



# CSE488- Ontologies and the Semantic Web

## Movie Ontology project Project Submission

Name	IDS
Rana Ahmed Abd-Elhalim	19P2468
Yara Nabil Mostafa	19P1881
Ahmed Mohamed Gamal	19P2057
Alaa Mohamed Galal	19P4206

## Table of Contents

GitHub link .....	3
1. Problem description.....	4
1.1 Problem Domain.....	4
1.2 Number Of Entities .....	4
1.3 Number of relations .....	6
1.3.1 Object Properties:.....	6
1.3.2 Inverse Object Properties: .....	6
1.3.3 Data Properties:.....	7
1.4 Logic used.....	9
2 Test Queries and their outputs .....	12
Query1: List the instances of the class Actor .....	12
Query2: List the instances of the class Writer .....	12
Query3: List the instances of the class Director .....	13
Query 4: List the name of all Thriller movies. For each one, display its director. ....	13
Query 5: List the name of all Crime Thriller movies. ....	14
Query 6: list the male actors in the movie in specific film.....	14
Query 7: How many movies have both "Action" and "Thriller" as genres? .....	15
Query 8: List all the movies written by a specific writer. ....	15
Query 9: Find movies with a certain language.....	16
Query 10: List the name of Actors older than 51 years. ....	16
Query 11: A query that contains at least 2 Optional Graph Patterns .....	17
Query 12: A query that contains at least 2 alternatives and conjunctions. ....	18
Query 13: A query that contains a CONSTRUCT query form .....	18
Query 14: A query that contains an ASK query form .....	19
Query 15: A query that contains a DESCRIBE query form .....	20
Query 16: list of movies that have either the genre "Action" or "Thriller" .....	21
Query 17: List of all movies and Order by years.....	21
Query 18: List the count of movies for each genre.....	22
Query 19: List of information about movies and their language if exists .....	22
Query 20 :List the top 5 directors with the most movies and counts the number of movies each director has directed. ....	23
3 Visualize ontology.....	23

4 Snapshots of the Interface (GUI).....	26
5 Test Cases.....	30
6 Data Flow Diagram (DFD).....	33
Figure 1 RDF for class hierarchy using protege .....	5
Figure 2 Screenshot for Entity and its subclasses .....	5
Figure 3 Screenshot for the Object property .....	7
Figure 4 Screenshot for the Data property.....	9
Figure 5 Movie class restriction. ....	10
Figure 6 Person class restrictions.....	11
Figure 7 Class hierarchy using protege .....	23
Figure 8 Ontology visualization .....	24
Figure 9 Visualize of ontology using ontograpgh .....	25
Figure 10 First Screen for GUI.....	26
Figure 11 Pressing on Display Existing Movies Button.....	26
Figure 12 Selecting Movie to show its details .....	27
Figure 13 Pressing Search button without selecting any restrictions .....	27
Figure 14 Search with including/ excluding/ Director and genre. ....	28
Figure 15 Searching with including actor and genre.....	28
Figure 16 Searching for Specific Genre .....	29
Figure 17 Searching by including actor .....	29
Figure 18 Answer for the first test case .....	30
Figure 19 Answer for the second test case.....	30
Figure 20 Answer for third test case.....	31
Figure 21 Answer for fourth test case.....	31
Figure 22 Answer for fifth test case.....	32
Figure 23 Answer for sixth test case.....	32
Figure 24 Context Diagram of DFD .....	33
Figure 25 Level 0 of DFD .....	33

## GitHub link

<https://github.com/ranaahmed04/Ontology-project.git>

# 1. Problem description

The movie ontology seeks to address the challenges associated with organizing, managing, and accessing movie-related information in a structured and efficient manner. These challenges include:

- **Data Organization:** Movie data, including details about films, actors, directors, writers, and genres, often exists in disparate sources and formats, making it difficult to integrate and analyze comprehensively.
- **Information Retrieval:** Traditional methods of accessing movie information, such as manual searches or unstructured databases, can be time-consuming and inefficient, hindering decision-making and analysis processes.
- **Knowledge Representation:** The complex relationships and hierarchies within the movie domain, such as actor-director collaborations and genre classifications, require a robust framework for accurate representation and inference.

By addressing these challenges, the movie ontology aims to provide a standardized, interoperable, and comprehensive solution for managing and accessing movie-related information effectively, benefiting stakeholders across the film industry, including movie enthusiasts, researchers, filmmakers, and industry professionals.

## 1.1 Problem Domain

The ontology addresses the domain of movie management and representation, encompassing various facets of the film industry. It provides a structured framework for organizing and accessing movie-related information efficiently. The problem domain includes the following components:

- 1) **Movies:** Each movie is represented as an entity within the ontology, with attributes title, release year, country of production, and language.
- 2) **Actors:** Actors, the individuals portraying characters in movies, are a key component. Information about actors, including their names, genders, ages, and their nationality.
- 3) **Directors:** Directors play a crucial role in overseeing the creative aspects of filmmaking. The ontology contains information about directors including their names, genders, ages, and their nationality.
- 4) **Writers:** Writers, responsible for creating screenplays or scripts, are also represented in the ontology. Details about writers, including their names, genders, ages, and their nationality.
- 5) **Genres:** Genres categorize movies based on common themes, styles, or subject matter. Ontology represents different genres and their characteristics. the available genres in our ontology which are: Thriller, Crime, Action, Drama or Comedy.

## 1.2 Number Of Entities

In the Movie ontology, there are a total of 11 entities encompassing various classes and their relationships. The ontology defines classes such as Movie, Person, Director, Writer, Actor, Genre and potentially others, each representing a distinct entity in the domain of movies. These classes are interconnected through relationships delineating their associations and attributes. For instance, the Movie class is linked to the Director class through the hasDirector property, indicating the

director responsible for a particular movie. Similarly, movies are associated with genres via the hasGenre property, specifying the genre(s) to which a movie belongs.

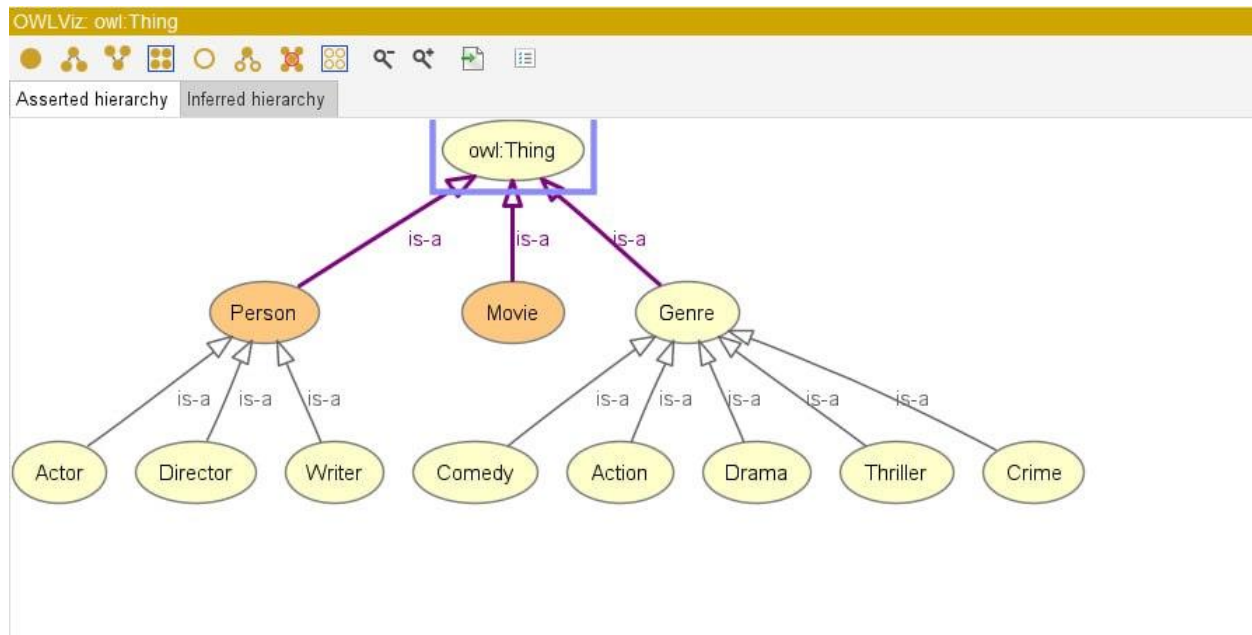


Figure 1 RDF for class hierarchy using protege

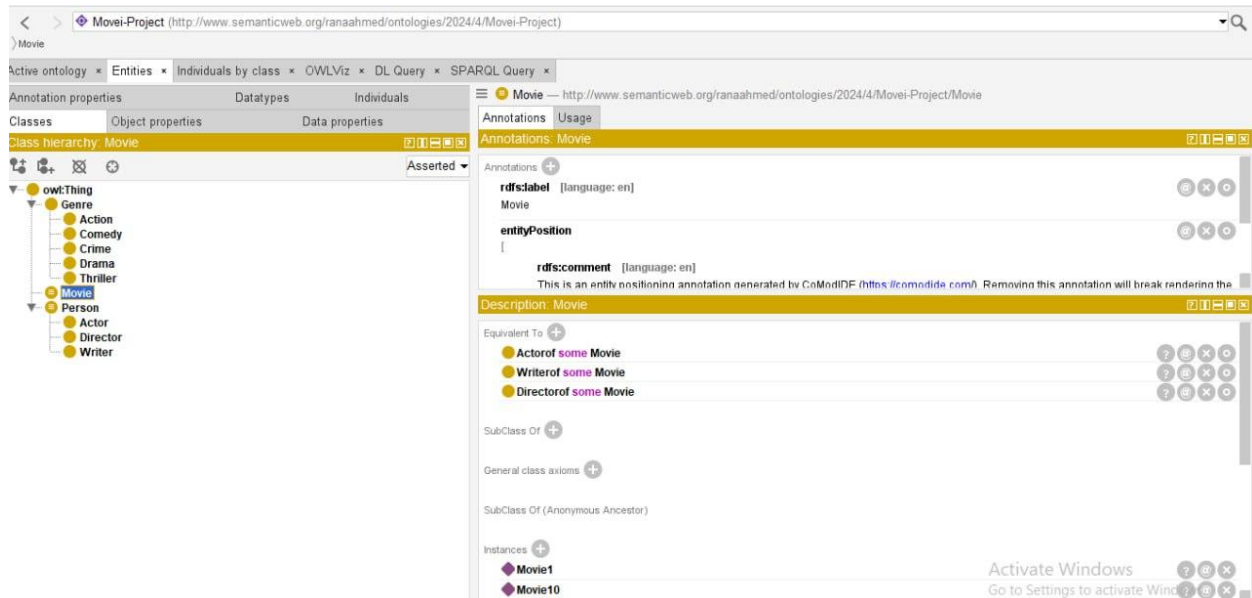


Figure 2 Screenshot for Entity and its subclasses

## 1.3 Number of relations

This is a detailed explanation of the relationships (properties) defined between entities in the ontology, including their types, domains, ranges, and significance within the domain. The ontology encompasses various aspects such as personnel (actors, directors, writers), genres, and movie details, and the relationships between these entities play a crucial role in representing the interconnected nature of the movie domain.

### 1.3.1 Object Properties:

- 1) hasActor:
  - Type: Object Property
  - Domain: Movie
  - Range: Actor
  - Explanation: This property denotes the relationship between a movie and its actors. It specifies which actors are associated with a particular movie.
- 2) hasDirector:
  - Type: Object Property
  - Domain: Movie
  - Range: Director
  - Explanation: Represents the relationship between a movie and its director. It indicates who directed a specific movie.
- 3) hasWriter:
  - Type: Object Property
  - Domain: Movie
  - Range: Writer
  - Explanation: Defines the relationship between a movie and its writer(s), specifying the individuals responsible for writing the screenplay or script.
- 4) hasGenre:
  - Type: Object Property
  - Domain: Movie
  - Range: Genre
  - Explanation: Specifies the genre(s) associated with a movie, indicating the category or type of the movie. It defines the relationship between a movie and its genre(s), allowing for categorization and classification based on thematic content or style.

### 1.3.2 Inverse Object Properties:

- 1) Actorof:
  - Type: Object Property
  - Domain: Actor

- Range: Movie
  - Explanation: An inverse property of hasActor, denoting the movies in which a particular actor has appeared.
- 2) Directorof:
- Type: Object Property
  - Domain: Director
  - Range: Movie
  - Explanation: Represents the inverse relationship of hasDirector, indicating the movies directed by a specific director.
- 3) Writerof:
- Type: Object Property
  - Domain: Writer
  - Range: Movie
  - Explanation: An inverse property of hasWriter, specifying the movies for which a particular individual served as a writer.

