MOVIES ONTOLOGIES

CSE488- Ontologies and the Semantic Web

Team 15 members:

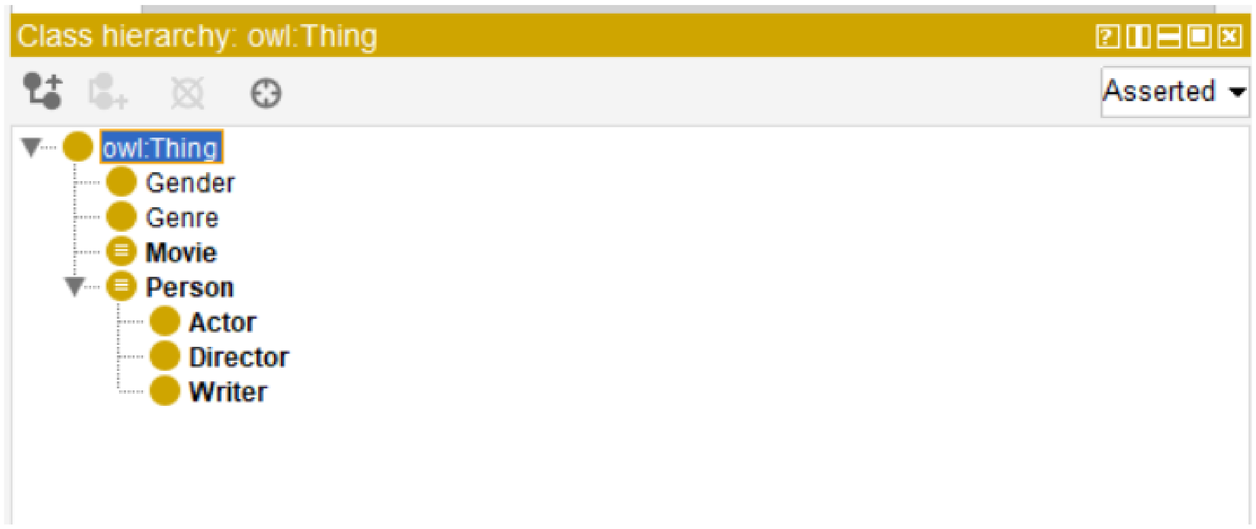
- Ahmed Gamil Fathy	19P4664
- Ahmed Osama Noaman	19P7926
- Mohamed Ehab Mansour	19P5241

Table of Contents

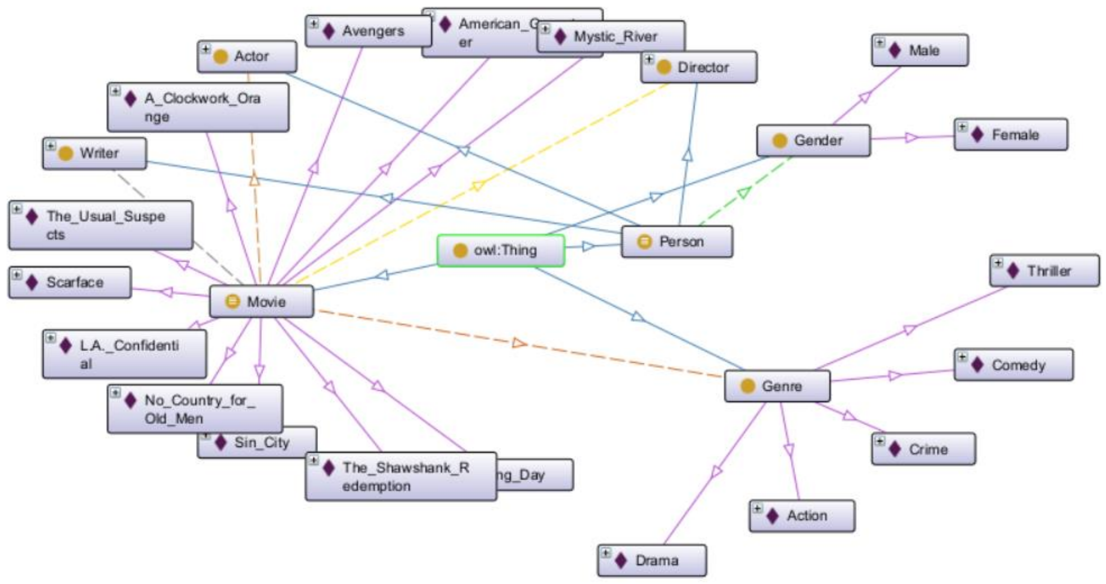
The Tree Hierarchy

Classes: Person, Movie, and Genre.

Subclasses: Actor, Director, Writer

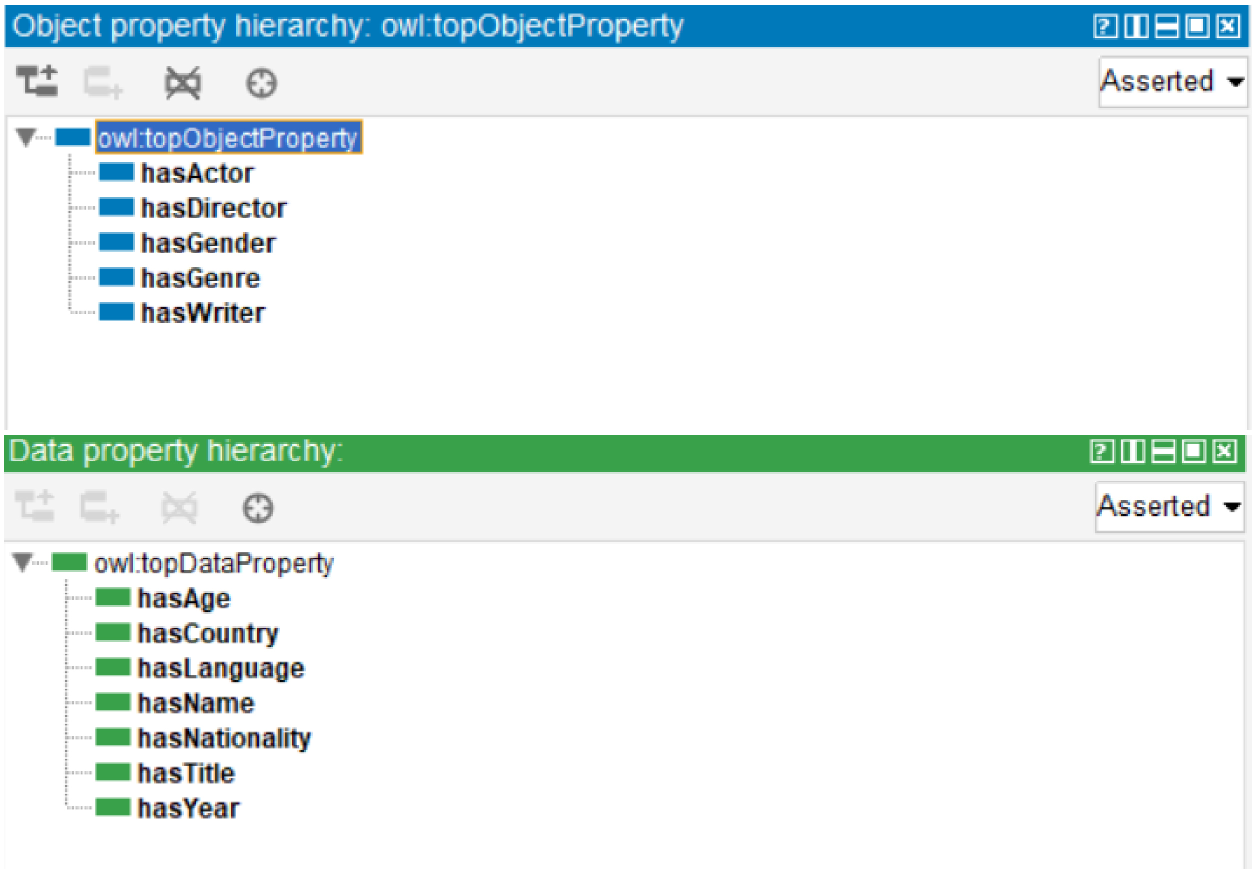


OntoGraf:


Avengers
American_Gangster
Mystic_River
Actor
Director
Male
A_Clockwork_Orange
Writer
Gender
Female
The_Usual_Suspects
owl:Thing
Person
Scarface
Movie
Thriller
L.A._Confidential
Comedy
Genre
No_Country_for_Old_Men
Sin_City
Crime
The_Shawshank_Redemption
ng_Day
Action
Drama

Object and Data Properties:

Object properties represent relationships between individuals or instances of classes in an ontology. These relationships typically involve other individuals or instances as their values. Data properties, on the other hand, represent attributes of individuals or instances in an ontology. These attributes typically have literal values, such as strings, numbers, or dates.

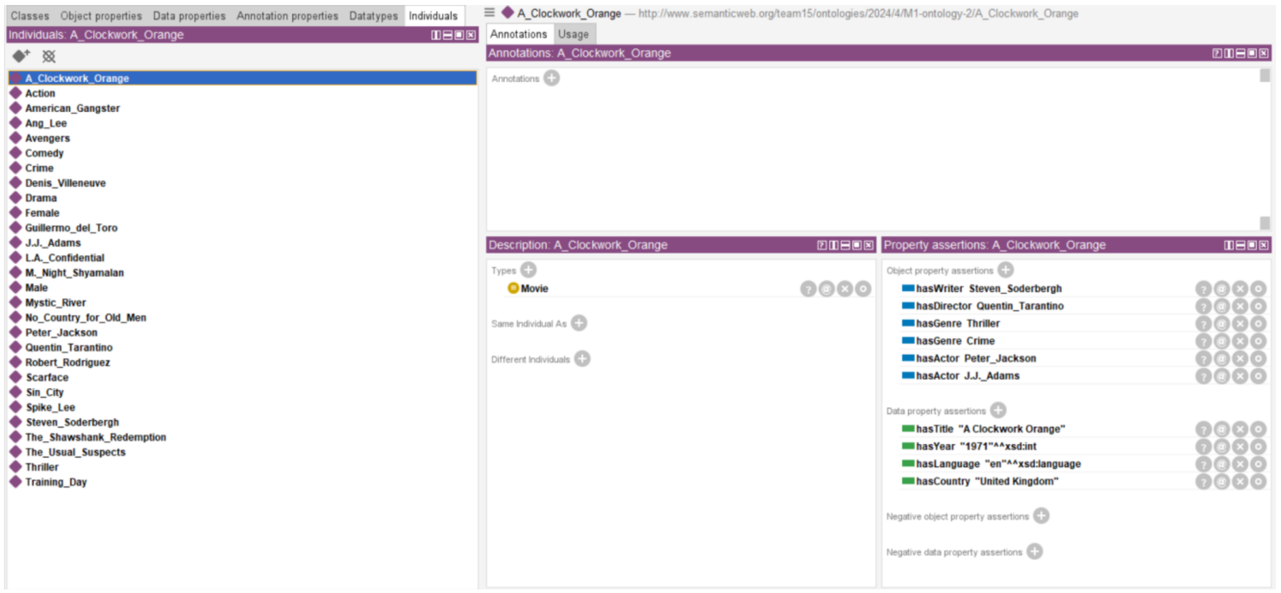


- owl:topObjectProperty
  - hasActor
  - hasDirector
  - hasGender
  - hasGenre
  - hasWriter

- owl:topDataProperty
  - hasAge
  - hasCountry
  - hasLanguage
  - hasName
  - hasNationality
  - hasTitle
  - hasYear

<u>**Populating the Ontology:**</u>

Individuals, or Examples, are added through the Individual tab in Protégé by using the classes, subclasses, and data and object properties.



