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.

1.1Problem 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.

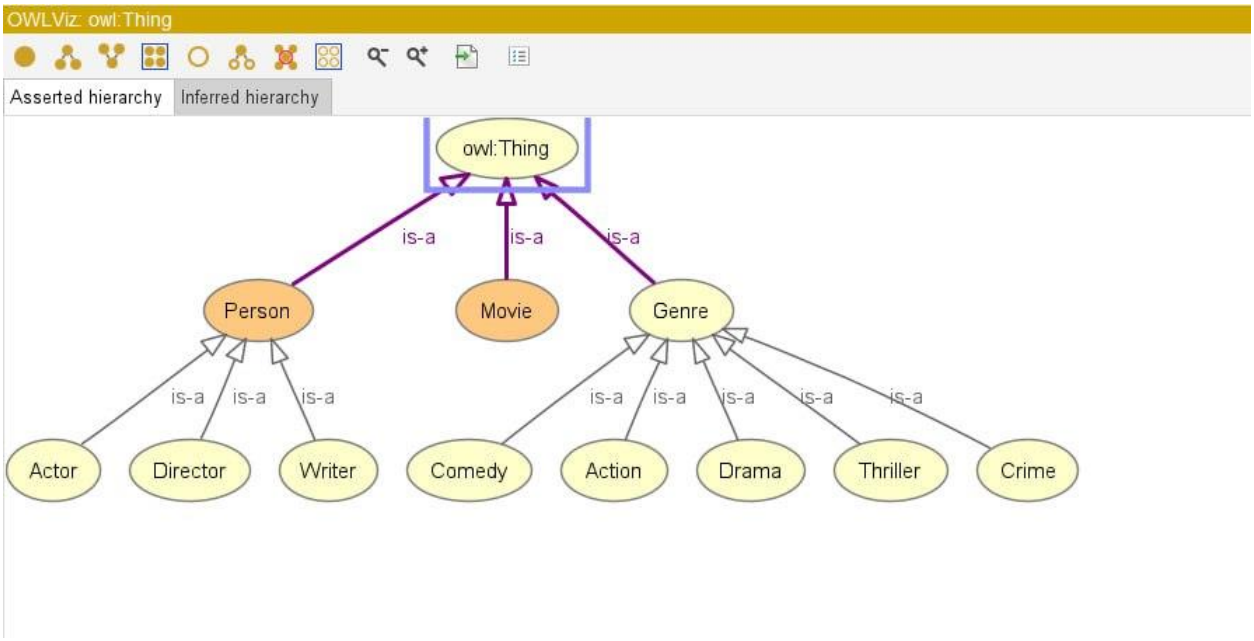


Figure 1 RDF for class hierarchy using protege

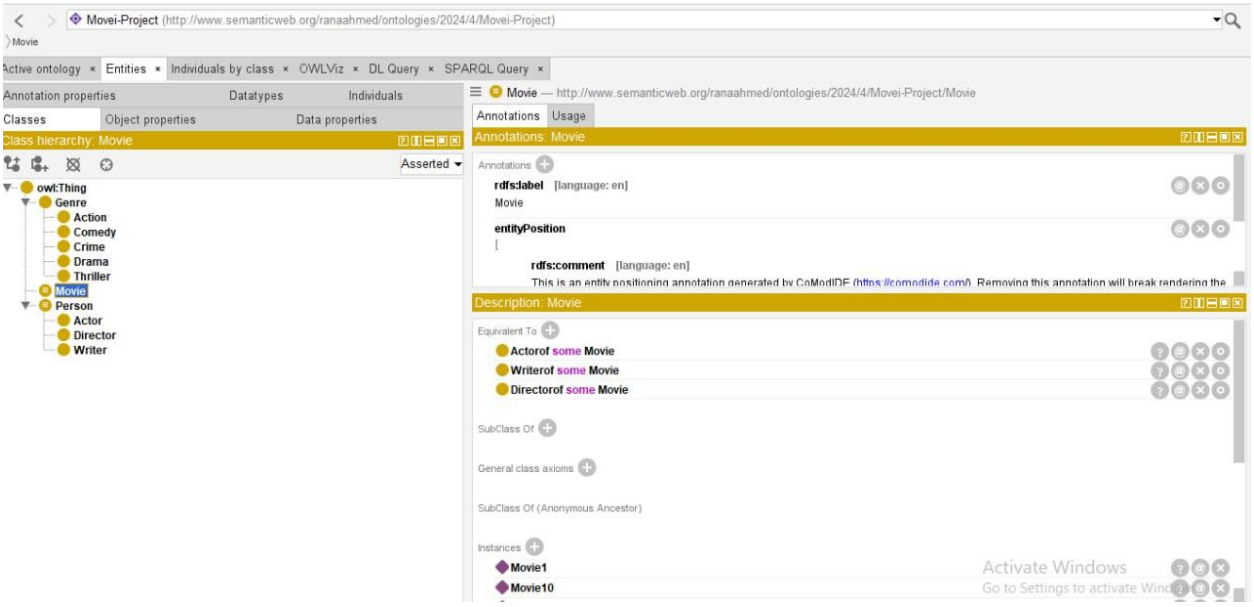


Figure 2 Screenshot for Entity and its subclasses

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.

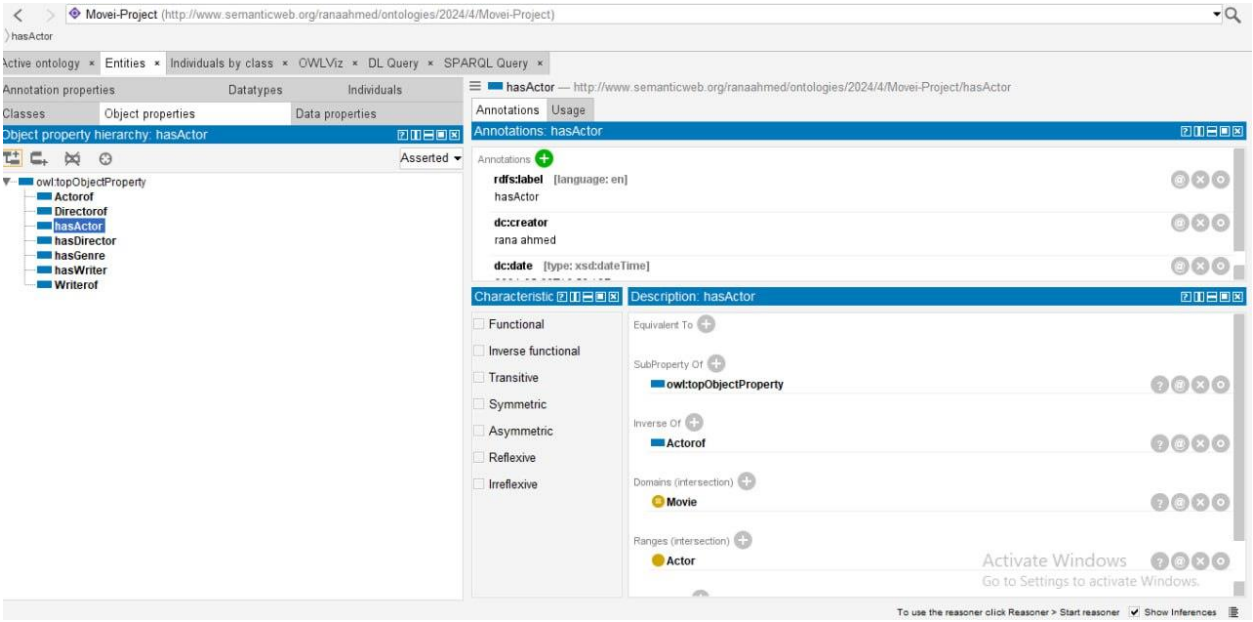


Figure 3 Screenshot for the Object property

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.

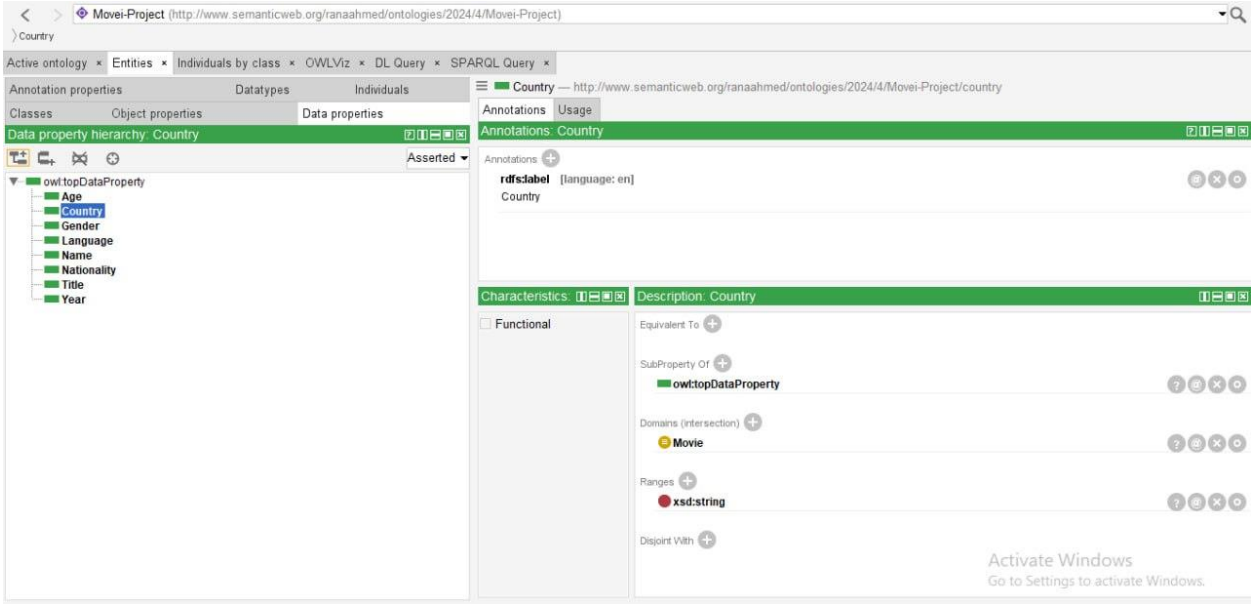


Figure 4 Screenshot for the Data property.

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.

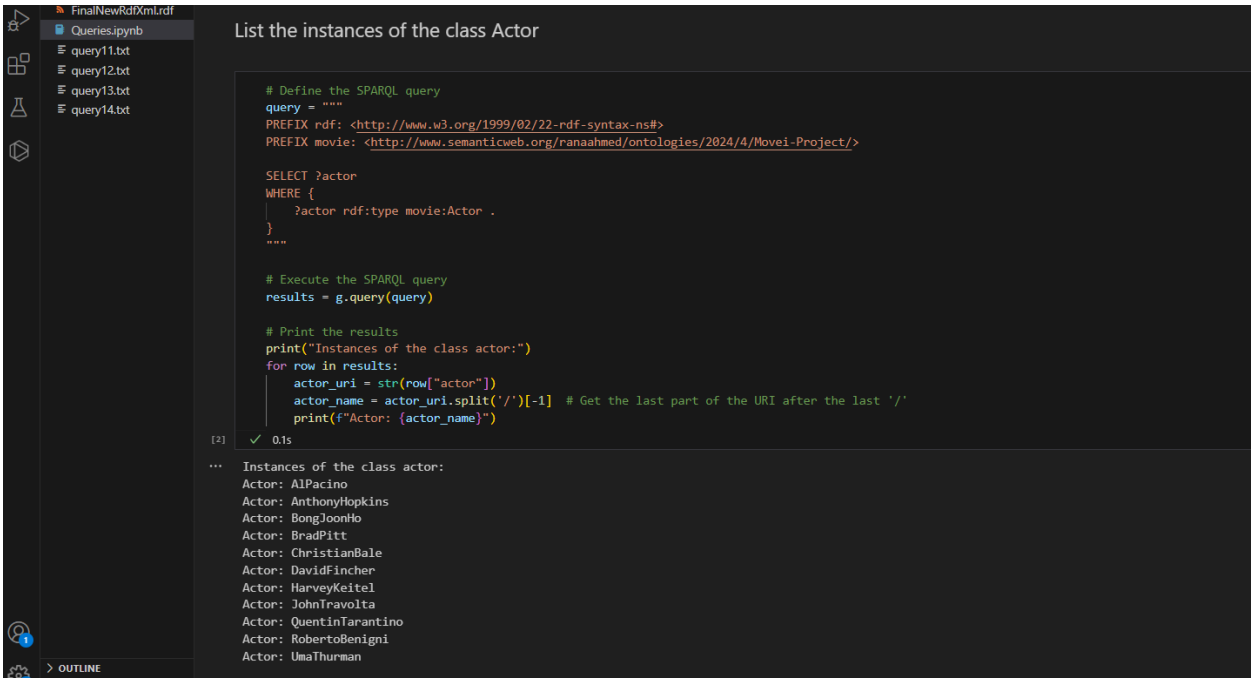
5) Inference Rules:

- 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.

