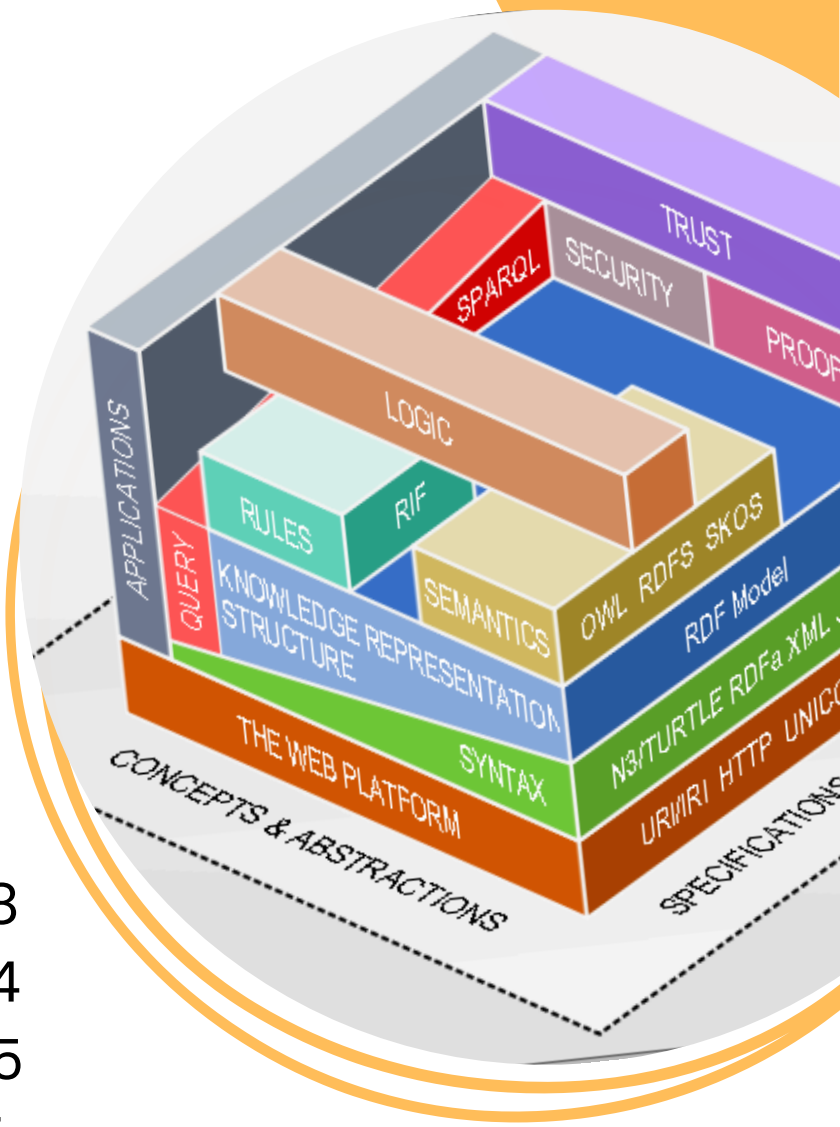
Ain Shams University

CSE488:

ONTOLOGIES AND THE SEMANTIC WEB



Presented to:

Dr. Ensaf Hussein

Eng. Eman Khaled

Prepared by:

Khaled Hassan	20p9978
Omar Ali Sobhy	20p2084
Habiba Yasser	20p9595
Laila Ihab	20p1005
Adham Yasser	20p9705

Table of Contents

Github link:

https://github.com/adham137/Ontology_application

Semantic Web Project

Problem Description:

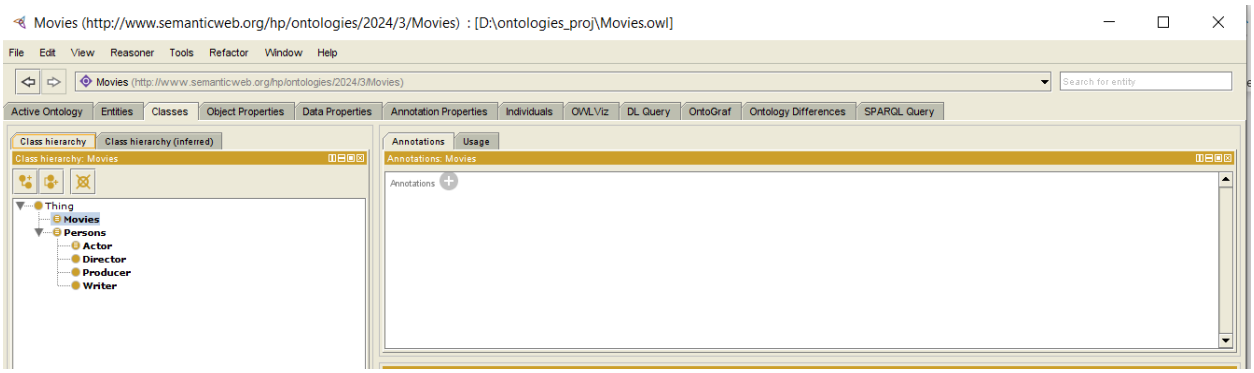
The problem domain that this ontology addresses revolves around the organization and representation of movie-related information. In the vast landscape of cinema, there exists a multitude of interconnected entities such as movies, directors, writers, actors, genres, and various attributes associated with them like titles, years, countries, and languages. Understanding and structuring this domain is crucial for various applications such as movie databases, recommendation systems, and analytical tools aimed at understanding trends in the film industry.

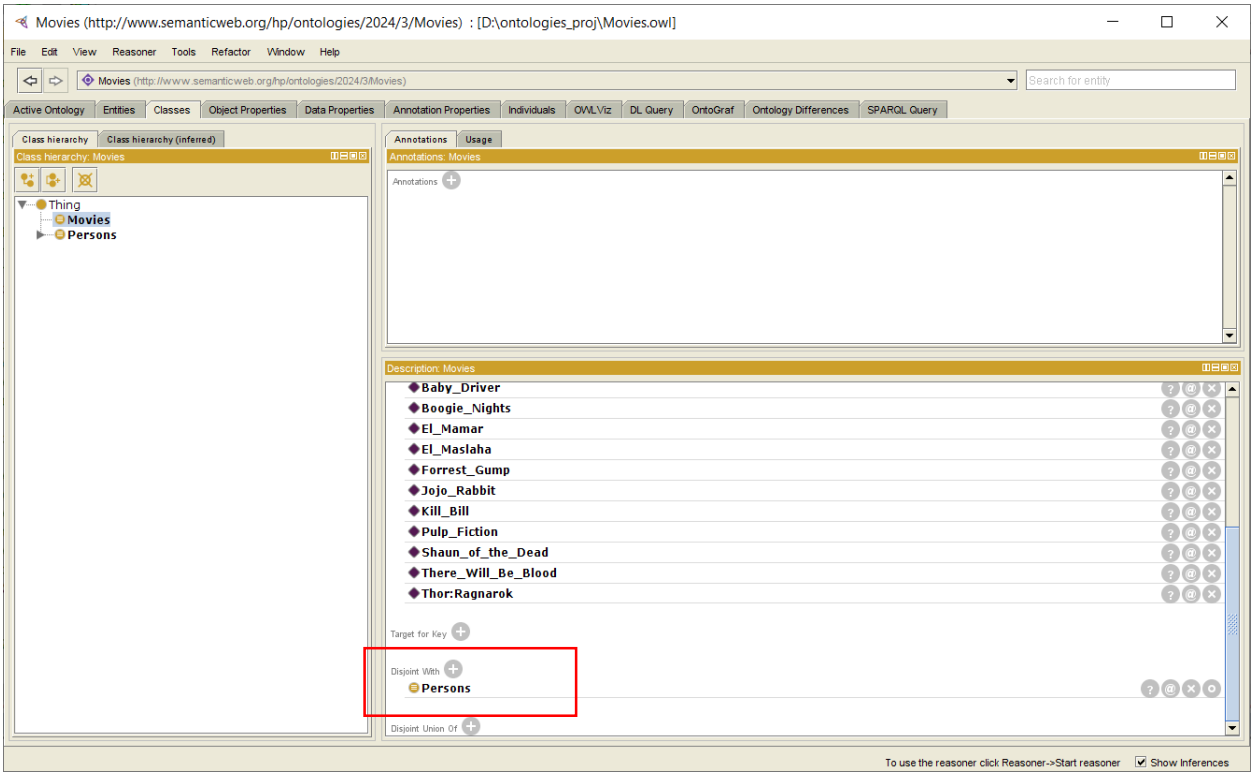
At its core, the problem entails creating a structured representation of movies and their associated entities to facilitate efficient querying, reasoning, and analysis. This involves defining the relationships between movies and their directors, writers, actors, genres, as well as capturing pertinent attributes of these entities such as names, genders, ages, and nationalities. Additionally, the ontology needs to encapsulate the taxonomy of movie genres to enable classification and categorization of movies based on their thematic content.

Finally, the problem domain encompasses considerations of data integrity, consistency, and interoperability. Ensuring that the ontology adheres to standard data modeling practices and is compatible with existing ontologies or data schemas within the domain of cinema is essential for promoting interoperability and facilitating seamless integration with other systems or datasets.

Number of Entities:

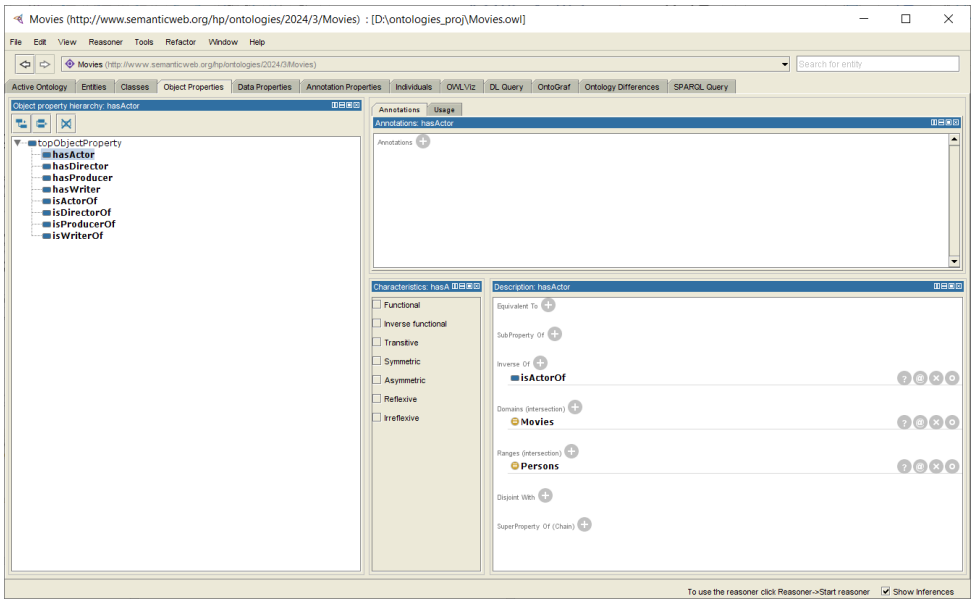
- Thing class is the superclass of any classes made in the ontology.
- 2 classes are made:
    a. Movies Class
    b. Persons Class: Class that represents persons that could possibly be working in any movie, including actors, directors, writers and producers.
    c. Movies and Persons are connected to each other via various object properties.
    d. Movies and Persons are disjoint with each other, meaning that they don't have any common individual or instance.