Individuals: A_Clockwork_Orange

- A_Clockwork_Orange
- Action
- American_Gangster
- Ang_Lee
- Avengers
- Comedy
- Crime
- Denis_Villeneuve
- Drama
- Female
- Guillermo_del_Toro
- J.J._Adams
- L.A._Confidential
- M._Night_Shyamalan
- Male
- Mystic_River
- No_Country_for_Old_Men
- Peter_Jackson
- Quentin_Tarantino
- Robert_Rodriguez
- Scarface
- Sin_City
- Spike_Lee
- Steven_Soderbergh
- The_Shawshank_Redemption
- The_Usual_Suspects
- Thriller
- Training_Day

A_Clockwork_Orange — http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/A_Clockwork_Orange

Annotations: A_Clockwork_Orange

Annotations

Description: A_Clockwork_Orange

Types
- Movie

Same Individual As

Different Individuals

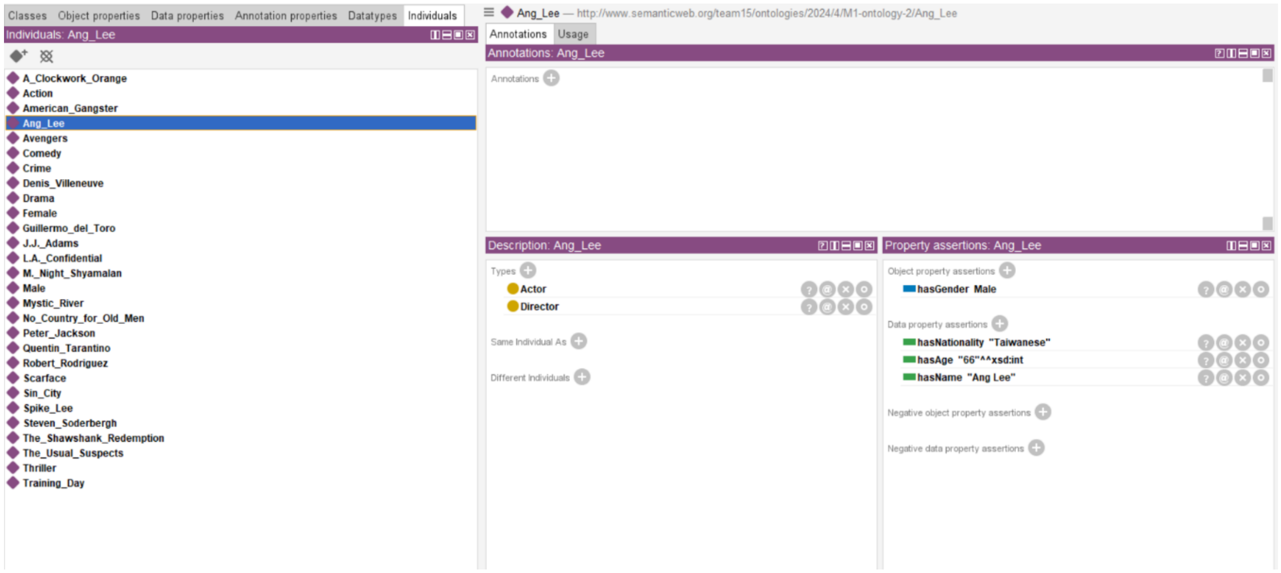
Property assertions: A_Clockwork_Orange

Object property assertions
- hasWriter Steven_Soderbergh
- hasDirector Quentin_Tarantino
- hasGenre Thriller
- hasGenre Crime
- hasActor Peter_Jackson
- hasActor J.J._Adams

Data property assertions
- hasTitle "A Clockwork Orange"
- hasYear "1971"^^xsd:int
- hasLanguage "en"^^xsd:language
- hasCountry "United Kingdom"

Negative object property assertions

Negative data property assertions



Individuals: Ang_Lee

- A_Clockwork_Orange
- Action
- American_Gangster
- Ang_Lee
- Avengers
- Comedy
- Crime
- Denis_Villeneuve
- Drama
- Female
- Guillermo_del_Toro
- J.J._Adams
- L.A._Confidential
- M._Night_Shyamalan
- Male
- Mystic_River
- No_Country_for_Old_Men
- Peter_Jackson
- Quentin_Tarantino
- Robert_Rodriguez
- Scarface
- Sin_City
- Spike_Lee
- Steven_Soderbergh
- The_Shawshank_Redemption
- The_Usual_Suspects
- Thriller
- Training_Day

Ang_Lee — http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee

Annotations: Ang_Lee

Annotations

Description: Ang_Lee

Types
- Actor
- Director

Same Individual As

Different Individuals

Property assertions: Ang_Lee

Object property assertions
- hasGender Male

Data property assertions
- hasNationality "Taiwanese"
- hasAge "66"^^xsd:int
- hasName "Ang Lee"

Negative object property assertions

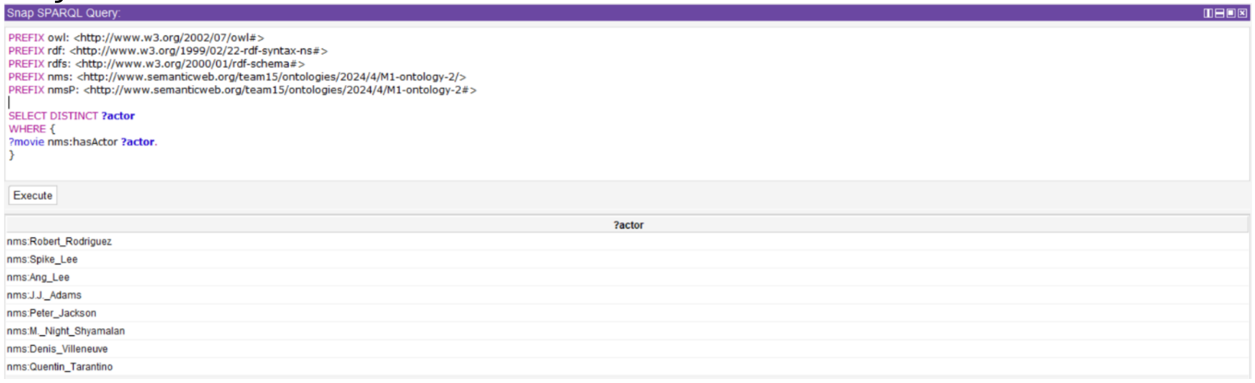
Negative data property assertions

Querying the Ontology:

SPARQL query is used for these examples:

1. List the instances of the class Actor

```
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 nms:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT DISTINCT ?actor
WHERE {
?movie nms:hasActor ?actor.
}
```

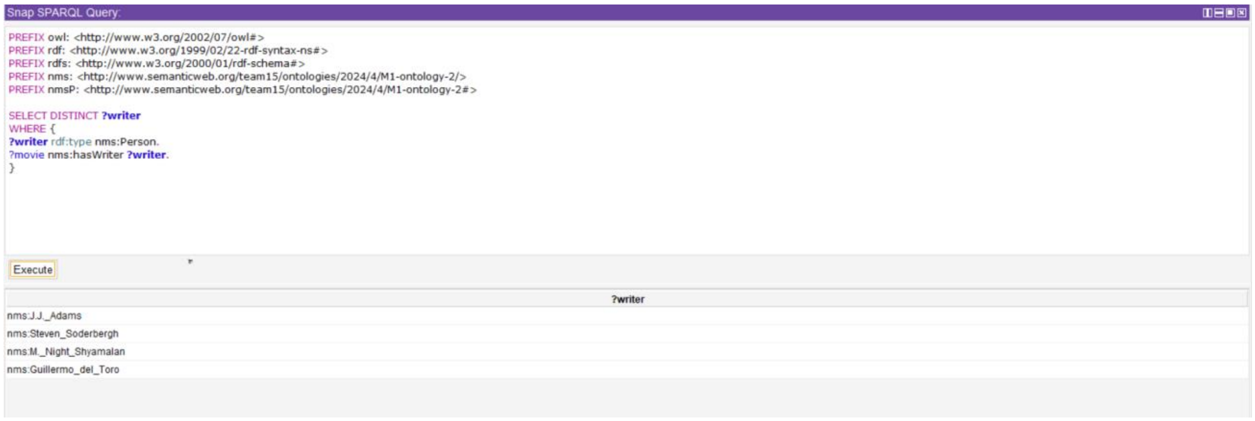


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT DISTINCT ?actor
WHERE {
?movie nms:hasActor ?actor.
}
```

?actor
nms:Robert_Rodriguez
nms:Spike_Lee
nms:Ang_Lee
nms:J.J._Adams
nms:Peter_Jackson
nms:M._Night_Shyamalan
nms:Denis_Villeneuve
nms:Quentin_Tarantino

2. List the instances of the class writer

```
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 nms:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT DISTINCT ?writer
WHERE {
?writer rdf:type nms:Person.
?movie nms:hasWriter ?writer.
}
```

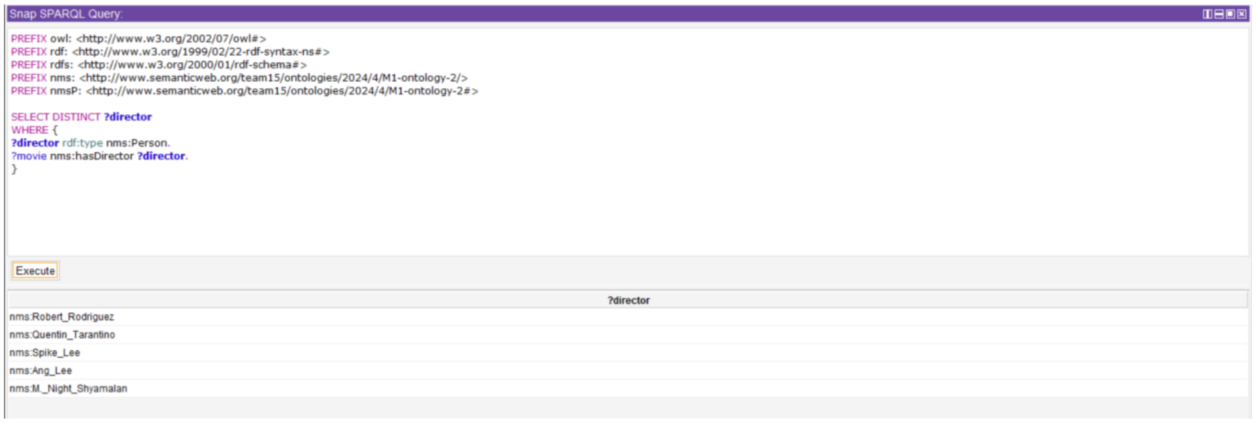


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT DISTINCT ?writer
WHERE {
?writer rdf:type nms:Person.
?movie nms:hasWriter ?writer.
}
```