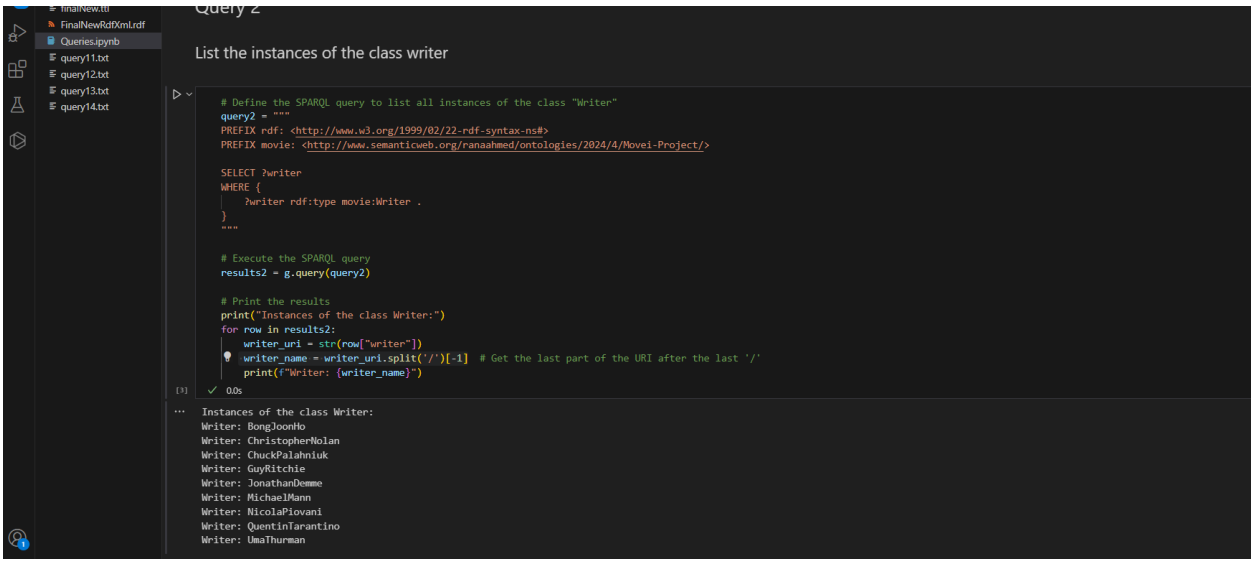
Once the inference rules are defined, they are applied to the RDF graph using SPARQL update queries. Each rule is executed separately, updating the graph with the inferred class memberships for movies. After applying the rules, we use SPARQL select queries to retrieve the inferred movie data based on 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

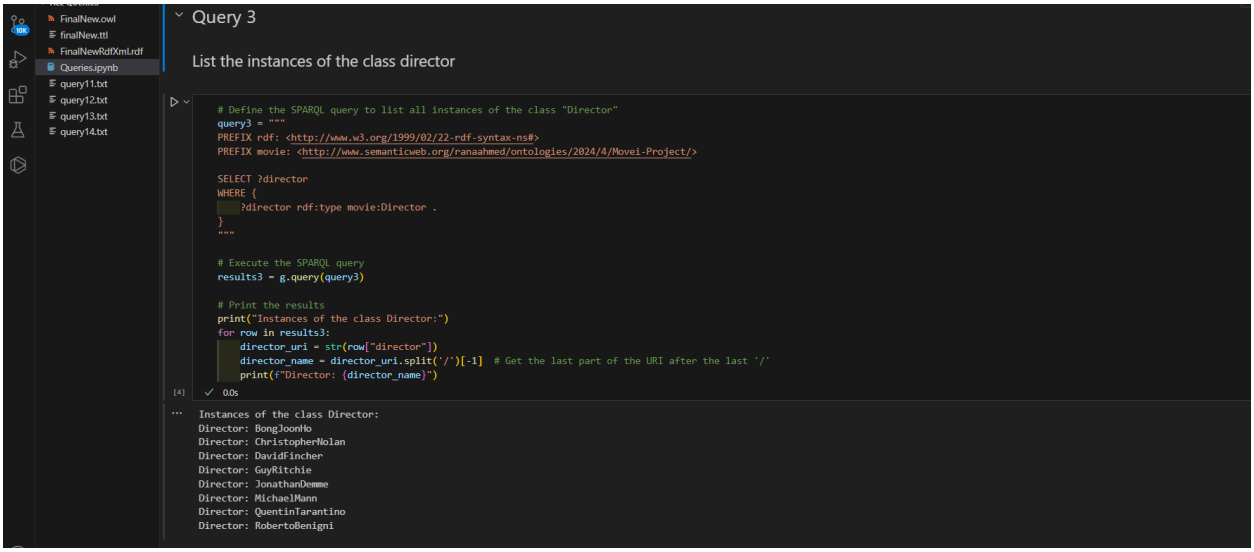
Query1: List the instances of the class Actor



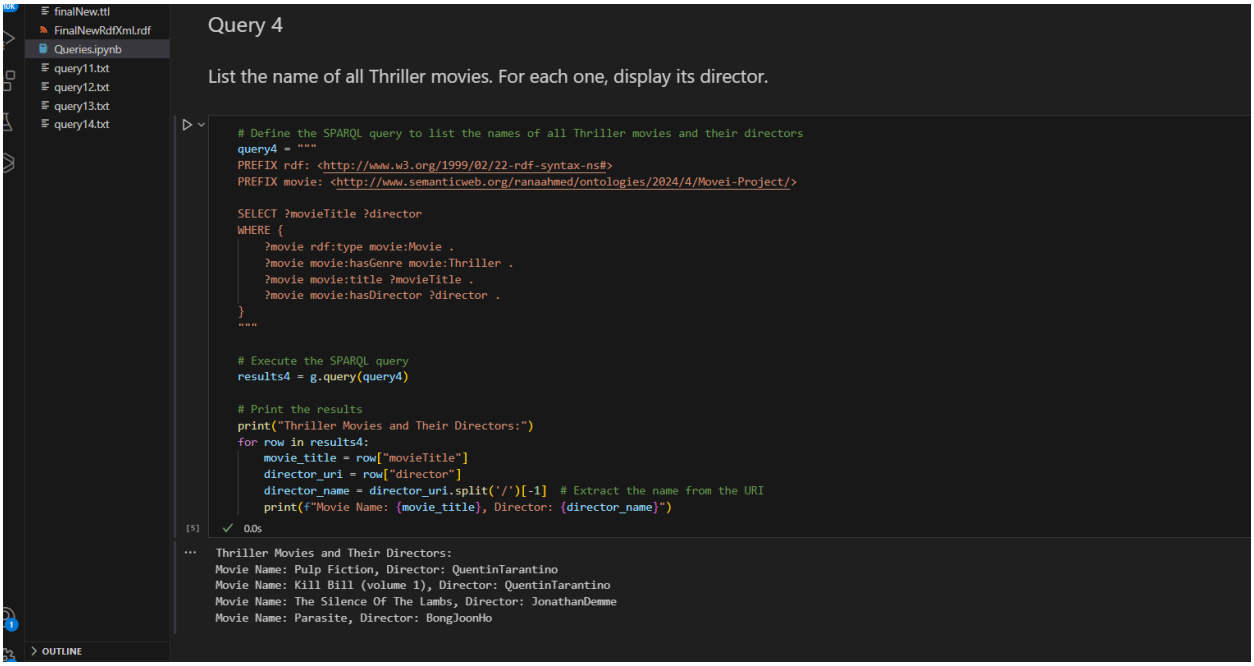
Query2: List the instances of the class Writer



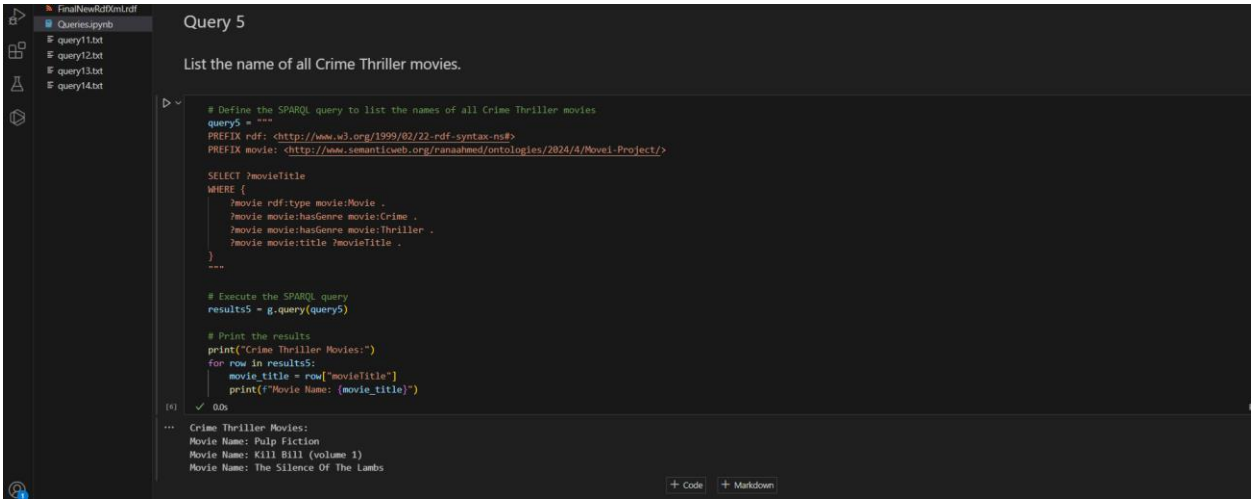
Query3: List the instances of the class Director



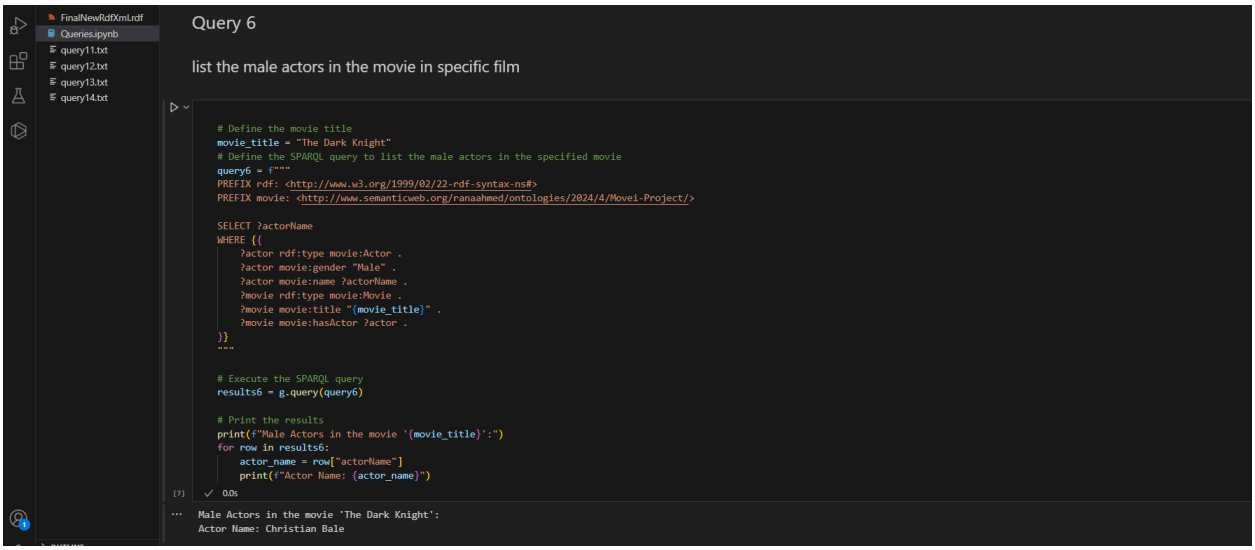
Query 4: List the name of all Thriller movies. For each one, display its director.