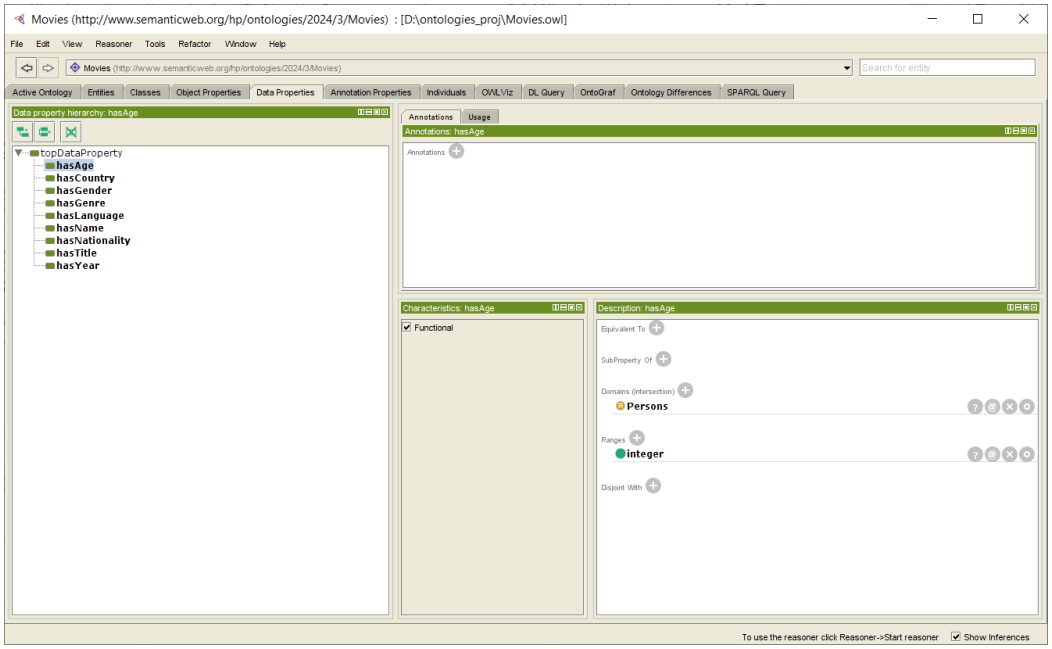


Number of Relations:

- The ontology consists of 8 Object Properties:
    a. has Actor: which has **Domain:** Movies and **Range**: Persons
    b. hasDirector: which has **Domain:** Movies and **Range**: Persons
    c. hasWriter: which has **Domain:** Movies and **Range**: Persons
    d. hasProducer: which has **Domain:** Movies and **Range**: Persons
    e. isActorOf: which has **Domain:** Actor and **Range**: Movies
    f. isDirectorOf: which has **Domain:** Director and **Range**: Movies
    g. isWriterOf: which has **Domain:** Writer and **Range**: Movies
    h. isProducerOf: which has **Domain:** Producer and **Range**: Movies



- The ontology also consists of 9 Data Properties:
    a. hasAge: which has **Domain:** Persons and **Range**: xsd:integer
    b. hasGender: which has **Domain:** Persons and **Range**: xsd:string
    c. hasName: which has **Domain:** Persons and **Range**: xsd:string
    d. hasNationality: which has **Domain:** Persons and **Range**: xsd:string
    e. hasTitle: which has **Domain:** Movies and **Range**: xsd:string
    f. hasGenre: which has **Domain:** Movies and **Range**: xsd:string
    g. hasYear: which has **Domain:** Movies and **Range**: xsd:integer
    h. hasCountry: which has **Domain:** Movies and **Range**: xsd:string
    i. hasLanguage: which has **Domain:** Movies and **Range**: literal



Logic:

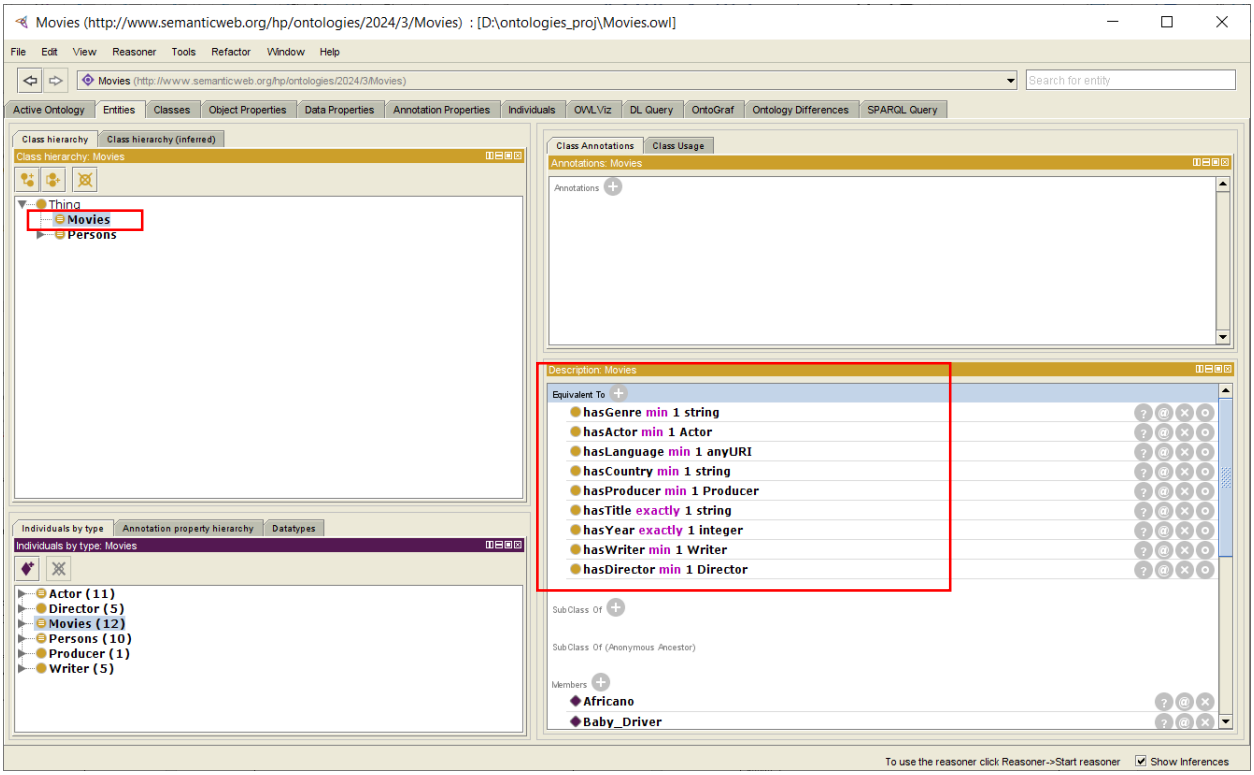
To define the ontology for representing movies and associated entities, we can employ Description Logic (DL), a formalism commonly used for ontology modeling. DL allows us to define classes, properties, and axioms to capture the relationships and constraints within the domain.

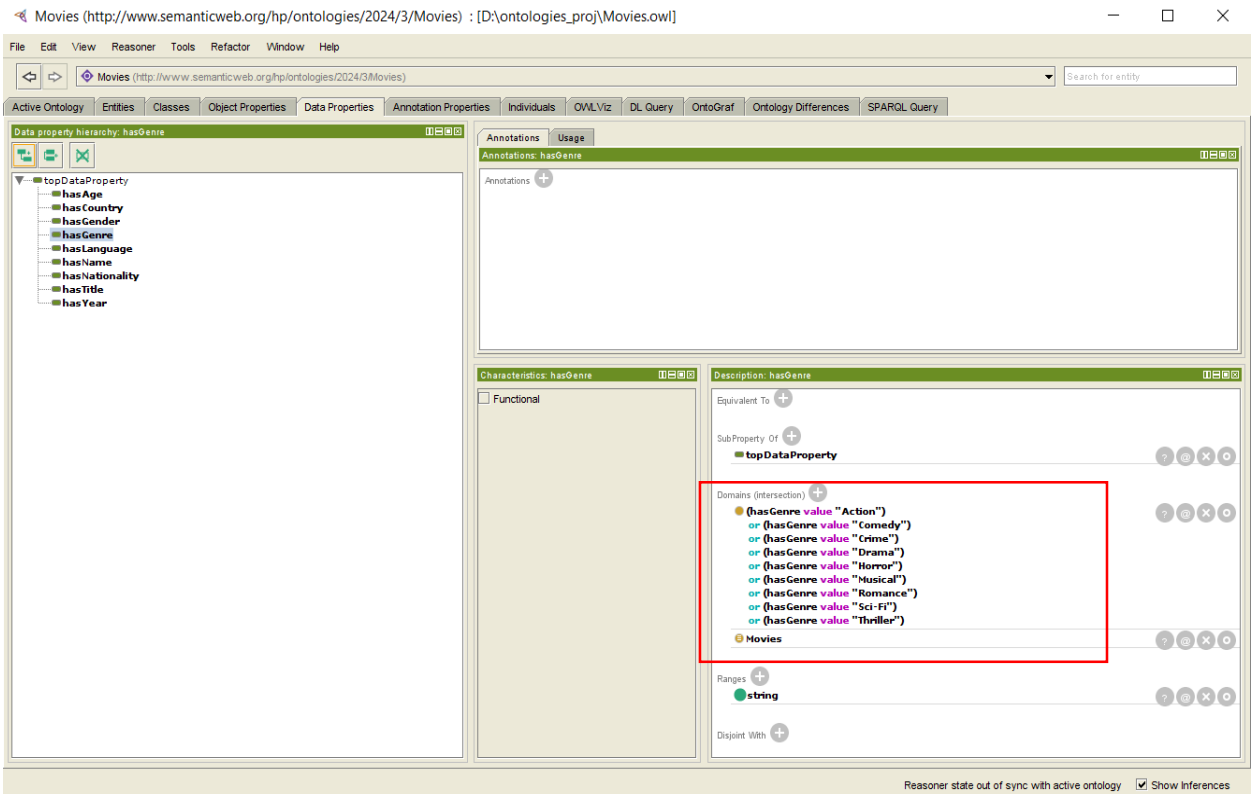
Axioms and Restrictions:

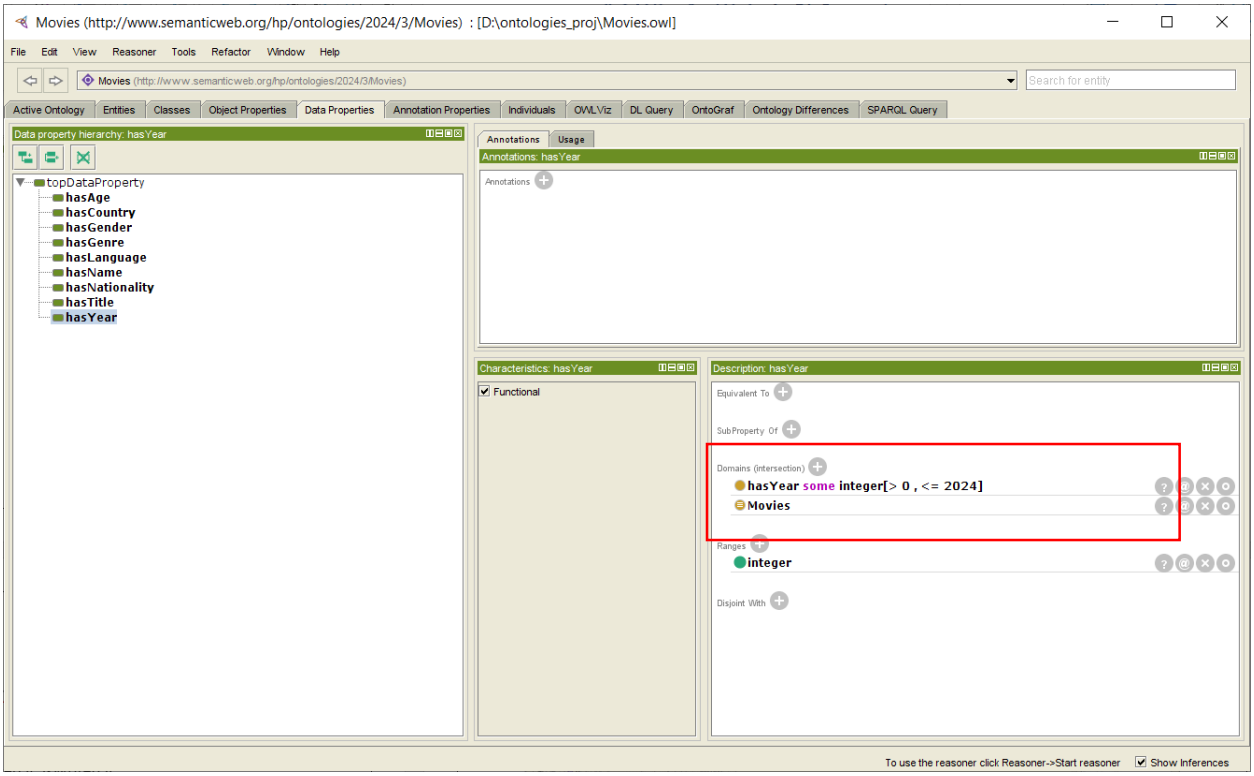
Define axioms to assert relationships and constraints between classes and properties. For example:

- Each Movie has one or several directors (hasDirector).
- Each Movie has one or several writers (hasWriter).
- Each Movie has one or several actors (hasActor).
- Each Movie has a title, a year, a country and a language.
- Each Movie belongs to one or several genres (hasGenre).
- To define the Genre of a movie, possible choices are: Thriller, Crime, Action, Drama or Comedy.

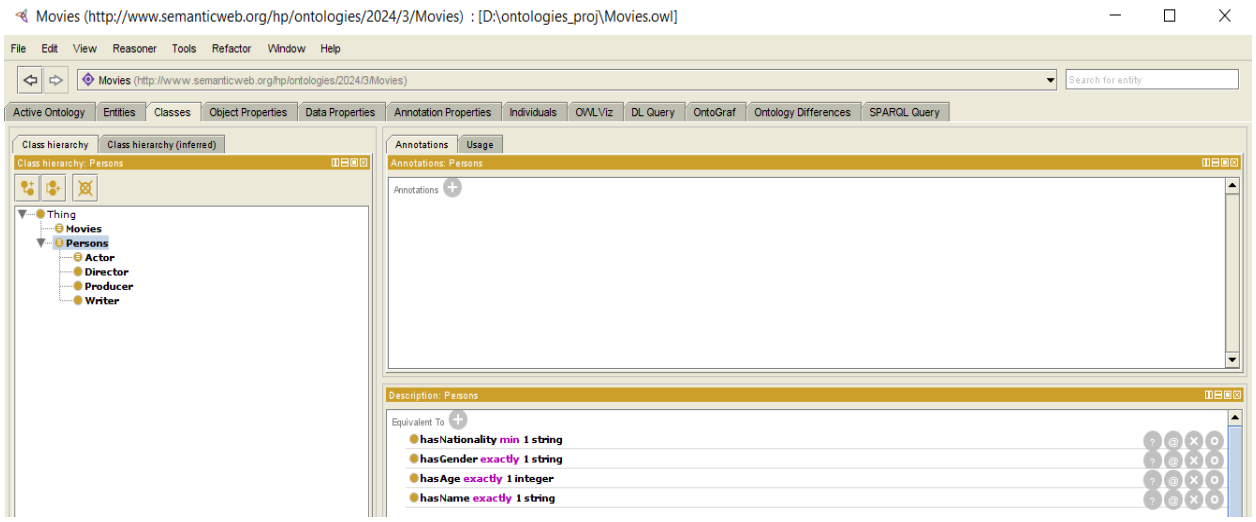
- Property hasYear is integer and from 0 to 2024

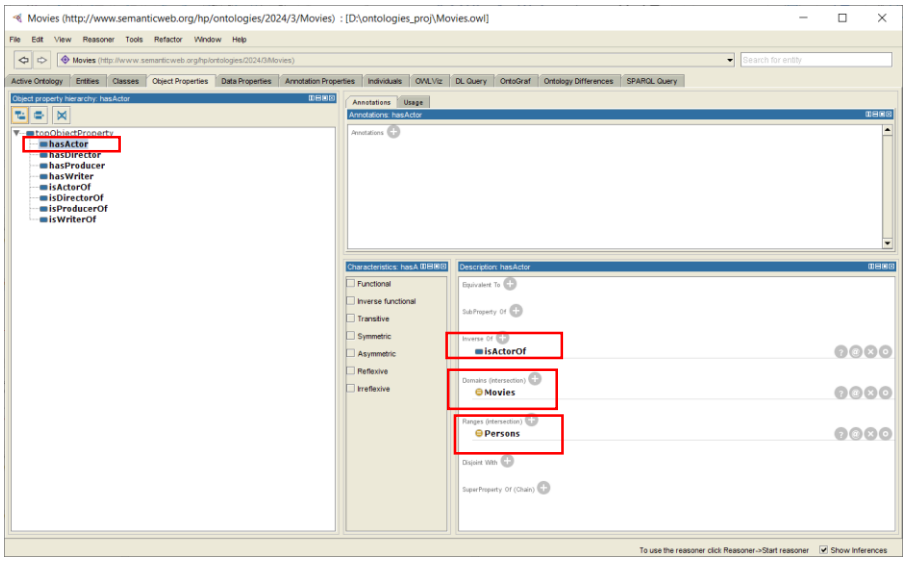


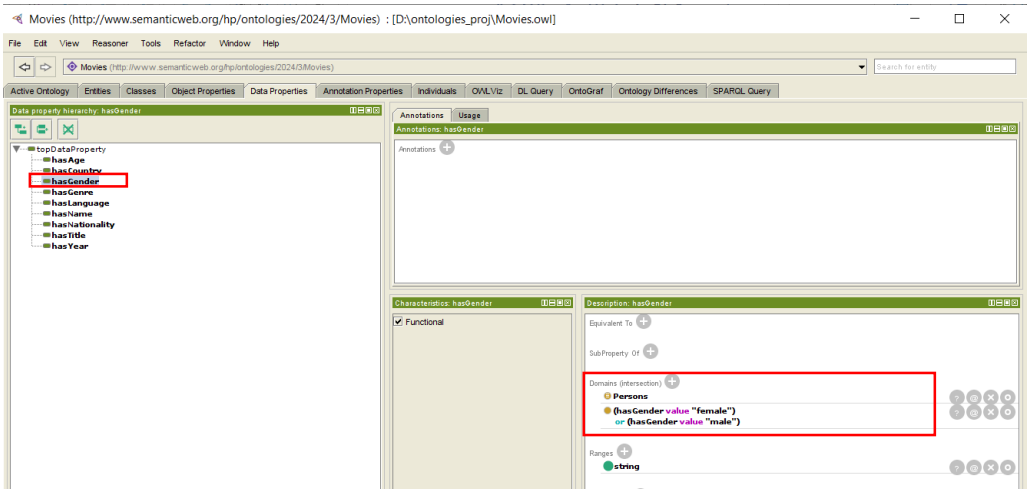


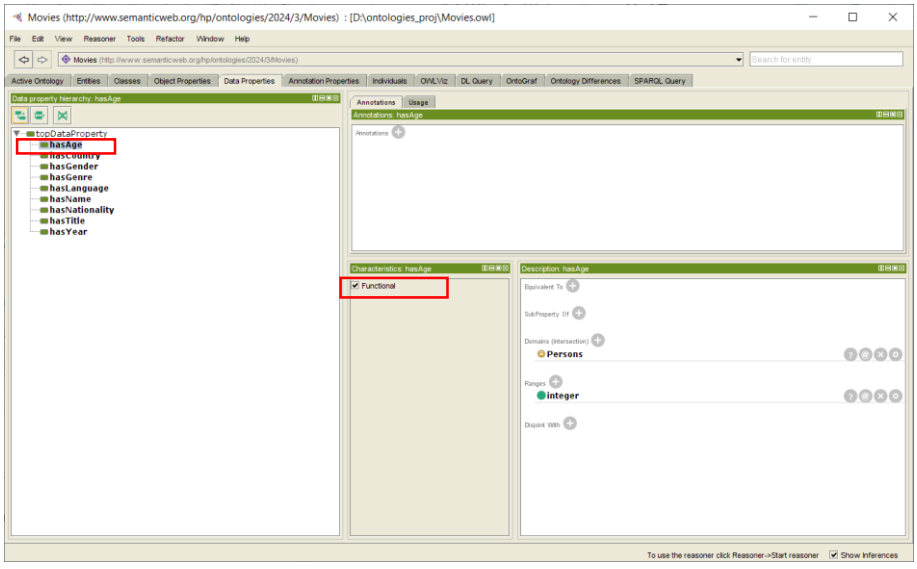


- Actors, directors and writers are persons.
- Each Person has a gender (male or female), name, age, and nationality.
- Defined hasAge, hasName, hasGender, hasTitle and hasYear as **Functional** data properties.
- Defined Movies Class as Disjoint with Persons class.
- Defined the domain and Range for all properties as illustrated above.
- Defined has Actor, hasDirector, hasWriter, hasProducer as inverseOf: isActorOf, isDirectorOf, isWriterOf, isProducerOf.




Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Object property hierarchy: hasActor
topObjectProperty
hasActor
hasDirector
hasProducer
hasWriter
isActorOf
isDirectorOf
isProducerOf
isWriterOf
Annotations Usage
Annotations: hasActor
Characteristics: hasA
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: hasActor
Equivalent To
SubProperty Of
Inverse Of
isActorOf
Domains (intersection)
Movies
Ranges (intersection)
Persons
Disjoint With
SuperProperty Of (Chain)
To use the reasoner click Reasoner->Start reasoner
Show Inferences

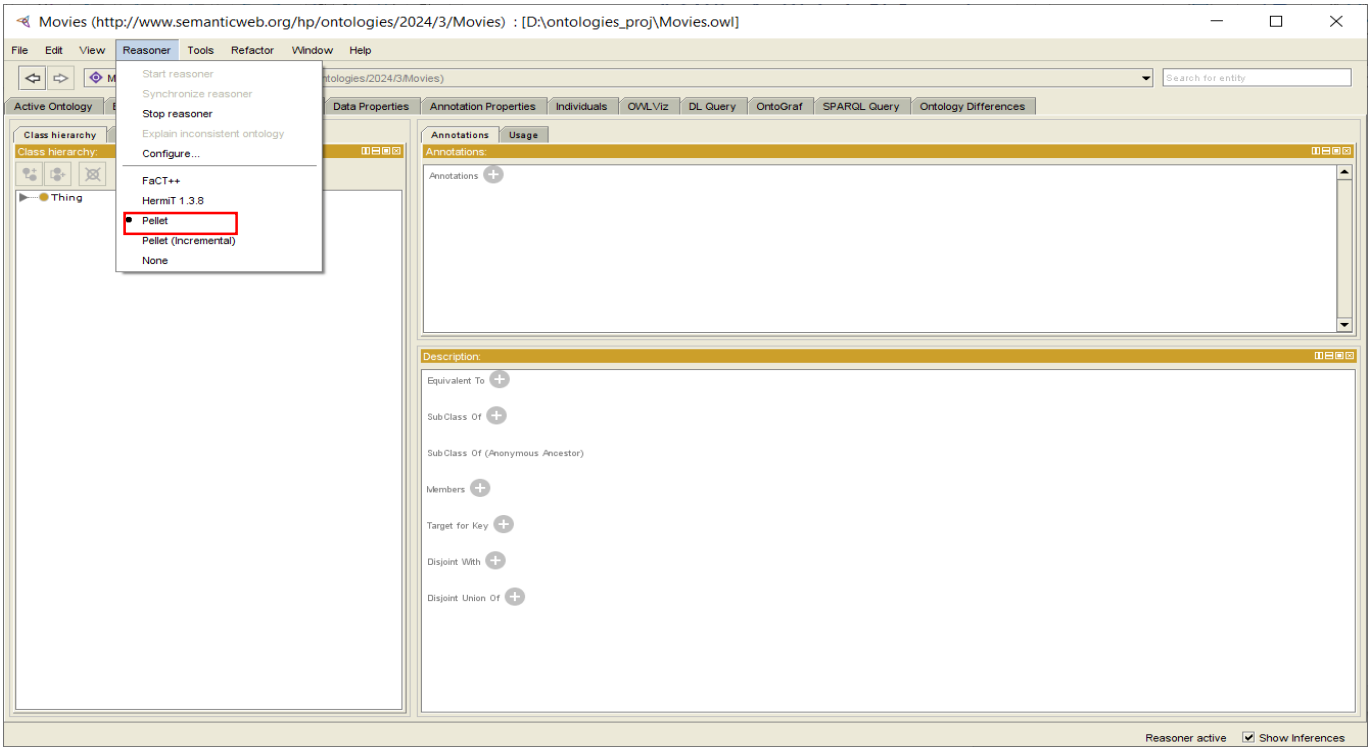

Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Data property hierarchy: hasGender
topDataProperty
hasAge
hasCountry
hasGender
hasGenre
hasLanguage
hasName
hasNationality
hasTitle
hasYear
Annotations Usage
Annotations: hasGender
Characteristics: hasGender
Functional
Description: hasGender
Equivalent To
SubProperty Of
Domains (intersection)
Persons
(hasGender value "female")
or (hasGender value "male")
Ranges
string


Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Data property hierarchy: hasAge
topDataProperty
hasAge
hasCountry
hasGender
hasGenre
hasLanguage
hasName
hasNationality
hasTitle
hasYear
Annotations Usage
Annotations: hasAge
Characteristics: hasAge
Functional
Description: hasAge
Equivalent To
SubProperty Of
Domains (intersection)
Persons
Ranges
integer
Disjoint With
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Use existential and universal quantifiers to specify constraints on relationships and attributes. For example (shown above):

- Existential quantifiers can be used to specify that each Movie must have at least one director, writer, actor and genre.
- Universal quantifiers can be used to specify that each Person must have a gender, name, age, and nationality.

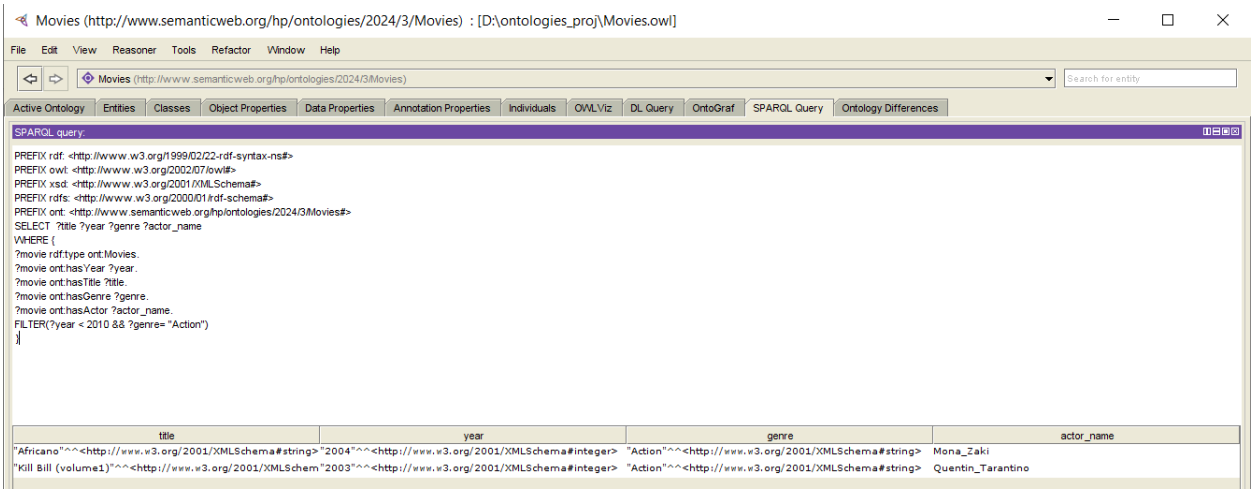
Check the consistency of the ontology:



Test Queries and Their Output:

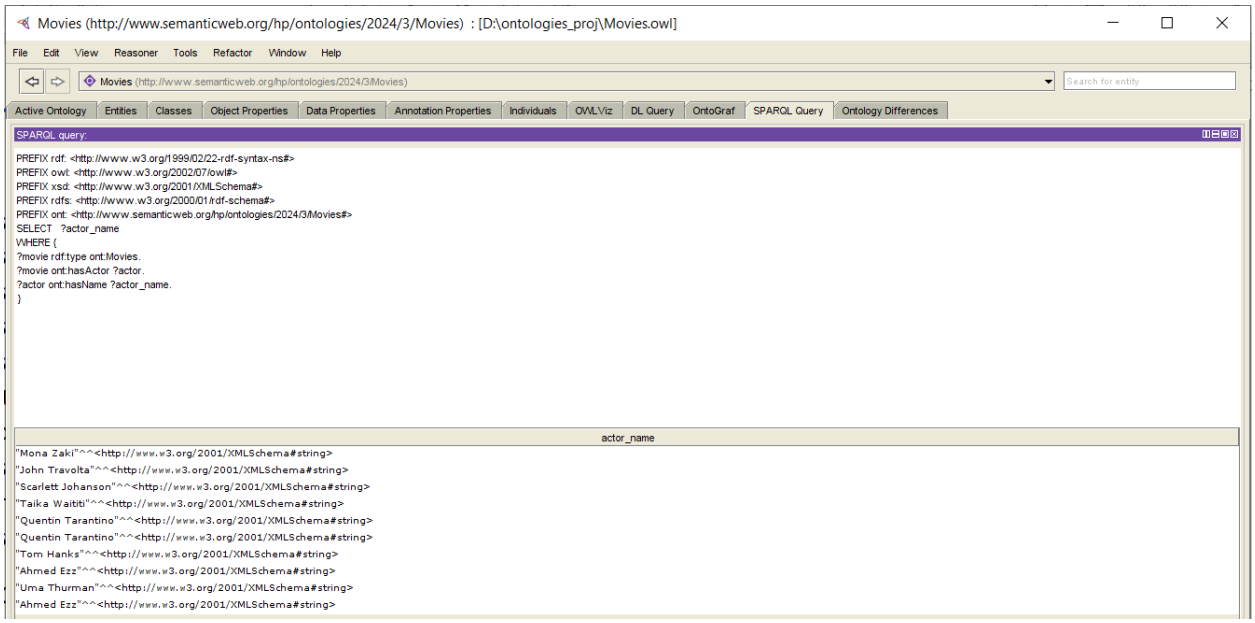
1) Find Movies that are before year 2010 and have genre "Action".

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT  ?title ?year ?genre ?actor_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasYear ?year.

?movie ont:hasTitle ?title.

?movie ont:hasGenre ?genre.

?movie ont:hasActor ?actor_name.