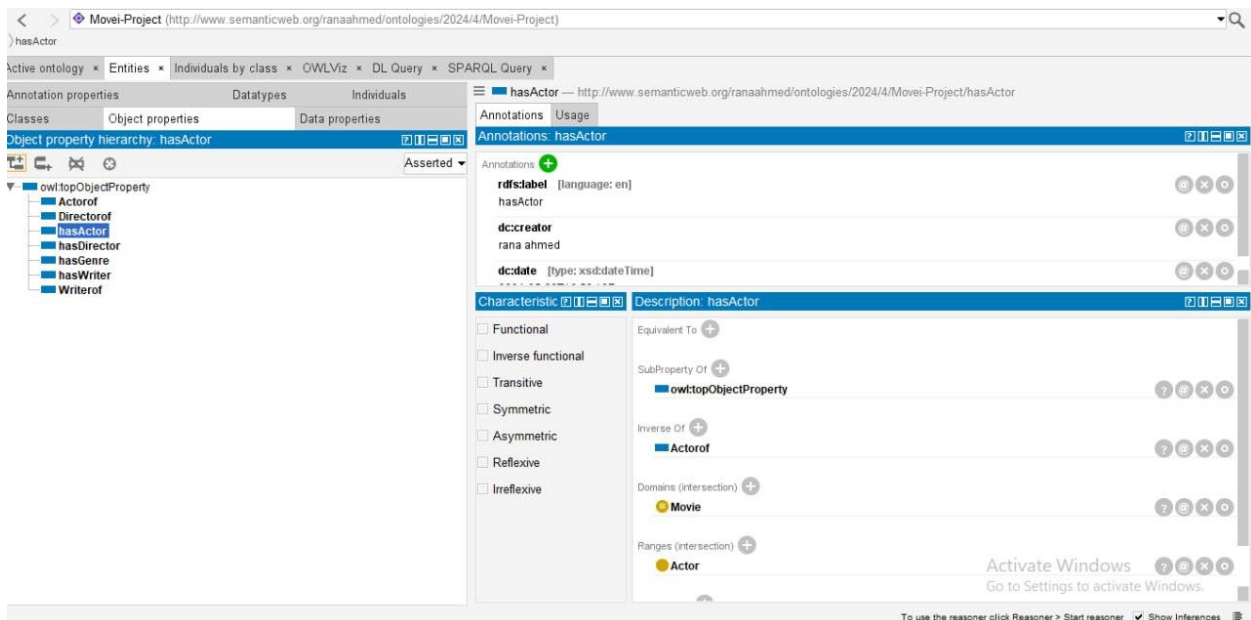


Figure 3 Screenshot for the Object property

### 1.3.3 Data Properties:

- 1) age:
- Type: Datatype Property
  - Domain: Person
  - Range: xsd:integer

- Explanation: Provides information about the age of a person associated with the movie domain, such as actors, directors, or writers.
- 2) country:
- Type: Datatype Property
  - Domain: Movie
  - Range: xsd:string
  - Explanation: Specifies the country associated with a movie, indicating its country of origin or filming location.
- 3) gender:
- Type: Datatype Property
  - Domain: Person
  - Range: xsd:string
  - Explanation: Represents the gender of a person involved in the movie domain, such as actors, directors, or writers.
- 4) language:
- Type: Datatype Property
  - Domain: Movie
  - Range: xsd:string
  - Explanation: Specifies the language(s) associated with a movie, indicating the primary language(s) in which the movie is presented or dubbed.
- 5) name:
- Type: Datatype Property
  - Domain: Person
  - Range: xsd:string
  - Explanation: Provides the name of a person associated with the movie domain, such as actors, directors, or writers.
- 6) nationality:
- Type: Datatype Property
  - Domain: Person
  - Range: xsd:string
  - Explanation: Specifies the nationality of a person involved in the movie domain, indicating their country of citizenship.
- 7) title:
- Type: Datatype Property
  - Domain: Movie
  - Range: xsd:string
  - Explanation: Represents the title of a movie, providing the name by which the movie is known or identified.
- 8) year:
- Type: Datatype Property
  - Domain: Movie
  - Range: xsd:integer



- Explanation: Indicates the year of release of a movie, specifying the year in which the movie was first made available to the public.

These data properties capture various attributes and details associated with individuals (e.g., actors, directors, writers) and movies within the movie ontology. They provide valuable information for querying, filtering, and organizing data related to the movie domain.

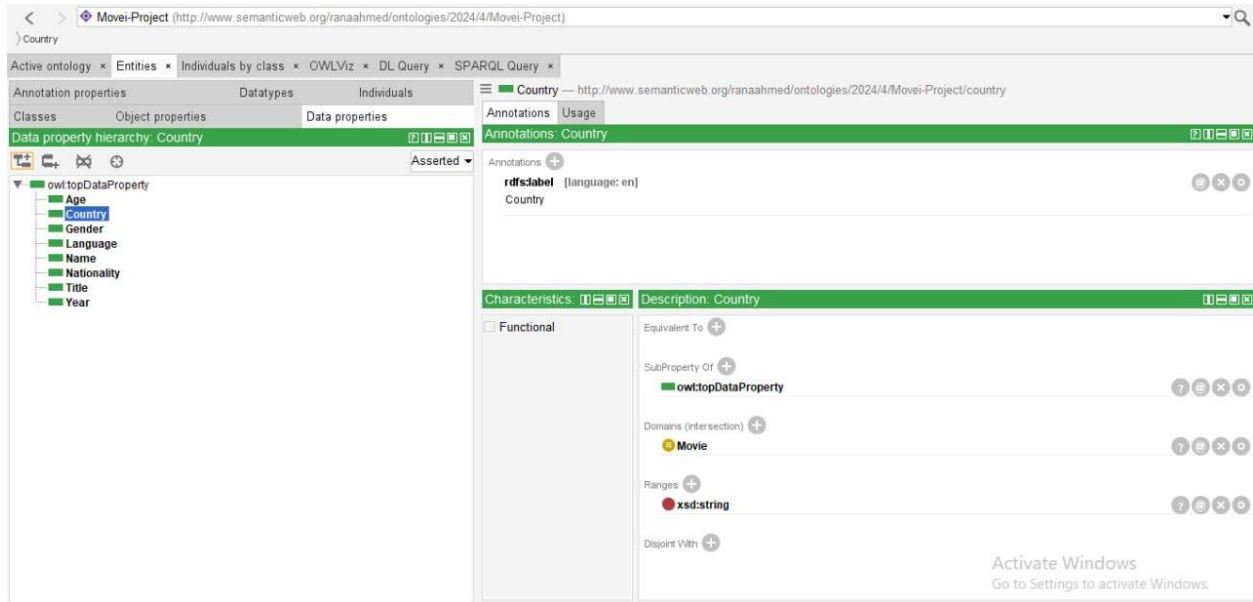


Figure 4 Screenshot for the Data property.

## 1.4 Logic used.

### 1) Axioms and Class Definitions:

- The ontology defines classes such as Actor, Director, Writer, Genre, Movie, and Person, representing various entities within the movie domain.
- Axioms are used to assert relationships between classes and define their characteristics. For example, the class Person is equivalent to the union of individuals involved in acting, directing, or writing movies.
- Class definitions are established through subclass relationships, specifying hierarchical structures within the ontology. For instance, Genre subclasses such as Action, Comedy, Crime, Drama, and Thriller inherit properties and characteristics from the parent class Genre.

2) Object Properties and Relationships:

- Object properties such as hasActor, hasDirector, hasWriter, and hasGenre define relationships between entities in the ontology.
- These properties establish connections between movies and individuals involved in their production, such as actors, directors, writers, and genres.
- Inverse object properties (e.g., Actorof, Directorof, Writerof) are employed to represent reverse relationships, facilitating querying and reasoning in both directions.

3) Data Properties and Attributes:

- Data properties capture attributes and characteristics associated with individuals and movies in the ontology.
- For instance, properties like age, gender, nationality, and language provide information about people involved in the movie domain.
- Properties such as title, year, and country offer details about movie titles, release years, and countries of origin.

4) Restrictions:

- Restrictions are utilized to impose constraints or conditions on classes and properties within the ontology.
- Movie Class Restriction:
  - ✓ Description: Each movie must have at least one actor, director, and writer.
  - ✓ Restriction in turtle:

```
### http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/Movie
:Movie rdf:type owl:Class ;
      owl:equivalentClass [ rdf:type owl:Restriction ;
                             owl:onProperty :Actorof ;
                             owl:someValuesFrom :Movie
                           ] ,
                           [ rdf:type owl:Restriction ;
                             owl:onProperty :Directorof ;
                             owl:someValuesFrom :Movie
                           ] ,
                           [ rdf:type owl:Restriction ;
                             owl:onProperty :Writerof ;
                             owl:someValuesFrom :Movie
                           ] ;
```

*Figure 5 Movie class restriction.*

- Person Class Restriction:
  - ✓ Description: Each person in the movie domain (actors, directors, writers) must be associated with at least one movie either as an actor, director, or writer.
  - ✓ Restriction in turtle:

```
### http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/Person
:Person rdf:type owl:Class ;
      owl:equivalentClass [ rdf:type owl:Restriction ;
                             owl:onProperty :hasActor ;
                             owl:someValuesFrom :Movie
                           ] ,
                           [ rdf:type owl:Restriction ;
                             owl:onProperty :hasDirector ;
                             owl:someValuesFrom :Movie
                           ] ,
                           [ rdf:type owl:Restriction ;
                             owl:onProperty :hasWriter ;
                             owl:someValuesFrom :Movie
                           ] ;
```

*Figure 6 Person class restrictions.*

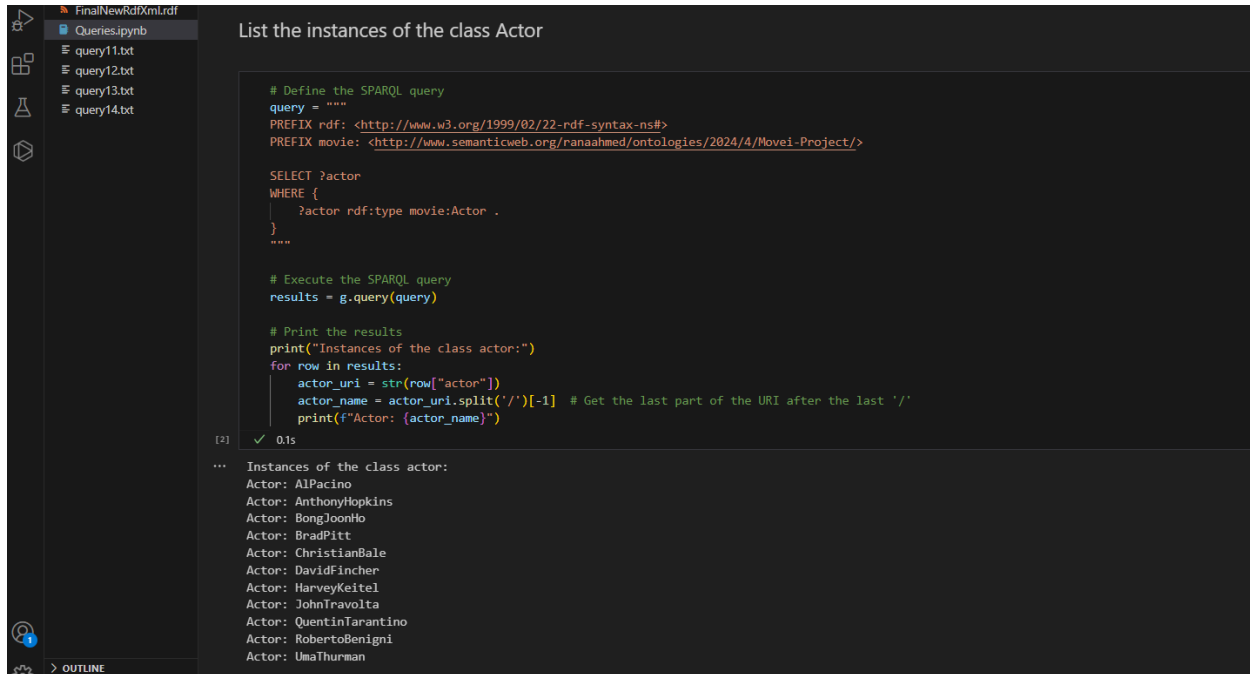
#### 5) Inference Rules:

- Movies with at least one actor and one director:
  - ✓ This rule identifies movies that have both actors and directors associated with them. To implement this rule, we use a SPARQL query to select movies (?movie) that have the properties movie\_proj:hasActor and movie\_proj:hasDirector. If a movie satisfies these conditions, it is inferred to belong to the class MovieWithActorsAndDirectors.
- Movies where the actor and director are the same person:
  - ✓ This rule identifies movies where the same person serves as both the actor and the director. Similarly, we use a SPARQL query to select movies that have the same individual (?person) associated with both movie\_proj:hasActor and movie\_proj:hasDirector. If such a movie is found, it is inferred to belong to the class SameActorAndDirector.
- Recent Movies (Released after 2010):
  - ✓ This rule identifies movies released after the year 2010. To implement this rule, we use a SPARQL filter to select movies that have a release year (movie\_proj:year) greater than 2010. If a movie satisfies this condition, it is inferred to belong to the class RecentMovie.

Once the inference rules are defined, they are applied to the RDF graph using SPARQL update queries. Each rule is executed separately, updating the graph with the inferred class memberships for movies. After applying the rules, we use SPARQL select queries to retrieve the inferred movie data based on each rule. The retrieved data includes the names of movies that satisfy the conditions specified in each rule. Finally, the inferred movie names are printed out for each rule to display the results of the inference process.