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

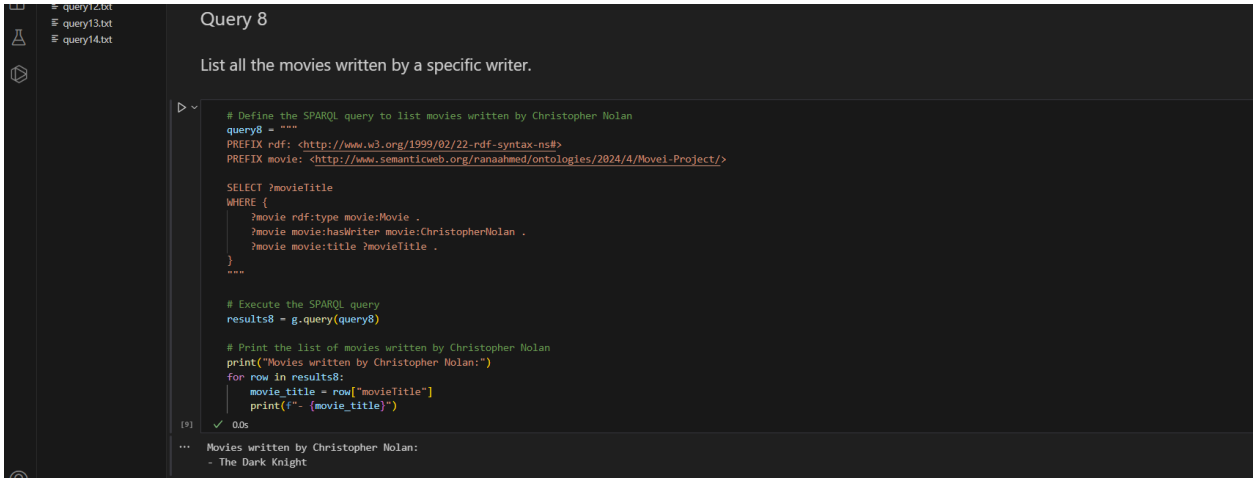


Query 6: list the male actors in the movie in specific film

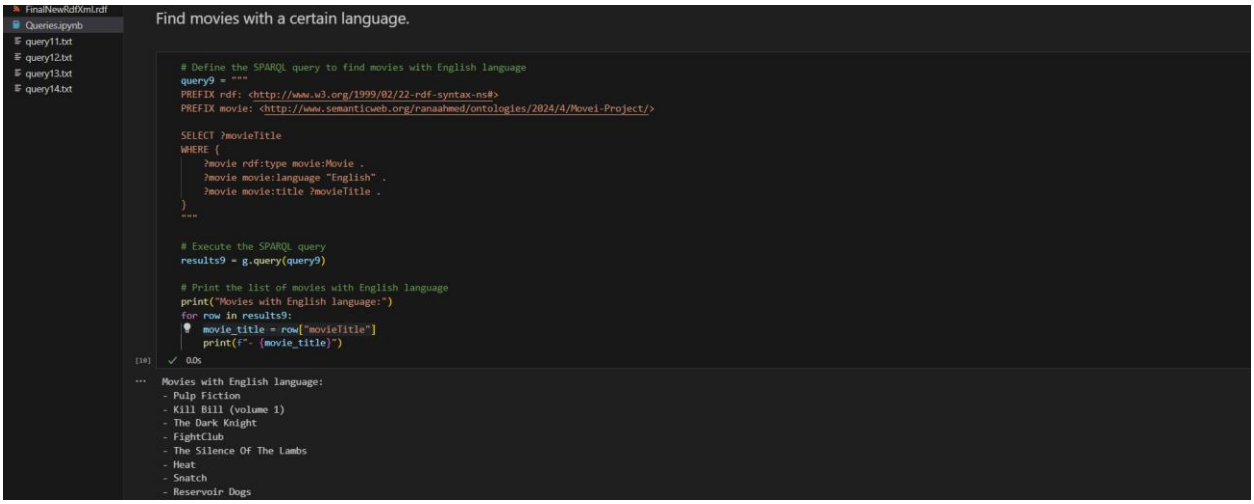


Query 7: How many movies have both "Action" and "Thriller" as genres?

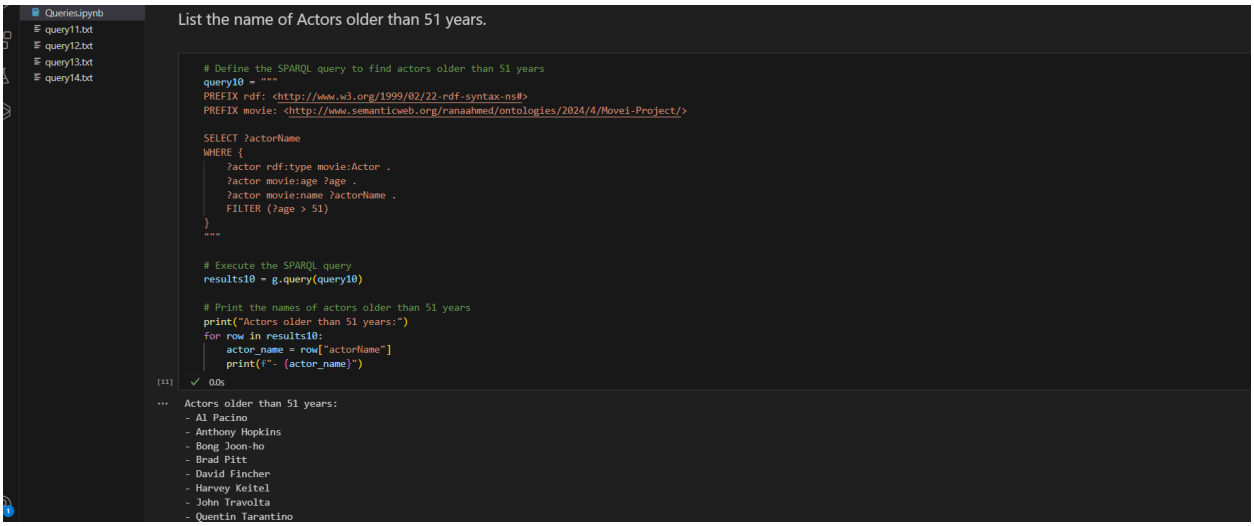
Query 8: List all the movies written by a specific writer.



Query 9: Find movies with a certain language.



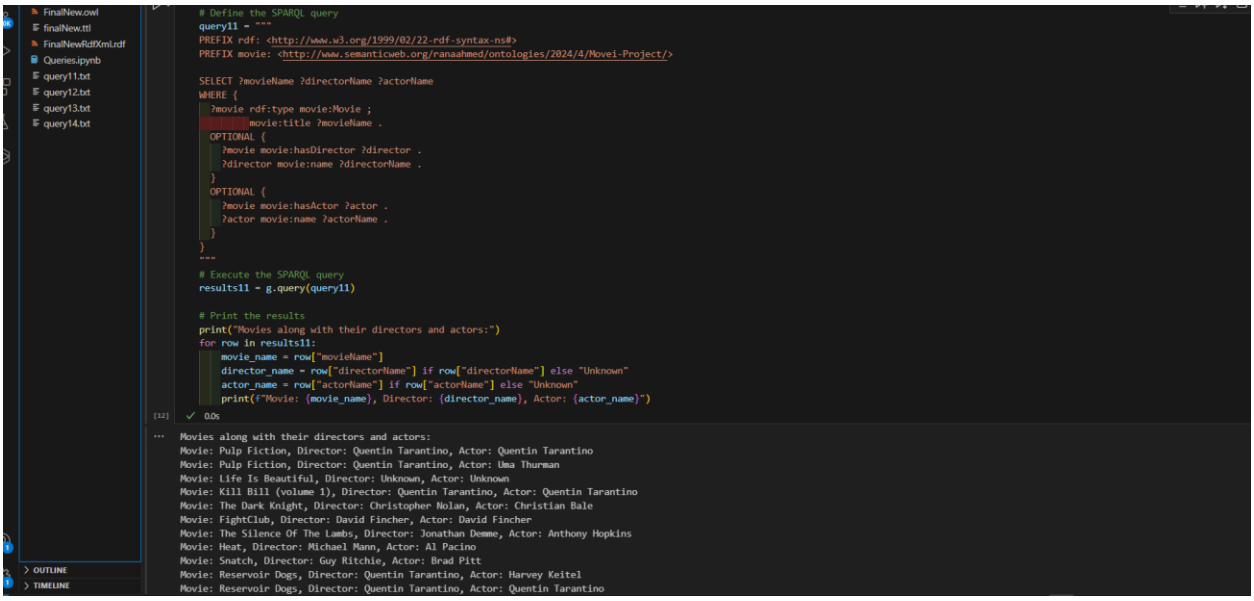
Query 10: List the name of Actors older than 51 years.



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

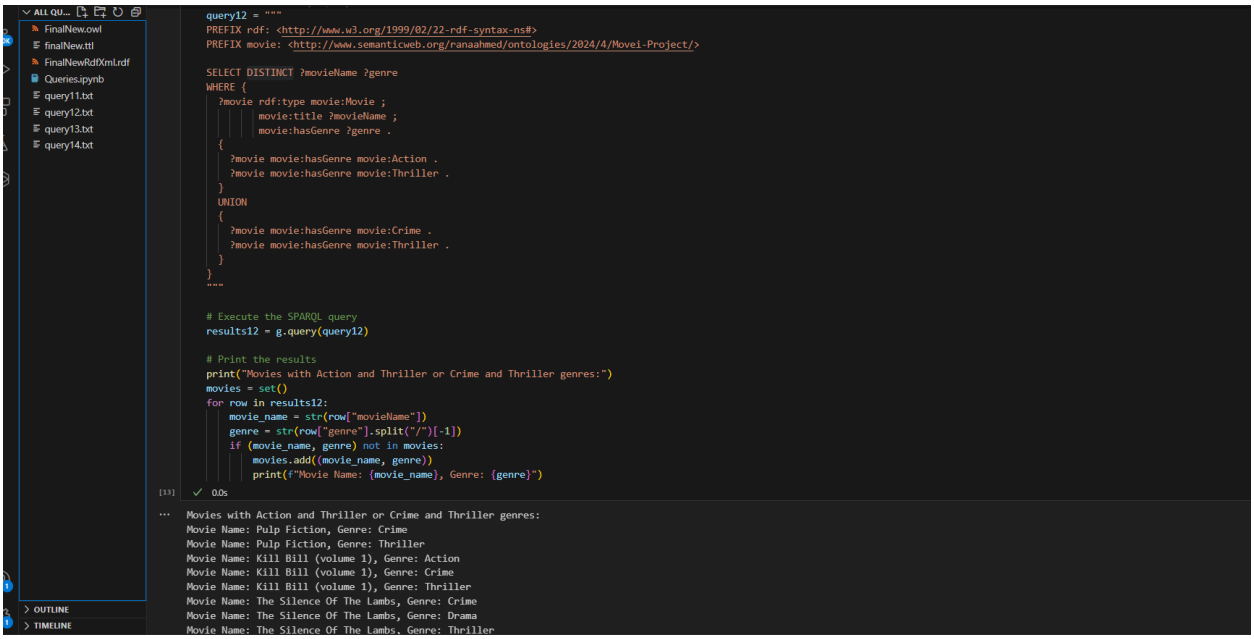
In this query:

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.



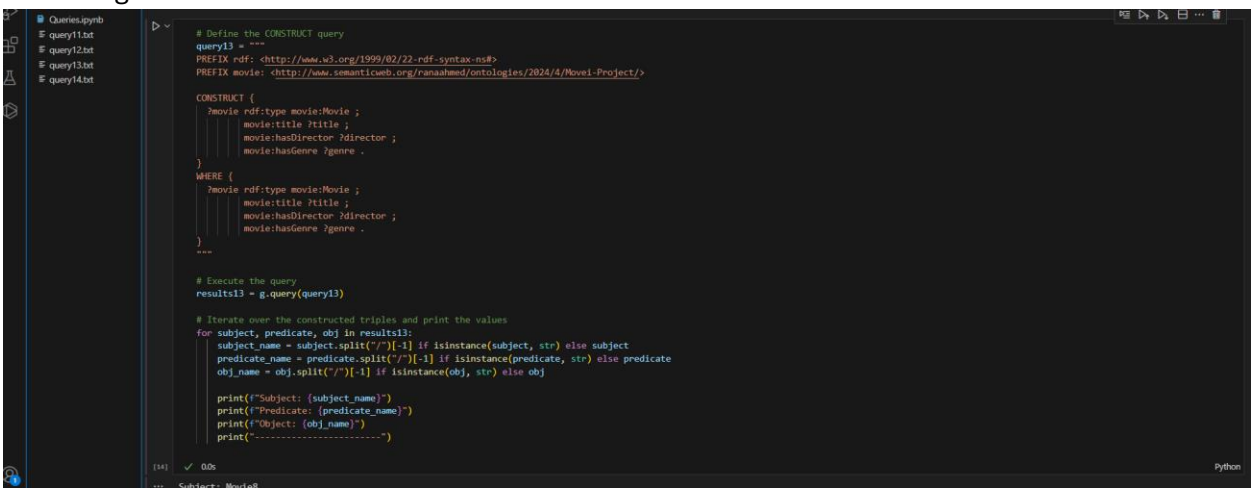
Query 12: A query that contains at least 2 alternatives and conjunctions.