?writer
nms:J.J._Adams
nms:Steven_Soderbergh
nms:M._Night_Shyamalan
nms:Guillermo_del_Toro

3. List the instances of the class 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 nms:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT DISTINCT ?director
WHERE {
?director rdf:type nms:Person.
?movie nms:hasDirector ?director.
}
```

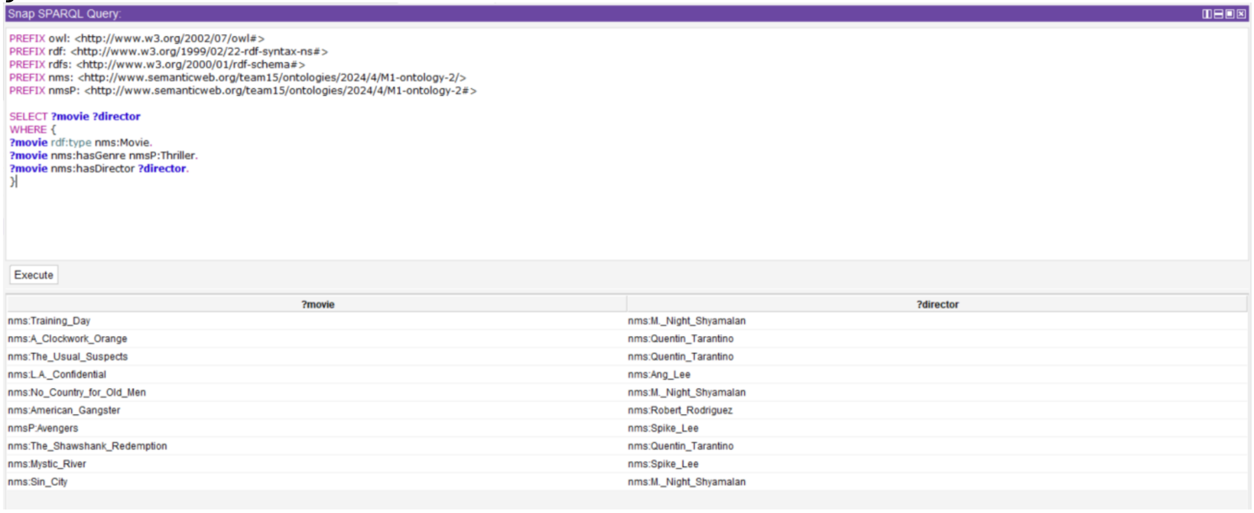

Snap SPARQL Query:

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT DISTINCT ?director
WHERE {
?director rdf:type nms:Person.
?movie nms:hasDirector ?director.
}
```

?director
nms:Robert_Rodriguez
nms:Quentin_Tarantino
nms:Spike_Lee
nms:Ang_Lee
nms:M._Night_Shyamalan

4. List the name of all Thriller movies. For each one, display its 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 nms:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT ?movie ?director
WHERE {
?movie rdf:type nms:Movie.
?movie nms:hasGenre nmsP:Thriller.
?movie nms:hasDirector ?director.
}
```

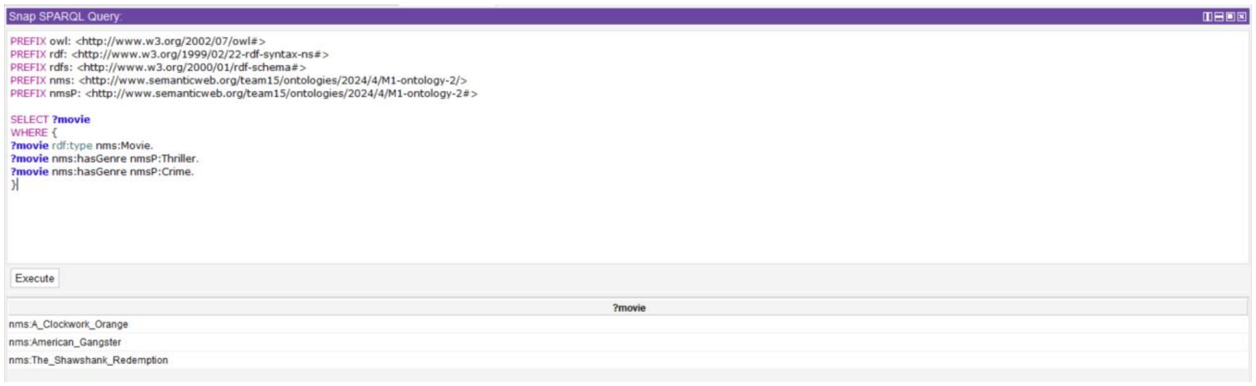


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT ?movie ?director
WHERE {
?movie rdf:type nms:Movie.
?movie nms:hasGenre nmsP:Thriller.
?movie nms:hasDirector ?director.
}
```

?movie	?director
nms:Training_Day	nms:M._Night_Shyamalan
nms:A_Clockwork_Orange	nms:Quentin_Tarantino
nms:The_Usual_Suspects	nms:Quentin_Tarantino
nms:L.A._Confidential	nms:Ang_Lee
nms:No_Country_for_Old_Men	nms:M._Night_Shyamalan
nms:American_Gangster	nms:Robert_Rodriguez
nmsP:Avengers	nms:Spike_Lee
nms:The_Shawshank_Redemption	nms:Quentin_Tarantino
nms:Mystic_River	nms:Spike_Lee
nms:Sin_City	nms:M._Night_Shyamalan

5. List the name of all Crime Thriller movies.

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT ?movie
WHERE {
?movie rdf:type nms:Movie.
?movie nms:hasGenre nmsP:Thriller.
?movie nms:hasGenre nmsP:Crime.
}
```

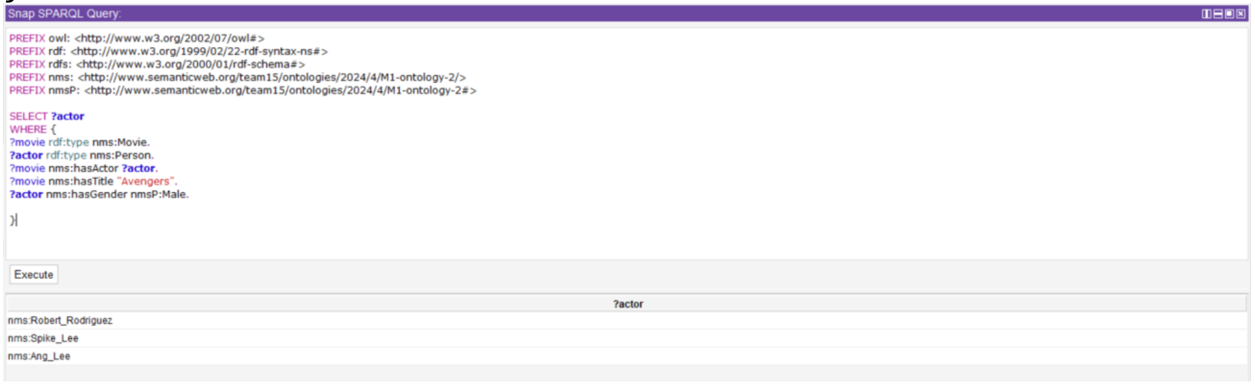


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT ?movie
WHERE {
?movie rdf:type nms:Movie.
?movie nms:hasGenre nmsP:Thriller.
?movie nms:hasGenre nmsP:Crime.
}
```

?movie
nms:A_Clockwork_Orange
nms:American_Gangster
nms:The_Shawshank_Redemption

6. List the male actors in the movie in specific film

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT ?actor
WHERE {
?movie rdf:type nms:Movie.
?actor rdf:type nms:Person.
?movie nms:hasActor ?actor.
?movie nms:hasTitle "Avengers".
?actor nms:hasGender nmsP:Male.

}
```

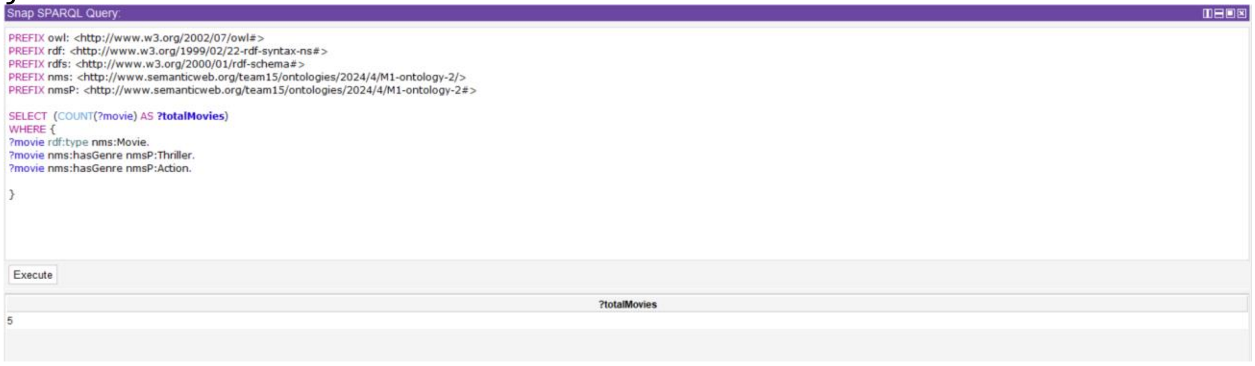


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT ?actor
WHERE {
?movie rdf:type nms:Movie.
?actor rdf:type nms:Person.
?movie nms:hasActor ?actor.
?movie nms:hasTitle "Avengers".
?actor nms:hasGender nmsP:Male.

}
```

?actor
nms:Robert_Rodriguez
nms:Spike_Lee
nms:Ang_Lee

7. How many movies have both "Action" and "Thriller" as 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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT  (COUNT(?movie) AS ?totalMovies)
WHERE {
?movie rdf:type nms:Movie.
?movie nms:hasGenre nmsP:Thriller.
?movie nms:hasGenre nmsP:Action.

