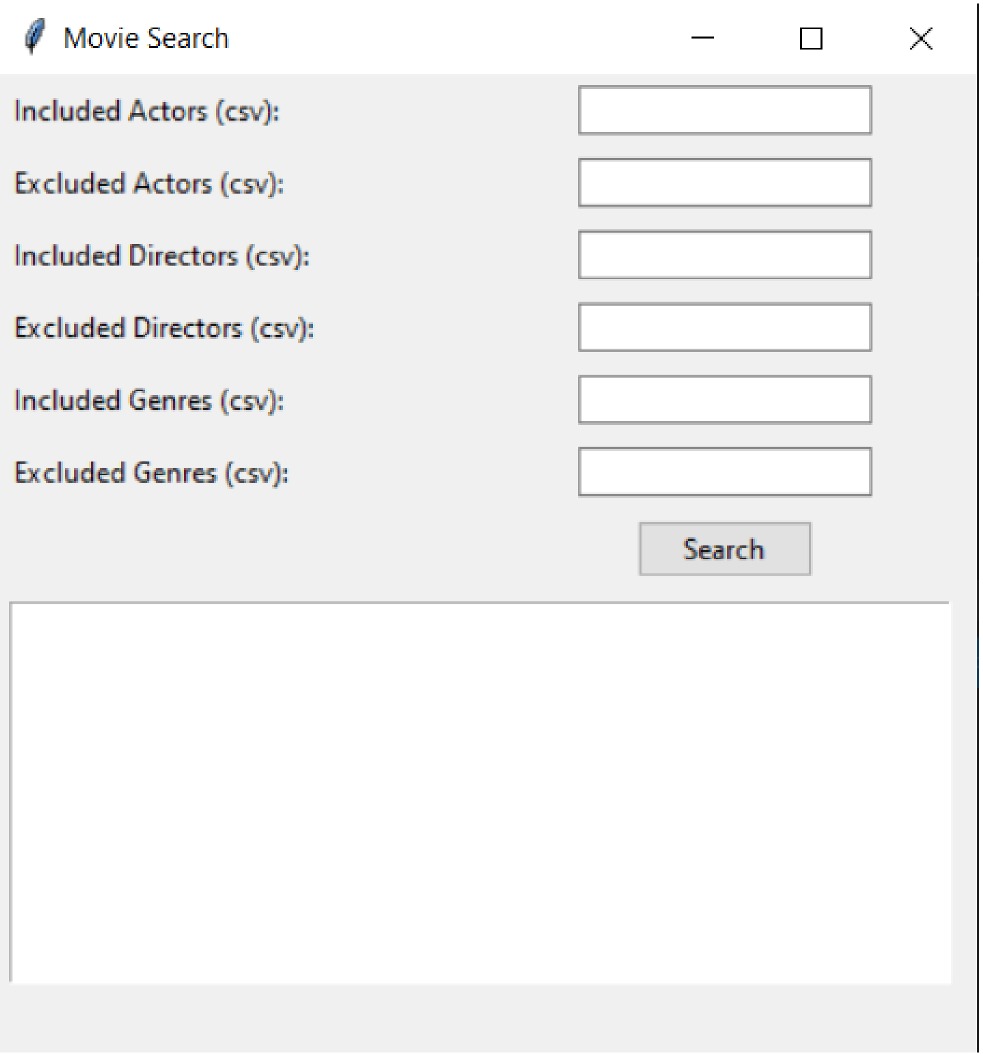

C:\Windows\system32\cmd.exe - "node" "C:\Users\Sohayla\AppData\Roaming\npm\\node_modules\@comunica\query-sparql\bin...

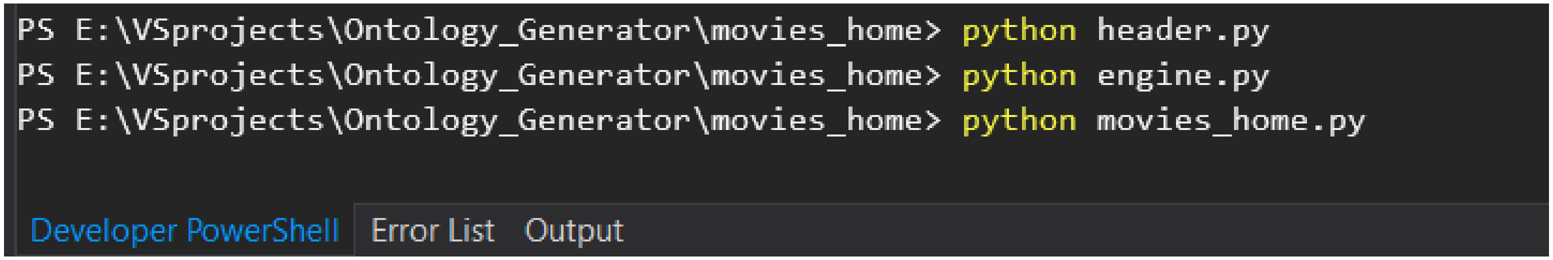
```
Microsoft Windows [Version 10.0.19045.4355]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Sohayla>npm --version
10.5.0

C:\Users\Sohayla>E:

E:\>comunica-sparql-http http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1/
Server running on http://localhost:3000/sparql
Server worker (11532) running on http://localhost:3000/sparql
```

`For more information:`

Comunica – Setting up a SPARQL endpoint

Desktop Application Interface



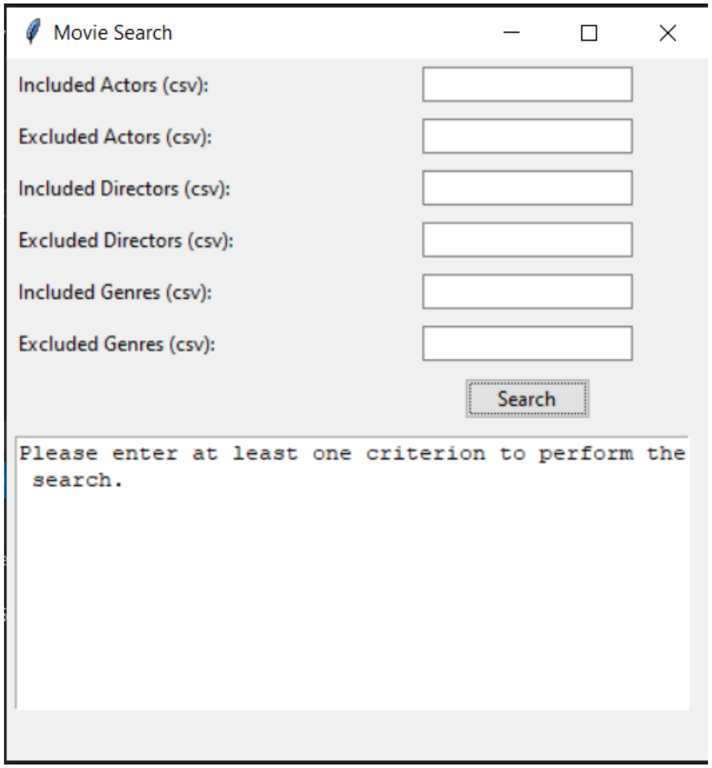


```
PS E:\VSprojects\Ontology_Generator\movies_home> python header.py
PS E:\VSprojects\Ontology_Generator\movies_home> python engine.py
PS E:\VSprojects\Ontology_Generator\movies_home> python movies_home.py
```