## 2 Test Queries and their outputs

### Query1: List the instances of the class Actor



The screenshot shows a Jupyter Notebook interface with a file explorer on the left containing files like 'FinalNewRdfXmlLrd', 'Queries.ipynb', and several 'query' files. The main area displays a Python script that defines a SPARQL query to retrieve instances of the 'Actor' class from an RDF graph. The script includes comments for defining the query, executing it, and printing the results. The output shows a list of actor names: AlPacino, AnthonyHopkins, BongJoonHo, BradPitt, ChristianBale, DavidFincher, HarveyKeitel, JohnTravolta, QuentinTarantino, RobertoBenigni, and UmaThurman.

```
# Define the SPARQL query
query = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Move1-Project/>

SELECT ?actor
WHERE {
  ?actor rdf:type movie:Actor .
}
"""

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

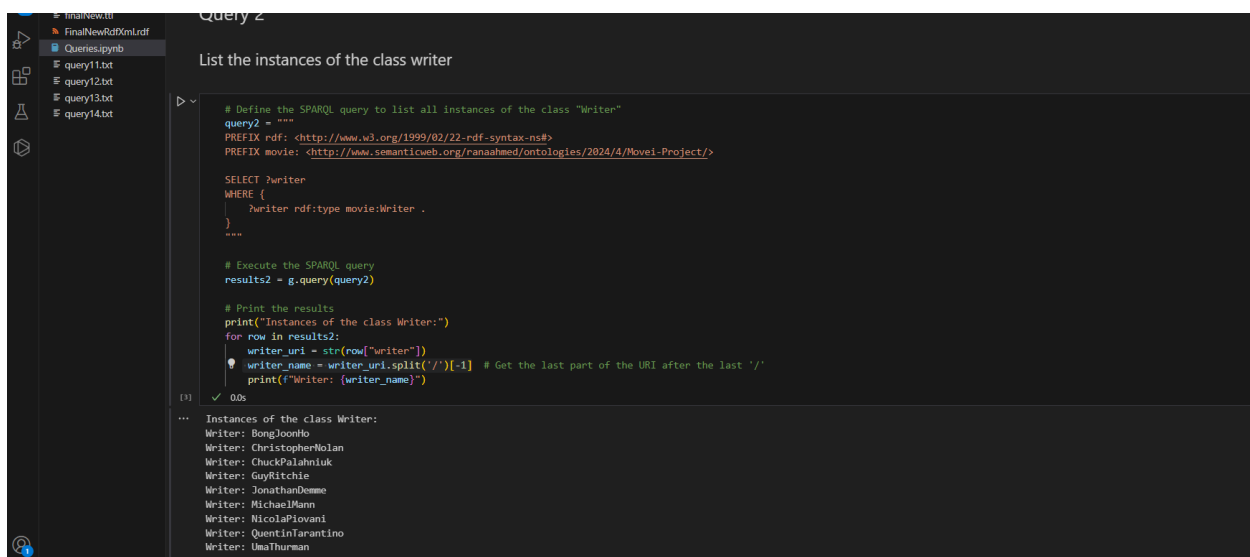
# Print the results
print("Instances of the class actor:")
for row in results:
    actor_uri = str(row["actor"])
    actor_name = actor_uri.split('/')[-1] # Get the last part of the URI after the last '/'
    print(f"Actor: {actor_name}")
```

[2] ✓ 0.1s

Instances of the class actor:

- Actor: AlPacino
- Actor: AnthonyHopkins
- Actor: BongJoonHo
- Actor: BradPitt
- Actor: ChristianBale
- Actor: DavidFincher
- Actor: HarveyKeitel
- Actor: JohnTravolta
- Actor: QuentinTarantino
- Actor: RobertoBenigni
- Actor: UmaThurman

### Query2: List the instances of the class Writer



The screenshot shows a Jupyter Notebook interface with a file explorer on the left. The main area displays a Python script that defines a SPARQL query to retrieve instances of the 'Writer' class from an RDF graph. The script includes comments for defining the query, executing it, and printing the results. The output shows a list of writer names: BongJoonHo, ChristopherNolan, ChuckPalahniuk, GuyRitchie, JonathanDemme, MichaelMann, NicolaPiovani, QuentinTarantino, and UmaThurman.

```
# Define the SPARQL query to list all instances of the class "Writer"
query2 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Move1-Project/>

SELECT ?writer
WHERE {
  ?writer rdf:type movie:Writer .
}
"""

# Execute the SPARQL query
results2 = g.query(query2)

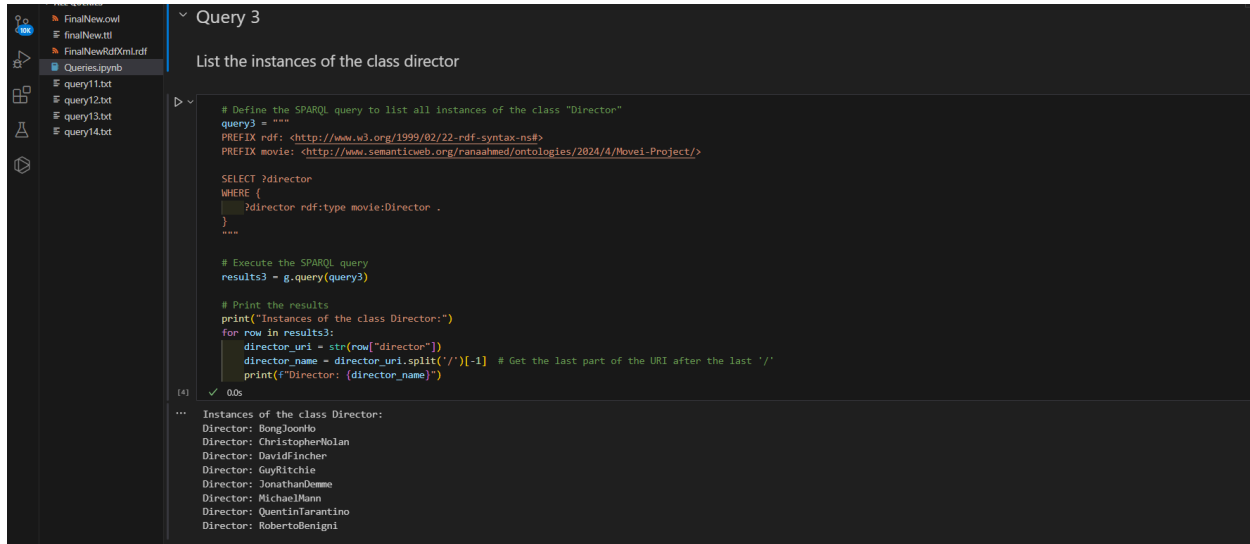
# Print the results
print("Instances of the class Writer:")
for row in results2:
    writer_uri = str(row["writer"])
    writer_name = writer_uri.split('/')[-1] # Get the last part of the URI after the last '/'
    print(f"Writer: {writer_name}")
```

[3] ✓ 0.0s

Instances of the class Writer:

- Writer: BongJoonHo
- Writer: ChristopherNolan
- Writer: ChuckPalahniuk
- Writer: GuyRitchie
- Writer: JonathanDemme
- Writer: MichaelMann
- Writer: NicolaPiovani
- Writer: QuentinTarantino
- Writer: UmaThurman

## Query3: List the instances of the class Director



```
# Define the SPARQL query to list all instances of the class "Director"
query3 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT ?director
WHERE {
  ?director rdf:type movie:Director .
}
"""

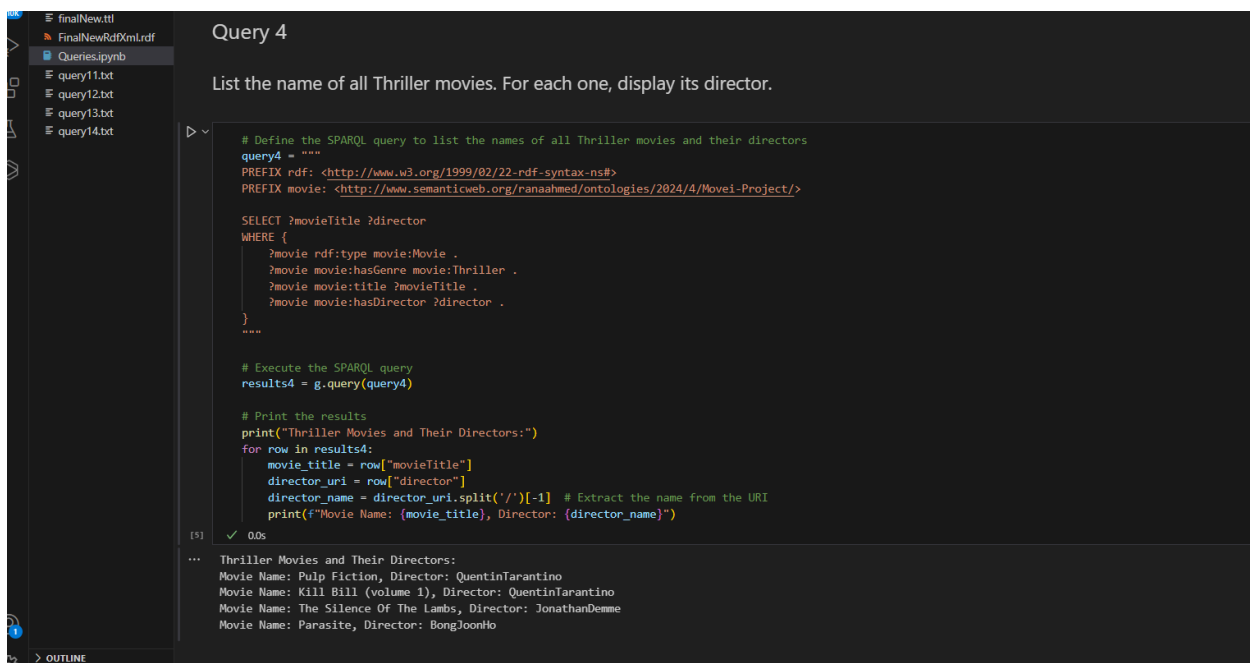
# Execute the SPARQL query
results3 = g.query(query3)

# Print the results
print("Instances of the class Director:")
for row in results3:
    director_uri = str(row["director"])
    director_name = director_uri.split('/')[-1] # Get the last part of the URI after the last '/'
    print(f"Director: {director_name}")
```

Instances of the class Director:

- Director: BongJoonho
- Director: ChristopherNolan
- Director: DavidFincher
- Director: GuyRitchie
- Director: JonathanDemme
- Director: MichaelMann
- Director: QuentinTarantino
- Director: RobertoBenigni

## Query 4: List the name of all Thriller movies. For each one, display its director.



```
# Define the SPARQL query to list the names of all Thriller movies and their directors
query4 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT ?movieTitle ?director
WHERE {
  ?movie rdf:type movie:Movie .
  ?movie movie:hasGenre movie:Thriller .
  ?movie movie:title ?movieTitle .
  ?movie movie:hasDirector ?director .
}
"""

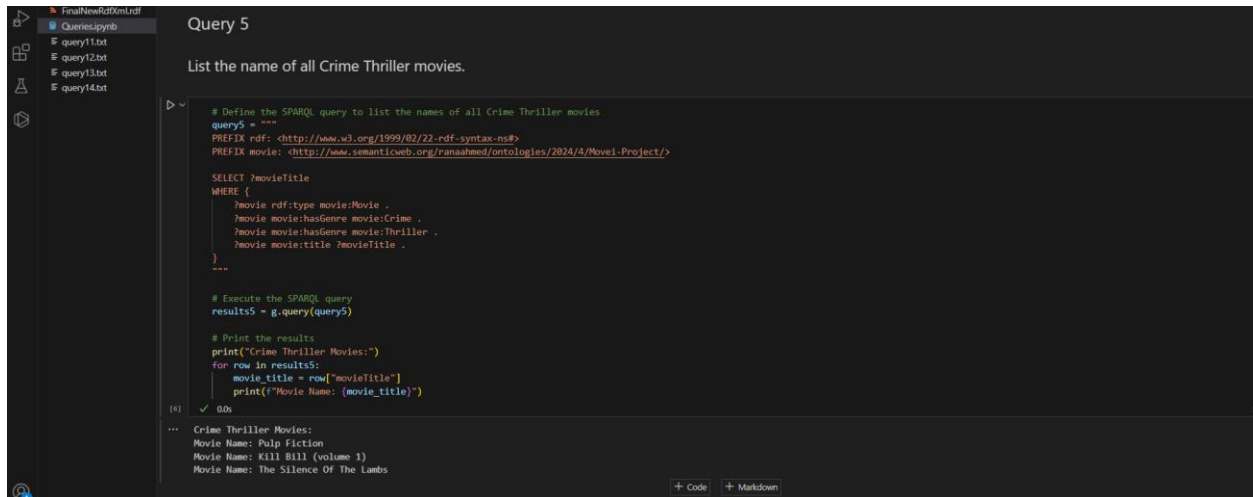
# Execute the SPARQL query
results4 = g.query(query4)

# Print the results
print("Thriller Movies and Their Directors:")
for row in results4:
    movie_title = row["movieTitle"]
    director_uri = row["director"]
    director_name = director_uri.split('/')[-1] # Extract the name from the URI
    print(f"Movie Name: {movie_title}, Director: {director_name}")
```

Thriller Movies and Their Directors:

- Movie Name: Pulp Fiction, Director: QuentinTarantino
- Movie Name: Kill Bill (volume 1), Director: QuentinTarantino
- Movie Name: The Silence Of The Lambs, Director: JonathanDemme
- Movie Name: Parasite, Director: BongJoonho