FILTER(?year < 2010 && ?genre= "Action")

}
```



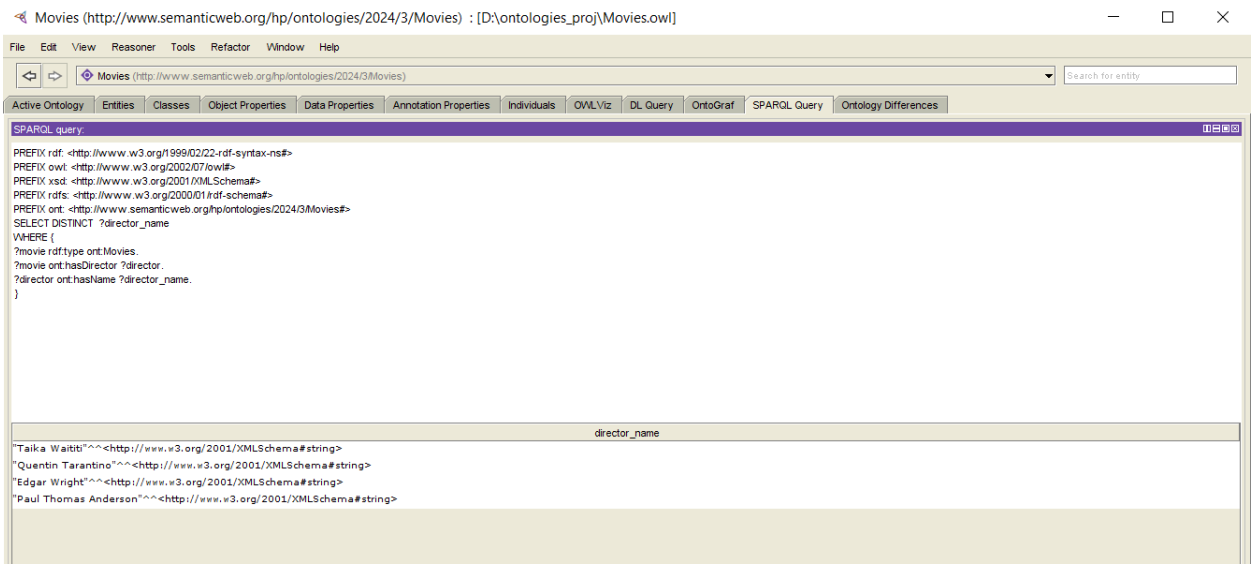
2) List the instances of the class Actor

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT   ?actor_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasActor ?actor.

?actor ont:hasName ?actor_name.

}
```



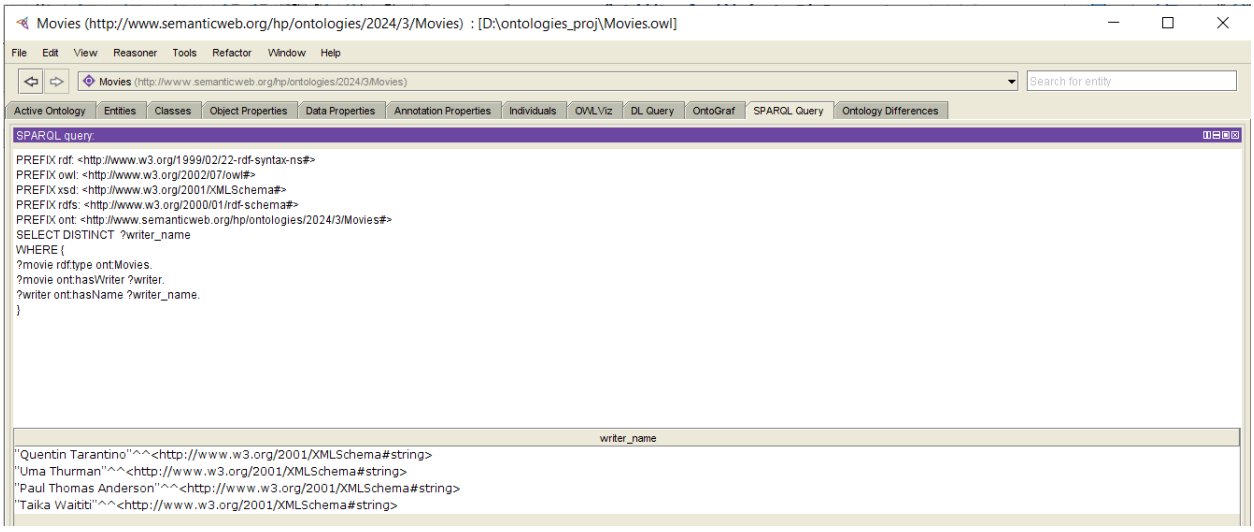
3) List the instances of the class director

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT DISTINCT  ?director_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasDirector ?director.

?director ont:hasName ?director_name.

}
```



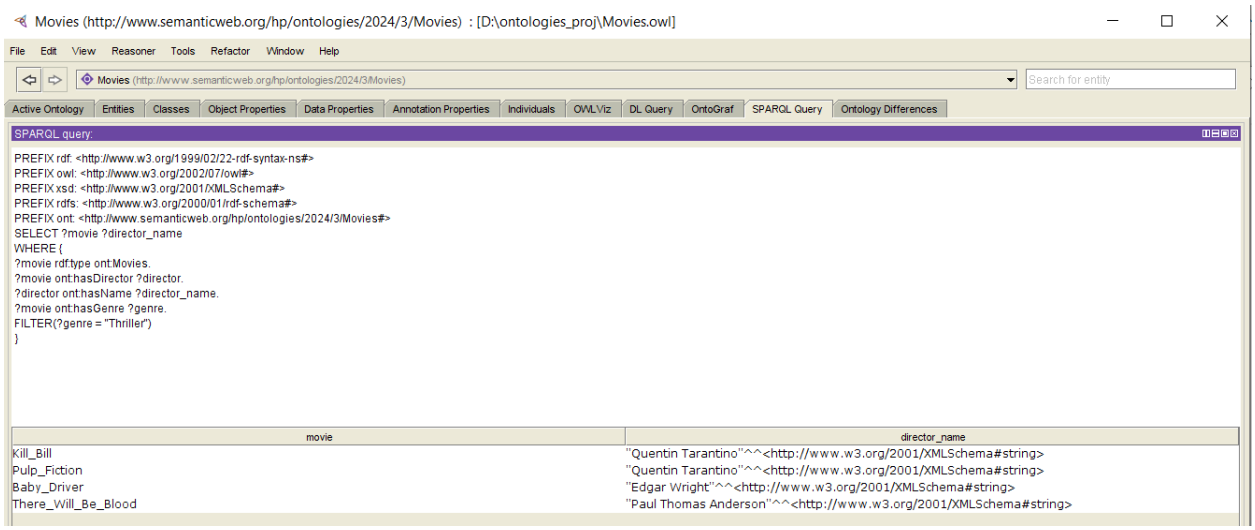
4) List the instances of the class writer

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT DISTINCT  ?writer_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasWriter ?writer.

?writer ont:hasName ?writer_name.

}
```



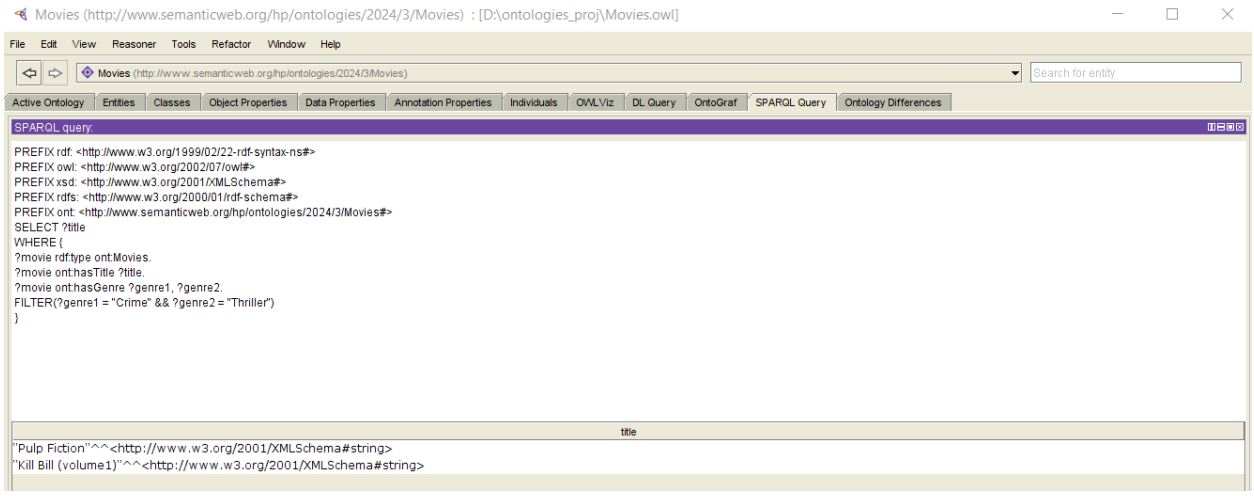
5) List the name of all Thriller movies. For each one, display its director.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?movie ?director_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasDirector ?director.

?director ont:hasName ?director_name.

?movie ont:hasGenre ?genre.

FILTER(?genre = "Thriller")

}
```



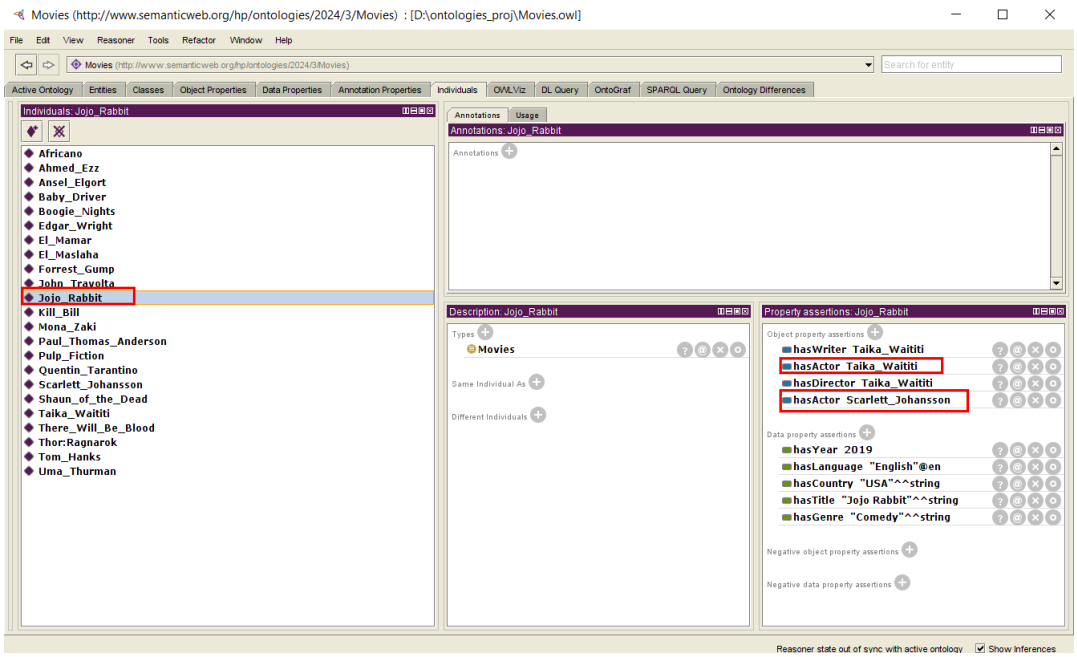
6) List the name of all Crime Thriller movies.

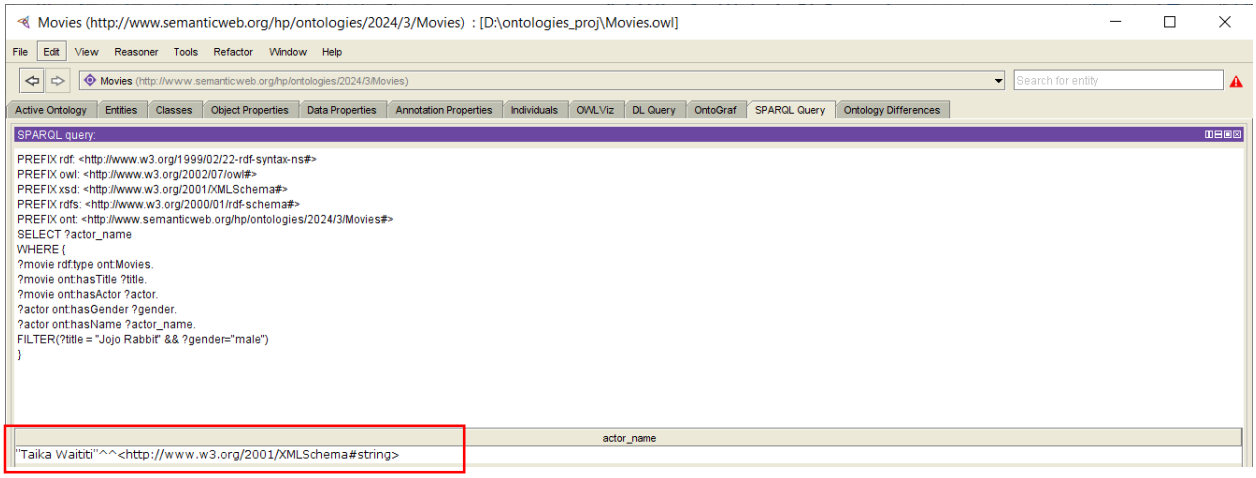
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?title

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasTitle ?title.

?movie ont:hasGenre ?genre1, ?genre2.

FILTER(?genre1 = "Crime" && ?genre2 = "Thriller")

}
```