}
```

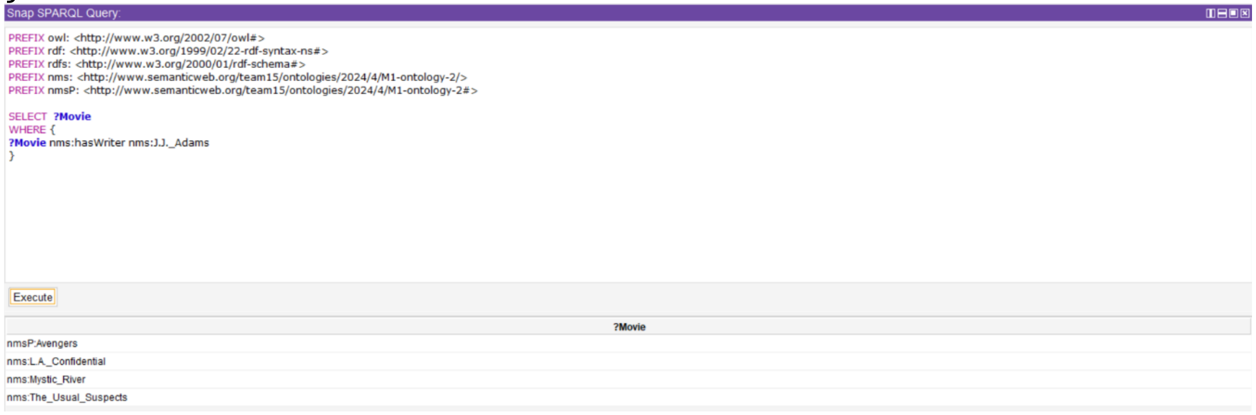


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  (COUNT(?movie) AS ?totalMovies)
WHERE {
?movie rdf:type nms:Movie.
?movie nms:hasGenre nmsP:Thriller.
?movie nms:hasGenre nmsP:Action.

}
```

?totalMovies
5

8. List all the movies written by a specific writer.

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  ?Movie
WHERE {
?Movie nms:hasWriter nms:J.J._Adams
}
```

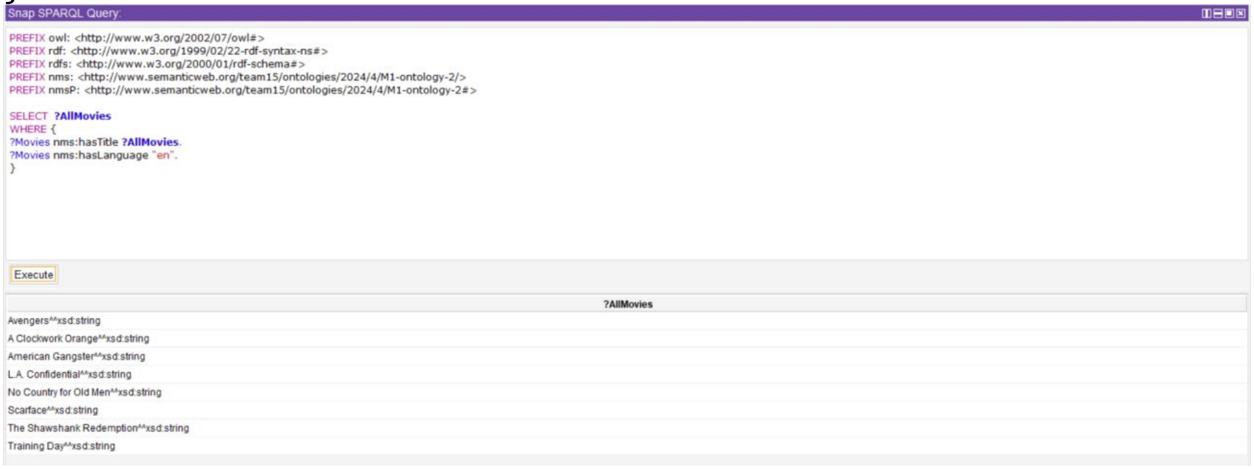


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  ?Movie
WHERE {
?Movie nms:hasWriter nms:J.J._Adams
}
```

?Movie
nmsP:Avengers
nms:L.A._Confidential
nms:Mystic_River
nms:The_Usual_Suspects

9. Find movies with a certain language.

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT  ?AllMovies
WHERE {
?Movies nms:hasTitle ?AllMovies.
?Movies nms:hasLanguage "en".
}
```



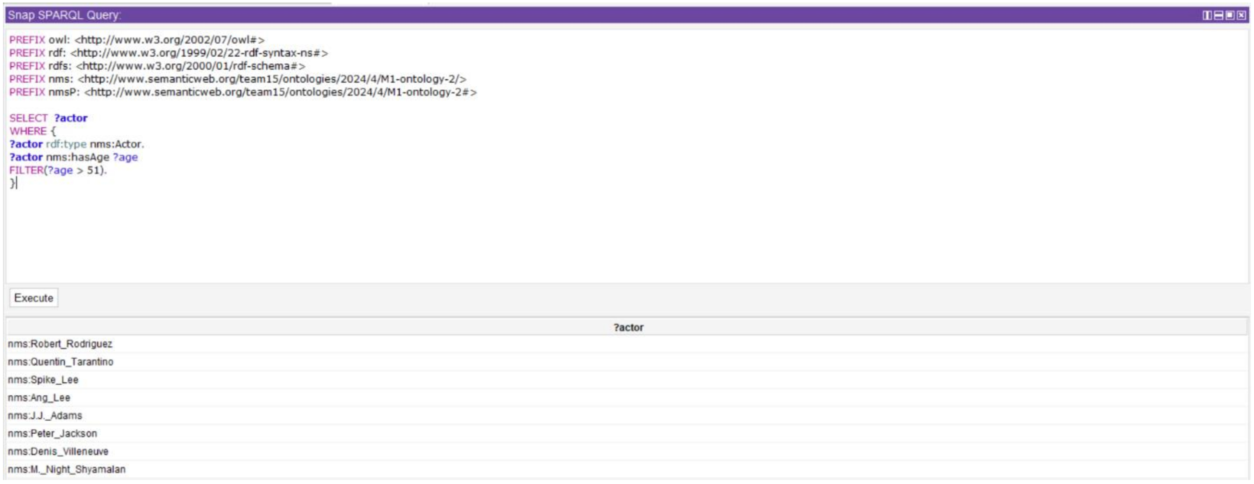
Snap SPARQL Query:

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  ?AllMovies
WHERE {
?Movies nms:hasTitle ?AllMovies.
?Movies nms:hasLanguage "en".
}
```

?AllMovies
Avengers^^xsd:string
A Clockwork Orange^^xsd:string
American Gangster^^xsd:string
L.A. Confidential^^xsd:string
No Country for Old Men^^xsd:string
Scarface^^xsd:string
The Shawshank Redemption^^xsd:string
Training Day^^xsd:string

10. List the name of Actors older than 51 years.

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT  ?actor
WHERE {
?actor rdf:type nms:Actor.
?actor nms:hasAge ?age
FILTER(?age > 51).
}
```



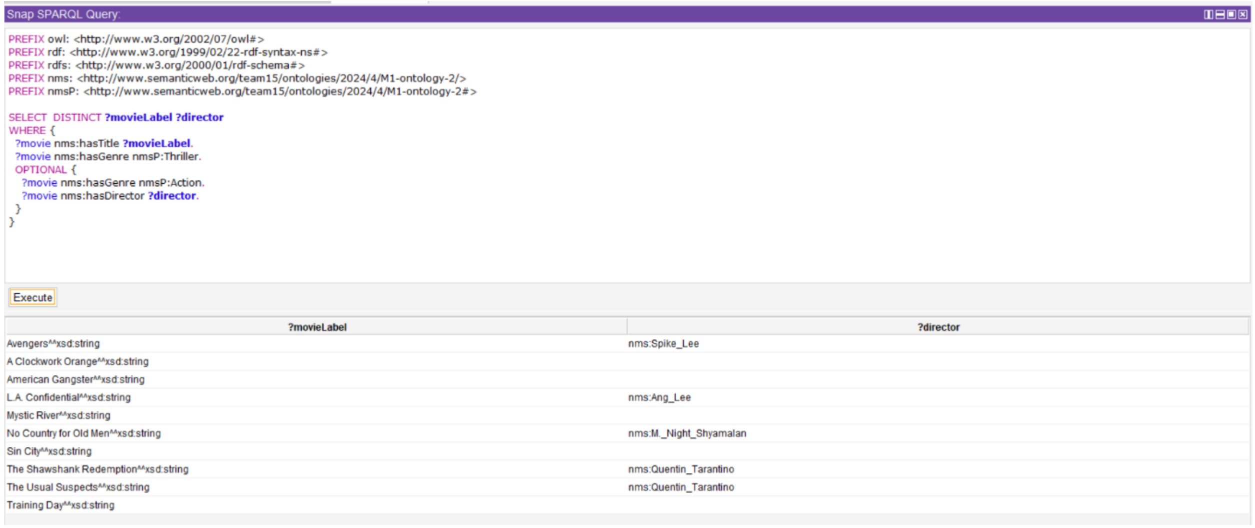
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  ?actor
WHERE {
?actor rdf:type nms:Actor.
?actor nms:hasAge ?age
FILTER(?age > 51).
}
```

?actor
nms:Robert_Rodriguez
nms:Quentin_Tarantino
nms:Spike_Lee
nms:Ang_Lee
nms:J.J._Adams
nms:Peter_Jackson
nms:Denis_Villeneuve
nms:M._Night_Shyamalan

More SPARQL Queries:

1. Output all the thriller movies, and if one of those thrillers is Action then also print its 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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  DISTINCT ?movieLabel ?director
WHERE {
  ?movie nms:hasTitle ?movieLabel.
  ?movie nms:hasGenre nmsP:Thriller.
  OPTIONAL {
    ?movie nms:hasGenre nmsP:Action.
    ?movie nms:hasDirector ?director.
  }
}
```

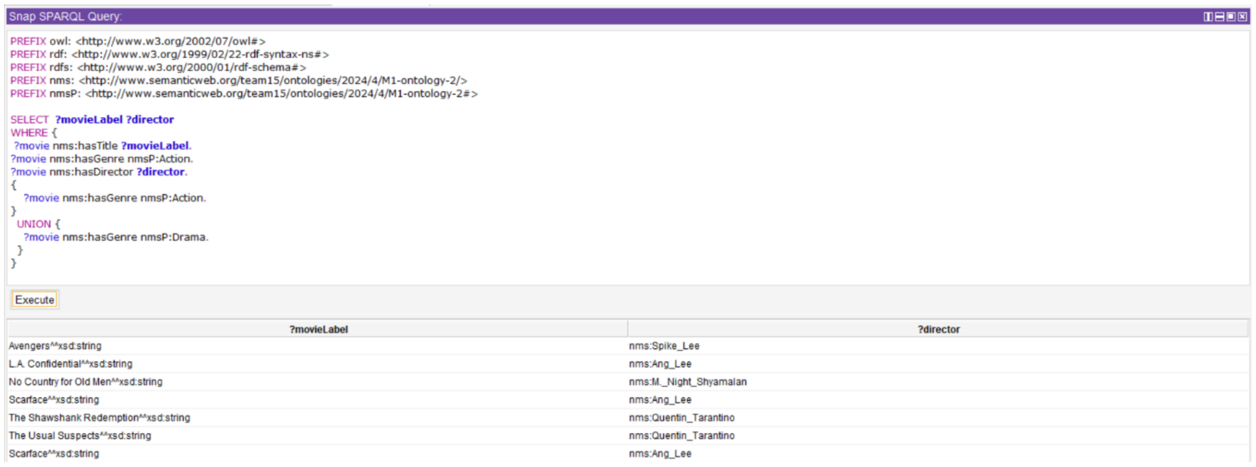


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  DISTINCT ?movieLabel ?director
WHERE {
  ?movie nms:hasTitle ?movieLabel.
  ?movie nms:hasGenre nmsP:Thriller.
  OPTIONAL {
    ?movie nms:hasGenre nmsP:Action.
    ?movie nms:hasDirector ?director.
  }
}
```

