

2024



# Milestone I: 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



## Part I: Modeling the Ontology

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. 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

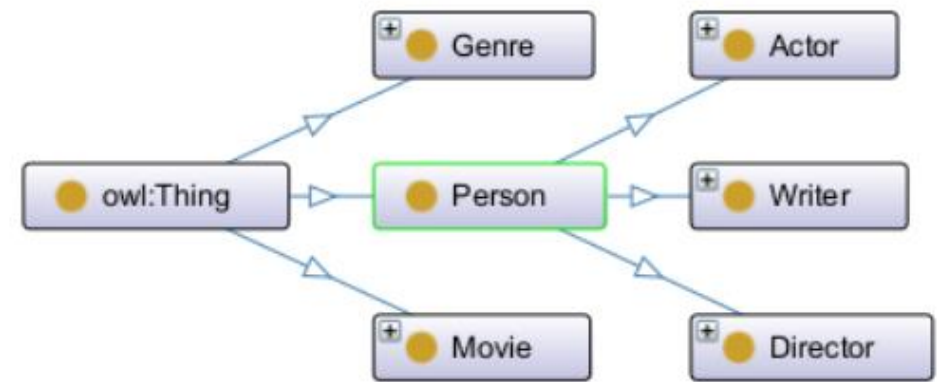
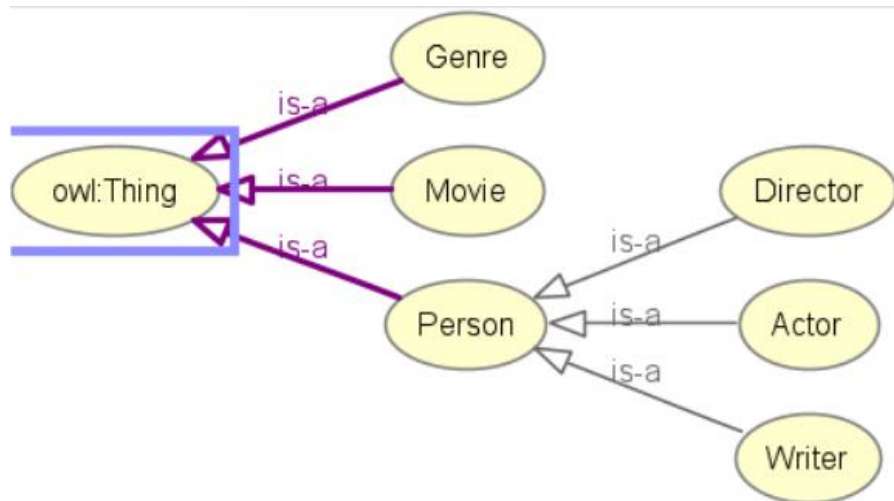
### 3) Data Property:

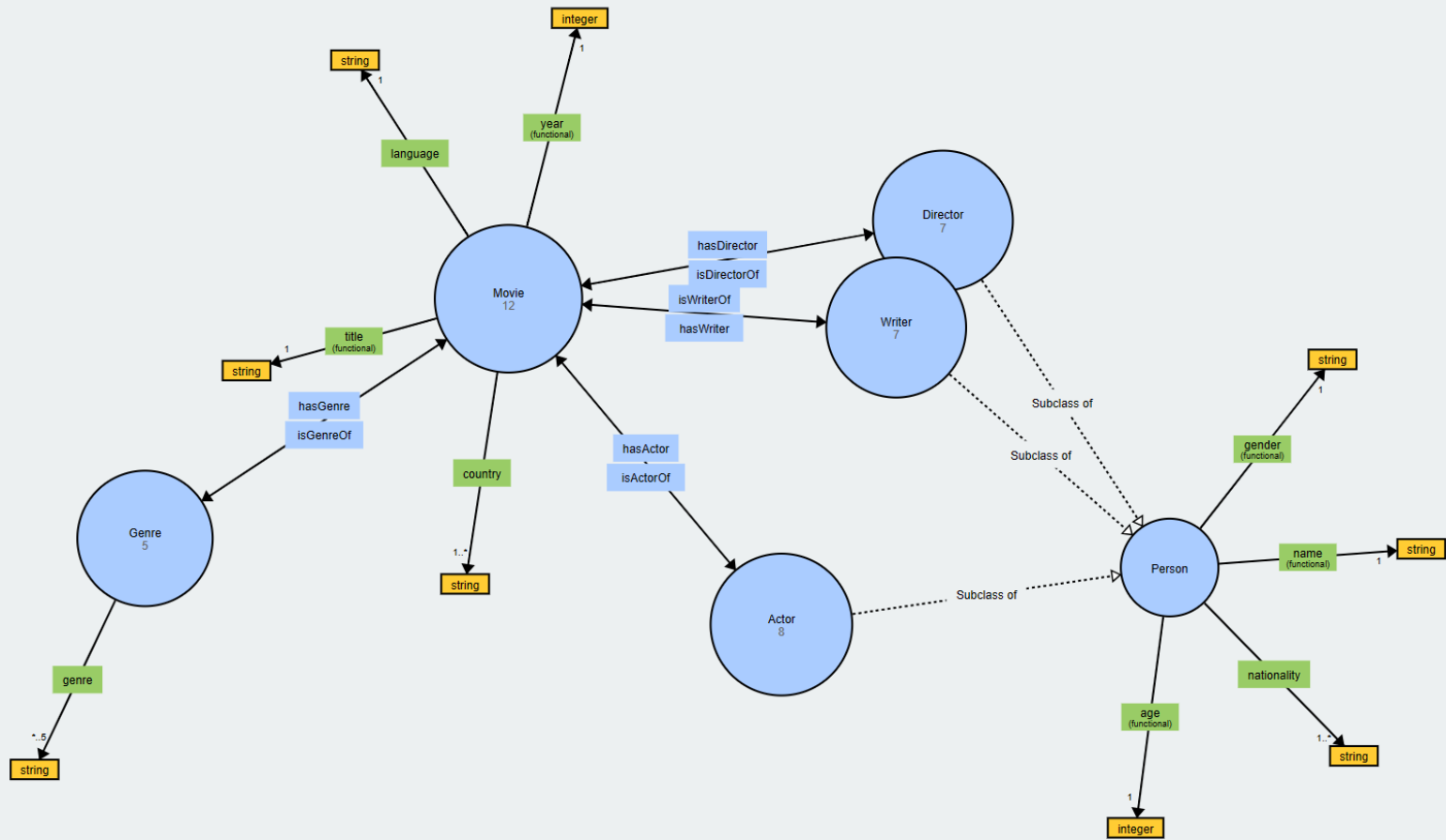
- a) name: Name of Person
- b) age: Age of Person
- c) nationality: Nationality(s) of Person
- d) gender: Gender of Person
- e) genre: Genre(s) of Movie
- f) title: Title of Movie
- g) country: Country(s) of Production of Movie
- h) language: Language(s) of Movie
- i) year: Year of Copyright of Movie

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) A Movie cannot have the same values of the Genre as Director, Writer, Actor
- 4) A Genre cannot have name of Person (Actor, Writer, Director) or Movie
- 5) hasGenre cannot be equivalent to hasActor, hasDirector, hasWriter between the same individuals.
- 6) Disjointedness between data properties is always between properties belonging to the same class.
- 7) Language and Country of a Movie are not equivalent values (ie: A Movie whose country is Canada can have the language France (in English tags only), but France should not be filtered as a country among others in a query searching for country == "France").
- 8) Only one name and one title for a Person or a Movie should be registered.
- 9) There are only 2 genders, male or female.
- 10) 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).
- 11) Only 1 type of language tags were used, the default English tags. There weren't explicitly declared.
- 12) Age 0 means the Person is deceased.
- 13) A Person (Actor, Director, Writer) can have multiple nationalities.
- 14) A Person with a minor role in a Movie is considered an Actor

Finally, view the ontology: [PerfectionistAF/Movies\\_Ontology \(github.com\)](https://github.com/PerfectionistAF/Movies_Ontology)