## Query 5: List the name of all Crime Thriller movies.



```
Query 5

List the name of all Crime Thriller movies.

# Define the SPARQL query to list the names of all Crime Thriller movies
query5 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

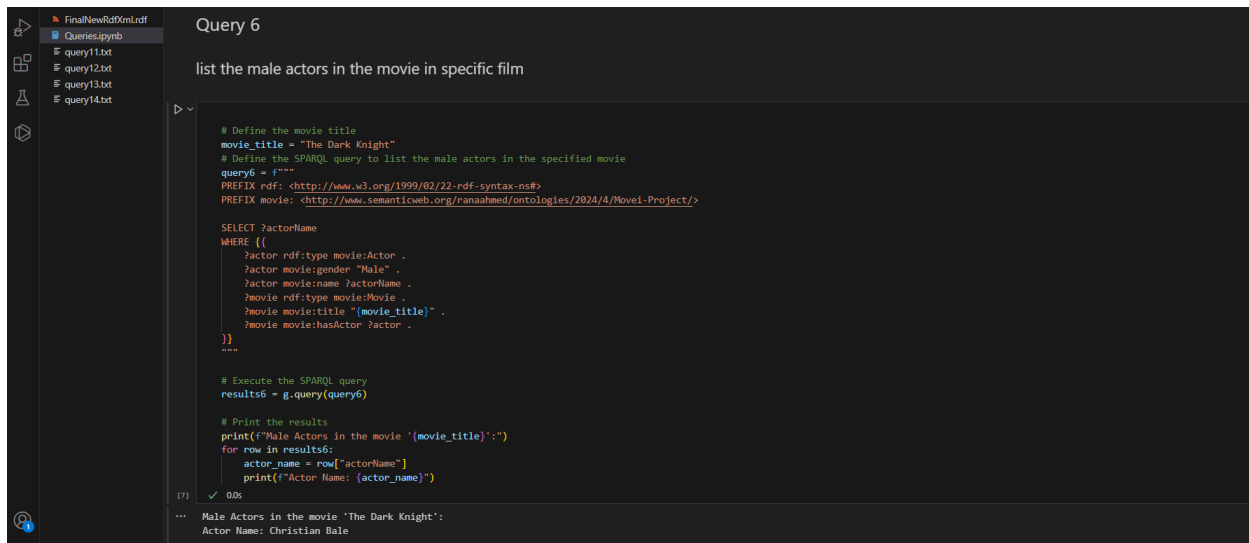
SELECT ?movieTitle
WHERE {
  ?movie rdf:type movie:Movie .
  ?movie movie:hasGenre movie:Crime .
  ?movie movie:hasGenre movie:Thriller .
  ?movie movie:title ?movieTitle .
}
"""

# Execute the SPARQL query
results5 = g.query(query5)

# Print the results
print("Crime Thriller Movies:")
for row in results5:
    movie_title = row["movieTitle"]
    print(f"Movie Name: {movie_title}")
```

Crime Thriller Movies:  
Movie Name: Pulp Fiction  
Movie Name: Kill Bill (volume 1)  
Movie Name: The Silence Of The Lambs

## Query 6: list the male actors in the movie in specific film



```
Query 6

list the male actors in the movie in specific film

# Define the movie title
movie_title = "The Dark Knight"
# Define the SPARQL query to list the male actors in the specified movie
query6 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

SELECT ?actorName
WHERE {
  ?actor rdf:type movie:Actor .
  ?actor movie:gender "Male" .
  ?actor movie:name ?actorName .
  ?movie rdf:type movie:Movie .
  ?movie movie:title "{movie_title}" .
  ?movie movie:hasActor ?actor .
}
"""

# Execute the SPARQL query
results6 = g.query(query6)

# Print the results
print(f"Male Actors in the movie '{movie_title}':")
for row in results6:
    actor_name = row["actorName"]
    print(f"Actor Name: {actor_name}")
```

Male Actors in the movie 'The Dark Knight':  
Actor Name: Christian Bale

## Query 7: How many movies have both "Action" and "Thriller" as genres?

```
query13.txt
query14.txt

Query 7

How many movies have both "Action" and "Thriller" as genres?

# Define the SPARQL query to count movies with both "Action" and "Thriller" as genres
query7 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT (COUNT(?movie) AS ?count)
WHERE {
    ?movie rdf:type movie:Movie .
    ?movie movie:hasGenre movie:Action .
    ?movie movie:hasGenre movie:Thriller .
}
"""

# Execute the SPARQL query
results7 = g.query(query7)

# Extract and print the count of movies with both "Action" and "Thriller" as genres
for row in results7:
    movie_count = row["count"]
    print(f"Number of movies with both 'Action' and 'Thriller' genres: {movie_count}")

[8] ✓ 0.0s
... Number of movies with both 'Action' and 'Thriller' genres: 1
```

## Query 8: List all the movies written by a specific writer.

```
query12.txt
query13.txt
query14.txt

Query 8

List all the movies written by a specific writer.

# Define the SPARQL query to list movies written by Christopher Nolan
query8 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT ?movieTitle
WHERE {
    ?movie rdf:type movie:Movie .
    ?movie movie:hasWriter movie:ChristopherNolan .
    ?movie movie:title ?movieTitle .
}
"""

# Execute the SPARQL query
results8 = g.query(query8)

# Print the list of movies written by Christopher Nolan
print("Movies written by Christopher Nolan:")
for row in results8:
    movie_title = row["movieTitle"]
    print(f"- {movie_title}")

[9] ✓ 0.0s
... Movies written by Christopher Nolan:
- The Dark Knight
```

## Query 9: Find movies with a certain language.

```
Find movies with a certain language.

# Define the SPARQL query to find movies with English language
query9 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

SELECT ?movieTitle
WHERE {
    ?movie rdf:type movie:Movie .
    ?movie movie:language "English" .
    ?movie movie:title ?movieTitle .
}
"""

# Execute the SPARQL query
results9 = g.query(query9)

# Print the list of movies with English language
print("Movies with English language:")
for row in results9:
    movie_title = row["movieTitle"]
    print(f"- {movie_title}")

[10] ✓ 0.0s

... Movies with English language:
- Pulp Fiction
- Kill Bill (volume 1)
- The Dark Knight
- Fight Club
- The Silence Of The Lambs
- Heat
- Snatch
- Reservoir Dogs
```

## Query 10: List the name of Actors older than 51 years.

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

# Define the SPARQL query to find actors older than 51 years
query10 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

SELECT ?actorName
WHERE {
    ?actor rdf:type movie:Actor .
    ?actor movie:age ?age .
    ?actor movie:name ?actorName .
    FILTER (?age > 51)
}
"""

# Execute the SPARQL query
results10 = g.query(query10)

# Print the names of actors older than 51 years
print("Actors older than 51 years:")
for row in results10:
    actor_name = row["actorName"]
    print(f"- {actor_name}")

[11] ✓ 0.0s

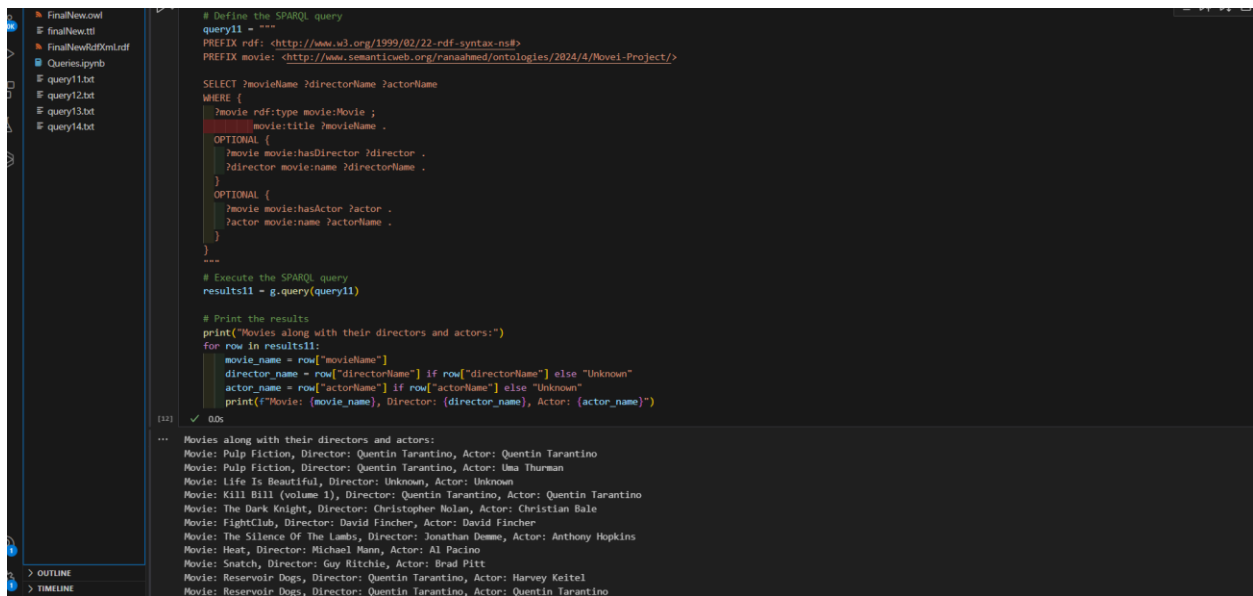
... Actors older than 51 years:
- Al Pacino
- Anthony Hopkins
- Bong Joon-ho
- Brad Pitt
- David Fincher
- Harvey Keitel
- John Travolta
- Quentin Tarantino
```



## Query 11: A query that contains at least 2 Optional Graph Patterns

In this query:

The first optional graph pattern retrieves the director's name for each movie, if available. The second optional graph pattern retrieves the actor's name for each movie, if available. This allows you to retrieve movies along with their directors and actors, but it will still return movies even if their directors or actors are not specified.



```
# Define the SPARQL query
query11 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/panaahmed/ontologies/2024/4/Novel-Project/>

SELECT ?movieName ?directorName ?actorName
WHERE {
  ?movie rdf:type movie:Movie ;
         movie:title ?movieName .
  OPTIONAL {
    ?movie movie:hasDirector ?director .
    ?director movie:name ?directorName .
  }
  OPTIONAL {
    ?movie movie:hasActor ?actor .
    ?actor movie:name ?actorName .
  }
}
"""

# Execute the SPARQL query
results11 = g.query(query11)

# Print the results
print("Movies along with their directors and actors:")
for row in results11:
    movie_name = row["movieName"]
    director_name = row["directorName"] if row["directorName"] else "Unknown"
    actor_name = row["actorName"] if row["actorName"] else "Unknown"
    print(f"Movie: {movie_name}, Director: {director_name}, Actor: {actor_name}")
```

1/21 ✓ 0.0s

... Movies along with their directors and actors:

Movie: Pulp Fiction, Director: Quentin Tarantino, Actor: Quentin Tarantino  
Movie: Pulp Fiction, Director: Quentin Tarantino, Actor: Uma Thurman  
Movie: Life Is Beautiful, Director: Unknown, Actor: Unknown  
Movie: Kill Bill (volume 1), Director: Quentin Tarantino, Actor: Quentin Tarantino  
Movie: The Dark Knight, Director: Christopher Nolan, Actor: Christian Bale  
Movie: Fight Club, Director: David Fincher, Actor: David Fincher  
Movie: The Silence Of The Lambs, Director: Jonathan Demme, Actor: Anthony Hopkins  
Movie: Heat, Director: Michael Mann, Actor: Al Pacino  
Movie: Snatch, Director: Guy Ritchie, Actor: Brad Pitt  
Movie: Reservoir Dogs, Director: Quentin Tarantino, Actor: Harvey Keitel  
Movie: Reservoir Dogs, Director: Quentin Tarantino, Actor: Quentin Tarantino

## Query 12: A query that contains at least 2 alternatives and conjunctions.

The SPARQL query retrieves movie names and their corresponding genres. It uses alternatives and conjunctions to find movies with specific combinations of genres: either both "Action" and "Thriller" or both "Crime" and "Thriller". The SELECT statement specifies the variables to retrieve, while the WHERE clause defines the conditions for matching movies and their genres. Finally, the DISTINCT keyword ensures that duplicate results are eliminated.

```
query12 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT DISTINCT ?movieName ?genre
WHERE {
    ?movie rdf:type movie:Movie ;
           movie:title ?movieName ;
           movie:hasGenre ?genre .
    {
        ?movie movie:hasGenre movie:Action .
        ?movie movie:hasGenre movie:Thriller .
    }
    UNION
    {
        ?movie movie:hasGenre movie:Crime .
        ?movie movie:hasGenre movie:Thriller .
    }
}
"""

# Execute the SPARQL query
results12 = g.query(query12)

# Print the results
print("Movies with Action and Thriller or Crime and Thriller genres:")
movies = set()
for row in results12:
    movie_name = str(row["movieName"])
    genre = str(row["genre"]).split("/")[-1]
    if (movie_name, genre) not in movies:
        movies.add((movie_name, genre))
        print(f"Movie Name: {movie_name}, Genre: {genre}")

[13] ✓ 0.0s

... Movies with Action and Thriller or Crime and Thriller genres:
Movie Name: Pulp Fiction, Genre: Crime
Movie Name: Pulp Fiction, Genre: Thriller
Movie Name: Kill Bill (volume 1), Genre: Action
Movie Name: Kill Bill (volume 1), Genre: Crime
Movie Name: Kill Bill (volume 1), Genre: Thriller
Movie Name: The Silence Of The Lambs, Genre: Crime
Movie Name: The Silence Of The Lambs, Genre: Drama
Movie Name: The Silence Of The Lambs, Genre: Thriller
```

## Query 13: A query that contains a CONSTRUCT query form

The provided SPARQL query is a CONSTRUCT query, which is used to create a new RDF graph based on patterns found in an existing RDF graph. This query constructs a new RDF graph by extracting specific information (movie titles, directors, and genres) from an existing RDF graph that conforms to certain patterns specified in the WHERE clause, and then structures them into new triples according to the CONSTRUCT clause.