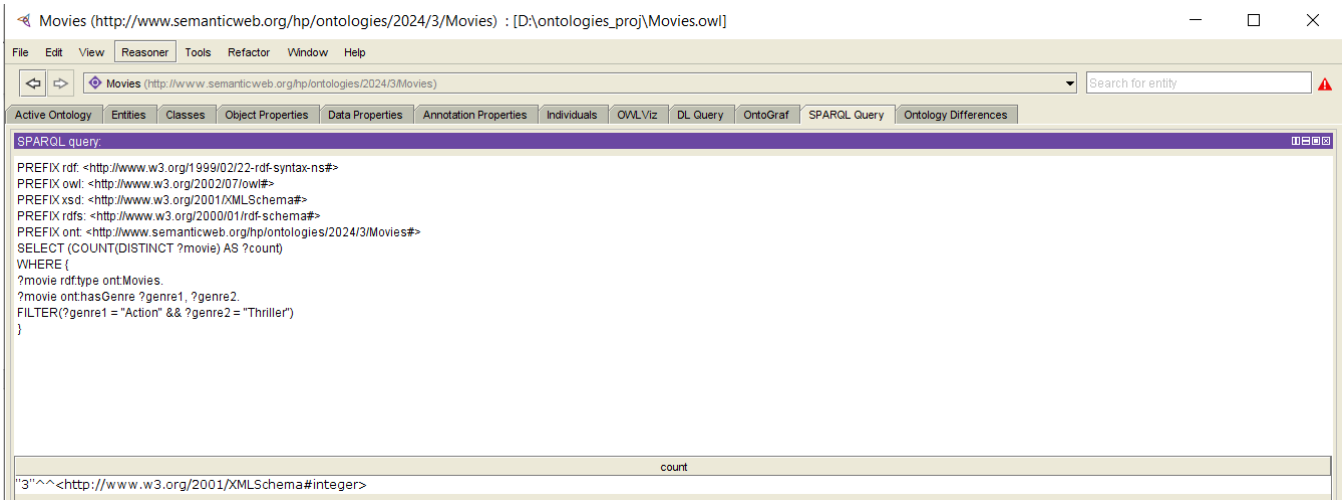
7) List the male actors in the movie in specific film

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?actor_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasTitle ?title.

?movie ont:hasActor ?actor.

?actor ont:hasGender ?gender.

?actor ont:hasName ?actor_name.

FILTER(?title = "Jojo Rabbit" && ?gender="male")

}
```



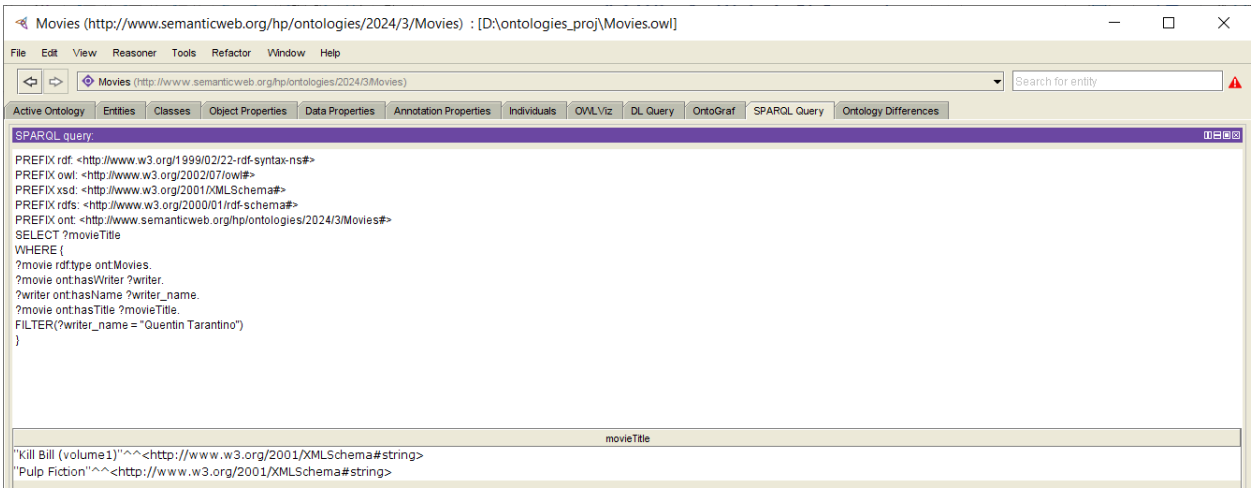


8) How many movies have both "Action" and "Thriller" as genres?



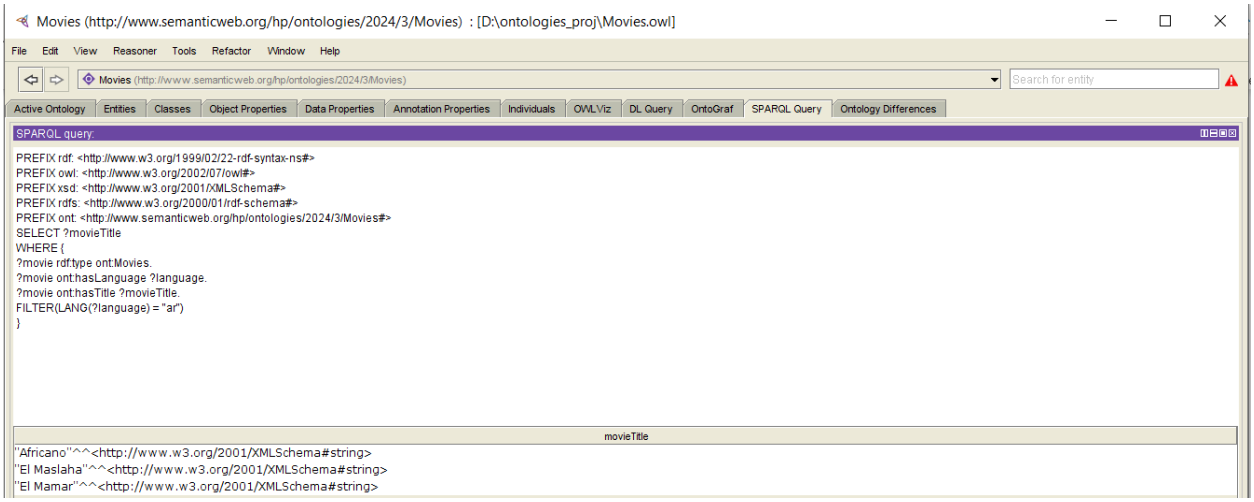
9) List all the movies written by a specific writer

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?movieTitle

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasWriter ?writer.

?writer ont:hasName ?writer_name.

?movie ont:hasTitle ?movieTitle.

FILTER(?writer_name = "Quentin Tarantino")

}
```

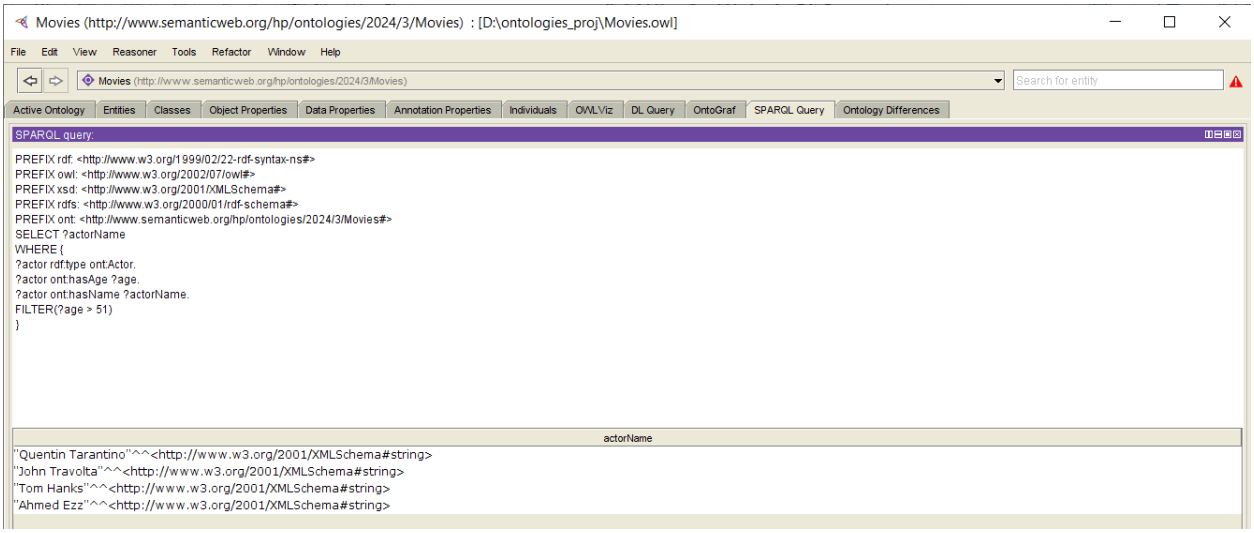


10) Find movies with a certain language.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT ?movieTitle

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasLanguage ?language.

?movie ont:hasTitle ?movieTitle.