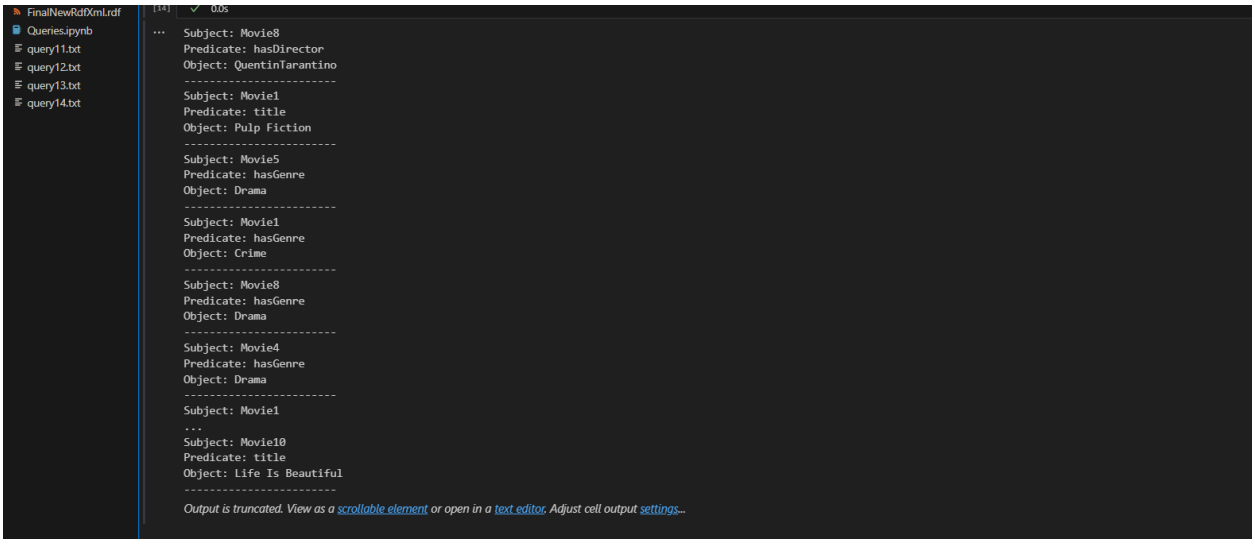
The SPARQL query retrieves movie names and their corresponding genres. It uses alternatives and conjunctions to find movies with specific combinations of genres: either both "Action" and "Thriller" or both "Crime" and "Thriller". The SELECT statement specifies the variables to retrieve, while the WHERE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



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.Tthis 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

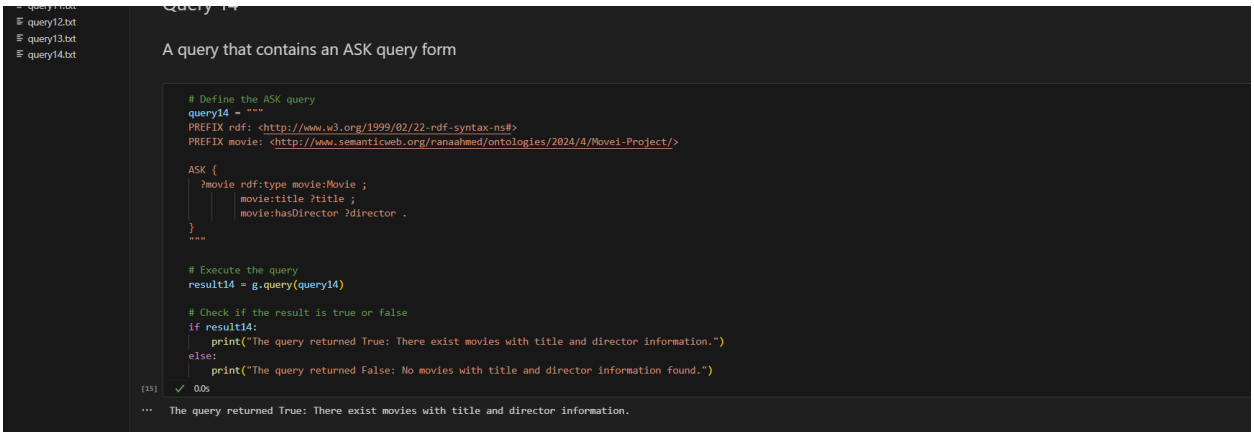




Query 14: A query that contains an ASK query form

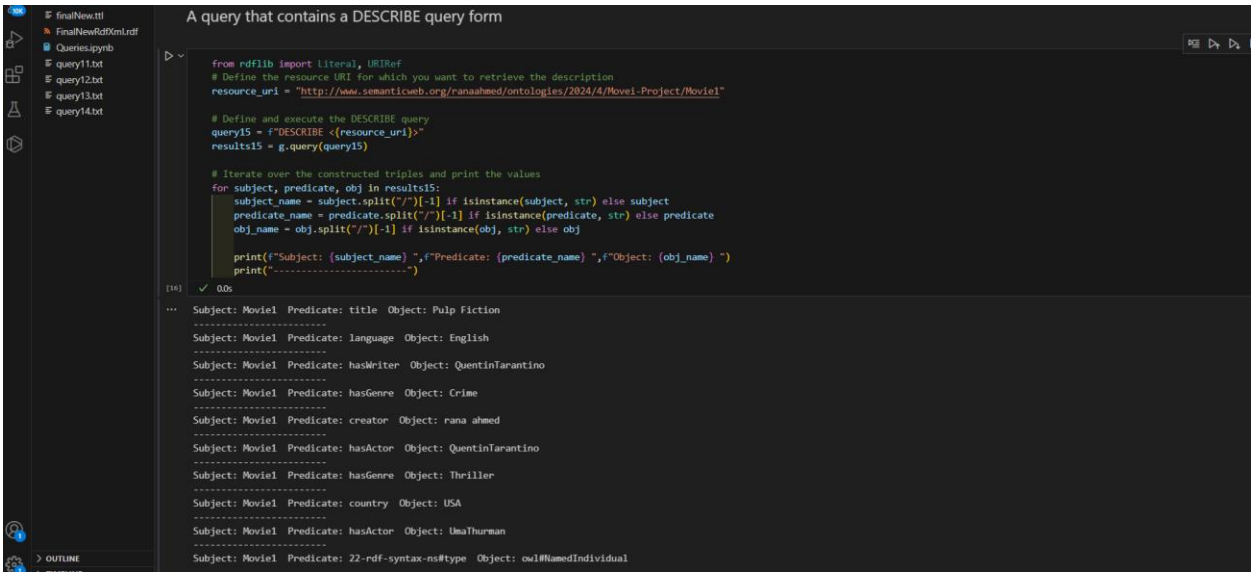
This query checks whether there exist any triples in the RDF graph where a resource is of type movie:Movie and has properties movie:title and movie:hasDirector. If there are any matches, the query will return true; otherwise, it will return false.

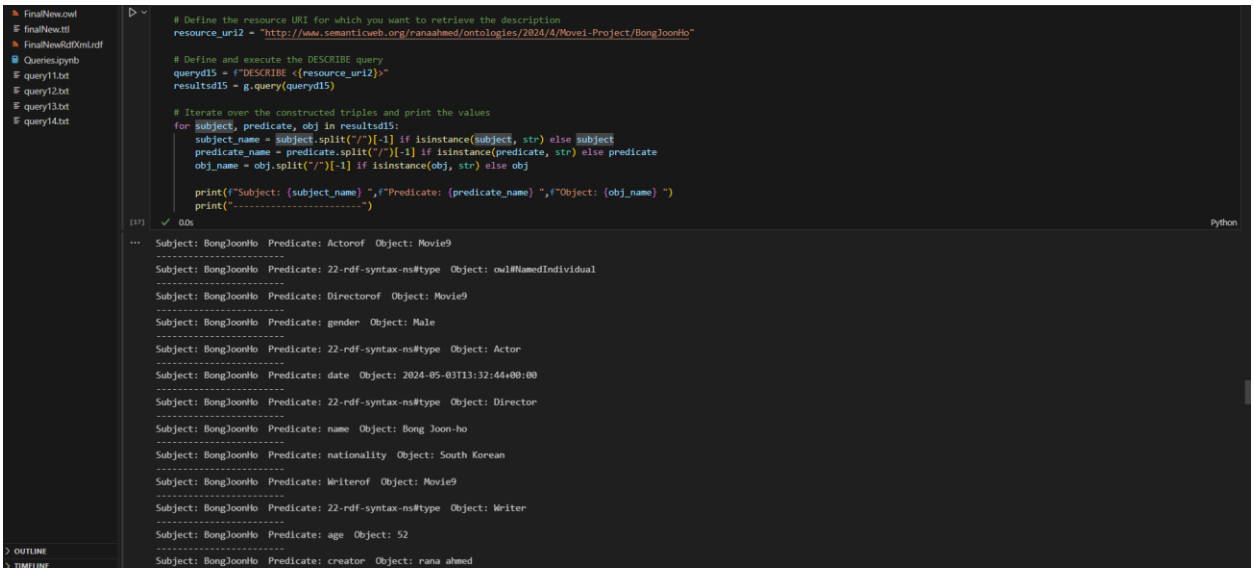
The ASK query in SPARQL returns a boolean value indicating whether the pattern described in the query matches any data in the RDF graph. Therefore, the result will be either True if the pattern exists in the graph, or False if it does not.



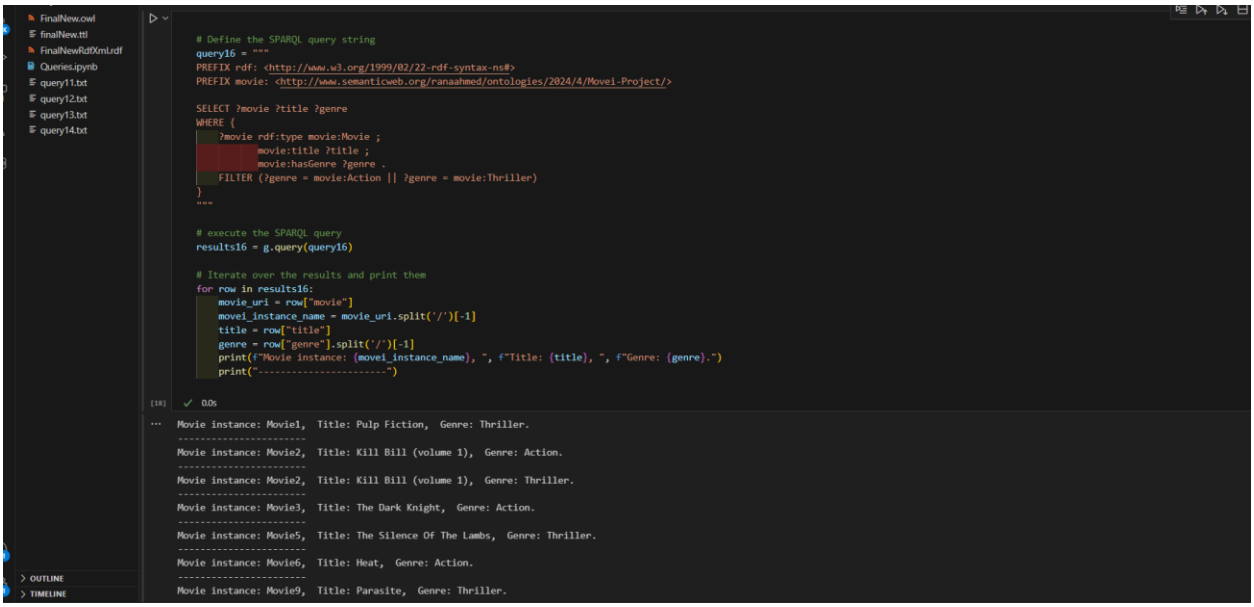
Query 15: A query that contains a DESCRIBE query form

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

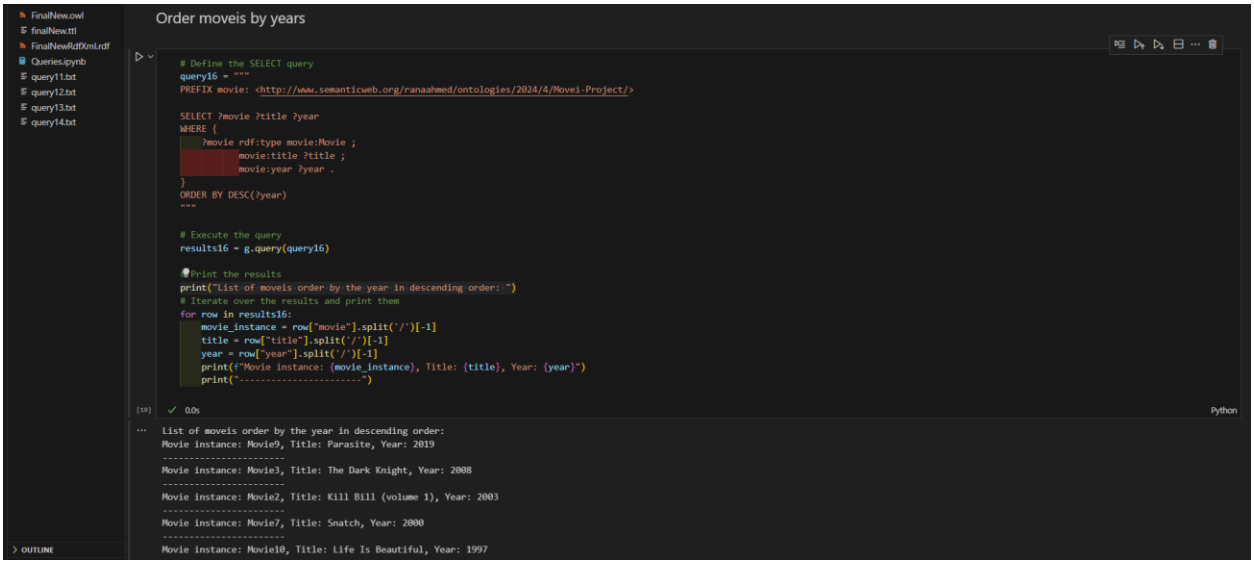




Query 16: list of movies that have either the genre "Action" or "Thriller"