?movieLabel	?director
Avengers^^xsd:string	nms:Spike_Lee
A Clockwork Orange^^xsd:string	
American Gangster^^xsd:string	
L.A. Confidential^^xsd:string	nms:Ang_Lee
Mystic River^^xsd:string	
No Country for Old Men^^xsd:string	nms:M._Night_Shyamalan
Sin City^^xsd:string	
The Shawshank Redemption^^xsd:string	nms:Quentin_Tarantino
The Usual Suspects^^xsd:string	nms:Quentin_Tarantino
Training Day^^xsd:string	

2. List the movie titles and their respective directors for movies that belong to the genre "Action" or "Drama."

```
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 nms:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT  ?movieLabel ?director
WHERE {
  ?movie nms:hasTitle ?movieLabel.
  ?movie nms:hasGenre nmsP:Action.
  ?movie nms:hasDirector ?director.
  {
    ?movie nms:hasGenre nmsP:Action.
  }
  UNION {
    ?movie nms:hasGenre nmsP:Drama.
  }
}
```

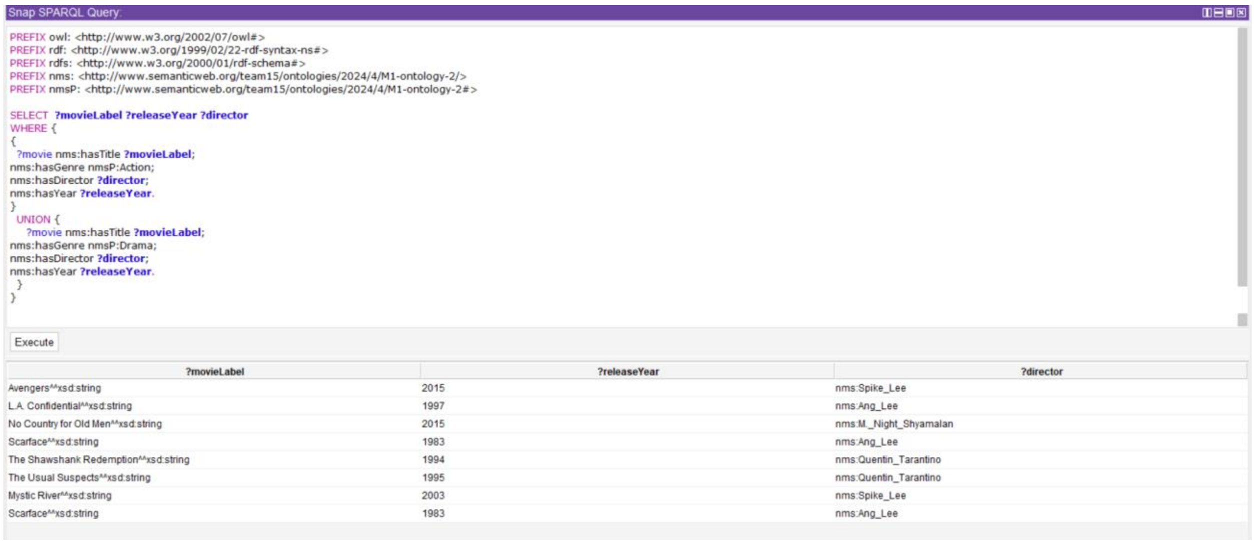


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  ?movieLabel ?director
WHERE {
 ?movie nms:hasTitle ?movieLabel.
?movie nms:hasGenre nmsP:Action.
?movie nms:hasDirector ?director.
{
   ?movie nms:hasGenre nmsP:Action.
}
 UNION {
   ?movie nms:hasGenre nmsP:Drama.
  }
}
```

?movieLabel	?director
Avengers^^xsd:string	nms:Spike_Lee
L.A. Confidential^^xsd:string	nms:Ang_Lee
No Country for Old Men^^xsd:string	nms:M._Night_Shyamalan
Scarface^^xsd:string	nms:Ang_Lee
The Shawshank Redemption^^xsd:string	nms:Quentin_Tarantino
The Usual Suspects^^xsd:string	nms:Quentin_Tarantino
Scarface^^xsd:string	nms:Ang_Lee

3. List movie titles, release years, and directors for movies that belong to either the "Action" or "Drama"

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT  ?movieLabel ?releaseYear ?director
WHERE {
{
  ?movie nms:hasTitle ?movieLabel;
nms:hasGenre nmsP:Action;
nms:hasDirector ?director;
nms:hasYear ?releaseYear.
}
  UNION {
     ?movie nms:hasTitle ?movieLabel;
nms:hasGenre nmsP:Drama;
nms:hasDirector ?director;
nms:hasYear ?releaseYear.
  }
}
```

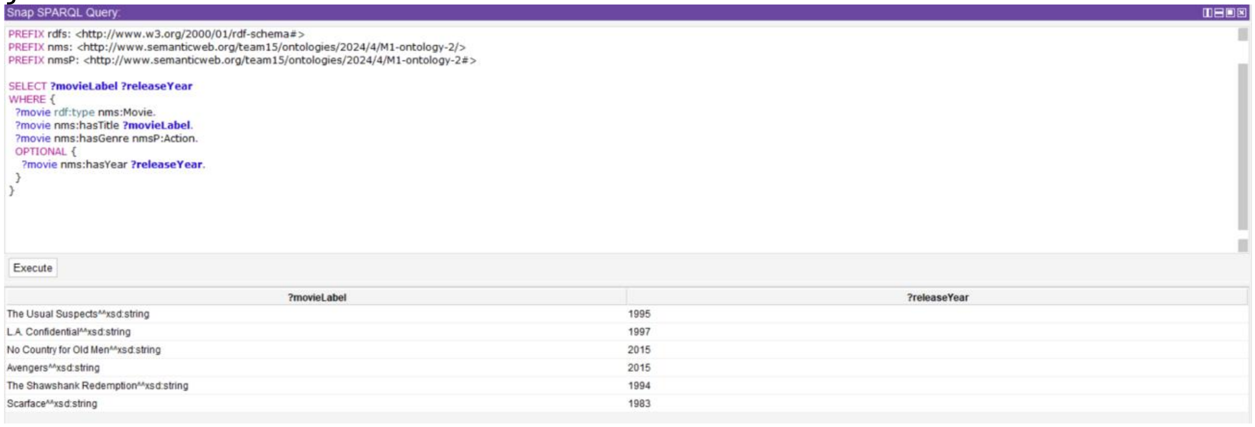


```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT  ?movieLabel ?releaseYear ?director
WHERE {
{
  ?movie nms:hasTitle ?movieLabel;
nms:hasGenre nmsP:Action;
nms:hasDirector ?director;
nms:hasYear ?releaseYear.
}
  UNION {
     ?movie nms:hasTitle ?movieLabel;
nms:hasGenre nmsP:Drama;
nms:hasDirector ?director;
nms:hasYear ?releaseYear.
  }
}
```

?movieLabel	?releaseYear	?director
Avengers^^xsd:string	2015	nms:Spike_Lee
L.A. Confidential^^xsd:string	1997	nms:Ang_Lee
No Country for Old Men^^xsd:string	2015	nms:M._Night_Shyamalan
Scarface^^xsd:string	1983	nms:Ang_Lee
The Shawshank Redemption^^xsd:string	1994	nms:Quentin_Tarantino
The Usual Suspects^^xsd:string	1995	nms:Quentin_Tarantino
Mystic River^^xsd:string	2003	nms:Spike_Lee
Scarface^^xsd:string	1983	nms:Ang_Lee

4. List movie titles that have genre action, and their release year if there is any.

```
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 nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/>
PREFIX nmsP:
<http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-
2#>

SELECT ?movieLabel ?releaseYear
WHERE {
  ?movie rdf:type nms:Movie.
  ?movie nms:hasTitle ?movieLabel.
  ?movie nms:hasGenre nmsP:Action.
  OPTIONAL {
    ?movie nms:hasYear ?releaseYear.
  }
}
```



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX nms: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/>
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT ?movieLabel ?releaseYear
WHERE {
  ?movie rdf:type nms:Movie.
  ?movie nms:hasTitle ?movieLabel.
  ?movie nms:hasGenre nmsP:Action.
  OPTIONAL {
    ?movie nms:hasYear ?releaseYear.
  }
}
```

?movieLabel	?releaseYear
The Usual Suspects^^xsd:string	1995
L.A. Confidential^^xsd:string	1997
No Country for Old Men^^xsd:string	2015
Avengers^^xsd:string	2015
The Shawshank Redemption^^xsd:string	1994
Scarface^^xsd:string	1983

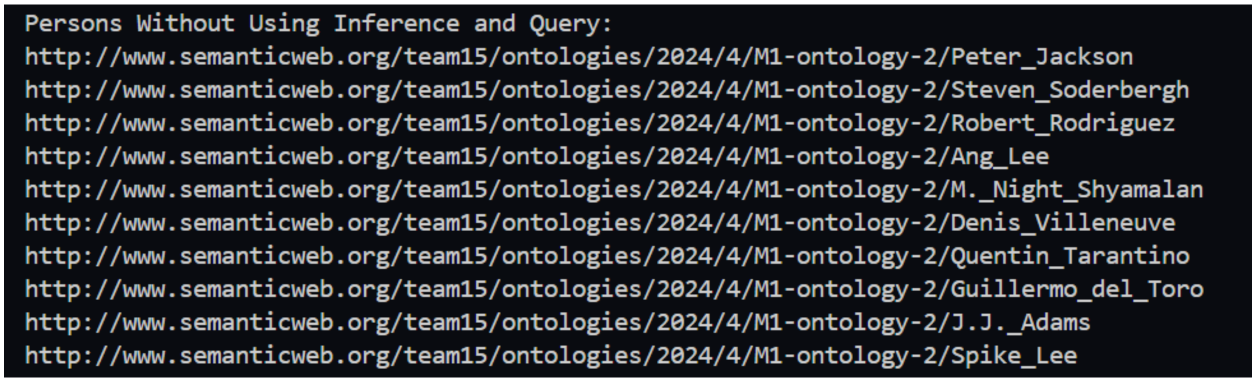
<u>Manipulating The Ontology in Python</u>

Ontology1

Code