Test Cases

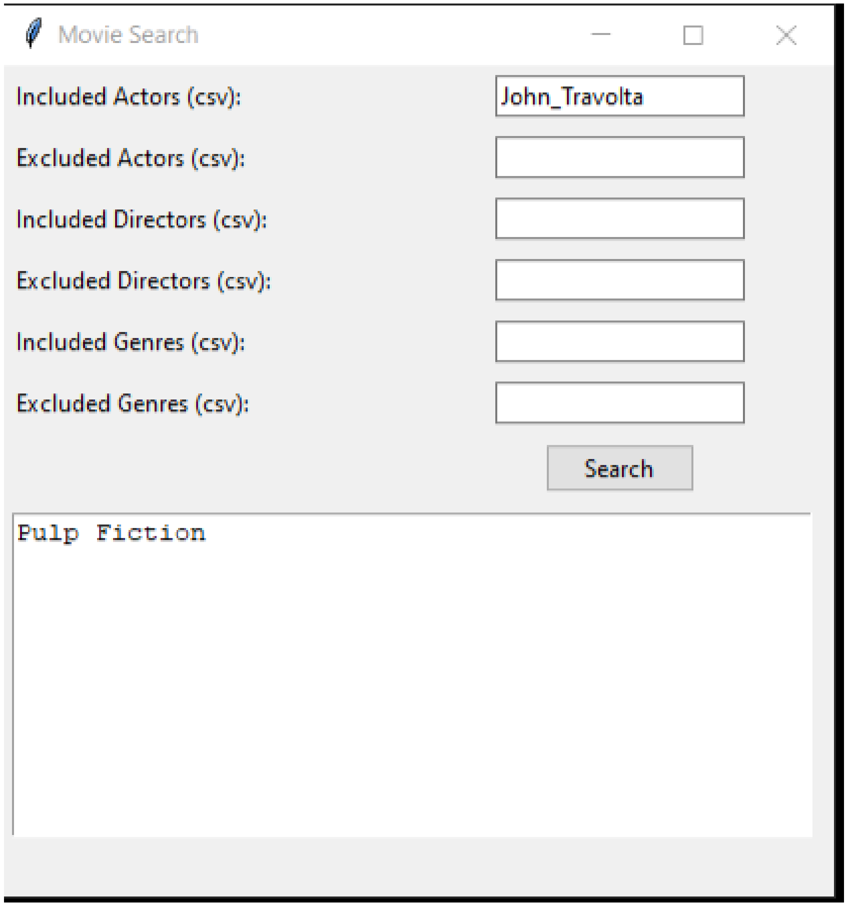
Test Case	Description	Result
Test Null value	Check null filled values	Success
Test Single value	Test include and exclude actor	Success
Test Single value	Test include and exclude director	Success
Test Single value	Test include and exclude genre	Success
Test Multiple values	Test same value for include 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:**



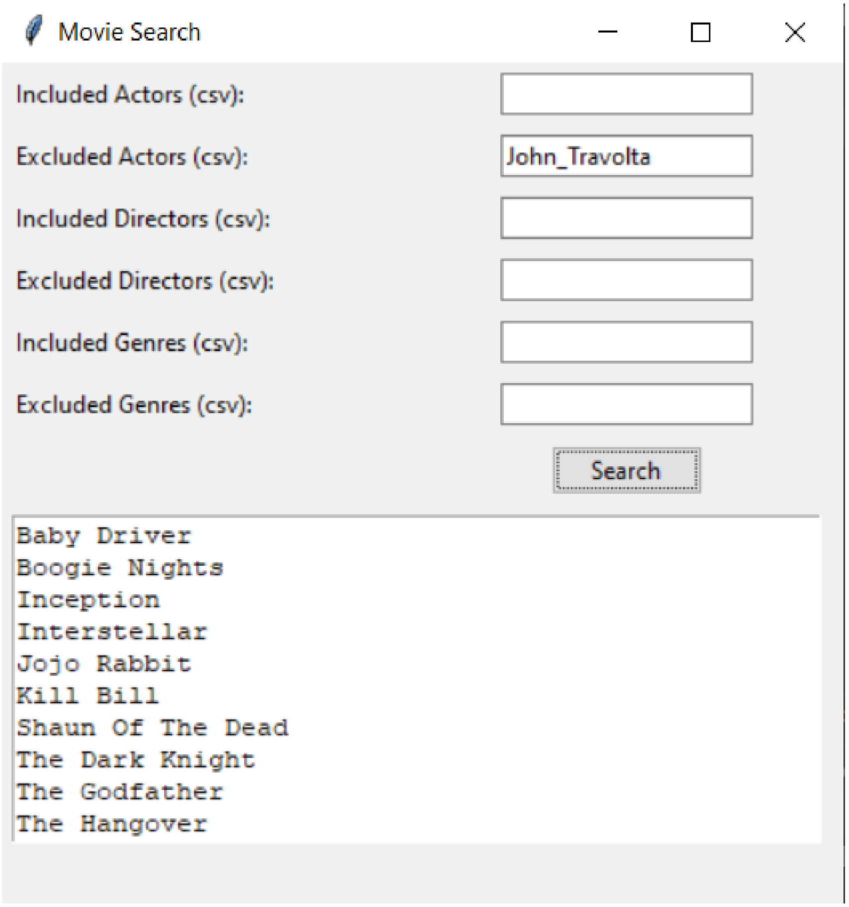
```
Please enter at least one criterion to perform the
 search.
```

2) Test Single value:



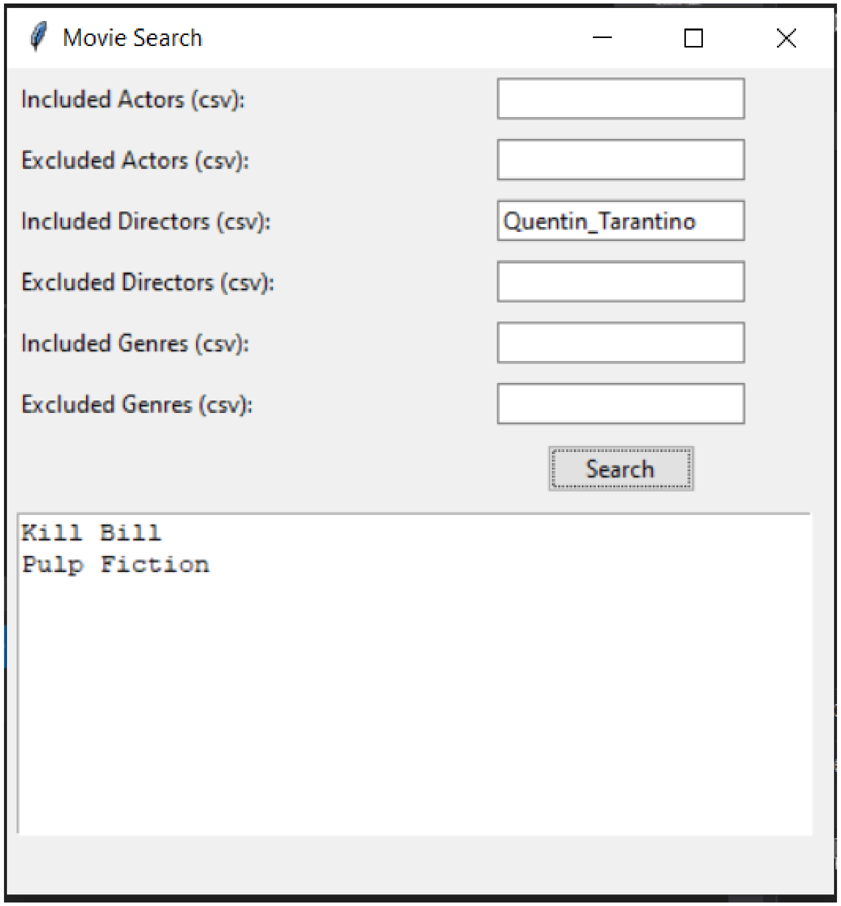
Included Actors (csv): John_Travolta

```
Pulp Fiction
```



Excluded Actors (csv): John_Travolta

```
Baby Driver
Boogie Nights
Inception
Interstellar
Jojo Rabbit
Kill Bill
Shaun Of The Dead
The Dark Knight
The Godfather
The Hangover
```



Included Actors (csv):

Excluded Actors (csv):

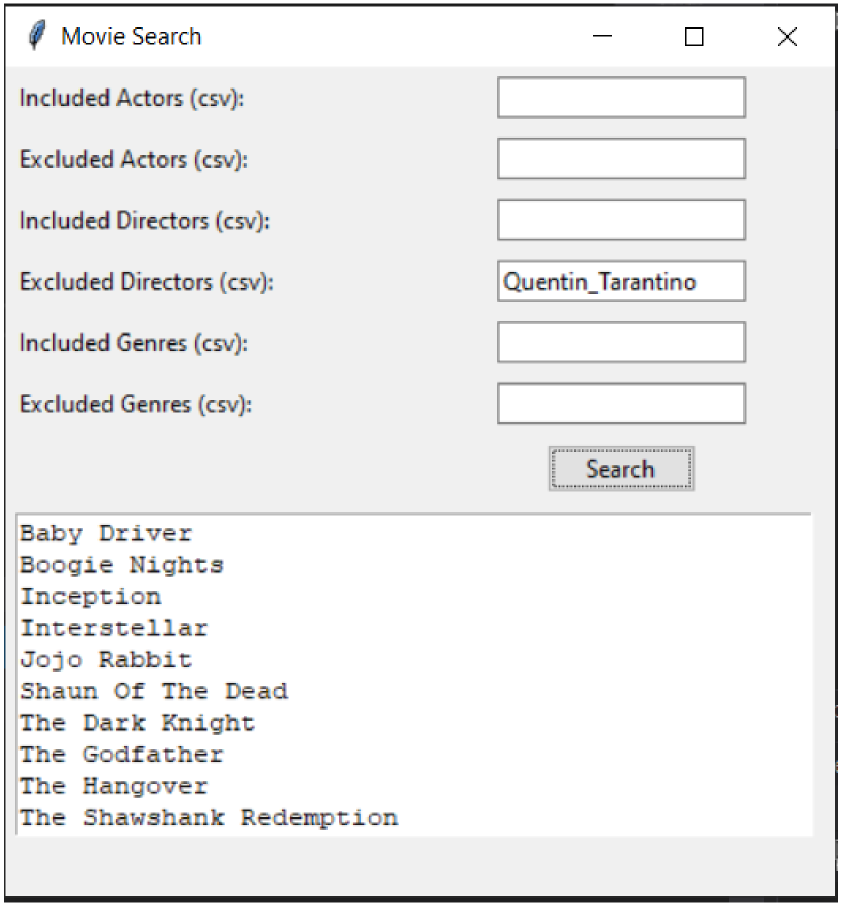
Included Directors (csv): Quentin_Tarantino

Excluded Directors (csv):

Included Genres (csv):

Excluded Genres (csv):

```
Kill Bill
Pulp Fiction
```



Included Actors (csv):

Excluded Actors (csv):

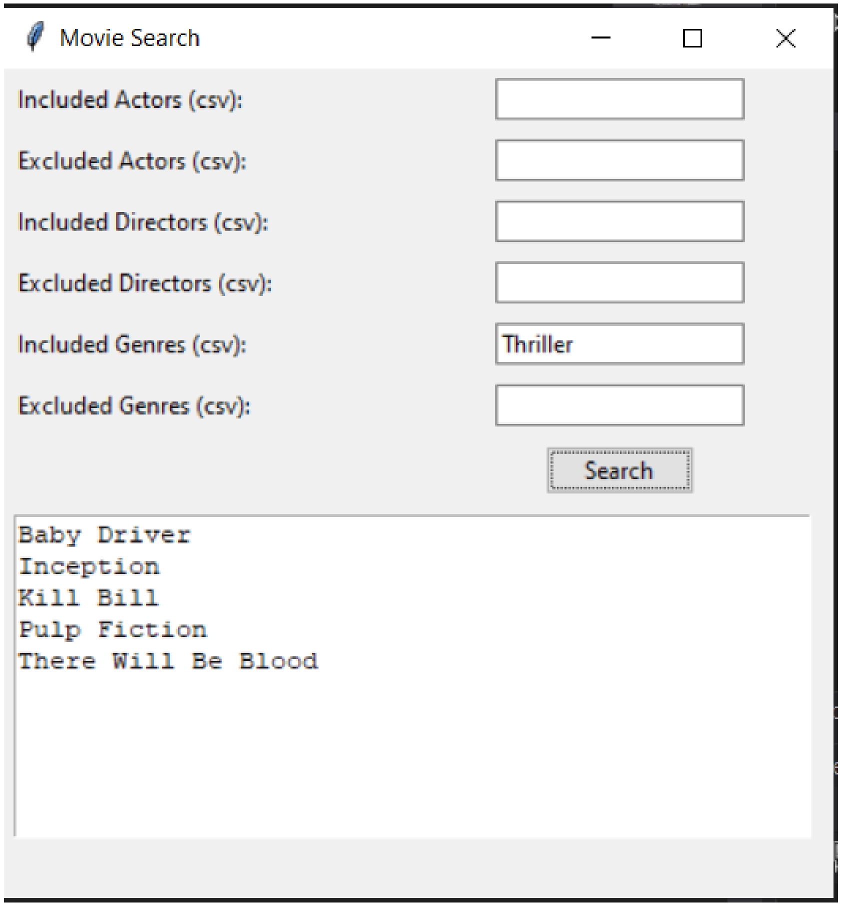
Included Directors (csv):