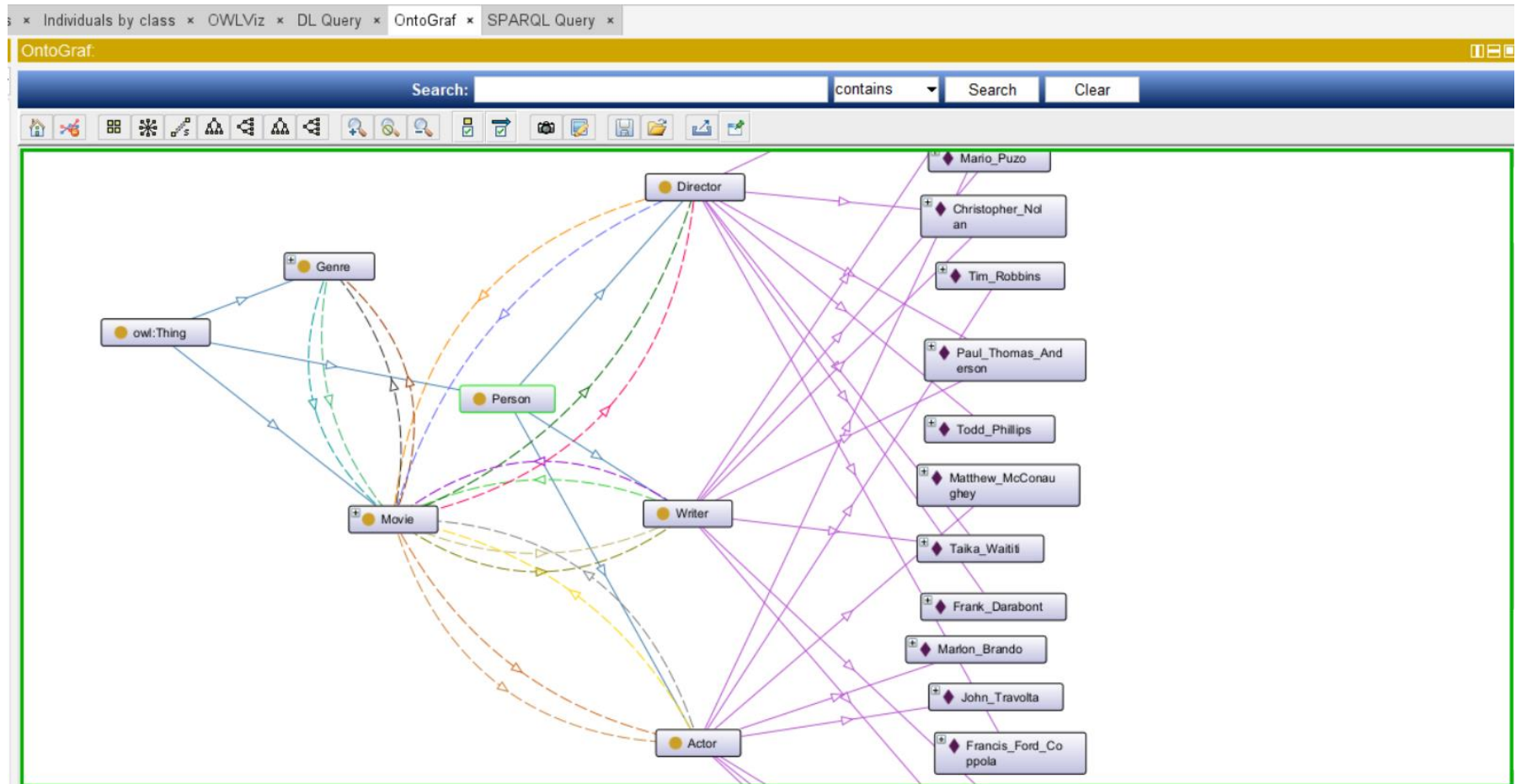




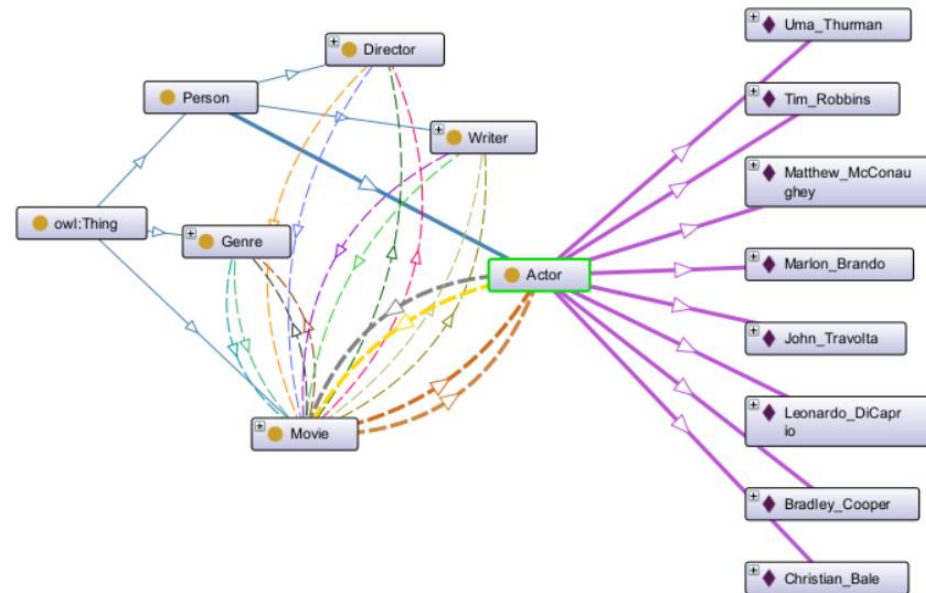
## Part II: Populating the Ontology