```
# Define the CONSTRUCT query
query13 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

CONSTRUCT {
    ?movie rdf:type movie:Movie ;
           movie:title ?title ;
           movie:hasDirector ?director ;
           movie:hasGenre ?genre .
}
WHERE {
    ?movie rdf:type movie:Movie ;
           movie:title ?title ;
           movie:hasDirector ?director ;
           movie:hasGenre ?genre .
}
"""

# Execute the query
results13 = g.query(query13)

# Iterate over the constructed triples and print the values
for subject, predicate, obj in results13:
    subject_name = subject.split("/")[-1] if isinstance(subject, str) else subject
    predicate_name = predicate.split("/")[-1] if isinstance(predicate, str) else predicate
    obj_name = obj.split("/")[-1] if isinstance(obj, str) else obj

    print(f"Subject: {subject_name}")
    print(f"Predicate: {predicate_name}")
    print(f"Object: {obj_name}")
    print("-----")

[14] ✓ 0.0s

Subject: Movie1
```

```
FinalNewRdXmlLrdf [14] ✓ 0.0s
...
Subject: Movie8
Predicate: hasDirector
Object: QuentinTarantino
-----
Subject: Movie1
Predicate: title
Object: Pulp Fiction
-----
Subject: Movie5
Predicate: hasGenre
Object: Drama
-----
Subject: Movie1
Predicate: hasGenre
Object: Crime
-----
Subject: Movie8
Predicate: hasGenre
Object: Drama
-----
Subject: Movie4
Predicate: hasGenre
Object: Drama
-----
Subject: Movie1
...
Subject: Movie10
Predicate: title
Object: Life Is Beautiful
-----
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

## Query 14: A query that contains an ASK query form

This query checks whether there exist any triples in the RDF graph where a resource is of type movie:Movie and has properties movie:title and movie:hasDirector. If there are any matches, the query will return true; otherwise, it will return false.

The ASK query in SPARQL returns a boolean value indicating whether the pattern described in the query matches any data in the RDF graph. Therefore, the result will be either True if the pattern exists in the graph, or False if it does not.

```
query11.txt
query12.txt
query13.txt
query14.txt

Query 14
A query that contains an ASK query form

# Define the ASK query
query14 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

ASK {
  ?movie rdf:type movie:Movie ;
  ?movie title ?title ;
  ?movie hasDirector ?director .
}
"""

# Execute the query
result14 = g.query(query14)

# Check if the result is true or false
if result14:
    print("The query returned True: There exist movies with title and director information.")
else:
    print("The query returned False: No movies with title and director information found.")

[14] ✓ 0.0s
... The query returned True: There exist movies with title and director information.
```

## Query 15: A query that contains a DESCRIBE query form

Query to retrieve a description of a specific resource (in this case, a movie, person) from the RDF graph.

```
A query that contains a DESCRIBE query form

from rdflib import Literal, URIRef
# Define the resource URI for which you want to retrieve the description
resource_uri = "http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/Movie1"

# Define and execute the DESCRIBE query
query15 = f'DESCRIBE <{resource_uri}>'
results15 = g.query(query15)

# Iterate over the constructed triples and print the values
for subject, predicate, obj in results15:
    subject_name = subject.split("/")[-1] if isinstance(subject, str) else subject
    predicate_name = predicate.split("/")[-1] if isinstance(predicate, str) else predicate
    obj_name = obj.split("/")[-1] if isinstance(obj, str) else obj
    print(f"Subject: {subject_name} ", f"Predicate: {predicate_name} ", f"Object: {obj_name} ")
    print("-----")

[14] ✓ 0.0s

... Subject: Movie1 Predicate: title Object: Pulp Fiction
-----
Subject: Movie1 Predicate: language Object: English
-----
Subject: Movie1 Predicate: hasWriter Object: QuentinTarantino
-----
Subject: Movie1 Predicate: hasGenre Object: Crime
-----
Subject: Movie1 Predicate: creator Object: rana ahmed
-----
Subject: Movie1 Predicate: hasActor Object: QuentinTarantino
-----
Subject: Movie1 Predicate: hasGenre Object: Thriller
-----
Subject: Movie1 Predicate: country Object: USA
-----
Subject: Movie1 Predicate: hasActor Object: UmaThurman
-----
Subject: Movie1 Predicate: 22-rdf-syntax-ns#type Object: owl:NamedIndividual
```

```
# Define the resource URI for which you want to retrieve the description
resource_uri2 = "http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/BongJoonho"

# Define and execute the DESCRIBE query
query15 = f'DESCRIBE <{resource_uri2}>'
results15 = g.query(query15)

# Iterate over the constructed triples and print the values
for subject, predicate, obj in results15:
    subject_name = subject.split("/")[-1] if isinstance(subject, str) else subject
    predicate_name = predicate.split("/")[-1] if isinstance(predicate, str) else predicate
    obj_name = obj.split("/")[-1] if isinstance(obj, str) else obj
    print(f"Subject: {subject_name} ", f"Predicate: {predicate_name} ", f"Object: {obj_name} ")
    print("-----")

[15] ✓ 0.0s Python

... Subject: BongJoonho Predicate: Actorof Object: Movie9
-----
Subject: BongJoonho Predicate: 22-rdf-syntax-ns#type Object: owl:NamedIndividual
-----
Subject: BongJoonho Predicate: Directorof Object: Movie9
-----
Subject: BongJoonho Predicate: gender Object: Male
-----
Subject: BongJoonho Predicate: 22-rdf-syntax-ns#type Object: Actor
-----
Subject: BongJoonho Predicate: date Object: 2024-05-03T13:32:44+00:00
-----
Subject: BongJoonho Predicate: 22-rdf-syntax-ns#type Object: Director
-----
Subject: BongJoonho Predicate: name Object: Bong Joon-ho
-----
Subject: BongJoonho Predicate: nationality Object: South Korean
-----
Subject: BongJoonho Predicate: Writerof Object: Movie9
-----
Subject: BongJoonho Predicate: 22-rdf-syntax-ns#type Object: Writer
-----
Subject: BongJoonho Predicate: age Object: 52
-----
Subject: BongJoonho Predicate: creator Object: rana ahmed
```

## Query 16: list of movies that have either the genre "Action" or "Thriller"

```
# Define the SPARQL query string
query16 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

SELECT ?movie ?title ?genre
WHERE {
    ?movie rdf:type movie:Movie ;
           movie:title ?title ;
           movie:hasGenre ?genre .
    FILTER (?genre = movie:Action || ?genre = movie:Thriller)
}
"""

# execute the SPARQL query
results16 = g.query(query16)

# Iterate over the results and print them
for row in results16:
    movie_instance = row["movie"]
    title = row["title"]
    genre = row["genre"]
    print(f"Movie instance: {movie_instance}, Title: {title}, Genre: {genre}.")
```

Movie instance: Movie1, Title: Pulp Fiction, Genre: Thriller.  
Movie instance: Movie2, Title: Kill Bill (volume 1), Genre: Action.  
Movie instance: Movie2, Title: Kill Bill (volume 1), Genre: Thriller.  
Movie instance: Movie3, Title: The Dark Knight, Genre: Action.  
Movie instance: Movie5, Title: The Silence Of The Lambs, Genre: Thriller.  
Movie instance: Movie6, Title: Heat, Genre: Action.  
Movie instance: Movie9, Title: Parasite, Genre: Thriller.

## Query 17: List of all movies and Order by years

```
# Define the SELECT query
query17 = """
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

SELECT ?movie ?title ?year
WHERE {
    ?movie rdf:type movie:Movie ;
           movie:title ?title ;
           movie:year ?year .
}
ORDER BY DESC(?year)
"""

# Execute the query
results17 = g.query(query17)

# Print the results
print("List of movies order by the year in descending order:")
for row in results17:
    movie_instance = row["movie"]
    title = row["title"]
    year = row["year"]
    print(f"Movie instance: {movie_instance}, Title: {title}, Year: {year}.")
```

List of movies order by the year in descending order:  
Movie instance: Movie9, Title: Parasite, Year: 2019  
Movie instance: Movie3, Title: The Dark Knight, Year: 2008  
Movie instance: Movie2, Title: Kill Bill (volume 1), Year: 2003  
Movie instance: Movie7, Title: Snatch, Year: 2000  
Movie instance: Movie10, Title: Life Is Beautiful, Year: 1997  
Movie instance: Movie6, Title: Heat, Year: 1995  
Movie instance: Movie1, Title: Pulp Fiction, Year: 1994  
Movie instance: Movie4, Title: Fight Club, Year: 1994  
Movie instance: Movie8, Title: Reservoir Dogs, Year: 1992  
Movie instance: Movie5, Title: The Silence Of The Lambs, Year: 1991

## Query 18: List the count of movies for each genre.

```
FinalNewRdfVml.rdf
Queries.ipynb
query11.txt
query12.txt
query13.txt
query14.txt

List the count of movies for each genre.

# Define the SELECT query
query18 = """
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT ?genre (COUNT(?movie) AS ?count)
WHERE {
  ?movie rdf:type movie:Movie ;
  movie:hasGenre ?genre .
}
GROUP BY ?genre
"""

# Execute the query
results18 = g.query(query18)

# Iterate over the results and print them
print("List count of movies for each genre: ")
for row in results18:
    genre = row["genre"].split("/")[-1]
    count = row["count"]
    print(f"Genre: {genre}, Count: {count}")
    print("-----")

[38] ✓ 0.0s Python

List count of movies for each genre:
Genre: Crime, Count: 7
-----
Genre: Thriller, Count: 4
-----
Genre: Comedy, Count: 3
-----
Genre: Drama, Count: 7
-----
Genre: Action, Count: 3
-----
```

## Query 19: List of information about movies and their language if exists

```
FinalNewRdfVml.rdf
Queries.ipynb
query11.txt
query12.txt
query13.txt
query14.txt

# Define the SPARQL query
query19 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movei-Project/>

SELECT DISTINCT ?movie ?title ?director ?genre ?language
WHERE {
  ?movie rdf:type movie:Movie ;
  movie:title ?title ;
  movie:hasDirector ?director ;
  movie:hasGenre ?genre .
  OPTIONAL { ?movie movie:language ?language }
}
"""

# Execute the query
results19 = g.query(query19)
movie_names = set()

# Iterate over the results and print them
for row in results19:
    movie_name = row["movie"].split("/")[-1]
    title = row["title"].split("/")[-1]
    director = row["director"].split("/")[-1]
    genre = row["genre"].split("/")[-1]
    language = row["language"].split("/")[-1] if row["language"] else "Not specified"

    # If movie name is not already printed, print all details
    if movie_name not in movie_names:
        print(f"Movie Name: {movie_name}, Title: {title}, Director: {director}, Genre: {genre}, Language: {language}")
        movie_names.add(movie_name)

[39] ✓ 0.0s Python

Movie Name: Movie1, Title: Pulp Fiction, Director: QuentinTarantino, Genre: Crime, Language: English
Movie Name: Movie10, Title: Life Is Beautiful, Director: RobertoBenigni, Genre: Comedy, Language: Italian
Movie Name: Movie2, Title: Kill Bill (volume 1), Director: QuentinTarantino, Genre: Action, Language: English
Movie Name: Movie3, Title: The Dark Knight, Director: ChristopherNolan, Genre: Action, Language: English
Movie Name: Movie4, Title: FightClub, Director: DavidFincher, Genre: Drama, Language: English
Movie Name: Movie5, Title: The Silence Of The Lambs, Director: JonathanDemme, Genre: Crime, Language: English
Movie Name: Movie6, Title: Heat, Director: MichaelMann, Genre: Action, Language: English
Movie Name: Movie7, Title: Snatch, Director: GuyRitchie, Genre: Comedy, Language: English
Movie Name: Movie8, Title: Reservoir Dogs, Director: QuentinTarantino, Genre: Crime, Language: English
Movie Name: Movie9, Title: Parasite, Director: BongJoonHo, Genre: Comedy, Language: Korean
```

Query 20 :List the top 5 directors with the most movies and counts the number of movies each director has directed.

```
Query 20
Lis the top 5 directors with the most movies and counts the number of movies each director has directed.

# Define the SPARQL query
query20 = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movie: <http://www.semanticweb.org/ranaahmed/ontologies/2024/4/Movie-Project/>

SELECT ?director (COUNT(?movie) AS ?movieCount)
WHERE {
  ?movie rdf:type movie:Movie ;
  ?movie movie:hasDirector ?director .
}
GROUP BY ?director
ORDER BY DESC(?movieCount)
LIMIT 5
"""

# Execute the query
results20 = g.query(query20)

# Iterate over the results and print them
for row in results20:
    director = row["director"].split("/")[-1]
    movie_count = row["movieCount"]
    print(f"Director: {director}, Movie Count: {movie_count}")
```

Director: QuentinTarantino, Movie Count: 3  
Director: RobertDeNiro, Movie Count: 1  
Director: ChristopherHolan, Movie Count: 1  
Director: DavidFincher, Movie Count: 1  
Director: JonathanDemme, Movie Count: 1