```python
from rdflib import Graph
from rdflib.tools.rdf2dot import rdf2dot

def load_ontology(file_path):
    # Load the TTL file into an RDF graph
    g = Graph()
    g.parse(file_path, format='ttl')
    return g

def get_persons(graph):
    persons = set()
    # Iterate through all triples in the graph
    for subj, pred, obj in graph:
        # Check if the predicate is 'rdf:type' and the object is 'Person'
        if pred.endswith('type') and (str(obj).endswith('Actor') or 
str(obj).endswith('Writer') or str(obj).endswith('Director')):
            persons.add(subj)
    return persons

def display_persons(file_path):
    # Load the ontology
    graph = load_ontology(file_path)
    # Get all the Persons from the ontology
    persons = get_persons(graph)
    # Display the Persons
    print("Persons Without Using Inference and Query:")
    for person in persons:
        print(person)

def visualize_graph(file_path, output_file):
    # Load the ontology
    graph = load_ontology(file_path)
    # Create a stream to write DOT data
    with open(output_file, 'w') as f:
        # Convert RDF graph to DOT format and write to the stream
        rdf2dot(graph, f)

# Example usage
if __name__ == "__main__":
    ttl_file = "Ontology_phase1_team15.ttl"  # Path to your TTL file
    output_dot_file = "graph.dot"  # Output DOT file
    display_persons(ttl_file)
    visualize_graph(ttl_file, output_dot_file)
```

output:



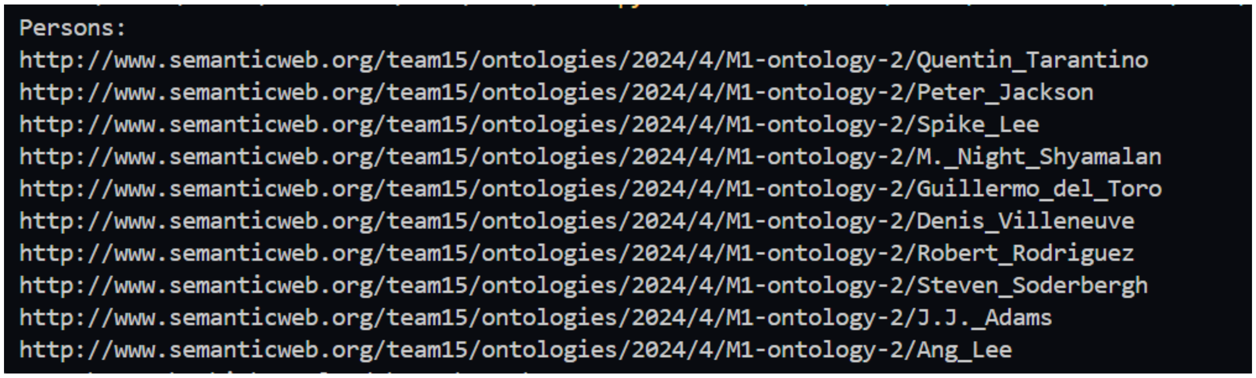
```
Persons Without Using Inference and Query:
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Peter_Jackson
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Steven_Soderbergh
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Robert_Rodriguez
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Denis_Villeneuve
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Quentin_Tarantino
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Guillermo_del_Toro
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Spike_Lee
```

Ontology2

Code

```
from rdflib import Graph, Namespace
from rdflib.plugins.sparql import prepareQuery

# Define namespaces
BASE = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/")
RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

def load_ontology(file_path):
    # Load the TTL file into an RDF graph
    g = Graph()
    g.parse(file_path, format='ttl')
    return g

def read_query_from_file(query_file):
    # Read the SPARQL query from the file
    with open(query_file, 'r') as f:
        query_text = f.read()
    return query_text

def get_persons_with_query(graph, query_text):
    persons = set()
    # Prepare a SPARQL query from the text
    query = prepareQuery(query_text, initNs={"rdf": RDF, "base": BASE})
    # Execute the query and collect the results
    results = graph.query(query)
    for row in results:
        persons.add(row.person)
    return persons

def display_persons_with_query(file_path, query_file):
    # Load the ontology
    graph = load_ontology(file_path)
    # Read the SPARQL query from the file
    query_text = read_query_from_file(query_file)
    # Get all the Persons who are actors, directors, or writers using SPARQL
query
    persons = get_persons_with_query(graph, query_text)
    # Display the Persons
    print("Persons:")
    for person in persons:
        print(person)

# Example usage
if __name__ == "__main__":
    ttl_file = "Ontology_phase1_team15.ttl"  # Path to your TTL file
    query_file = "query.txt"    # Path to your SPARQL query text file
    display_persons_with_query(ttl_file, query_file)

query file
SELECT ?person
WHERE {
    {
        ?person rdf:type base:Actor .
    }
    UNION
    {
        ?person rdf:type base:Director .
    }
    UNION
    {
        ?person rdf:type base:Writer .
    }
}
```

Output



```
Persons:
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Quentin_Tarantino
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Peter_Jackson
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Spike_Lee
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Guillermo_del_Toro
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Denis_Villeneuve
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Robert_Rodriguez
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Steven_Soderbergh
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee
```

Ontology3

Code

```python
from rdflib import Graph, RDF, RDFS, OWL, Namespace
from owlrl import DeductiveClosure, RDFS_Semantics

def load_ontology(file_path):
    # Load the TTL file into an RDF graph
    g = Graph()
    g.parse(file_path, format='ttl')

    # Apply Deductive Closure reasoning with RDFS semantics to the graph
    DeductiveClosure(RDFS_Semantics).expand(g)
    return g

def get_actors(graph, ns):
    actors = set()
    # Iterate through all inferred triples in the graph
    for subj, pred, obj in graph:
        # Check if the subject is an Actor
        if (subj, RDF.type, ns.Actor) in graph:
            actors.add(subj)
    return actors

def display_actors(file_path):
    # Load the ontology and apply Deductive Closure reasoning
    graph = load_ontology(file_path)
    # Define namespace
    ns = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/")
    # Get all the Actors from the ontology
    actors = get_actors(graph, ns)
    # Display the Actors
    print("Actors Using Inference without query")
    for actor in actors:
        print(actor)

# Example usage
if __name__ == "__main__":
    ttl_file = "Ontology_phase1_team15.ttl"  # Path to your TTL file
    display_actors(ttl_file)
```

Output

```
Actors Using Inference without query
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Spike_Lee
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Denis_Villeneuve
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Quentin_Tarantino
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Peter_Jackson
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Robert_Rodriguez
```

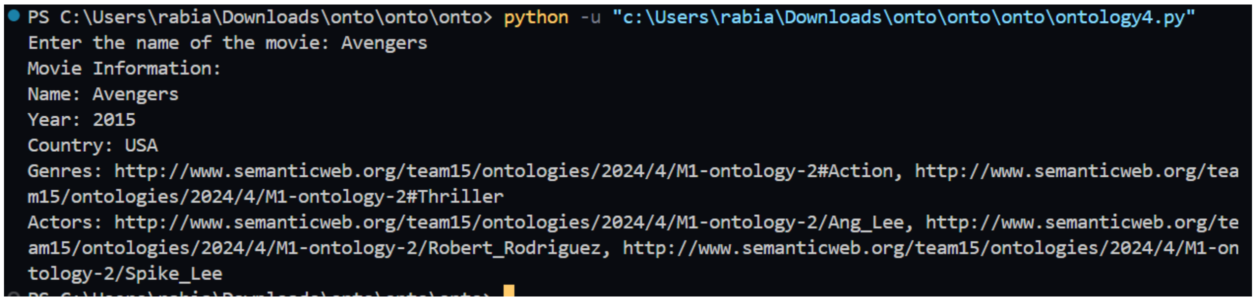
Ontology4

Code

```
from rdflib import Graph, RDF, RDFS, OWL, Namespace, Literal
from owlrl import DeductiveClosure, RDFS_Semantics

def load_ontology(file_path):
    # Load the TTL file into an RDF graph
    g = Graph()
    g.parse(file_path, format='ttl')

    # Apply Deductive Closure reasoning with RDFS semantics to the graph
    DeductiveClosure(RDFS_Semantics).expand(g)
```

```python
    return g

def get_movie_info(graph, ns, movie_name):
    movie_info = {}
    # Iterate through all inferred triples in the graph
    for subj, pred, obj in graph:
        # Check if the subject is an instance of Movie with the given name
        if (subj, RDF.type, ns.Movie) in graph and (subj, ns.hasTitle,
Literal(movie_name)) in graph:
            # Get movie year, country, genres, and actors
            year = graph.value(subj, ns.hasYear)
            country = graph.value(subj, ns.hasCountry)
            genres = [genre for genre in graph.objects(subj, ns.hasGenre)]
            actors = [actor for actor in graph.objects(subj, ns.hasActor)]
            movie_info = {
                'Year': year,
                'Country': country,
                'Genres': genres,
                'Actors': actors
            }
            break
    return movie_info

def display_movie_info(file_path, movie_name):
    # Load the ontology and apply Deductive Closure reasoning
    graph = load_ontology(file_path)
    # Define namespace
    ns = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/")
    # Get movie information
    movie_info = get_movie_info(graph, ns, movie_name)
    # Display movie information
    if movie_info:
        print("Movie Information:")
        print("Name:", movie_name)
        print("Year:", movie_info.get('Year'))
        print("Country:", movie_info.get('Country'))
        print("Genres:", ', '.join(movie_info.get('Genres')))
        print("Actors:", ', '.join(movie_info.get('Actors')))
    else:
        print("Error: Movie '{}' not found.".format(movie_name))