Excluded Directors (csv): Quentin_Tarantino

Included Genres (csv):

Excluded Genres (csv):

```
Baby Driver
Boogie Nights
Inception
Interstellar
Jojo Rabbit
Shaun Of The Dead
The Dark Knight
The Godfather
The Hangover
The Shawshank Redemption
```



Included Actors (csv):

Excluded Actors (csv):

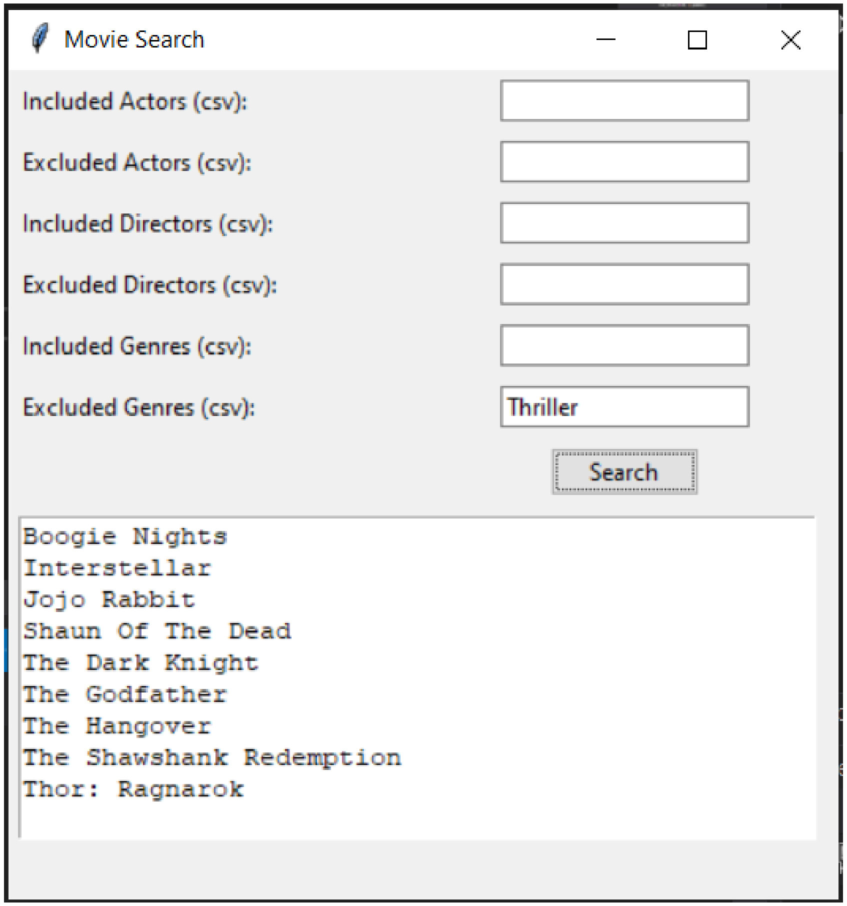
Included Directors (csv):

Excluded Directors (csv):

Included Genres (csv): Thriller

Excluded Genres (csv):

```
Baby Driver
Inception
Kill Bill
Pulp Fiction
There Will Be Blood
```



Included Actors (csv):

Excluded Actors (csv):

Included Directors (csv):

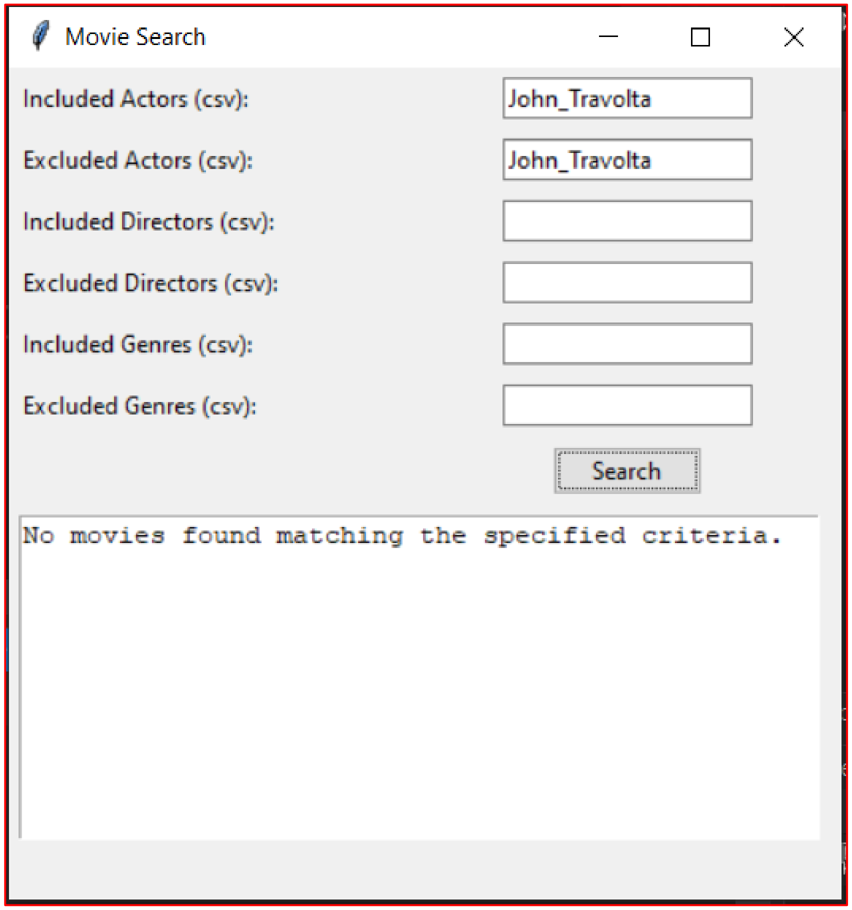
Excluded Directors (csv):

Included Genres (csv):

Excluded Genres (csv): Thriller

```
Boogie Nights
Interstellar
Jojo Rabbit
Shaun Of The Dead
The Dark Knight
The Godfather
The Hangover
The Shawshank Redemption
Thor: Ragnarok
```

3) Test Multiple values:



Included Actors (csv): John_Travolta

Excluded Actors (csv): John_Travolta

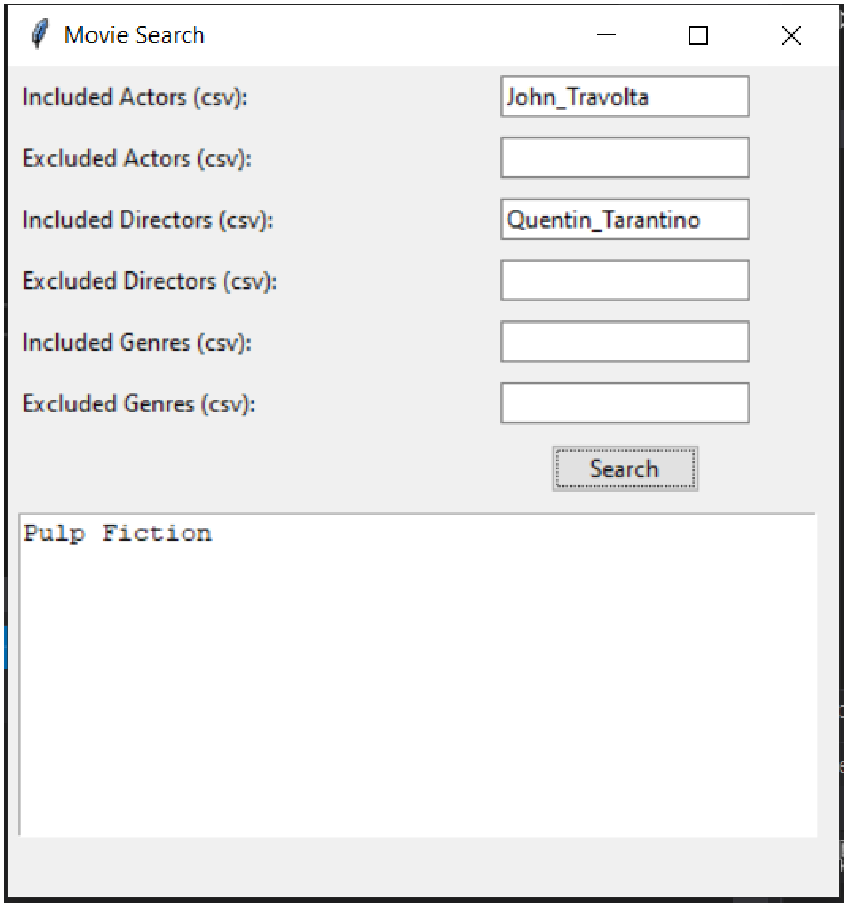
Included Directors (csv):

Excluded Directors (csv):

Included Genres (csv):

Excluded Genres (csv):

```
No movies found matching the specified criteria.
```



Included Actors (csv): John_Travolta

Excluded Actors (csv):

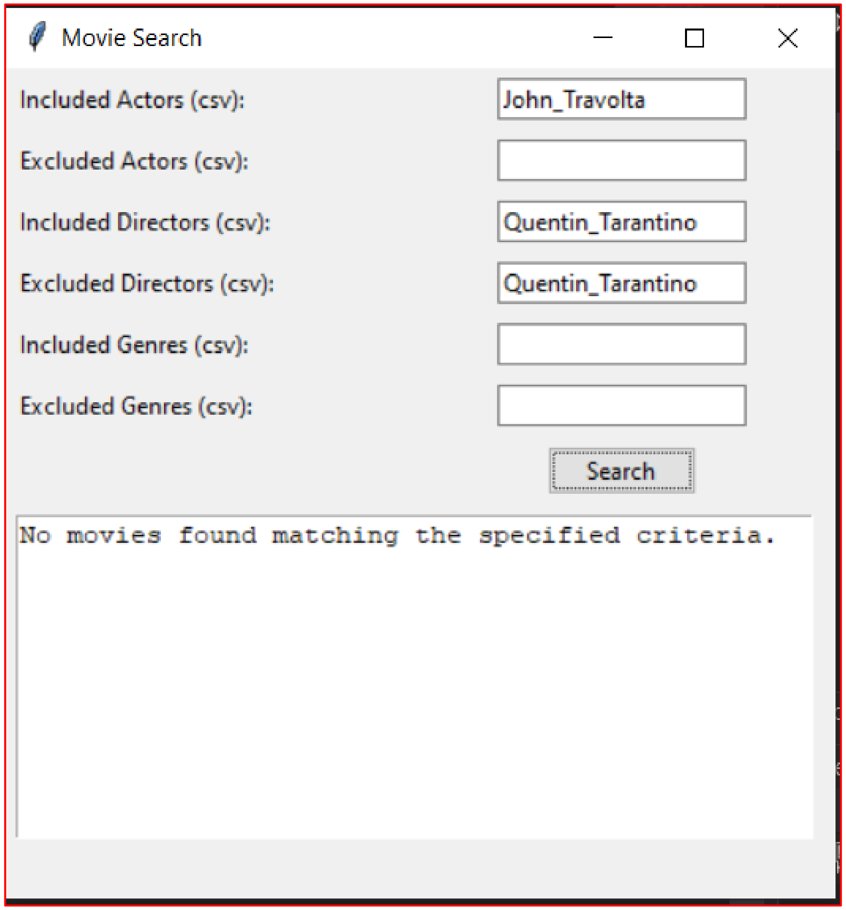
Included Directors (csv): Quentin_Tarantino

Excluded Directors (csv):

Included Genres (csv):

Excluded Genres (csv):

```
Pulp Fiction
```



Included Actors (csv): John_Travolta

Excluded Actors (csv):

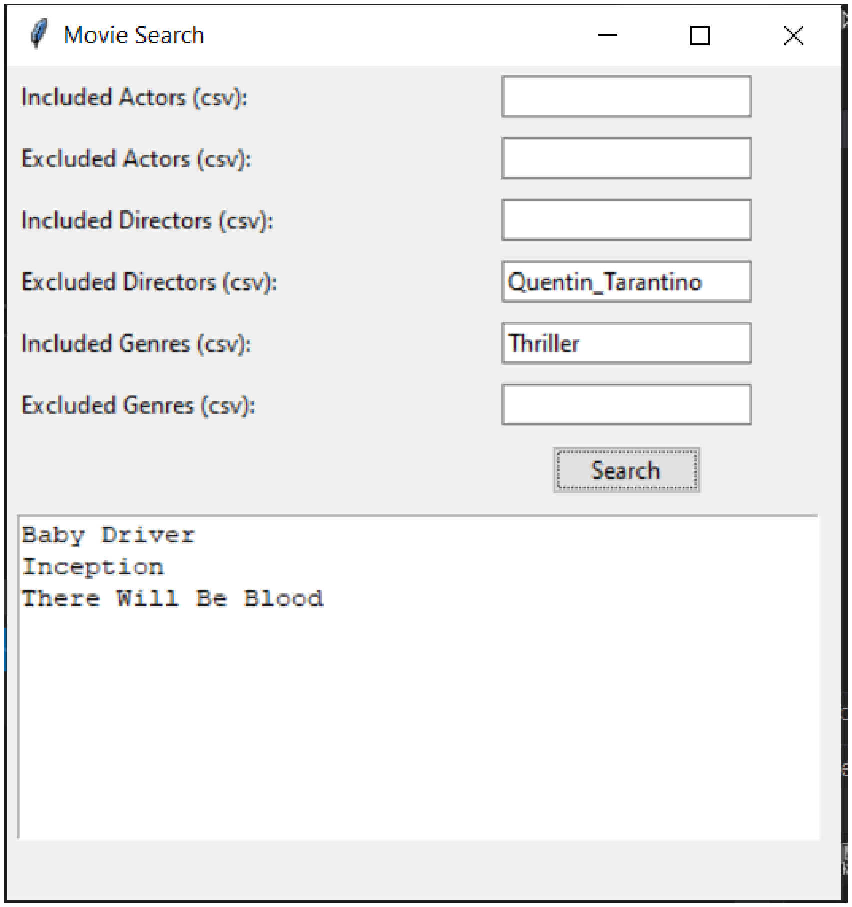
Included Directors (csv): Quentin_Tarantino