### 3 Visualize ontology.

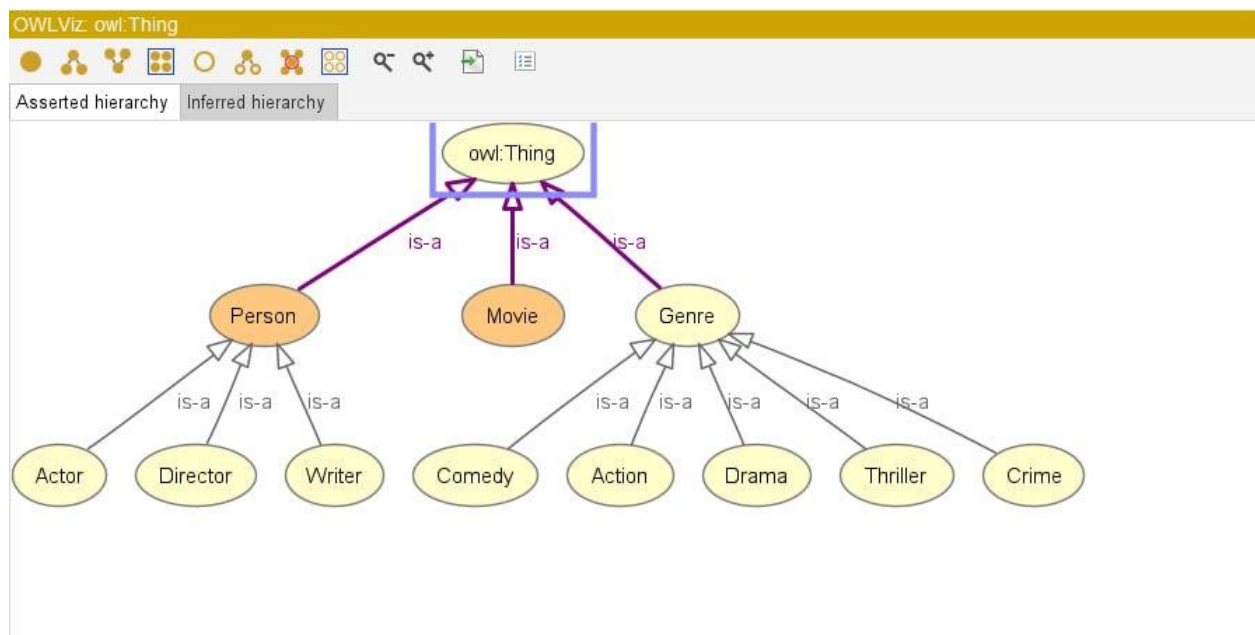


Figure 7 Class hierarchy using protege

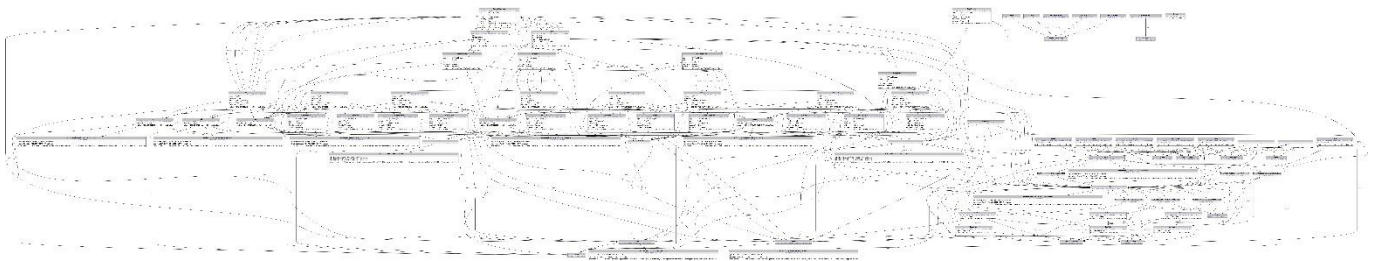
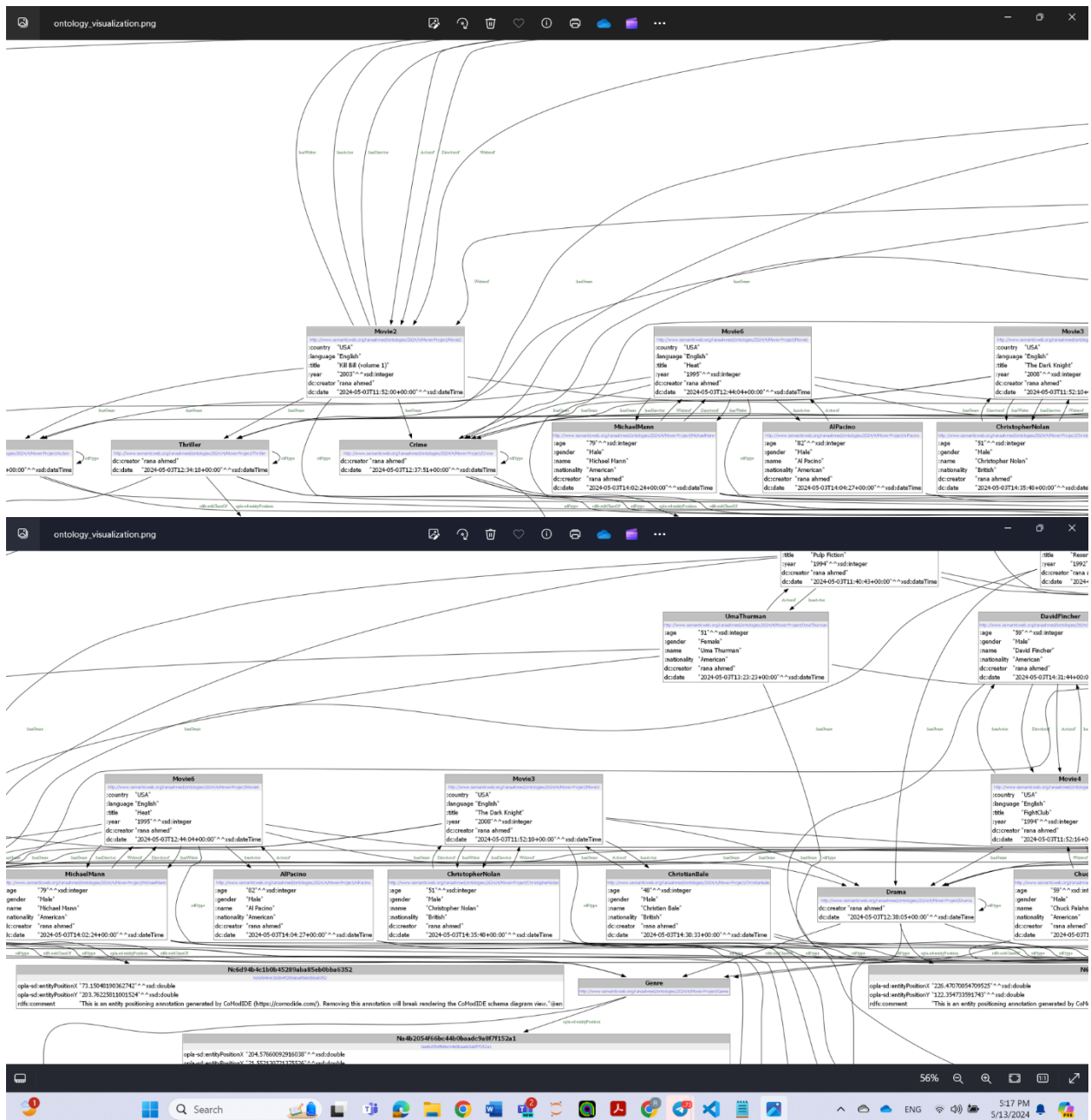