# Example usage
if __name__ == "__main__":
    ttl_file = "Ontology_phase1_team15.ttl"  # Path to your TTL file
    movie_name = input("Enter the name of the movie: ")
    display_movie_info(ttl_file, movie_name)
```

Output



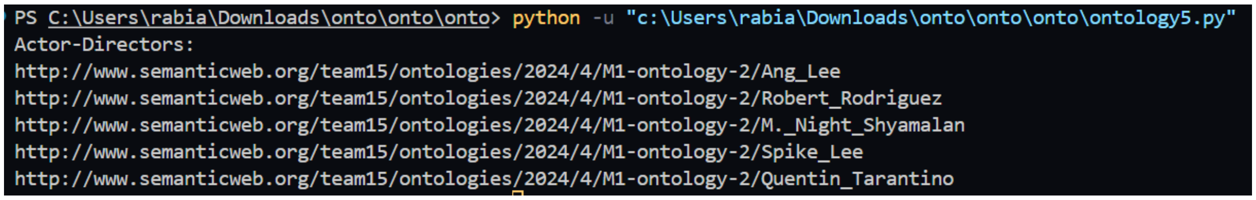
```
PS C:\Users\rabia\Downloads\onto\onto\onto> python -u "c:\Users\rabia\Downloads\onto\onto\onto\ontology4.py"
Enter the name of the movie: Avengers
Movie Information:
Name: Avengers
Year: 2015
Country: USA
Genres: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#Action, http://www.semanticweb.org/tea
m15/ontologies/2024/4/M1-ontology-2#Thriller
Actors: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee, http://www.semanticweb.org/te
am15/ontologies/2024/4/M1-ontology-2/Robert_Rodriguez, http://www.semanticweb.org/team15/ontologies/2024/4/M1-on
tology-2/Spike_Lee
PS C:\Users\rabia\Downloads\onto\onto\onto>
```

Ontology5

Code

```
from rdflib import Graph, RDF, Namespace
from rdflib.plugins.sparql import prepareQuery
from owlrl import DeductiveClosure, RDFS_Semantics, OWLRL_Semantics

# Define namespaces
BASE = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2#")
RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

def load_ontology(file_path, rule_path):
    # Load the TTL file into an RDF graph
    g = Graph()
    g.parse(file_path, format='ttl')
    g.parse(rule_path, format='ttl')
    # Apply Deductive Closure reasoning with RDFS semantics to the graph
    DeductiveClosure(OWLRL_Semantics).expand(g)
    return g

def get_actor_directors(graph):
    persons = set()
    # Prepare a SPARQL query from the text
    query = prepareQuery("""
        SELECT DISTINCT ?actorDirector
        WHERE {
            ?actorDirector rdf:type base:ActorDirector .
        }
        """, initNs={"rdf": RDF, "base": BASE})
    # Execute the query and collect the results
    results = graph.query(query)
    for row in results:
        persons.add(row["actorDirector"])
    return persons

def display_actor_directors(file_path, rule_path):
    # Load the ontology
    graph = load_ontology(file_path, rule_path)

    # Get all the actor-directors from the ontology
    actor_directors = get_actor_directors(graph)

    # Display the actor-directors
    print("Actor-Directors:")
    for actor_director in actor_directors:
        print(actor_director)

# Example usage
if __name__ == "__main__":
    ttl_file = "Ontology_phase1_team15.ttl"  # Path to your TTL file
    rule_file = "rule.ttl"
    display_actor_directors(ttl_file, rule_file)


Rule File
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/> .
@prefix ns: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>
.



# Define ActorDirector as an intersection of Actor and Director
ns:ActorDirector a owl:Class ;
    owl:equivalentClass [
        a owl:Class ;
        owl:intersectionOf ( :Actor :Director )
    ] .
```

Output



```
PS C:\Users\rabia\Downloads\onto\onto\onto> python -u "c:\Users\rabia\Downloads\onto\onto\onto\ontology5.py"
Actor-Directors:
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Robert_Rodriguez
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Spike_Lee
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Quentin_Tarantino
```

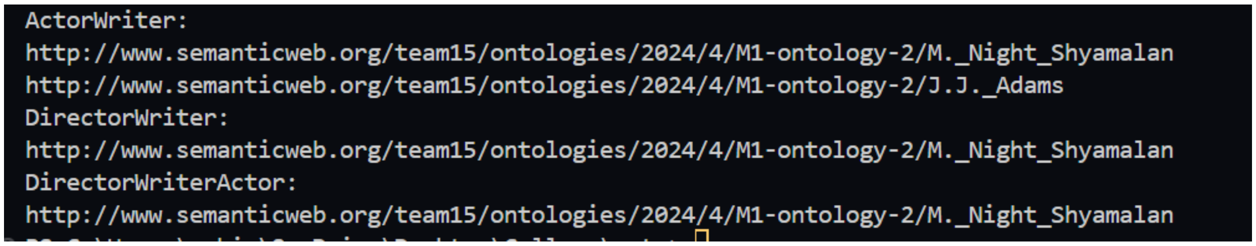
Ontology6

Code

```python
from rdflib import Graph, RDF, Namespace, OWL
from owlrl import DeductiveClosure, RDFS_Semantics, OWLRL_Semantics

# Define namespaces
BASE = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2#")
RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

def load_ontology(file_path, rule_path):
    # Load the TTL file into an RDF graph
    g = Graph()
    g.parse(file_path, format='ttl')
    g.parse(rule_path, format='ttl')

    # Apply Deductive Closure reasoning with RDFS semantics to the graph
    DeductiveClosure(OWLRL_Semantics).expand(g)
    return g

def get_actor_writer(graph):
    actor_writers = set()
    # Iterate through all inferred triples in the graph
    for subj, pred, obj in graph:
        # Check if the subject is an ActorWriter
        if (subj, RDF.type, BASE.ActorWriter) in graph:
            actor_writers.add(subj)
    return actor_writers
def display_actor_writer(file_path, rule_path):
    # Load the ontology
    graph = load_ontology(file_path, rule_path)

    # Get all the actor-directors from the ontology
    actor_writers = get_actor_writer(graph)

    # Display the actor-directors
    print("ActorWriter:")
    for actor_writer in actor_writers:
        print(actor_writer)

def get_director_writer(graph):
    director_writers = set()
    # Iterate through all inferred triples in the graph
    for subj, pred, obj in graph:
        # Check if the subject is an ActorWriter
        if (subj, RDF.type, BASE.WriterDirector) in graph:
            director_writers.add(subj)
    return director_writers
def display_director_writer(file_path, rule_path):
    # Load the ontology
    graph = load_ontology(file_path, rule_path)

    # Get all the actor-directors from the ontology
    director_writers = get_director_writer(graph)

    # Display the actor-directors
    print("DirectorWriter:")
    for director_writer in director_writers:
        print(director_writer)

def get_director_writer_actor(graph):
    director_writers_actor = set()
    # Iterate through all inferred triples in the graph
    for subj, pred, obj in graph:
        # Check if the subject is an ActorWriter
        if (subj, RDF.type, BASE.ActorWriterDirector) in graph:
            director_writers_actor.add(subj)
    return director_writers_actor
def display_director_writer_actor(file_path, rule_path):
    # Load the ontology
    graph = load_ontology(file_path, rule_path)

    # Get all the actor-directors from the ontology
    director_writers_actor = get_director_writer_actor(graph)

    # Display the actor-directors
    print("DirectorWriterActor:")
    for director_writer_actor in director_writers_actor:
        print(director_writer_actor)

# Example usage
```

```
if __name__ == "__main__":
    ttl_file = "Ontology_phase1_team15.ttl"  # Path to your TTL file
    rule_file = "rule.ttl"
    display_actor_writer(ttl_file, rule_file)
    display_director_writer(ttl_file, rule_file)
    display_director_writer_actor(ttl_file,rule_file)

Rule File

Rule File
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/> .
@prefix ns: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>
.

# Define ActorWriter as an intersection of Actor and Writer
ns:ActorWriter a owl:Class ;
    owl:equivalentClass [
        a owl:Class ;
        owl:intersectionOf ( :Actor :Writer )
    ] .

# Define WriterDirector as an intersection of Actor and Writer
ns:WriterDirector a owl:Class ;
    owl:equivalentClass [
        a owl:Class ;
        owl:intersectionOf ( :Director :Writer )
    ] .

# Define ActorWriterDirector as an intersection of Actor and Writer and Director
ns:ActorWriterDirector a owl:Class ;
    owl:equivalentClass [
        a owl:Class ;
        owl:intersectionOf ( :Actor :Director :Writer )
    ] .
```

Output



```
ActorWriter:
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
DirectorWriter:
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
DirectorWriterActor:
http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/M._Night_Shyamalan
[illegible]
```

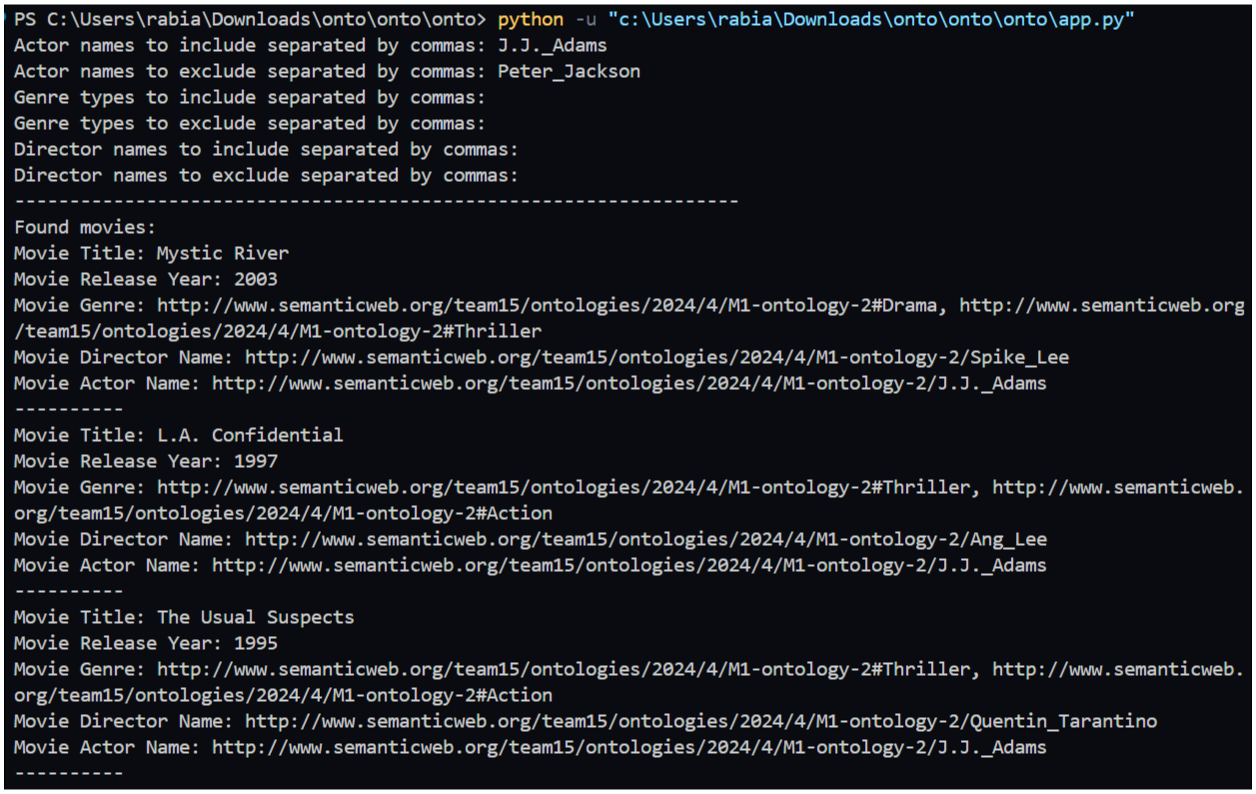
Application

Code

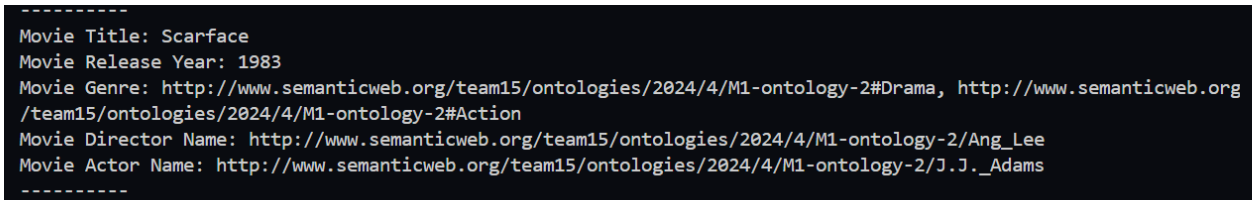
```python
from rdflib import Graph, URIRef, Namespace
from rdflib.namespace import RDF, RDFS, XSD