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

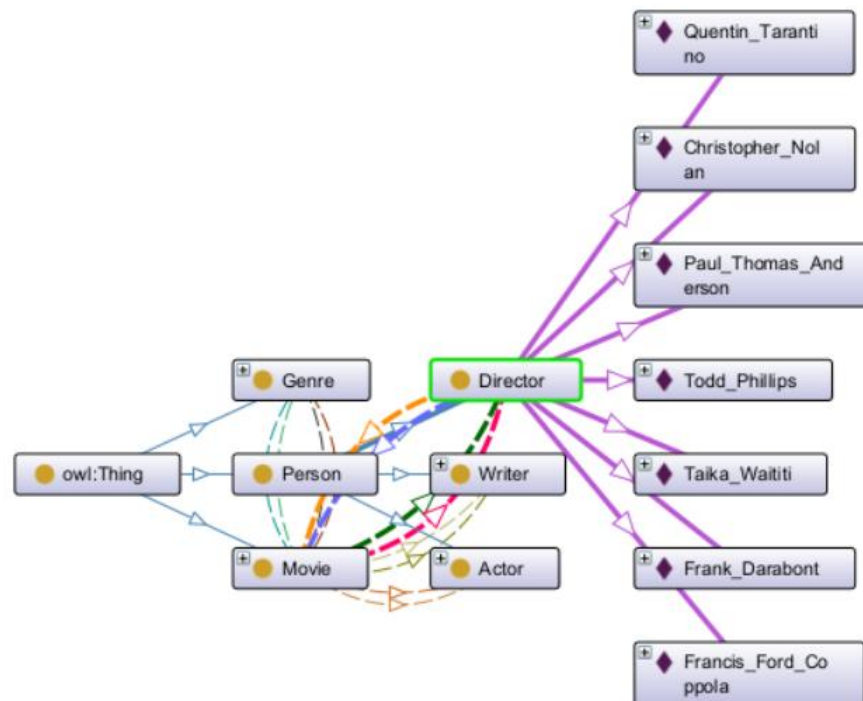
### 1) Person (Actor, Director, Writer)