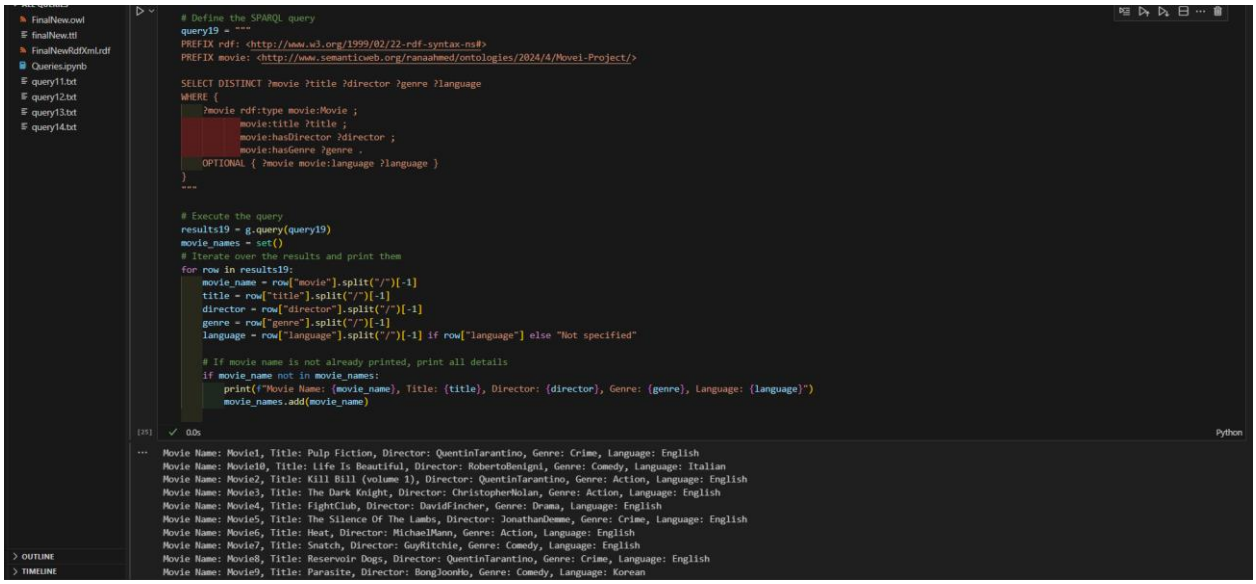


Query 17: List of all movies and Order by years

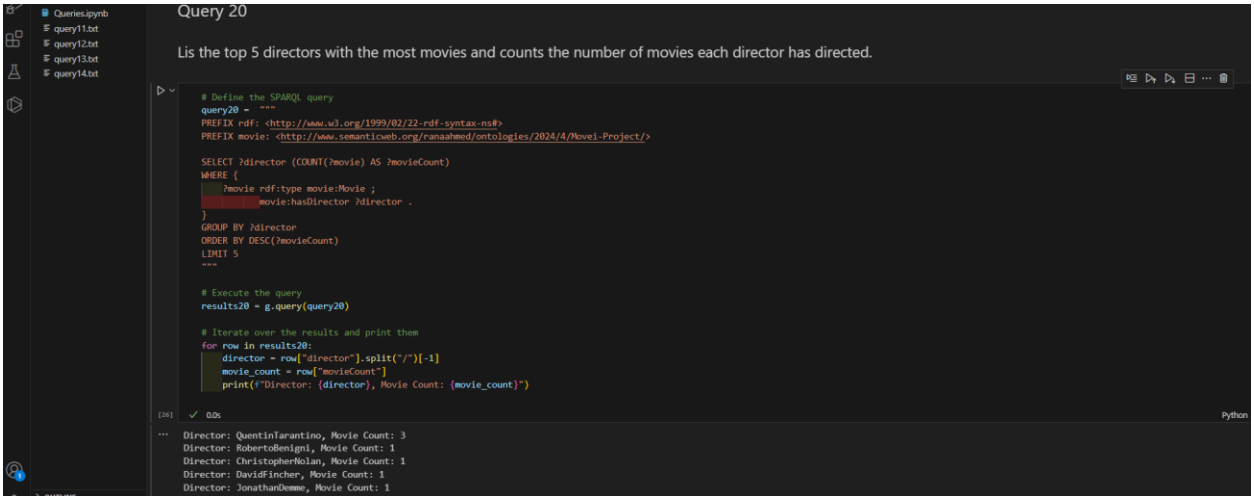


Query 18: List the count of movies for each genre.

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



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



3 Visualize ontology.

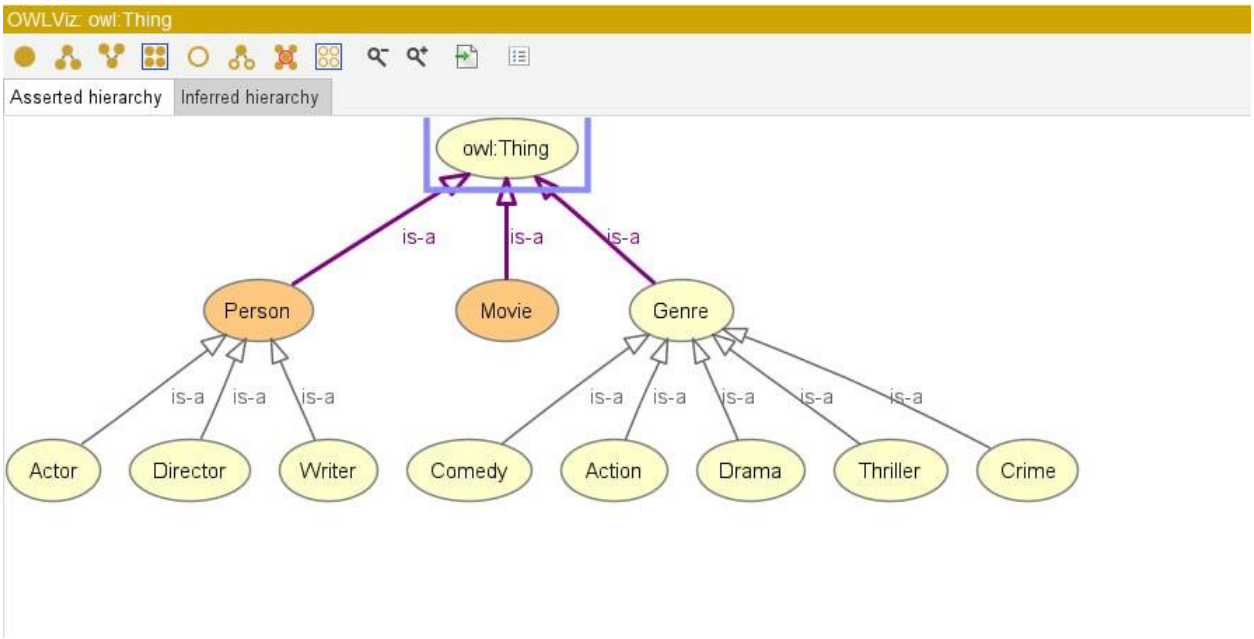


Figure 7 Class hierarchy using protege

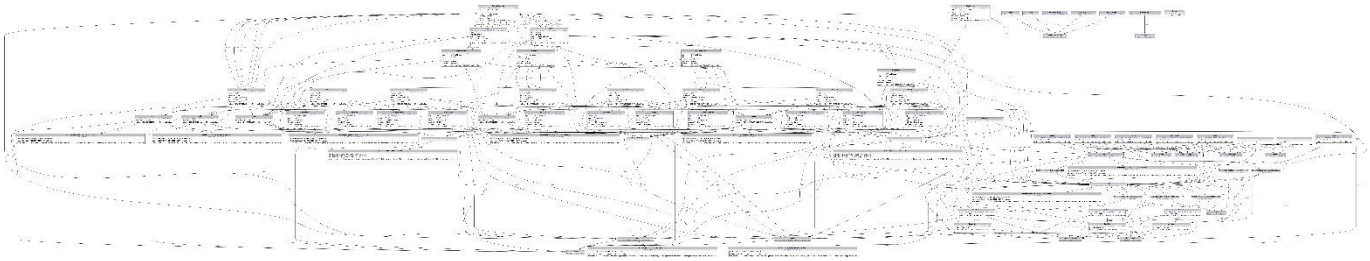
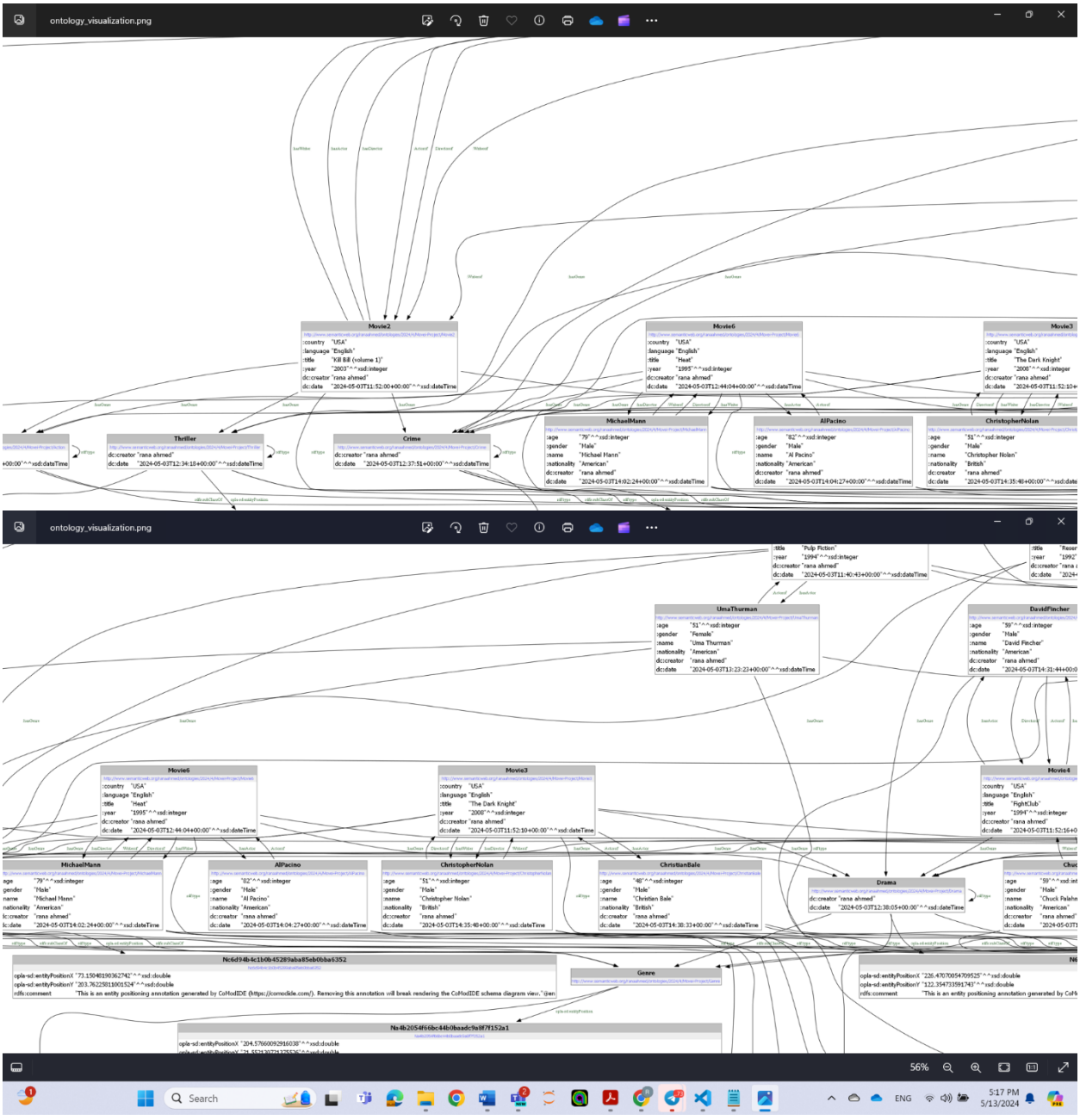