Excluded Directors (csv): Quentin_Tarantino

Included Genres (csv):

Excluded Genres (csv):

```
No movies found matching the specified criteria.
```



Included Actors (csv):

Excluded Actors (csv):

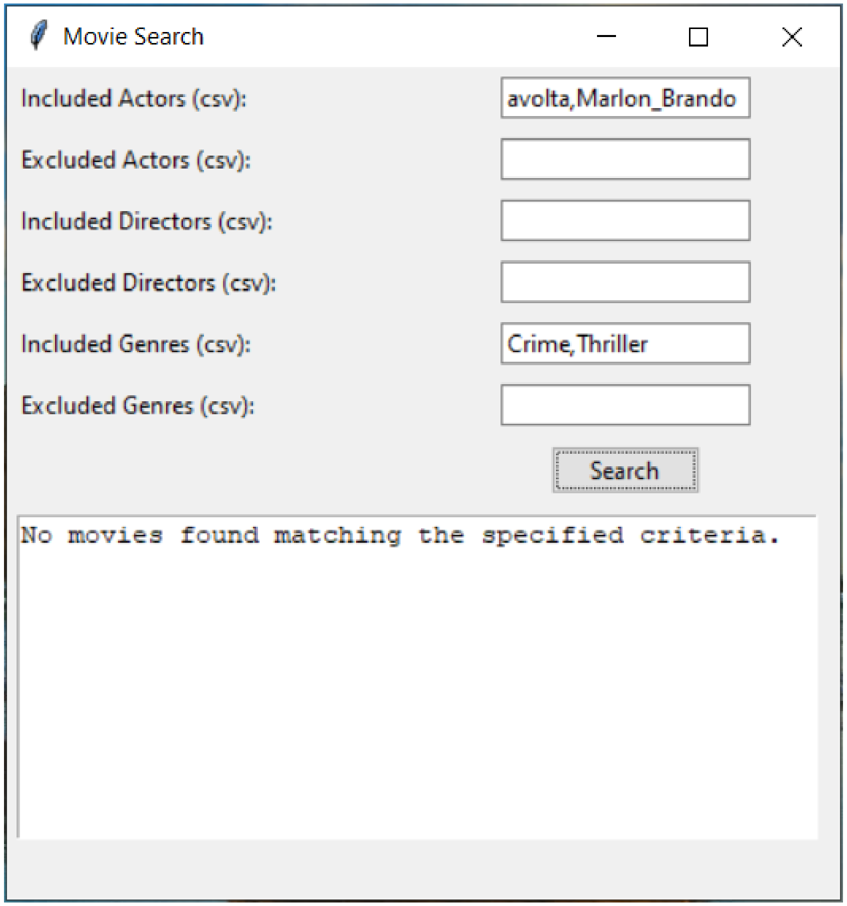
Included Directors (csv):

Excluded Directors (csv): Quentin_Tarantino

Included Genres (csv): Thriller

Excluded Genres (csv):

```
Baby Driver
Inception
There Will Be Blood
```



Included Actors (csv): avolta,Marlon_Brando

Excluded Actors (csv):

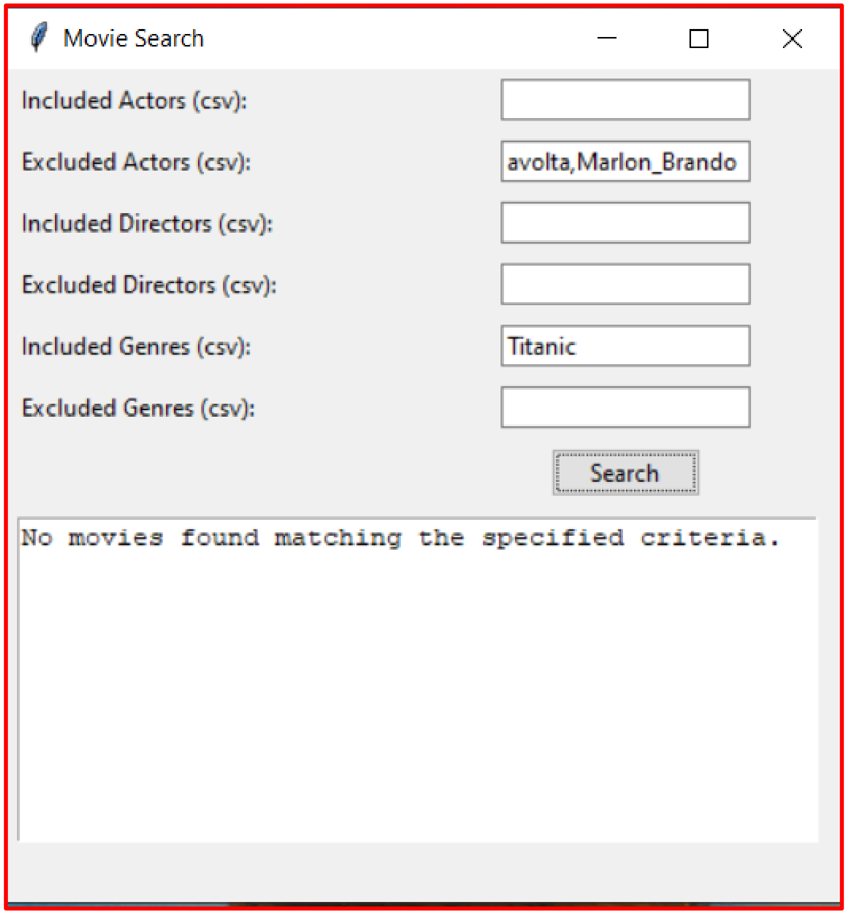
Included Directors (csv):

Excluded Directors (csv):

Included Genres (csv): Crime,Thriller

Excluded Genres (csv):

```
No movies found matching the specified criteria.
```



Included Actors (csv):

Excluded Actors (csv): avolta,Marlon_Brando

Included Directors (csv):