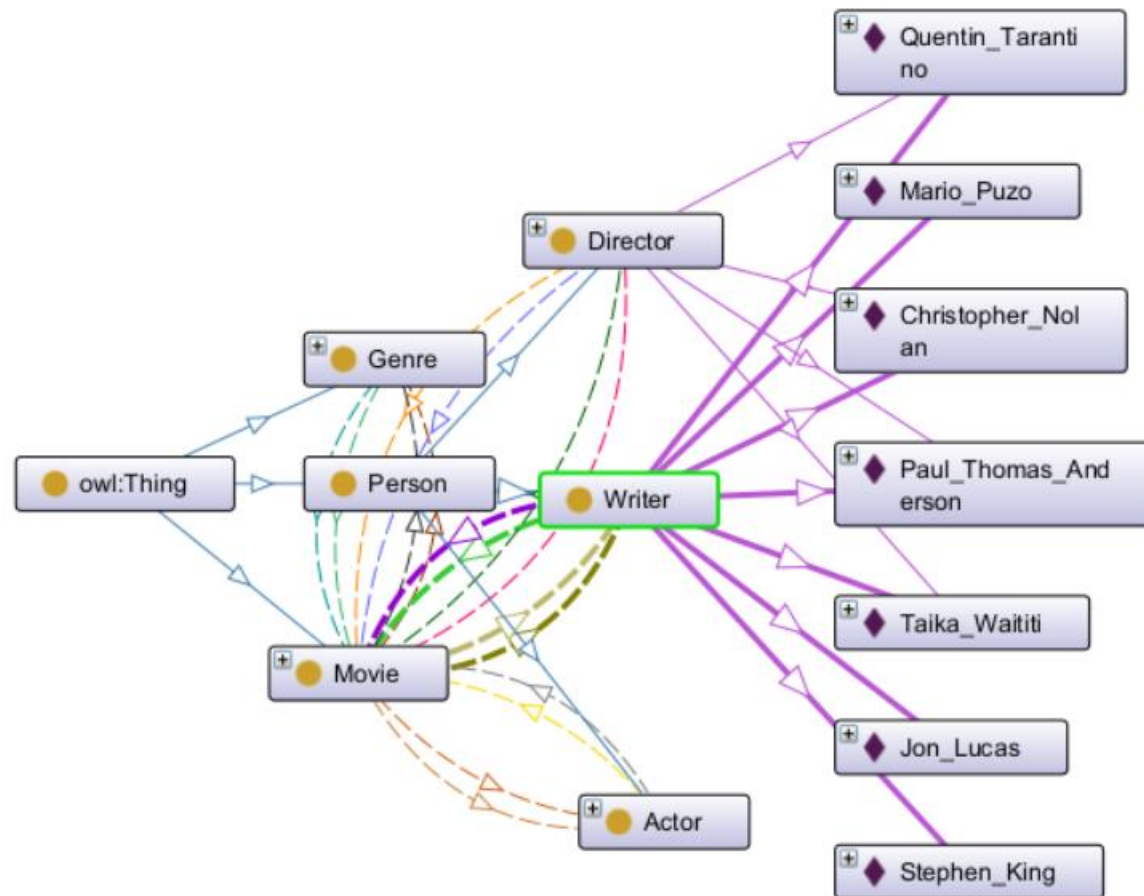
### a) Actor



### b) Director

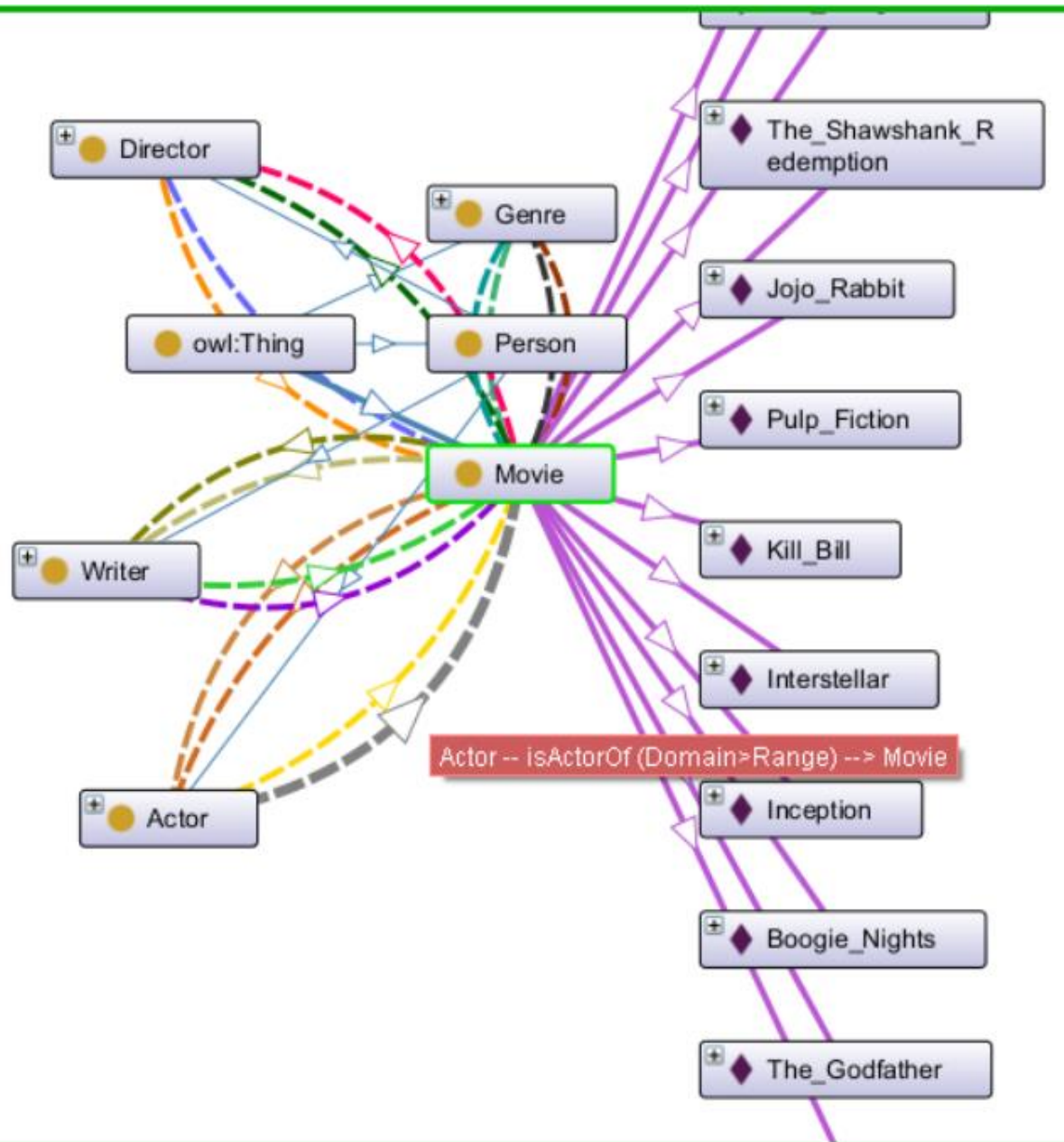


c) Writer





## 2) Movie





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

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

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  $\sqsubseteq$  **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)

## 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`  $\sqsubseteq$  `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`  $\sqsubseteq$  `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]"))  
}

Execute

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

## 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 .
    }
}
```

The screenshot shows a SPARQL query editor with the following 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#>
CONSTRUCT {
    ?person rdf:type ont:Actor;
    rdf:type ont:Director .
}
WHERE {
    {
        ?person rdf:type ont:Actor .
    }
    UNION
    {
        ?person rdf:type ont:Director .
    }
}
```

The results section displays 18 individuals:

- 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

The modal window "Add axioms to selected ontology" is open, showing a list of individuals with checkboxes for "Actor" and "Director" roles:

Individual	Actor	Director
John_Travolta	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
John_Travolta	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Leonardo_DiCaprio	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Leonardo_DiCaprio	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Marlon_Brando	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Marlon_Brando	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Matthew_McConaughey	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Matthew_McConaughey	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Paul_Thomas_Anderson	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Paul_Thomas_Anderson	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Quentin_Tarantino	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Quentin_Tarantino	