Figure 8 Ontology visualization





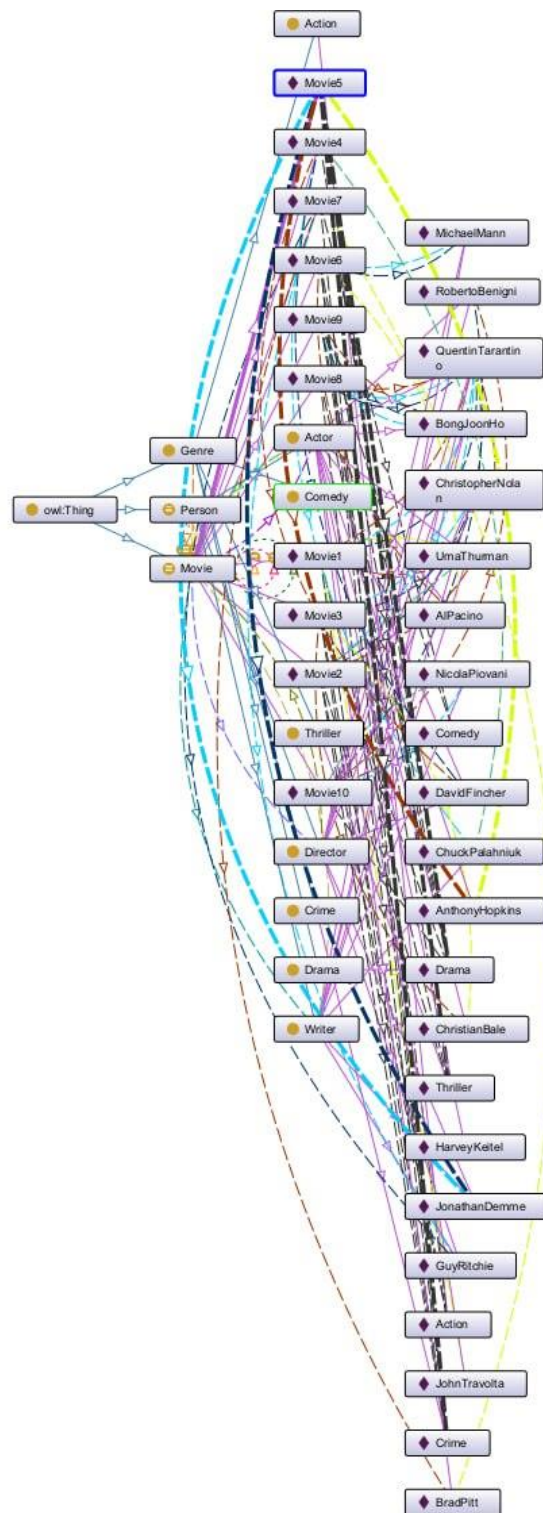


Figure 9 Visualize of ontology using ontograph

## 4 Snapshots of the Interface (GUI)

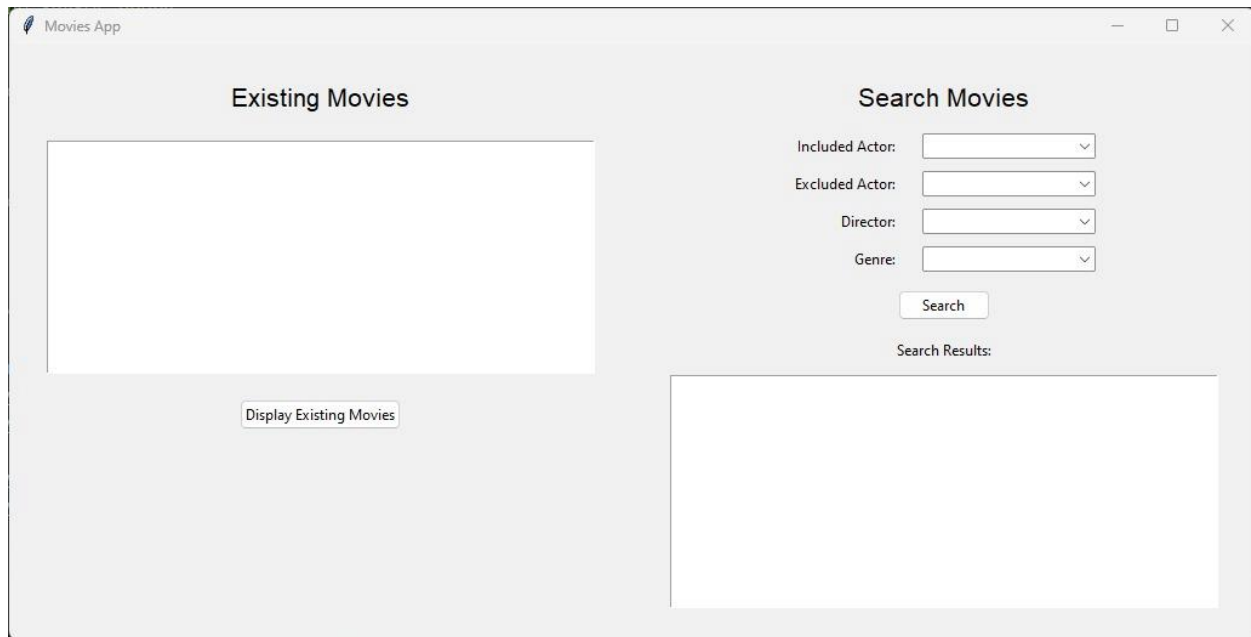


Figure 10 First Screen for GUI

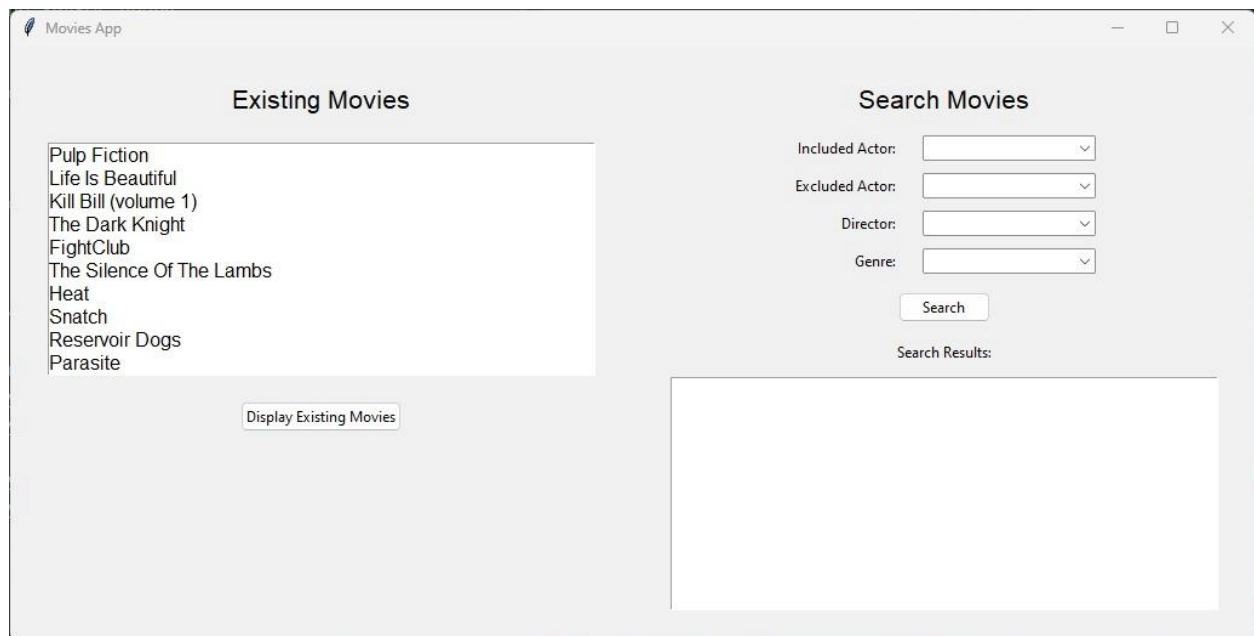


Figure 11 Pressing on Display Existing Movies Button

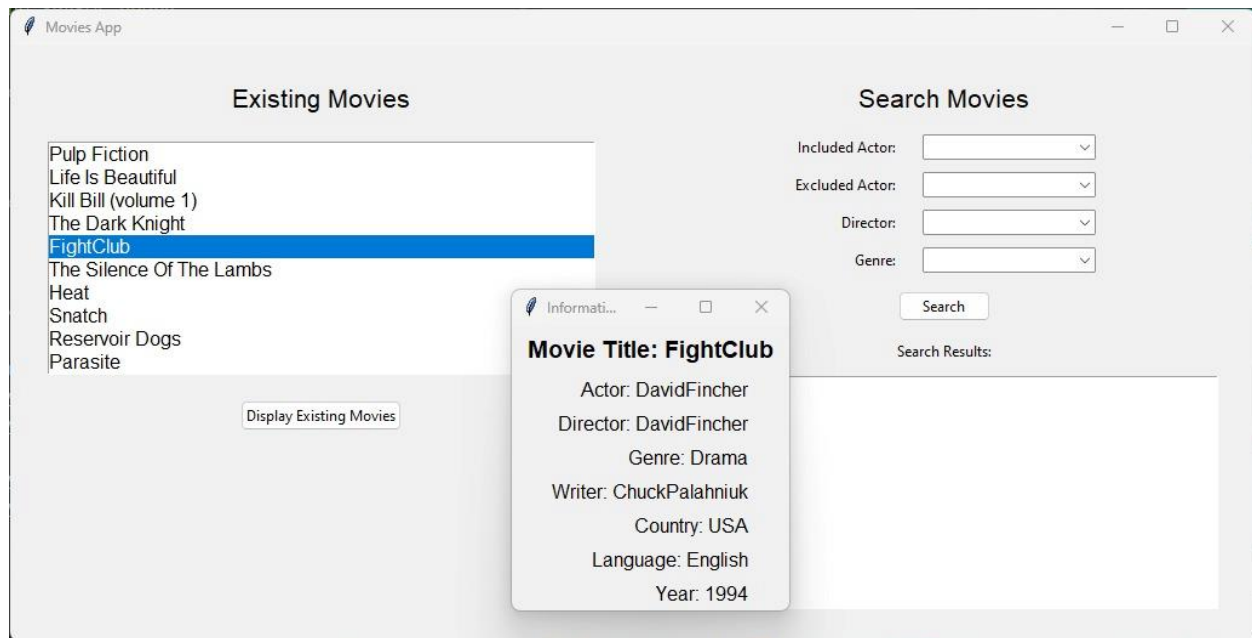


Figure 12 Selecting Movie to show its details

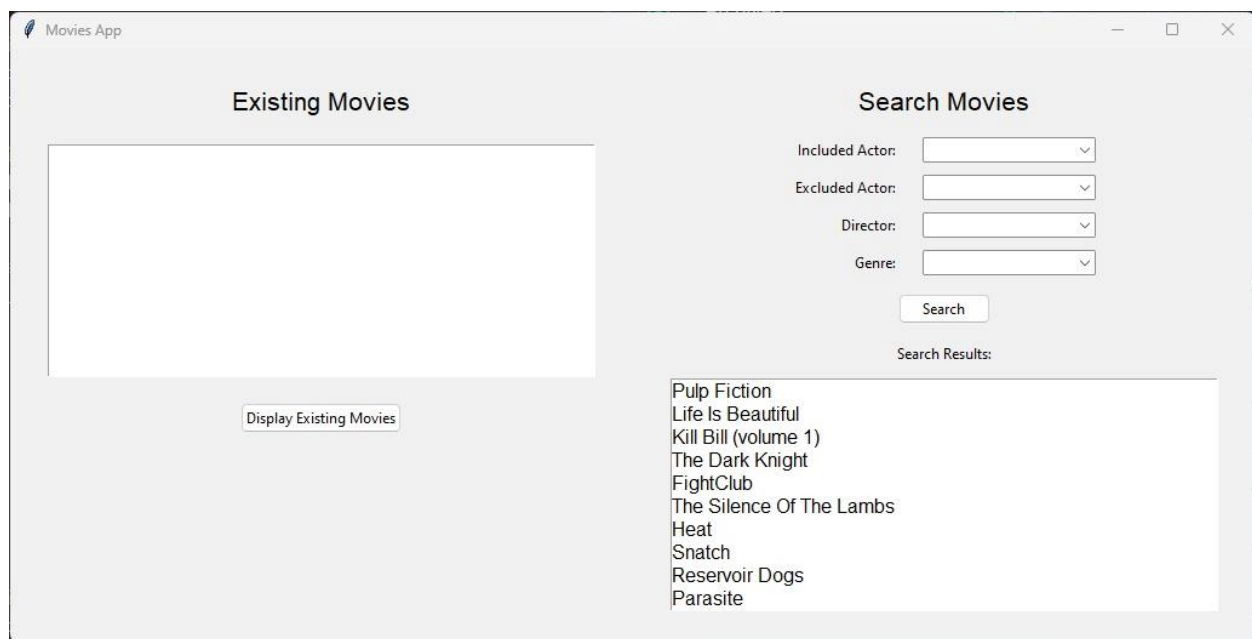


Figure 13 Pressing Search button without selecting any restrictions

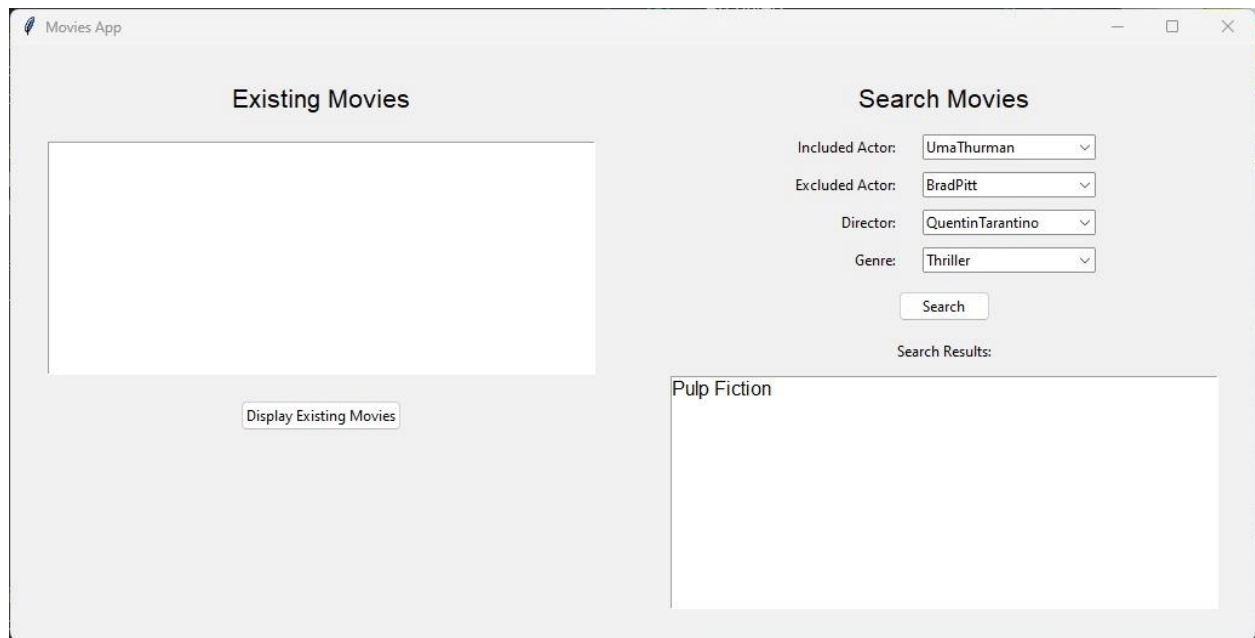


Figure 14 Search with including/ excluding/ Director and genre.

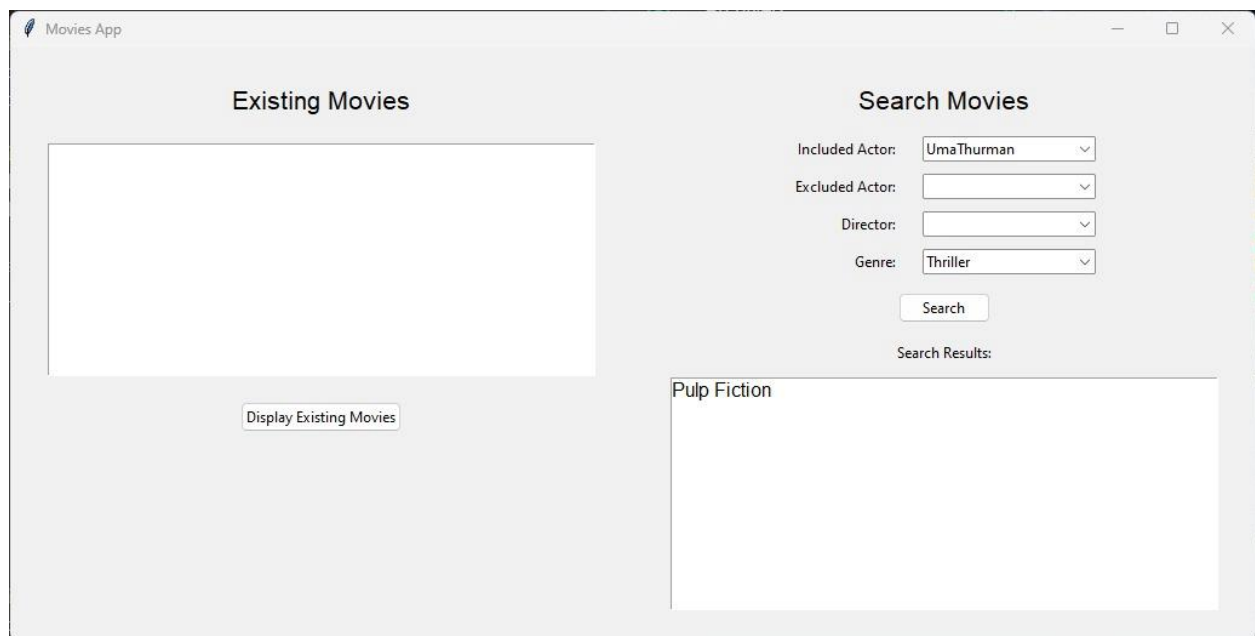


Figure 15 Searching with including actor and genre.