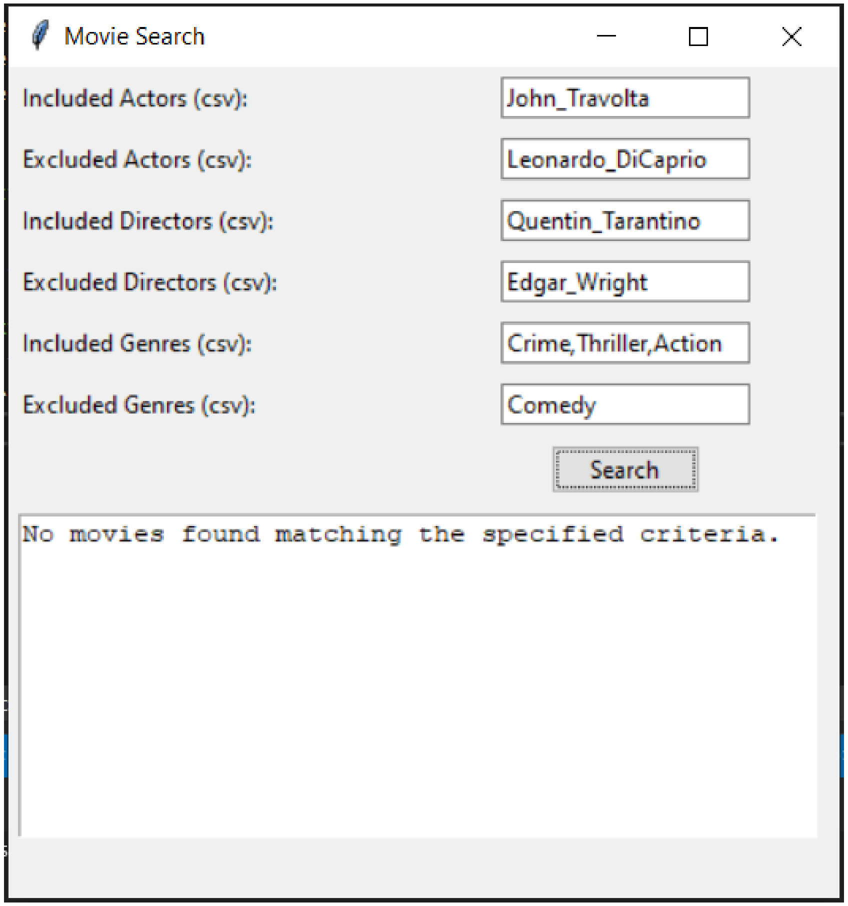
Excluded Directors (csv):

Included Genres (csv): Titanic

Excluded Genres (csv):

```
No movies found matching the specified criteria.
```

4) Test All values:



Included Actors (csv): John_Travolta

Excluded Actors (csv): Leonardo_DiCaprio

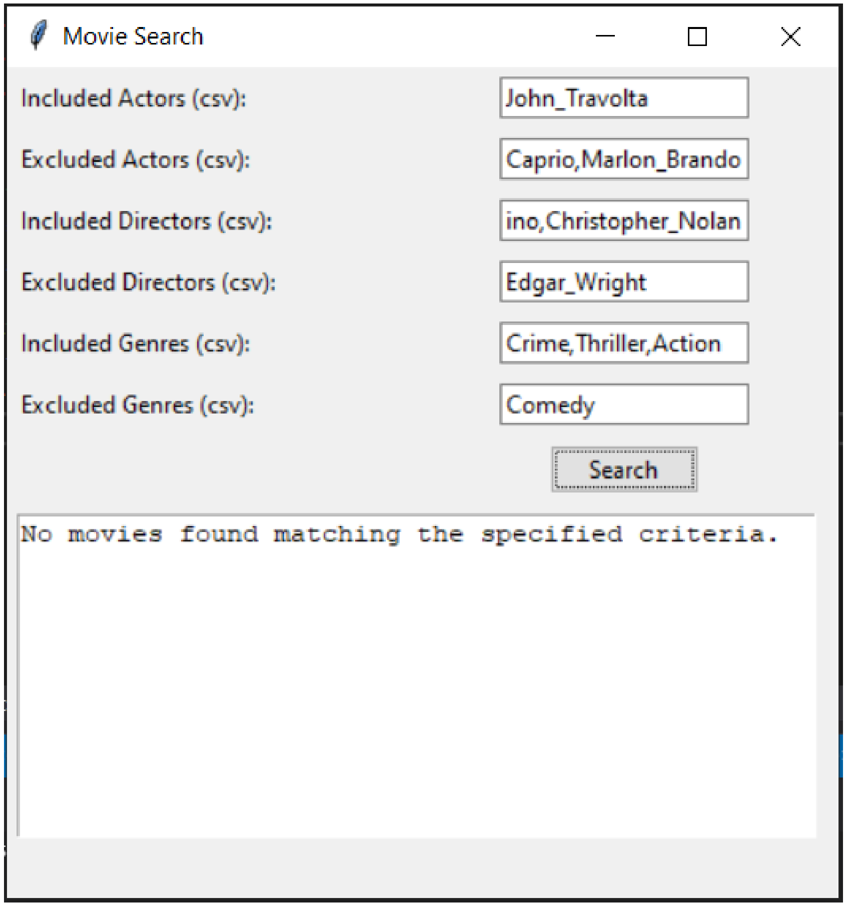
Included Directors (csv): Quentin_Tarantino

Excluded Directors (csv): Edgar_Wright

Included Genres (csv): Crime,Thriller,Action

Excluded Genres (csv): Comedy

```
No movies found matching the specified criteria.
```



Included Actors (csv): John_Travolta

Excluded Actors (csv): Caprio,Marlon_Brando

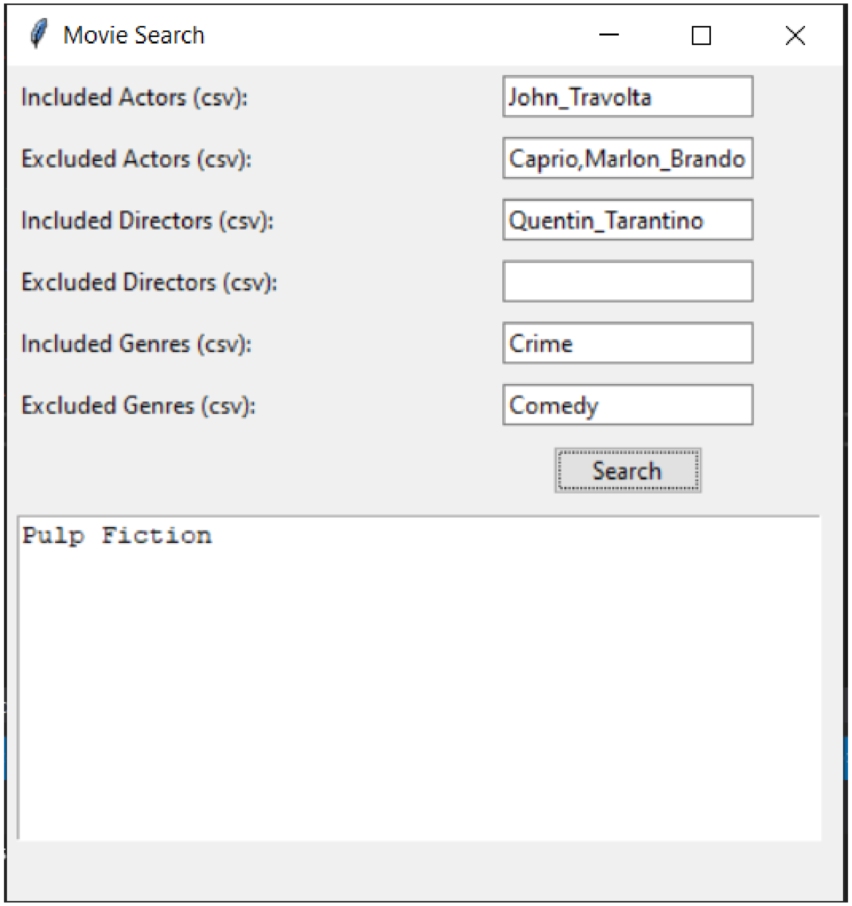
Included Directors (csv): ino,Christopher_Nolan