Figure 8 Ontology visualization



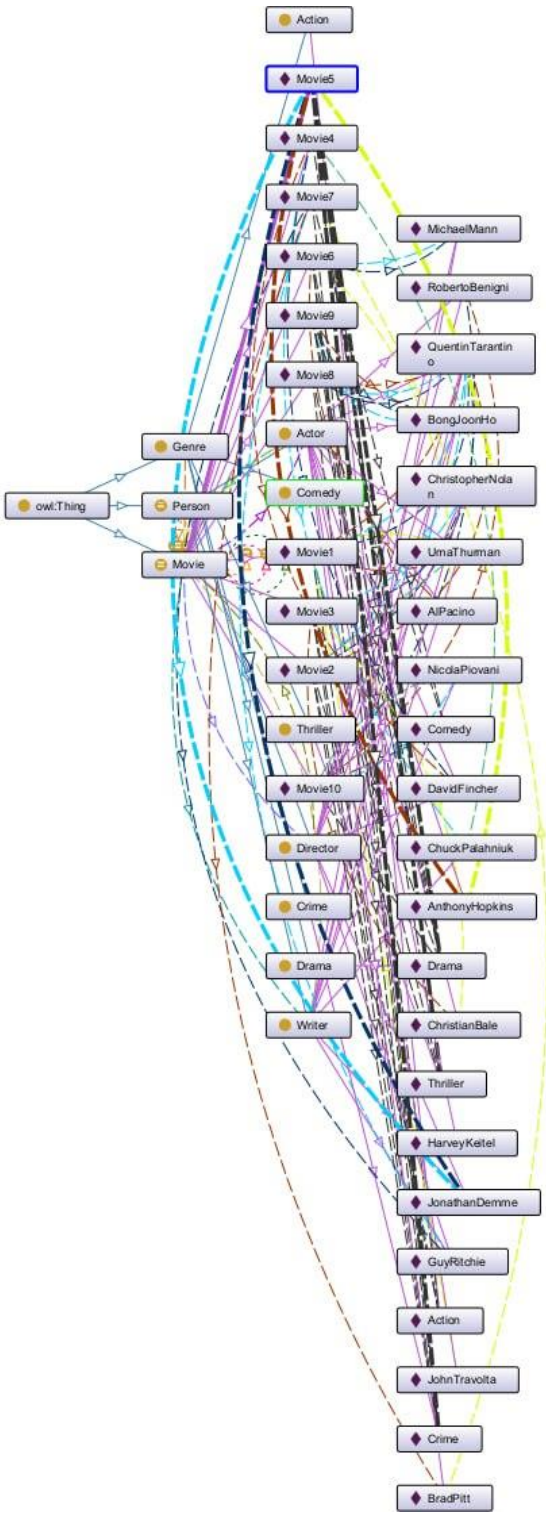


Figure 9 Visualize of ontology using ontograpgh

4 Snapshots of the Interface (GUI)

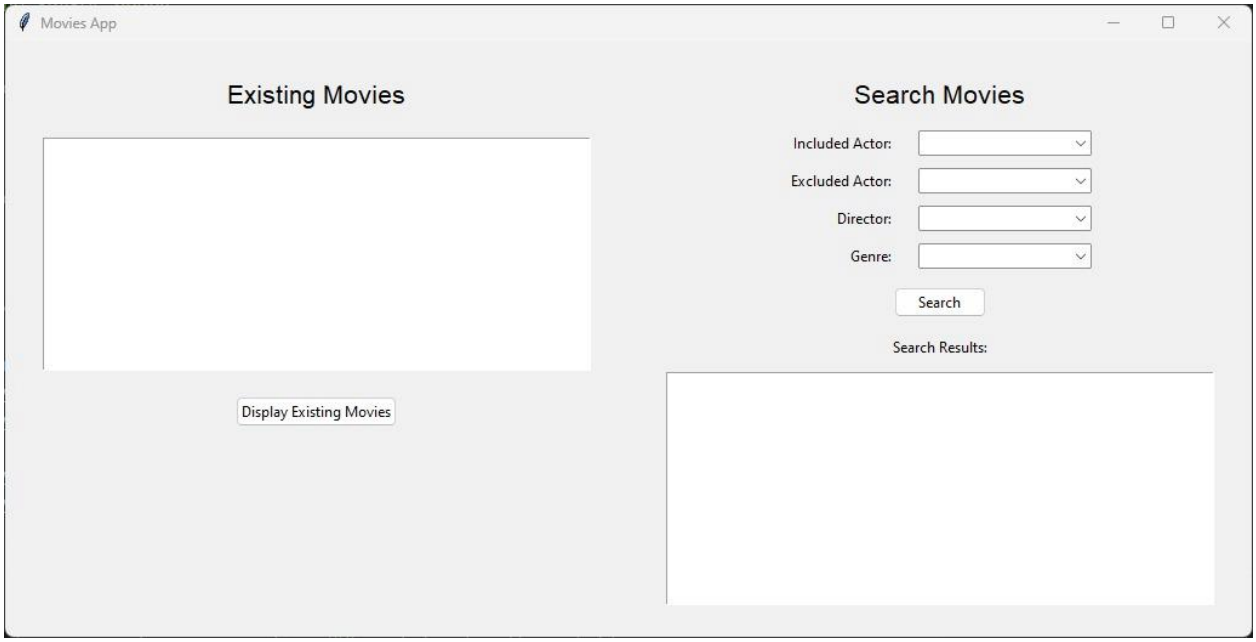


Figure 10 First Screen for GUI

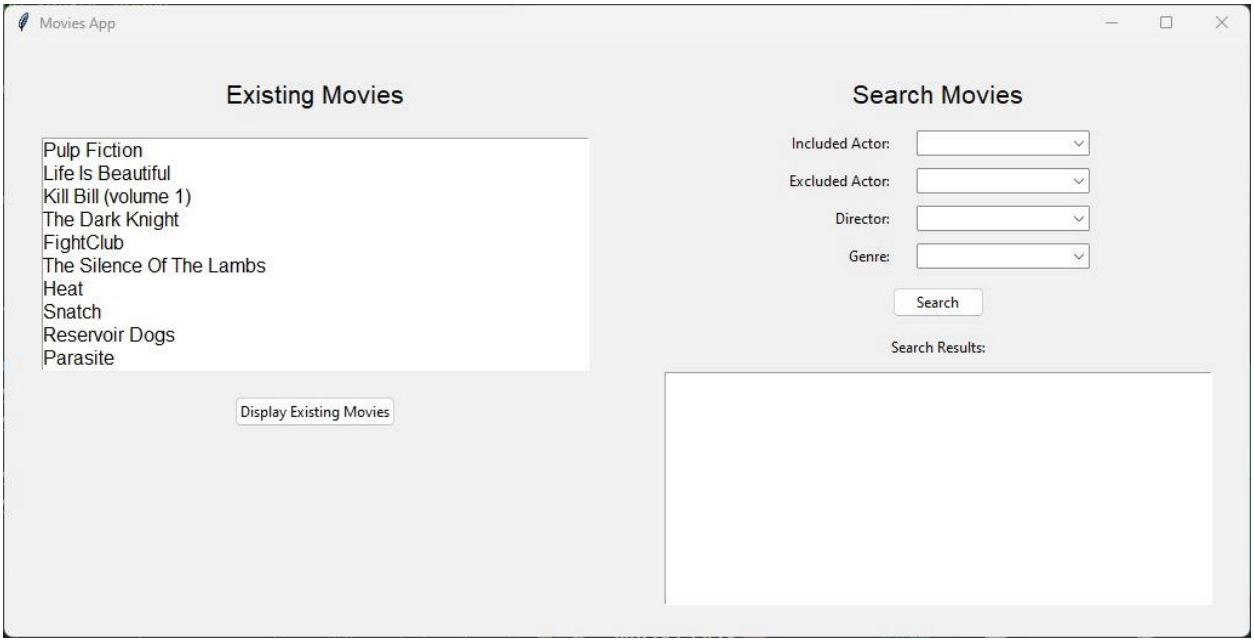


Figure 11 Pressing on Display Existing Movies Button

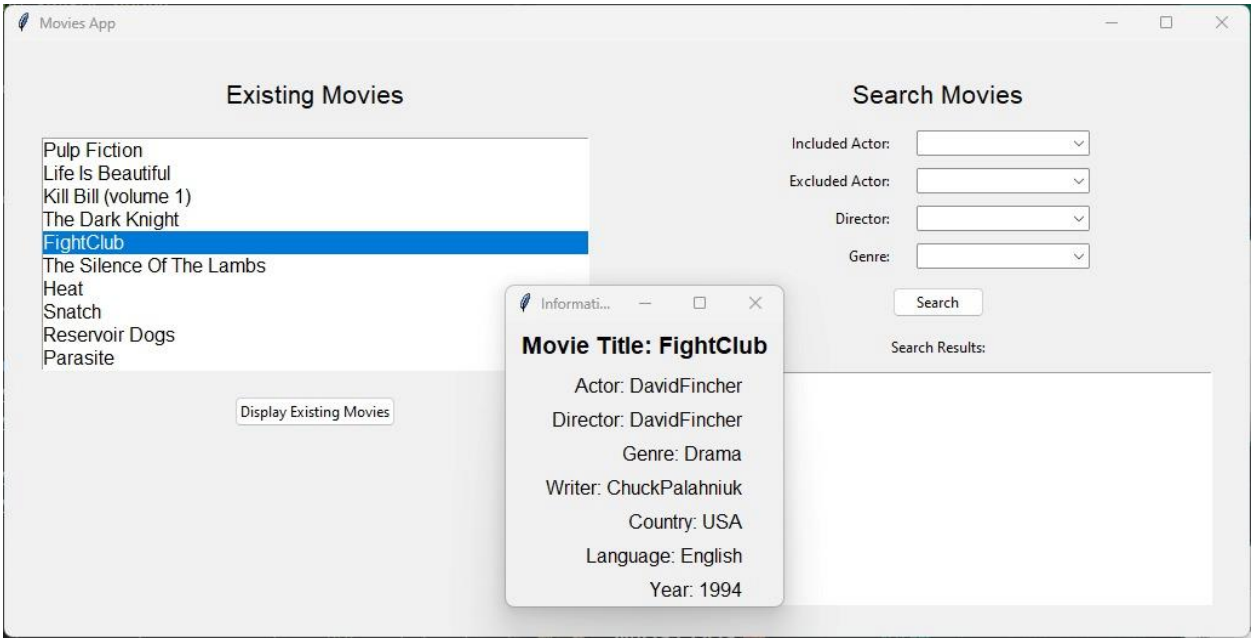


Figure 12 Selecting Movie to show its details

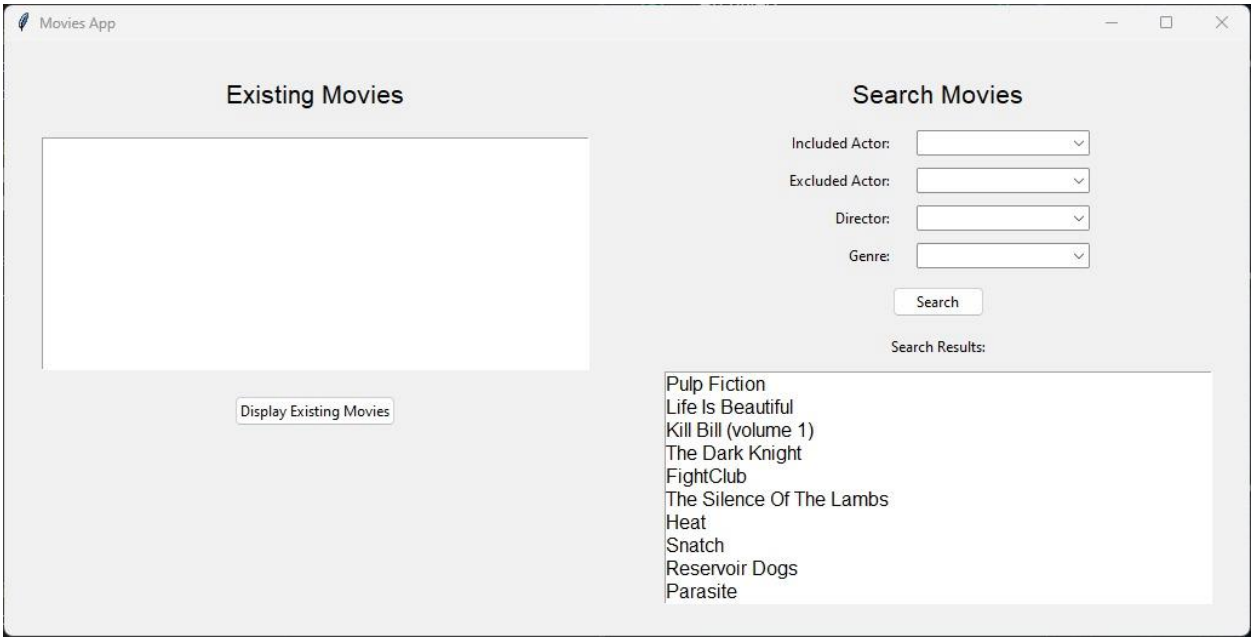


Figure 13 Pressing Search button without selecting any restrictions

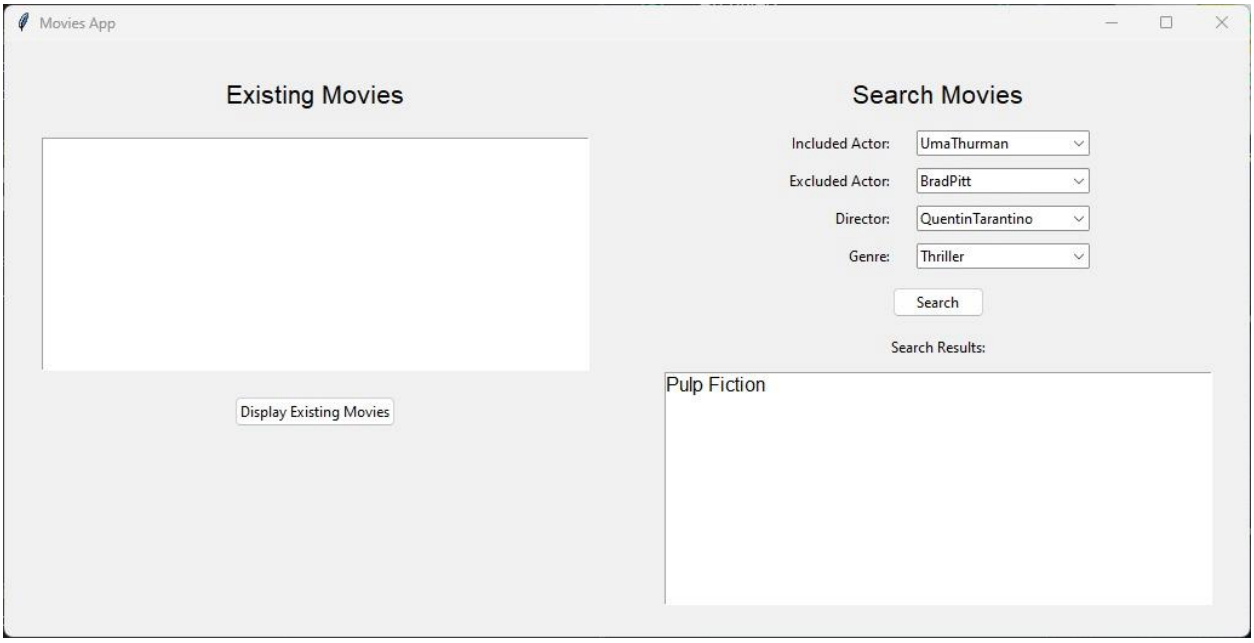


Figure 14 Search with including/ excluding/ Director and genre.