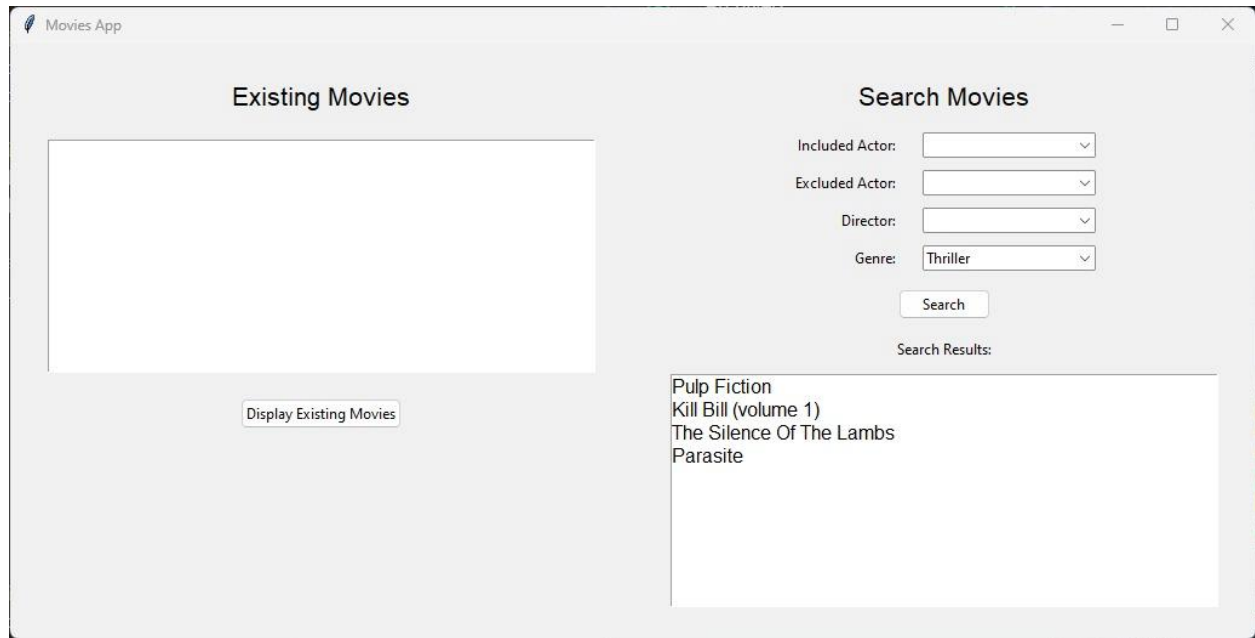


Figure 16 Searching for Specific Genre

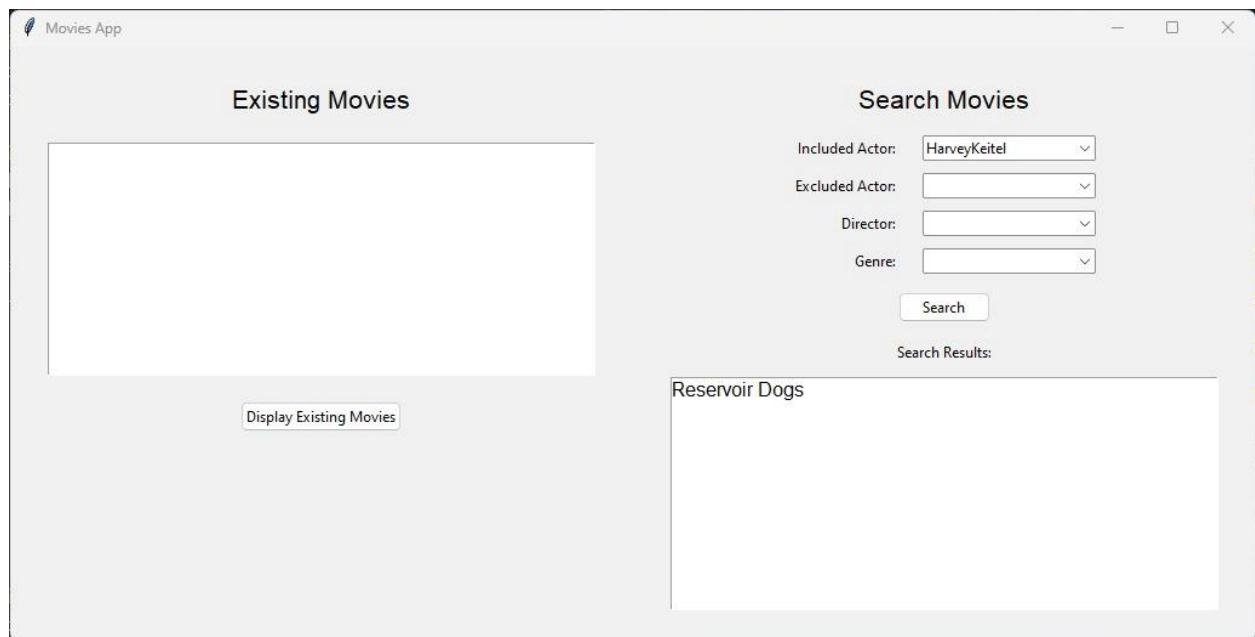


Figure 17 Searching by including actor

## 5 Test Cases

- a. Showing all films that Excluding specific actor (Quentin Tarantino)

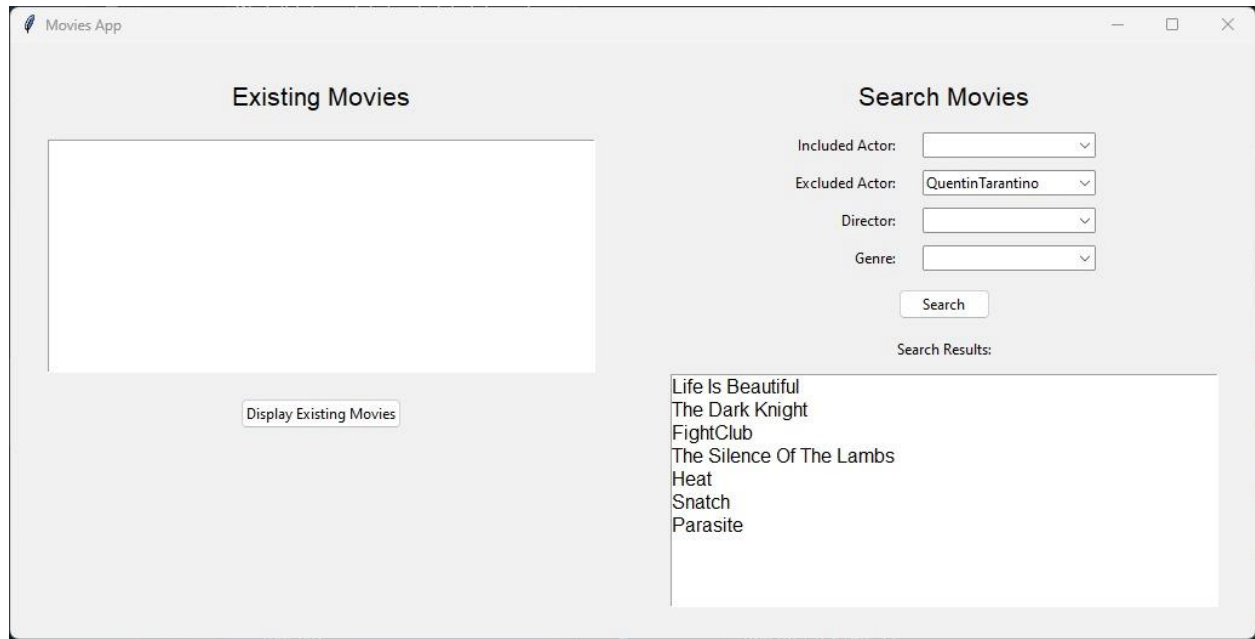


Figure 18 Answer for the first test case

- b. Showing all movies that Including specific actor (Harvey Keitel)

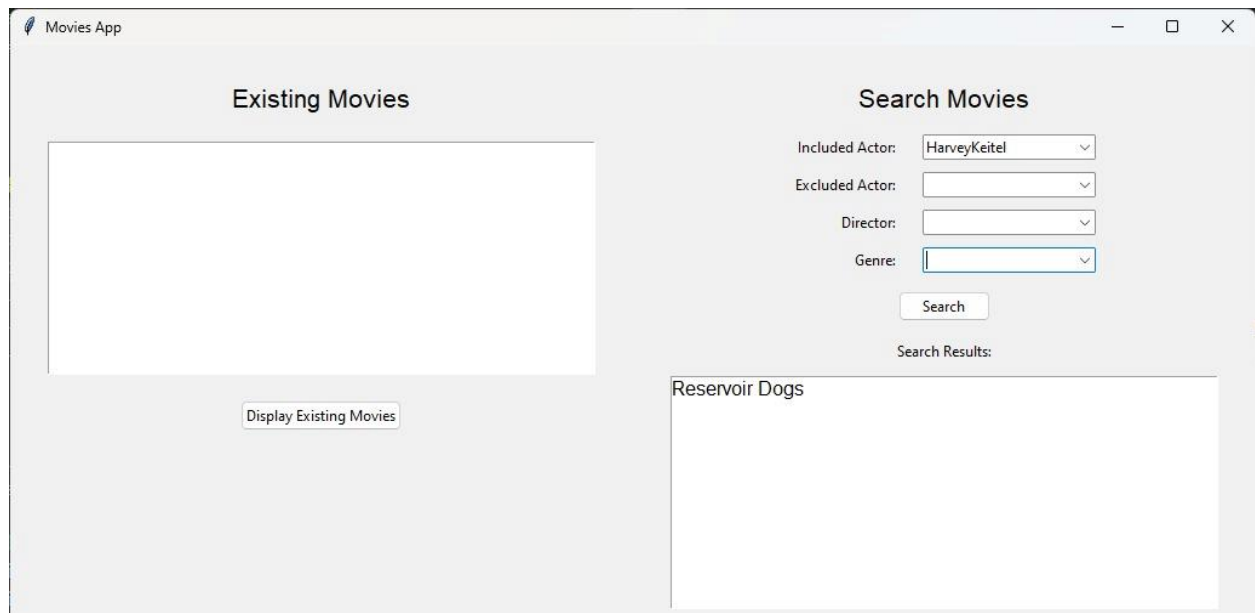


Figure 19 Answer for the second test case

- c. Showing all movies that Including specific actor (Quentin Tarantino)

The screenshot shows the 'Movies App' interface. On the left, under 'Existing Movies', there is a large empty rectangular box and a button labeled 'Display Existing Movies'. On the right, under 'Search Movies', there are four dropdown menus: 'Included Actor' (set to 'QuentinTarantino'), 'Excluded Actor' (empty), 'Director' (empty), and 'Genre' (empty). Below these is a 'Search' button. Under 'Search Results:', a list of movies is displayed: 'Pulp Fiction', 'Kill Bill (volume 1)', and 'Reservoir Dogs'.

Figure 20 Answer for third test case.

- d. Showing all movies that Including specific actor (Quentin Tarantino) and with genre thriller

The screenshot shows the 'Movies App' interface with the 'Genre' dropdown menu set to 'Thriller'. The 'Existing Movies' section on the left remains the same with an empty box and a 'Display Existing Movies' button. The 'Search Movies' section on the right has the 'Included Actor' set to 'QuentinTarantino' and 'Genre' set to 'Thriller'. The 'Search' button is present. The 'Search Results:' list now only contains 'Pulp Fiction' and 'Kill Bill (volume 1)', as 'Reservoir Dogs' is filtered out by the 'Thriller' genre selection.

Figure 21 Answer for fourth test case.

- e. Showing all movies that Including specific actor (Quentin Tarantino), with genre thriller, and excluding Actor Uma Thurman

The screenshot shows the 'Movies App' window. On the left, the 'Existing Movies' section has a large empty box and a 'Display Existing Movies' button. On the right, the 'Search Movies' section contains four dropdown menus: 'Included Actor' (set to 'QuentinTarantino'), 'Excluded Actor' (set to 'UmaThurman'), 'Director' (empty), and 'Genre' (set to 'Thriller'). Below these is a 'Search' button. Underneath the search filters, the 'Search Results:' label is followed by a box containing the text 'Kill Bill (volume 1)'.

Figure 22 Answer for fifth test case.

- f. Showing all movies that Including specific actor (Quentin Tarantino), with genre thriller, excluding Actor Uma Thurman , and specifying the director is Quentin Tarantino

This screenshot is similar to the previous one, but the 'Director' dropdown menu in the 'Search Movies' section is now set to 'QuentinTarantino'. The 'Search Results:' box still displays 'Kill Bill (volume 1)'.

Figure 23 Answer for sixth test case.



## 6 Data Flow Diagram (DFD)

### Context Diagram

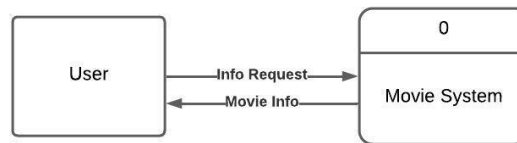


Figure 24 Context Diagram of DFD

### Level 0

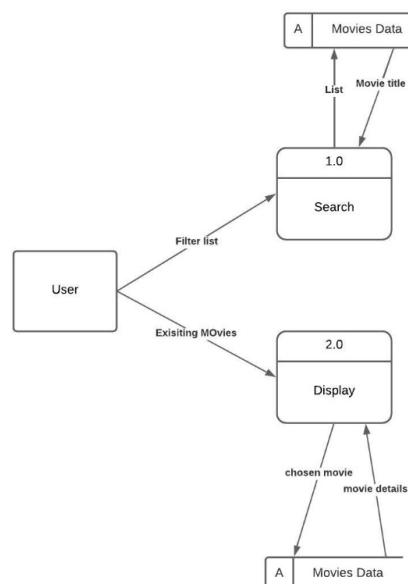


Figure 25 Level 0 of DFD

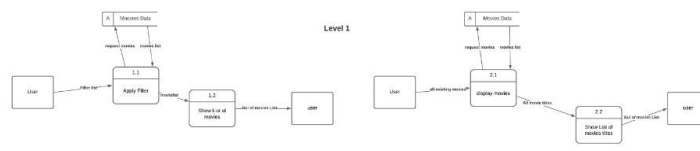


Figure 26 level 1 DFD