FILTER(LANG(?language) = "ar")

}
```



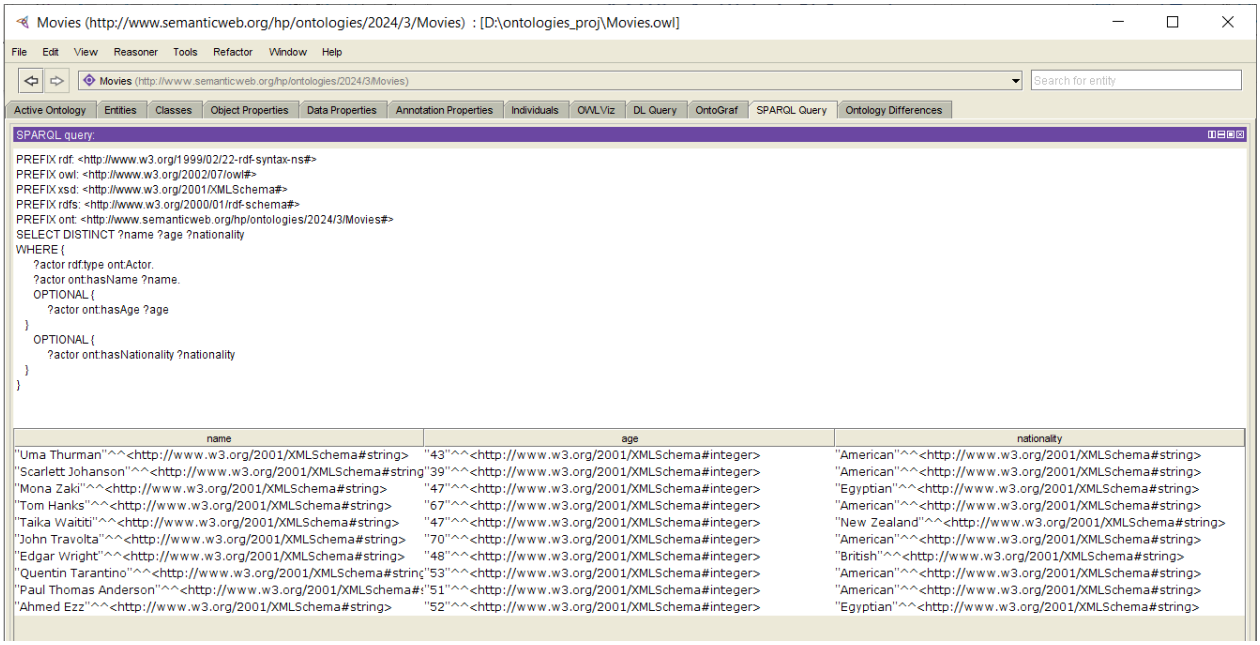
11) List the name of Actors older than 51 years.



12) A query that contains at least 2 Optional Graph Patterns

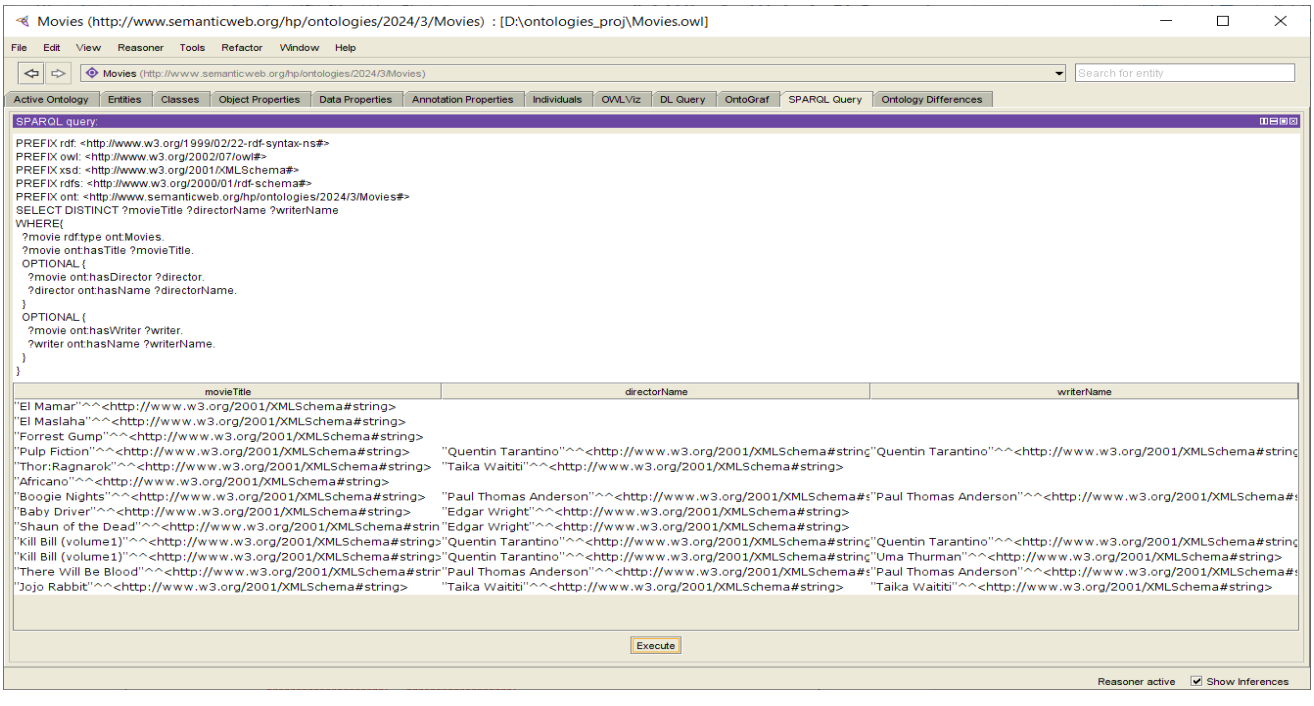
a) The query selects actors and includes optional graph patterns to retrieve age and nationality if available. The OPTIONAL keyword is used to specify the optional graph patterns.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT DISTINCT ?name ?age ?nationality

WHERE {

      ?actor rdf:type ont:Actor.

      ?actor ont:hasName ?name.

      OPTIONAL {

            ?actor ont:hasAge ?age

   }

      OPTIONAL {

            ?actor ont:hasNationality ?nationality

   }

}
```



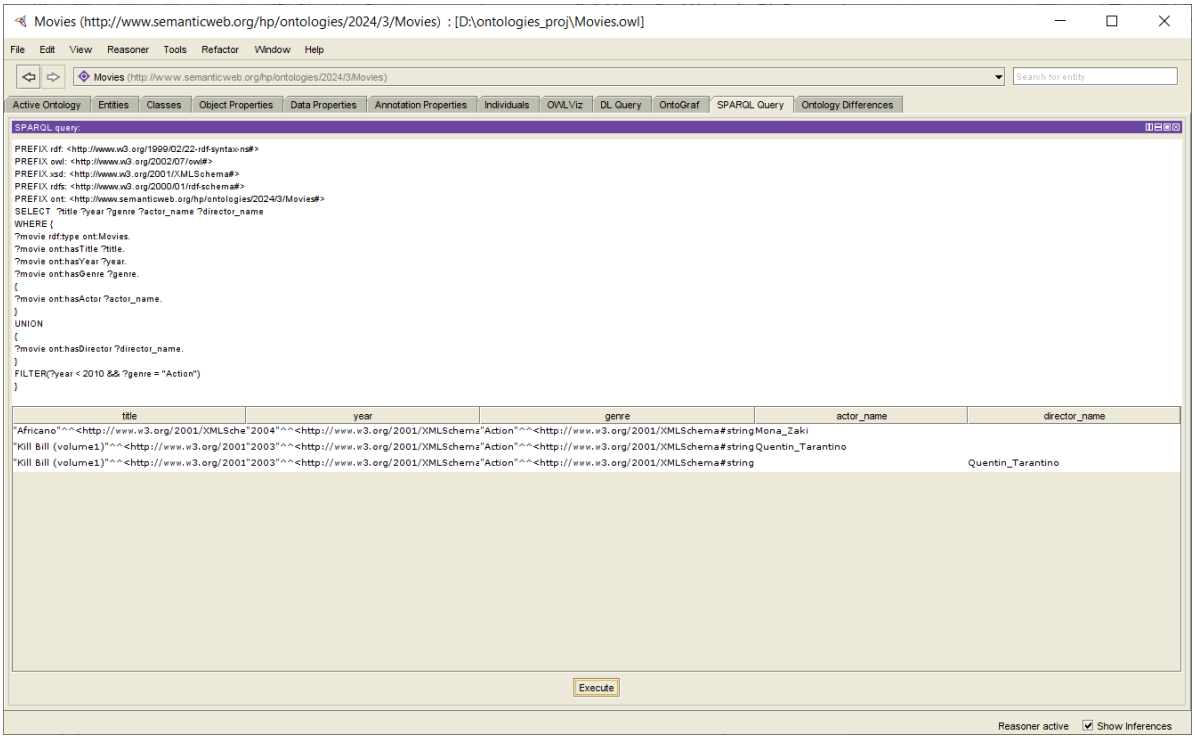
b) A query that selects movies and includes optional graph to retrieve the director's name and writer for each movie, if available.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT DISTINCT ?movieTitle ?directorName ?writerName

WHERE{

  ?movie rdf:type ont:Movies.

  ?movie ont:hasTitle ?movieTitle.

  OPTIONAL {

    ?movie ont:hasDirector ?director.

    ?director ont:hasName ?directorName.

  }

  OPTIONAL {

    ?movie ont:hasWriter ?writer.

    ?writer ont:hasName ?writerName.

  }

}
```



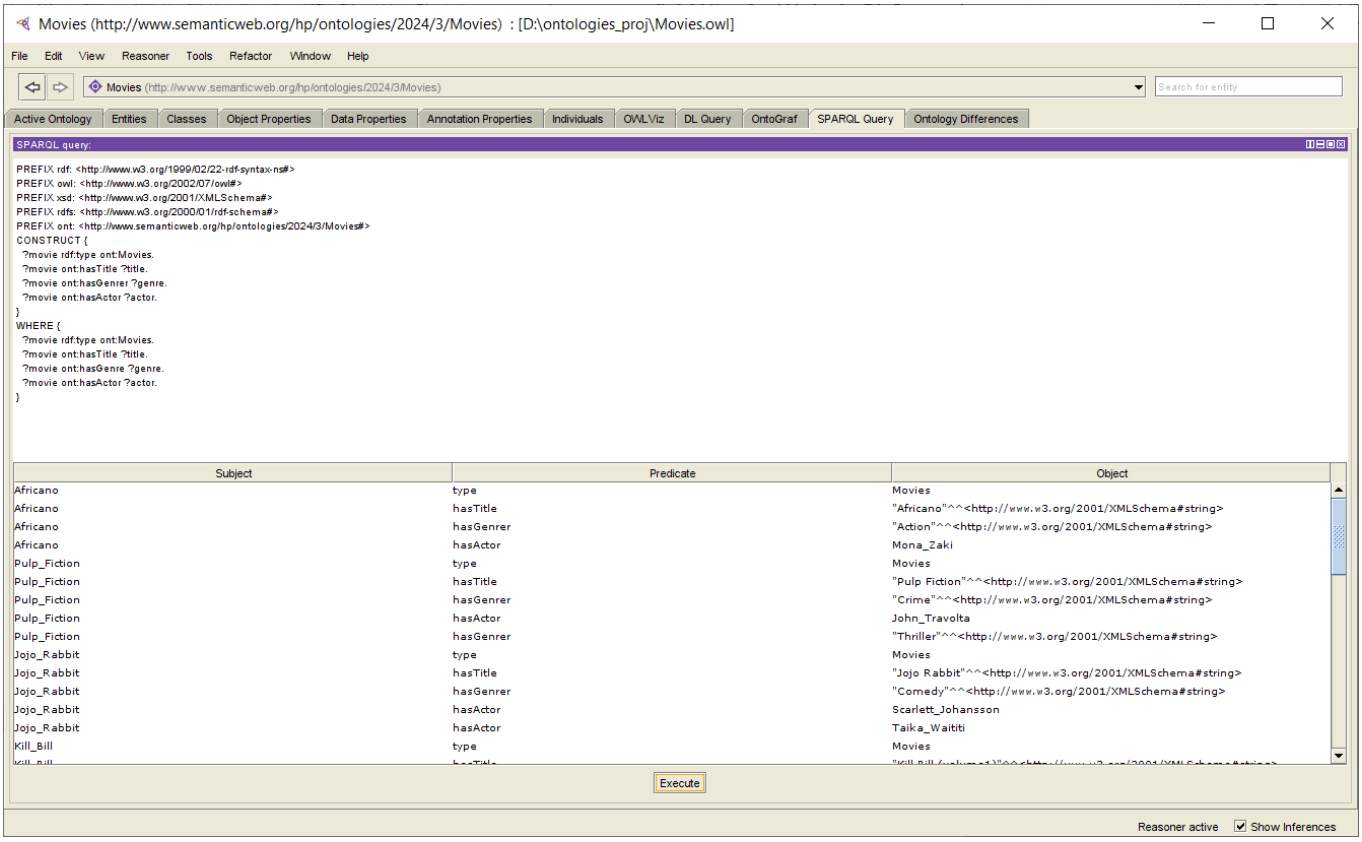
13) A query that contains at least 2 alternatives(UNION) and conjunctions(AND). The query selects movies and retrieves their titles, along with either the associated actors or the directors The query also includes a conjunction by using the FILTER function to make sure movie was before 2010 and has genre “Action”

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

SELECT  ?title ?year ?genre ?actor_name ?director_name

WHERE {

?movie rdf:type ont:Movies.

?movie ont:hasTitle ?title.

?movie ont:hasYear ?year.

?movie ont:hasGenre ?genre.

{

?movie ont:hasActor ?actor_name.

}

UNION

{

?movie ont:hasDirector ?director_name.

}

FILTER(?year < 2010 && ?genre = "Action")

}
```