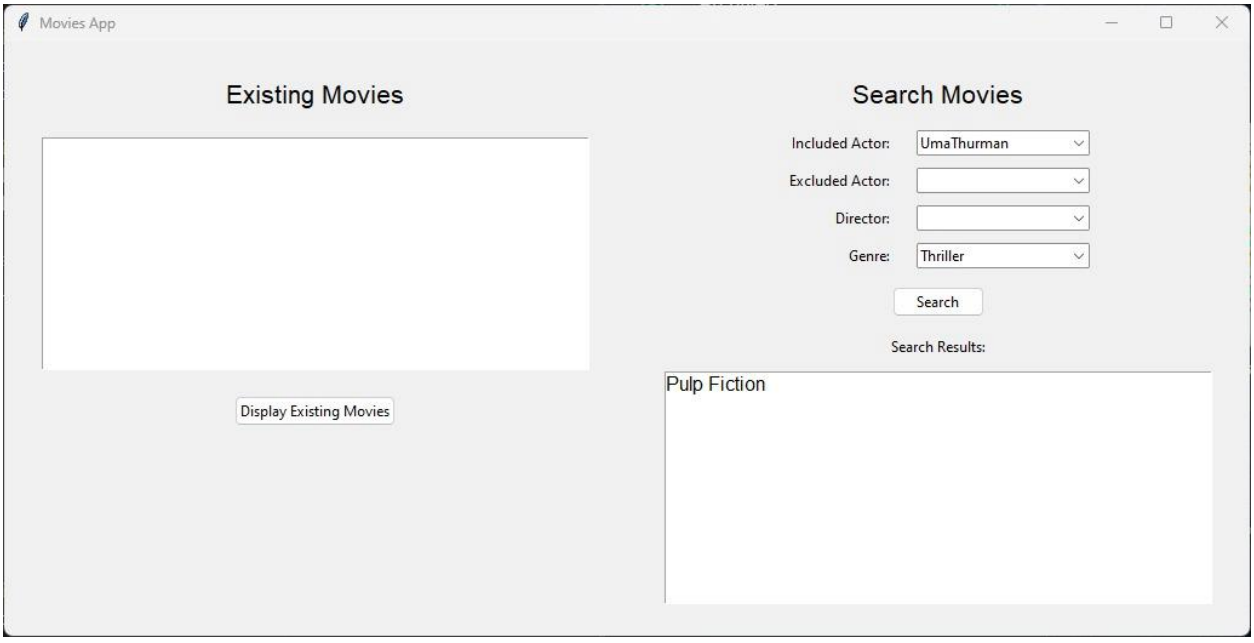


Figure 15 Searching with including actor and genre.