Excluded Directors (csv): Edgar_Wright

Included Genres (csv): Crime,Thriller,Action

Excluded Genres (csv): Comedy

```
No movies found matching the specified criteria.
```



Included Actors (csv): John_Travolta

Excluded Actors (csv): Caprio,Marlon_Brando

Included Directors (csv): Quentin_Tarantino

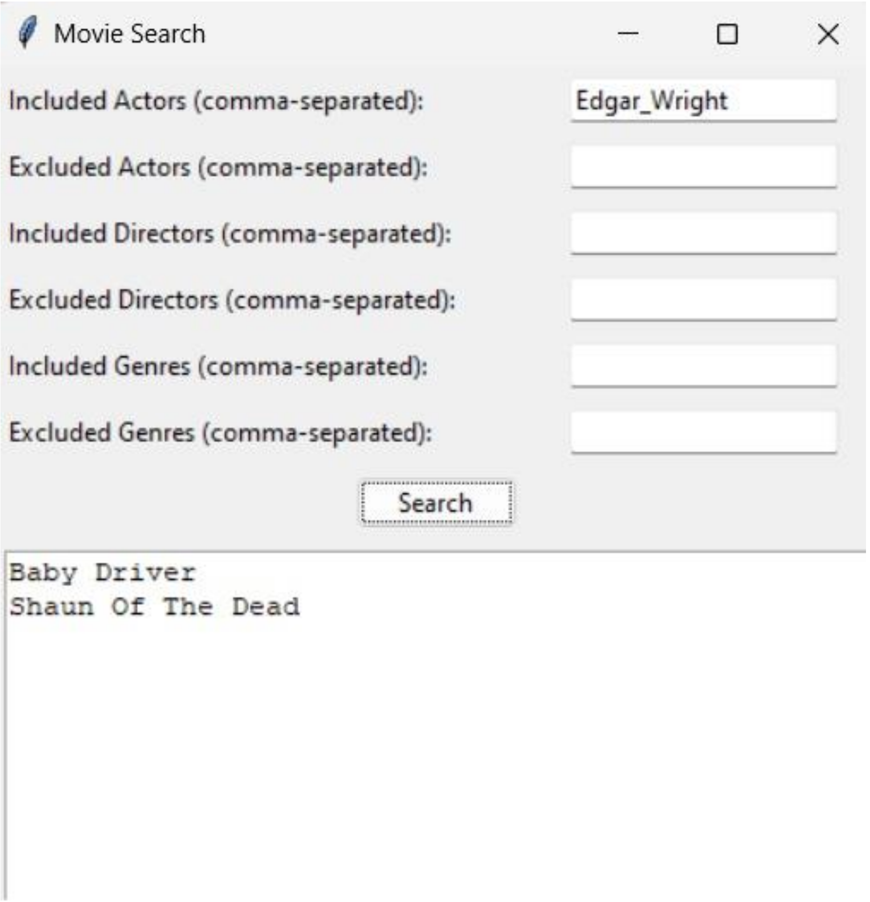
Excluded Directors (csv):

Included Genres (csv): Crime

Excluded Genres (csv): Comedy

```
Pulp Fiction
```

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



Included Actors (comma-separated): Edgar_Wright

Excluded Actors (comma-separated):

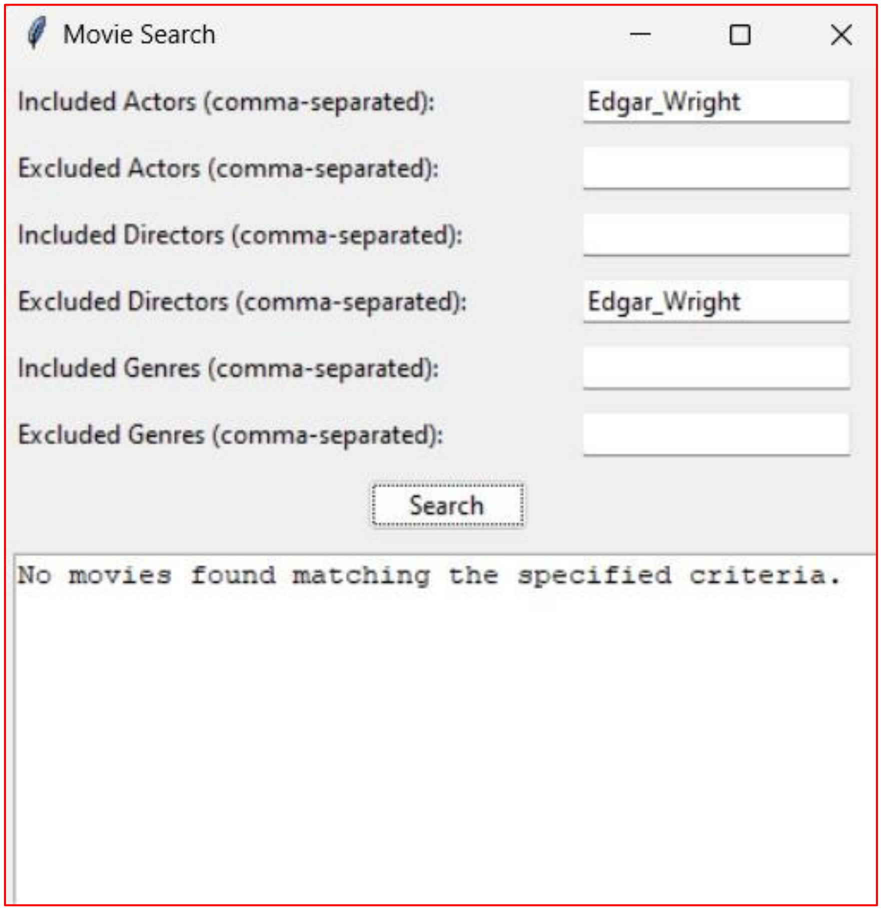
Included Directors (comma-separated):

Excluded Directors (comma-separated):

Included Genres (comma-separated):

Excluded Genres (comma-separated):

```
Baby Driver
Shaun Of The Dead
```



Included Actors (comma-separated): Edgar_Wright

Excluded Actors (comma-separated):

Included Directors (comma-separated):

Excluded Directors (comma-separated): Edgar_Wright

Included Genres (comma-separated):

Excluded Genres (comma-separated):

```
No movies found matching the specified criteria.
```