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Taika_Waititi	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Taika_Waititi	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Tim_Robbins	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Tim_Robbins	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Todd_Phillips	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Todd_Phillips	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Uma_Thurman	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Uma_Thurman	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Add to ontology: moviesV1

OK Cancel

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)
}
```

Execute

?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")
}
```

Snap SPARQL Query:	
<pre>PREFIX owl: &lt;http://www.w3.org/2002/07/owl#&gt; PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt; PREFIX xsd: &lt;http://www.w3.org/2001/XMLSchema#&gt; PREFIX ont: &lt;http://www.semanticweb.org/dataset/ontologies/2024/4/moviesV1#&gt;  SELECT ?title ?release_date WHERE {     ?movie rdf:type ont:Movie ;         ont:title ?title ;         ont:year ?release_date .     FILTER(STR(?release_date) &gt; "2000-01-01") }</pre>	
<button>Execute</button>	
?title	?release_date
Inception <sup>^xsd:string</sup>	2010
There Will Be Blood <sup>^xsd:string</sup>	2007
Thor: Ragnarok <sup>^xsd:string</sup>	2017
Jojo Rabbit <sup>^xsd:string</sup>	2019
The Hangover <sup>^xsd:string</sup>	2009
Kill Bill <sup>^xsd:string</sup>	2003
The Dark Knight <sup>^xsd:string</sup>	2008
Interstellar <sup>^xsd:string</sup>	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 .
}
```

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 ?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 .
}
```

Execute

?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)
}
```

Execute

?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 .
}
```

Execute

?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")
  }
}
```

**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>
```