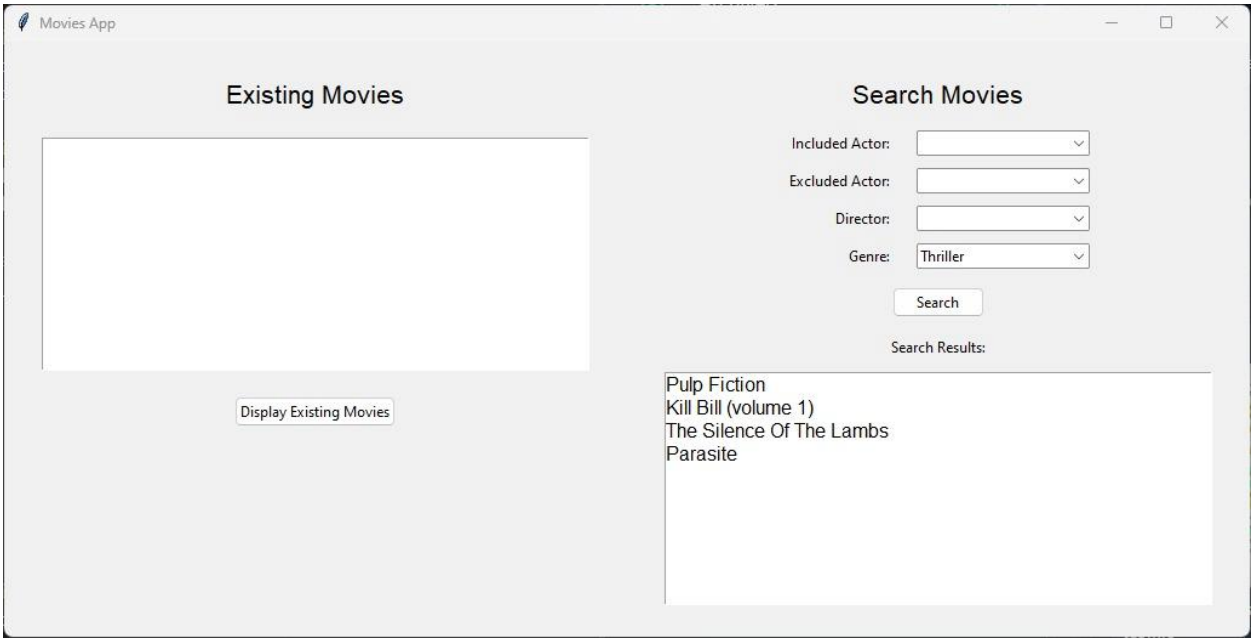


Figure 16 Searching for Specific Genre

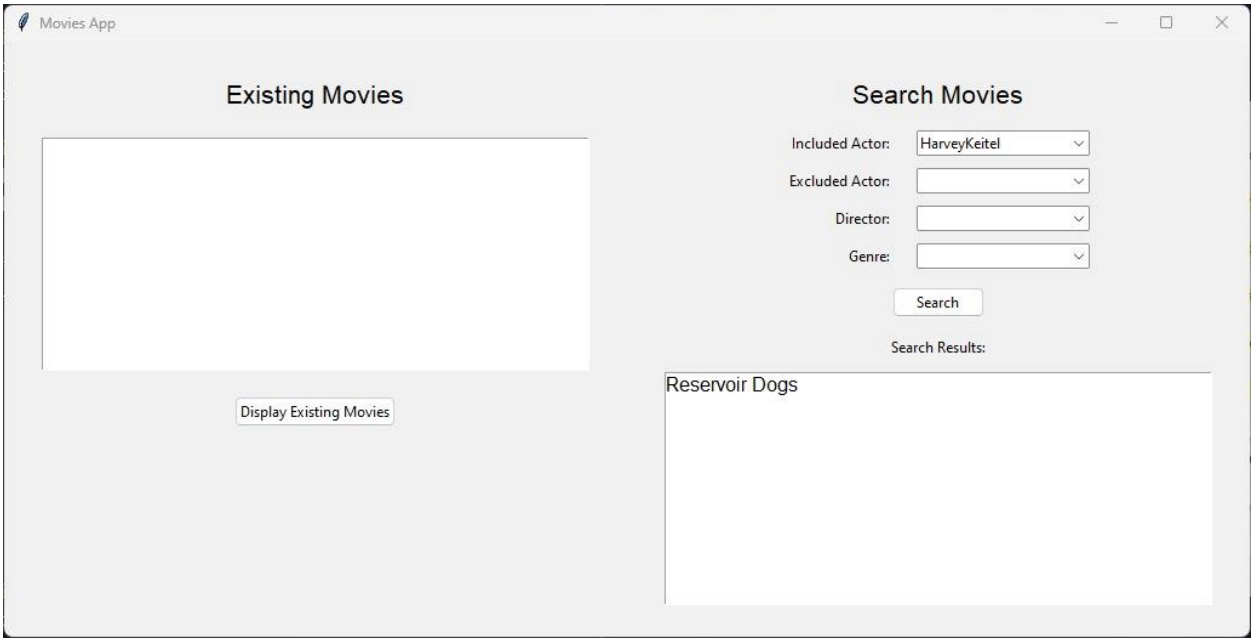


Figure 17 Searching by including actor

5 Test Cases

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

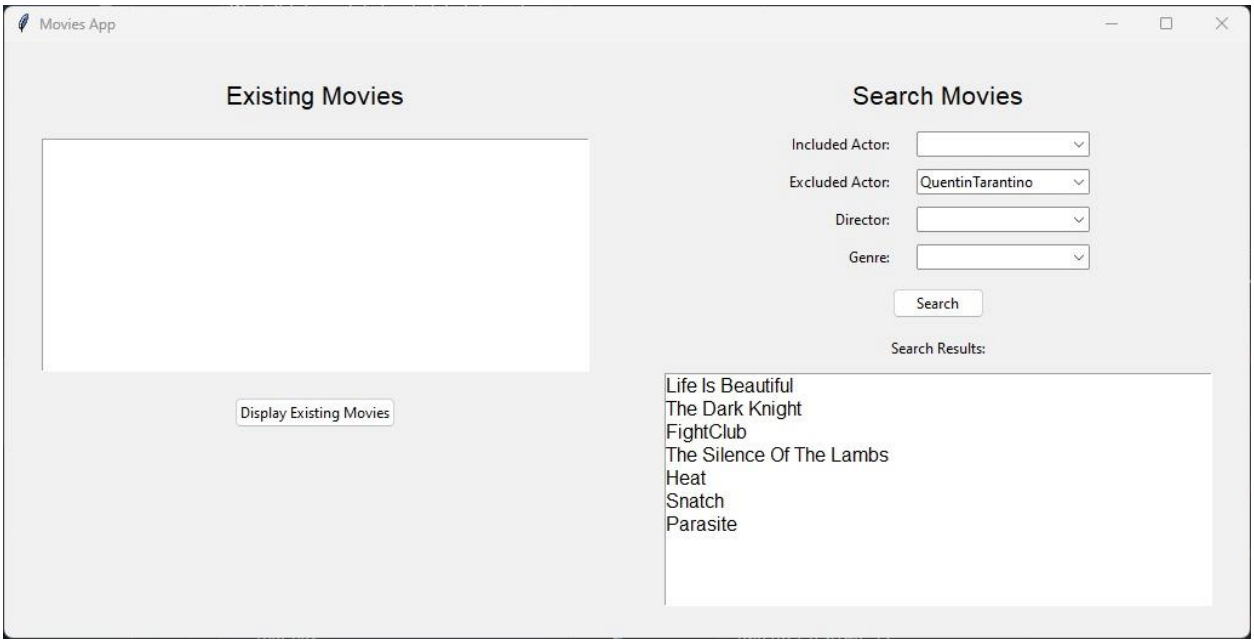


Figure 18 Answer for the first test case

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

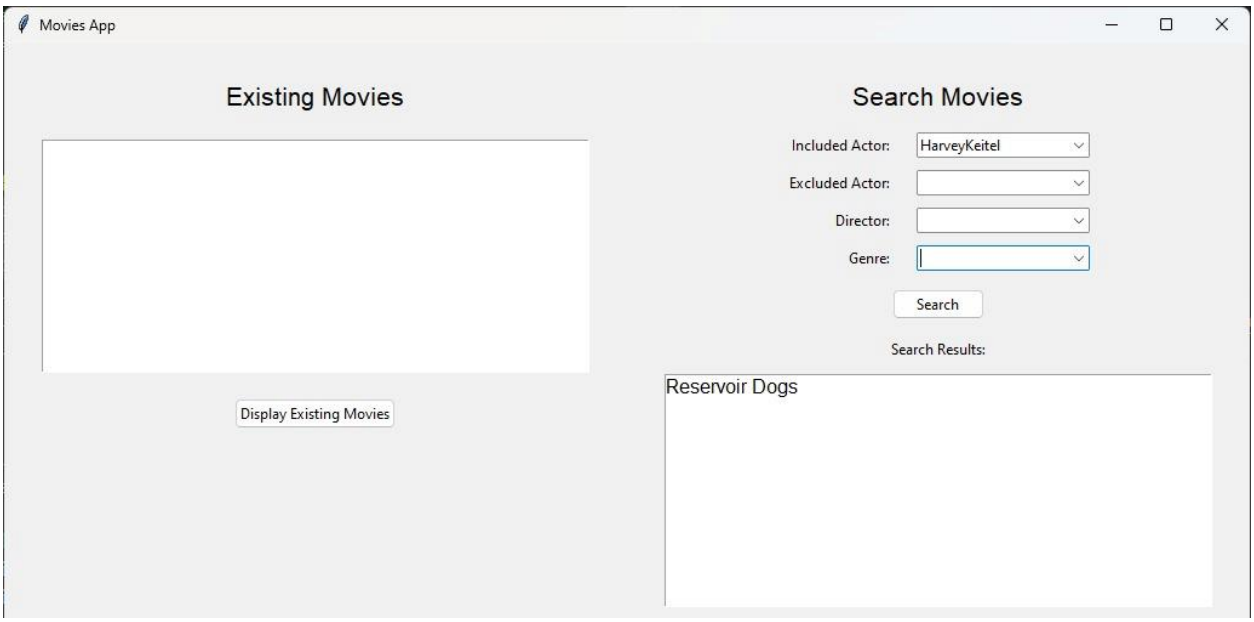


Figure 19 Answer for the second test case

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

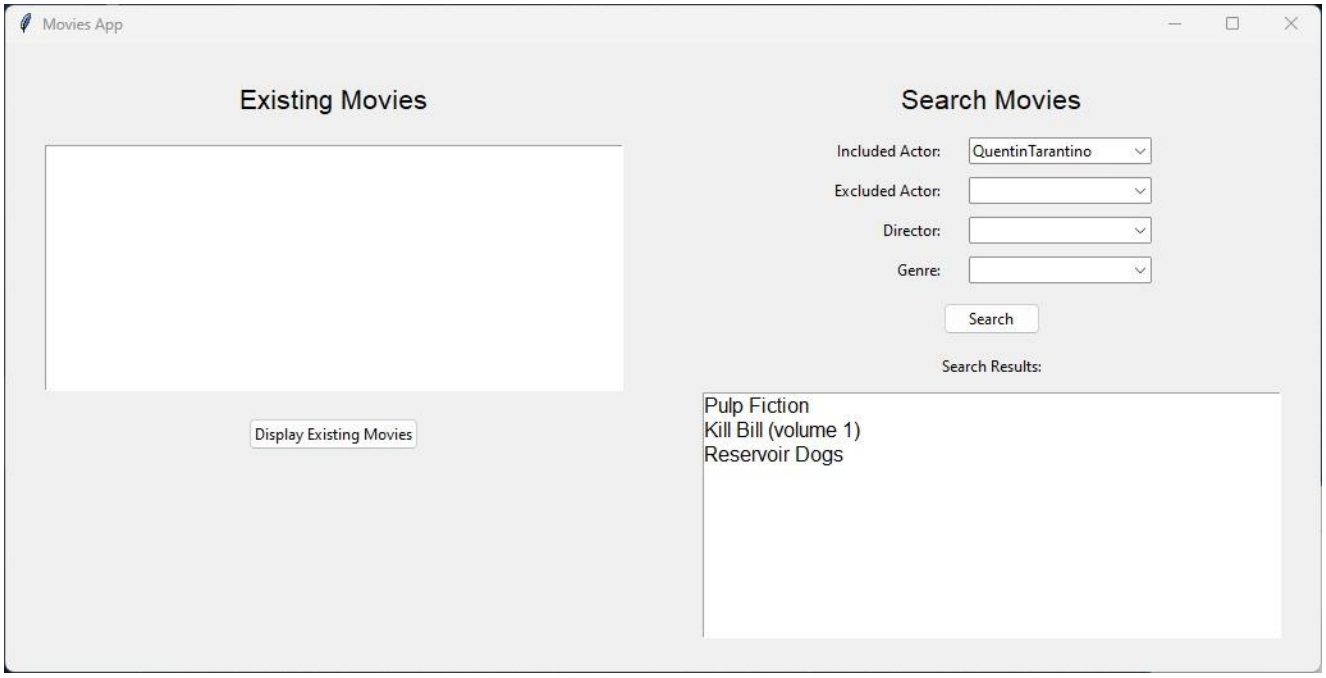


Figure 20 Answer for third test case.

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

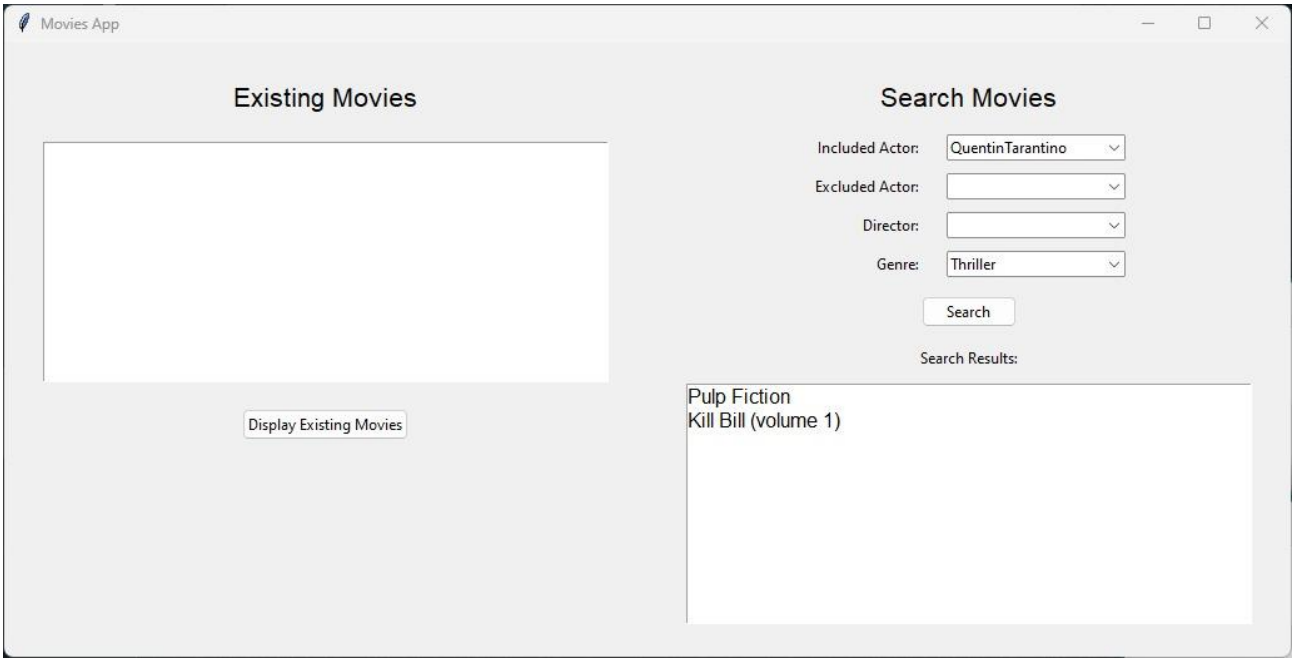


Figure 21 Answer for fourth test case.

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

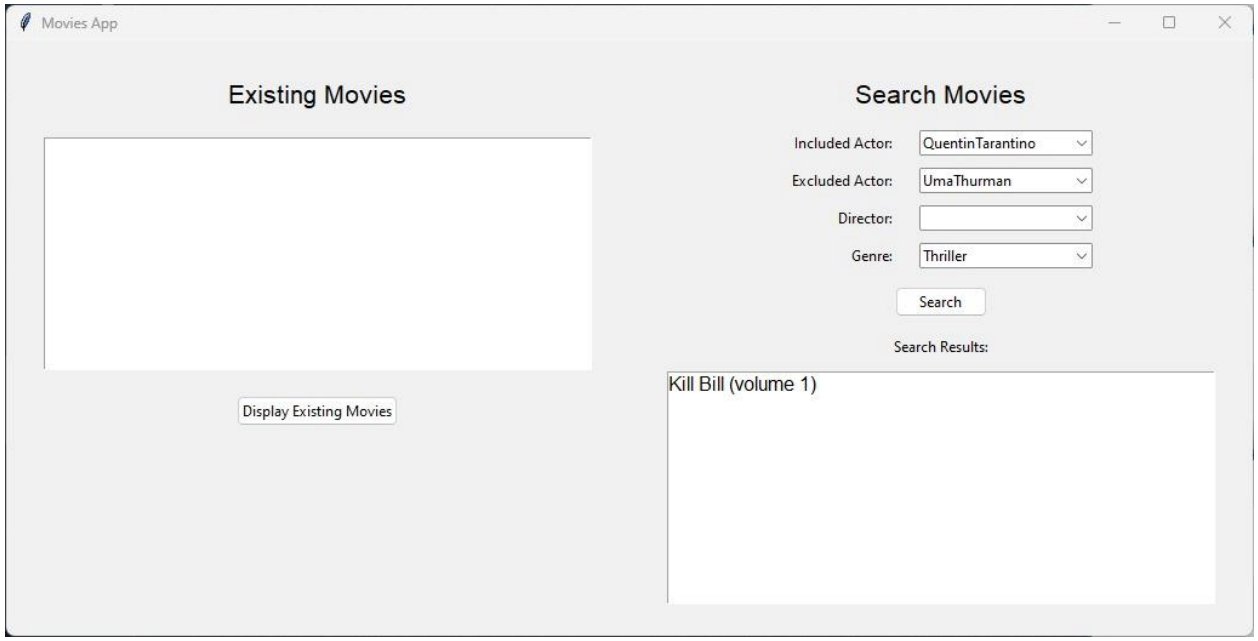


Figure 22 Answer for fifth test case.

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

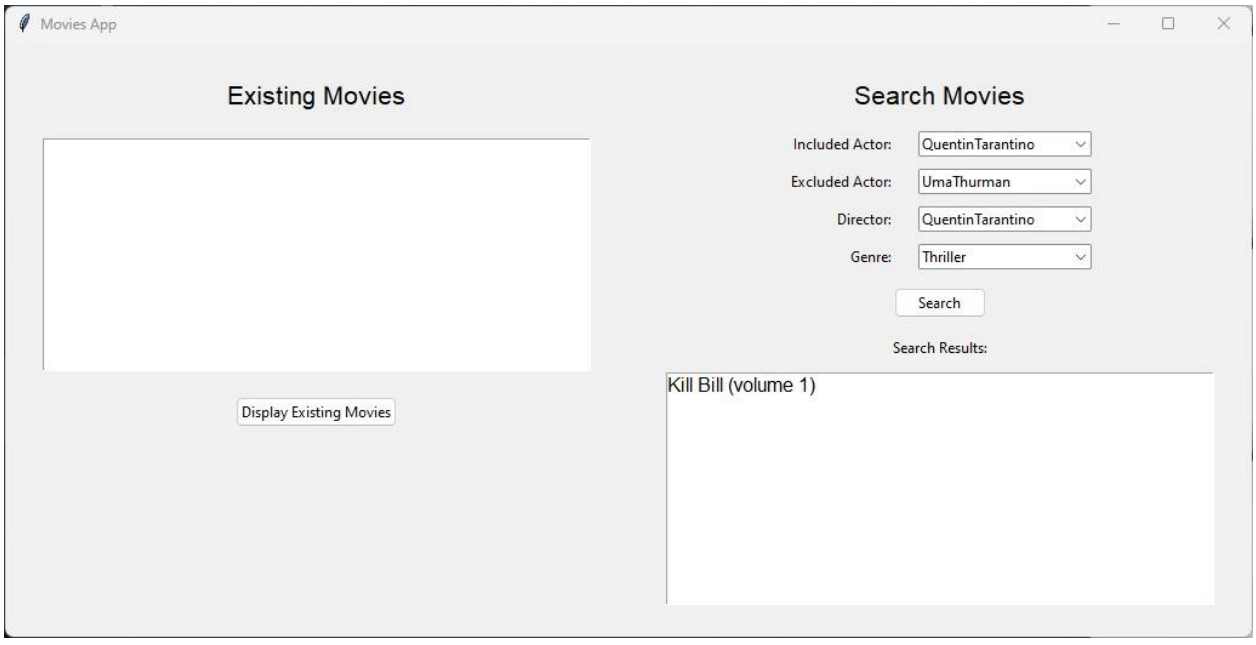


Figure 23 Answer for sixth test case.

6 Data Flow Diagram (DFD)

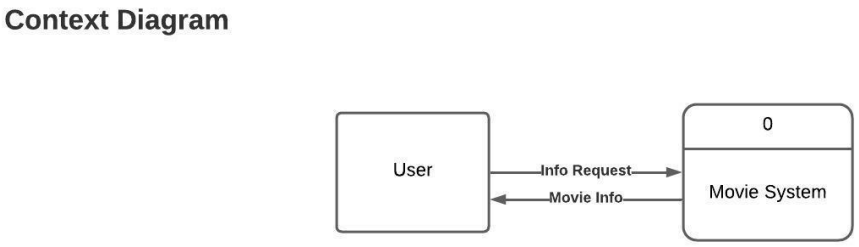


Figure 24 Context Diagram of DFD

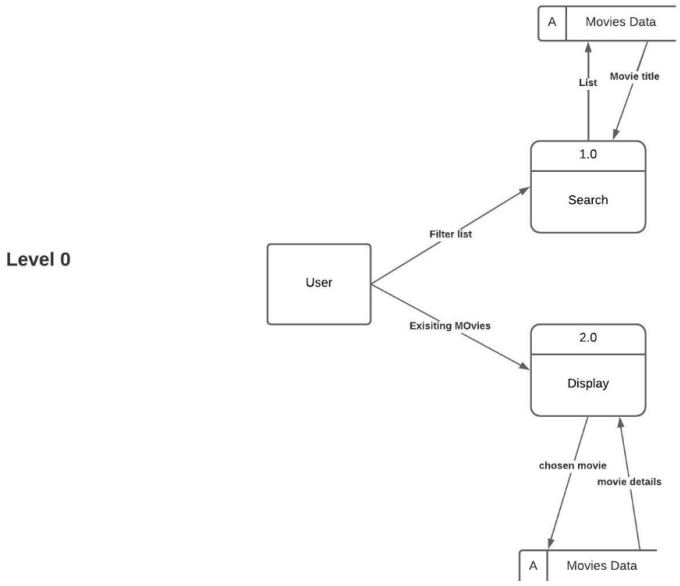


Figure 25 Level 0 of DFD

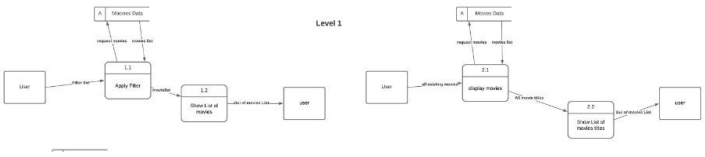


Figure 26 level 1 DFD