BASE = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2/")
BASE2 = Namespace("http://www.semanticweb.org/team15/ontologies/2024/4/M1-
ontology-2#")

# Define a Movie class to represent each movie
class Movie:
    def __init__(self, title, actors, director, genres, year):
        self.title = title
        self.actors = actors
        self.director = director
        self.genres = genres
        self.year = year

# Define a class to manage the movie database
class MovieDatabase:
    def __init__(self):
        self.movies = []

    def load_from_ttl(self, ttl_file):
        g = Graph()
        g.parse(ttl_file, format="ttl")

        # Iterate over all triples in the graph
        for subj, pred, obj in g:
            # Check if the subject is a movie
            if (subj, RDF.type, BASE.Movie) in g:
                # Initialize variables to store movie details
                title = None
                actors = set()
                director = None
                genres = set()
                year = None
                # Extract movie details from triples
                if (subj, BASE.hasTitle, None) in g:
                    title = str(g.value(subj, BASE.hasTitle))
                if (subj, BASE.hasActor, None) in g:
                    actors.update(str(actor) for actor in g.objects(subj,
BASE.hasActor))
                if (subj, BASE.hasDirector, None) in g:
                    director = str(g.value(subj, BASE.hasDirector))
                if (subj, BASE.hasGenre, None) in g:
                    genres.update(str(genre) for genre in g.objects(subj,
BASE.hasGenre))
                if (subj, BASE.hasYear, None) in g:
                    year = int(g.value(subj, BASE.hasYear))

                # Create a Movie object and append it to the list of movies
                movie = Movie(title, list(actors), director, list(genres), year)
                self.movies.append(movie)


    def search_movies(self, included_actors=[], included_directors=[],
included_genres=[], excluded_actors=[], excluded_directors=[],
excluded_genres=[]):
        result = []

        for movie in self.movies:
            if (not included_actors or
set(movie.actors).intersection(included_actors)) \
                    and (not included_directors or movie.director in
included_directors) \
                    and (not included_genres or
set(movie.genres).intersection(included_genres)) \
                    and (not excluded_actors or not
set(movie.actors).intersection(excluded_actors)) \
                    and (not excluded_directors or movie.director not in
excluded_directors) \
                    and (not excluded_genres or not
set(movie.genres).intersection(excluded_genres)):
                result.append(movie)

        return result

def prompt_and_process_input(prompt):
    """
    Prompt the user for input and process it into URIs.
    """
    names = input(prompt).split(',')
    if not names:
        return []
    else:
        return [BASE + name.strip() for name in names]

def prompt_and_process_input_for_genres(prompt):
    """
    Prompt the user for input and process it into URIs.
    """
    names = input(prompt).split(',')
    if not names:
        return []
    else:
        return [BASE2 + name.strip() for name in names]

# Main function
def main():
    # Create a movie database
    movie_database = MovieDatabase()

    # Load movie data from TTL file
    movie_database.load_from_ttl("Ontology_phase1_team15.ttl")

    # Take input from User
    included_actor_uris = prompt_and_process_input("Actor names to include
separated by commas: ")
    excluded_actor_uris = prompt_and_process_input("Actor names to exclude
separated by commas: ")
    included_genre_uris = prompt_and_process_input_for_genres("Genre types to
include separated by commas: ")
    excluded_genre_uris = prompt_and_process_input_for_genres("Genre types to
exclude separated by commas: ")
    included_director_uris = prompt_and_process_input("Director names to include
separated by commas: ")
    excluded_director_uris = prompt_and_process_input("Director names to exclude
separated by commas: ")

    # Check if any of the lists are empty and replace them with []
    included_actor_uris = included_actor_uris if included_actor_uris!=[BASE] else
[]
    excluded_actor_uris = excluded_actor_uris if excluded_actor_uris!=[BASE] else
[]
    included_director_uris = included_director_uris if
included_director_uris!=[BASE] else []
```

```
    excluded_director_uris = excluded_director_uris if
excluded_director_uris!=[BASE] else []
    included_genre_uris = included_genre_uris if included_genre_uris!=[BASE2]
else []
    excluded_genre_uris = excluded_genre_uris if excluded_genre_uris!=[BASE2]
else []

    # Search for movies based on criteria
    found_movies = movie_database.search_movies(
                                                included_actors=included_actor_ur
is,
                                                included_genres=included_genre_ur
is,
                                                included_directors=included_direc
tor_uris,
                                                excluded_actors=excluded_actor_ur
is,
                                                excluded_genres=excluded_genre_ur
is,
                                                excluded_directors=excluded_direc
tor_uris,
                                            )

    # Display the found movies
    print("------------------------------------------------------------------")
    if found_movies:
        print("Found movies:")
        printed_titles = set()  # Set to store encountered titles
        for movie in found_movies:
            if movie.title not in printed_titles:  # Check if title is not
already printed
                print("Movie Title:", movie.title)
                print("Movie Release Year:", movie.year)
                print("Movie Genre:", ", ".join(movie.genres))
                print("Movie Director Name:", movie.director)
                print("Movie Actor Name:", ", ".join(movie.actors))
                print("----------")
                printed_titles.add(movie.title)  # Add title to printed set
    else:
        print("There are no movies with this info.")


if __name__ == "__main__":
    main()
```

Output

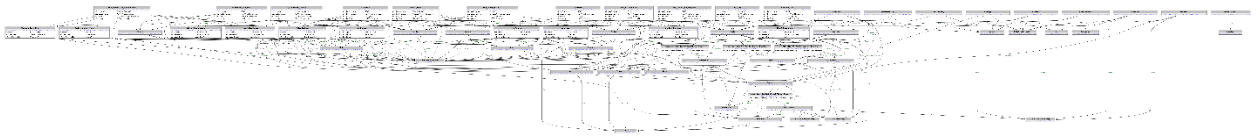


```
PS C:\Users\rabia\Downloads\onto\onto\onto> python -u "c:\Users\rabia\Downloads\onto\onto\onto\app.py"
Actor names to include separated by commas: J.J._Adams
Actor names to exclude separated by commas: Peter_Jackson
Genre types to include separated by commas:
Genre types to exclude separated by commas:
Director names to include separated by commas:
Director names to exclude separated by commas:
-----------------------------------------------------------------
Found movies:
Movie Title: Mystic River
Movie Release Year: 2003
Movie Genre: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#Drama, http://www.semanticweb.org
/team15/ontologies/2024/4/M1-ontology-2#Thriller
Movie Director Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Spike_Lee
Movie Actor Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
----------
Movie Title: L.A. Confidential
Movie Release Year: 1997
Movie Genre: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#Thriller, http://www.semanticweb.
org/team15/ontologies/2024/4/M1-ontology-2#Action
Movie Director Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee
Movie Actor Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
----------
Movie Title: The Usual Suspects
Movie Release Year: 1995
Movie Genre: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#Thriller, http://www.semanticweb.
org/team15/ontologies/2024/4/M1-ontology-2#Action
Movie Director Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Quentin_Tarantino
Movie Actor Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
----------
```

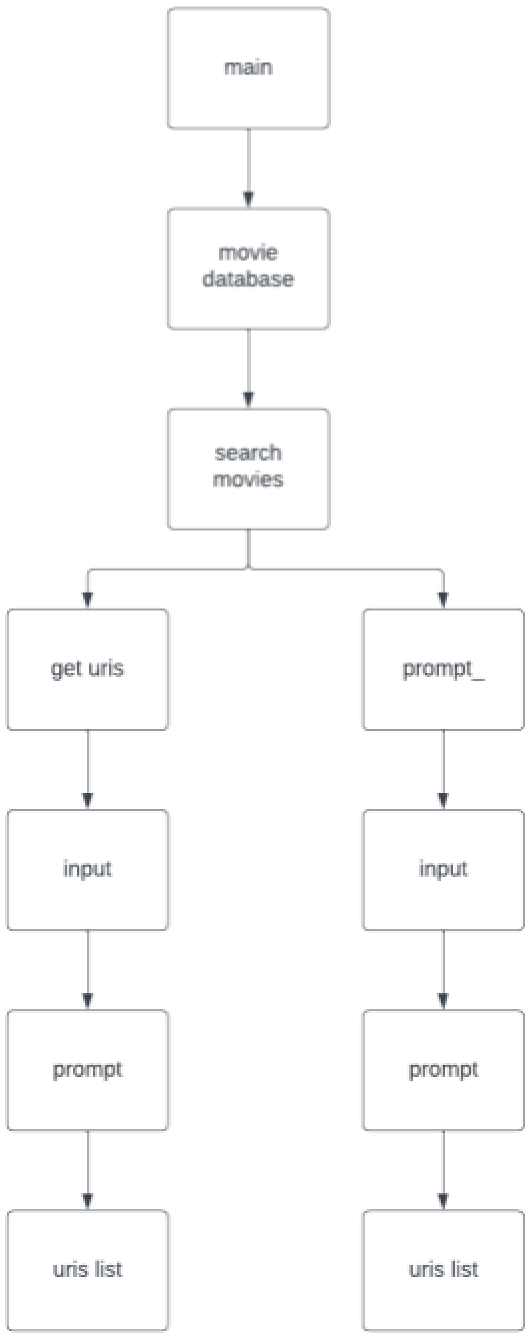


```
----------
Movie Title: Scarface
Movie Release Year: 1983
Movie Genre: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#Drama, http://www.semanticweb.org
/team15/ontologies/2024/4/M1-ontology-2#Action
Movie Director Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/Ang_Lee
Movie Actor Name: http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/J.J._Adams
----------
```

RDF GRAPH USING PYTHON



DFD


main
movie database
search movies
get uris
prompt_
input
input
prompt
prompt
uris list
uris list