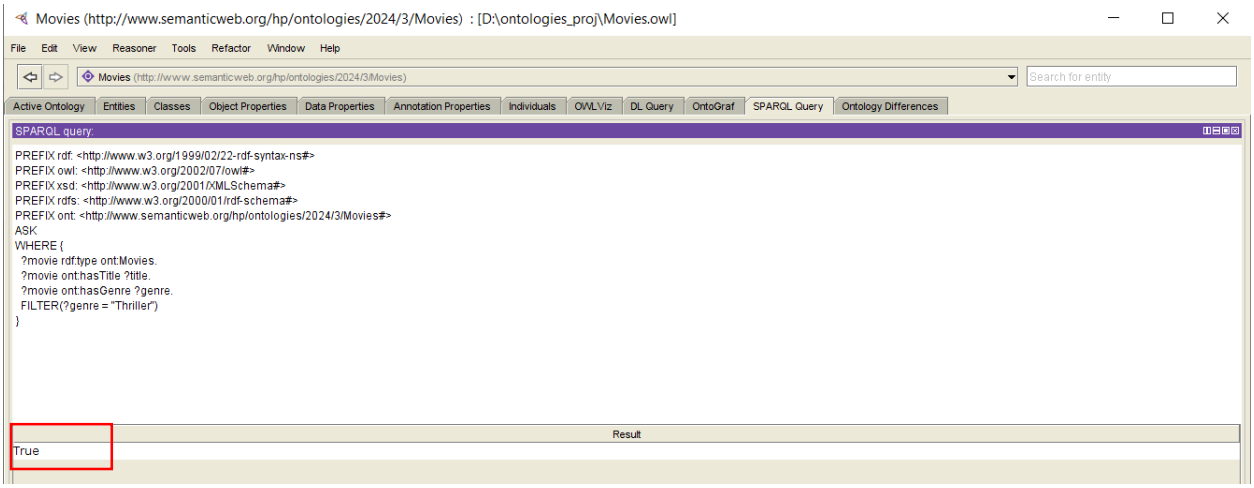
14) A query that contains a CONSTRUCT query form, construct triples for each movie, including its type, title, genre, and actor.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

CONSTRUCT {

  ?movie rdf:type ont:Movies.

  ?movie ont:hasTitle ?title.

  ?movie ont:hasGenrer ?genre.

  ?movie ont:hasActor ?actor.

}

WHERE {

  ?movie rdf:type ont:Movies.

  ?movie ont:hasTitle ?title.

  ?movie ont:hasGenre ?genre.

  ?movie ont:hasActor ?actor.

}
```



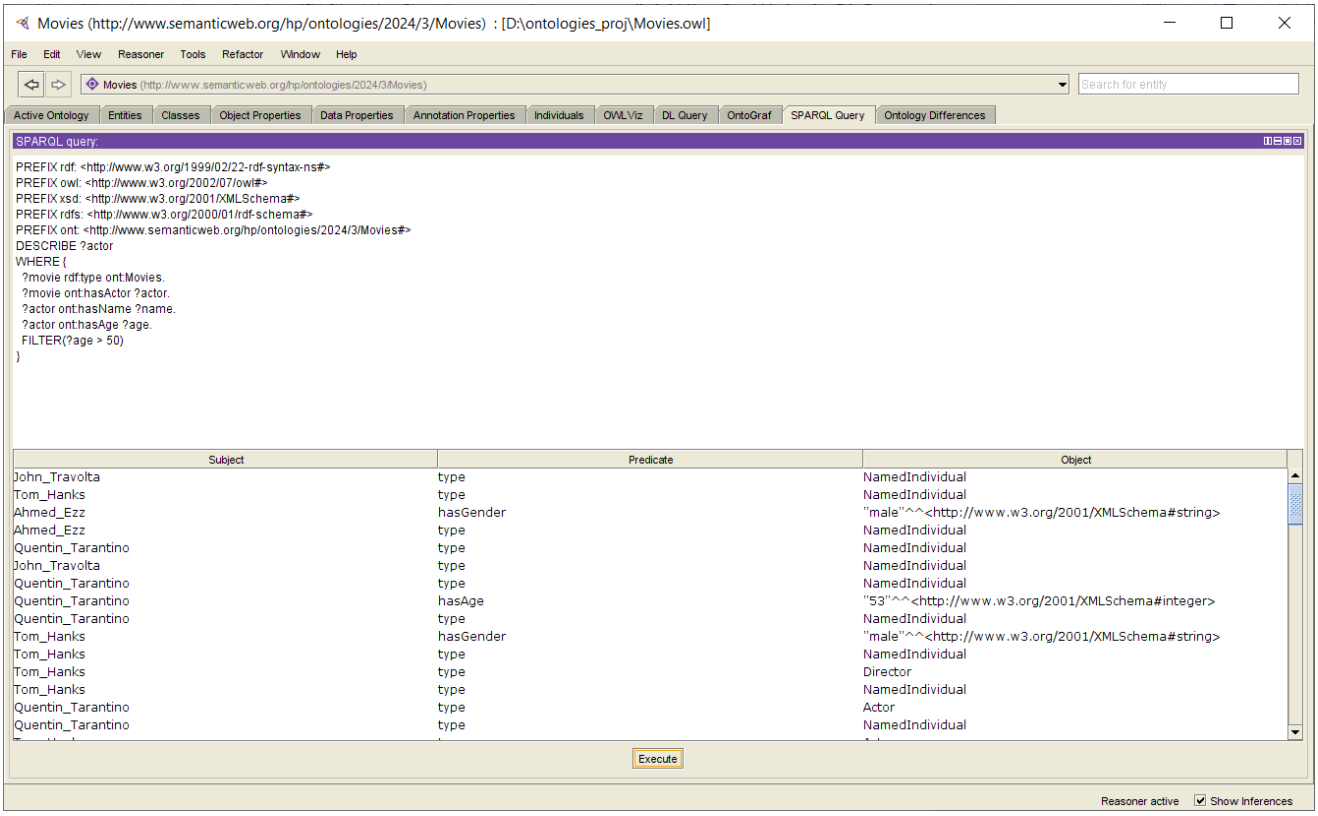
15) A query that contains an ASK query form: The ASK query form in SPARQL is used to ask a true/false question about the existence of a certain pattern in the data. The ASK query form will return either true or false, indicating whether there are movies with the genre "Thriller" in the data.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

ASK

WHERE {

  ?movie rdf:type ont:Movies.

  ?movie ont:hasTitle ?title.

  ?movie ont:hasGenre ?genre.

  FILTER(?genre = "Thriller")

}
```



16) A query that contains a DESCRIBE query form to retrieve RDF data that describes a specific resource (?actor).

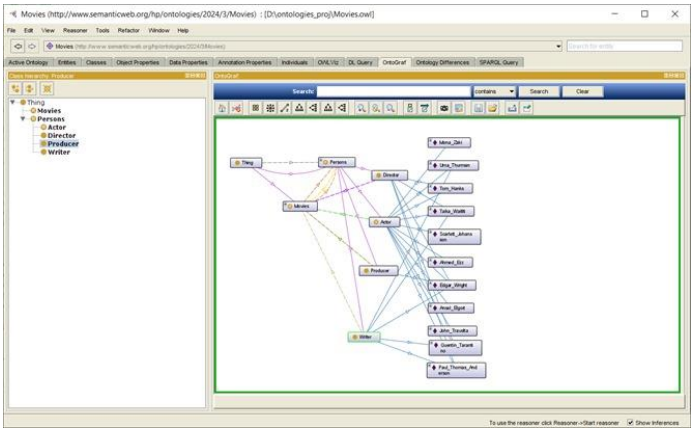
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

DESCRIBE ?actor

WHERE {

  ?movie rdf:type ont:Movies.

  ?movie ont:hasActor ?actor.

  ?actor ont:hasName ?name.

  ?actor ont:hasAge ?age.

  FILTER(?age > 50)
```

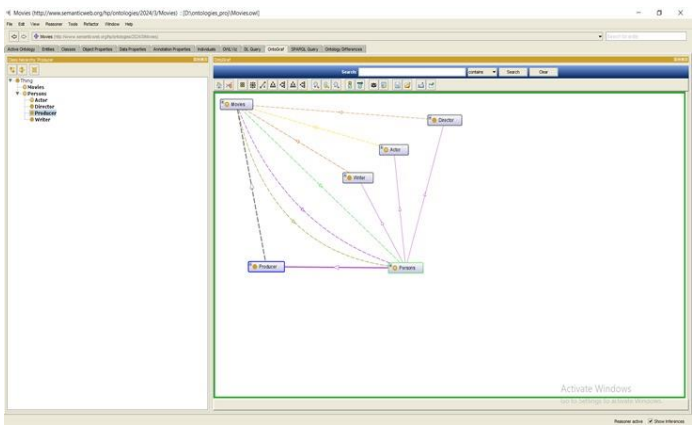


Visualize the Ontology:

Tool used:

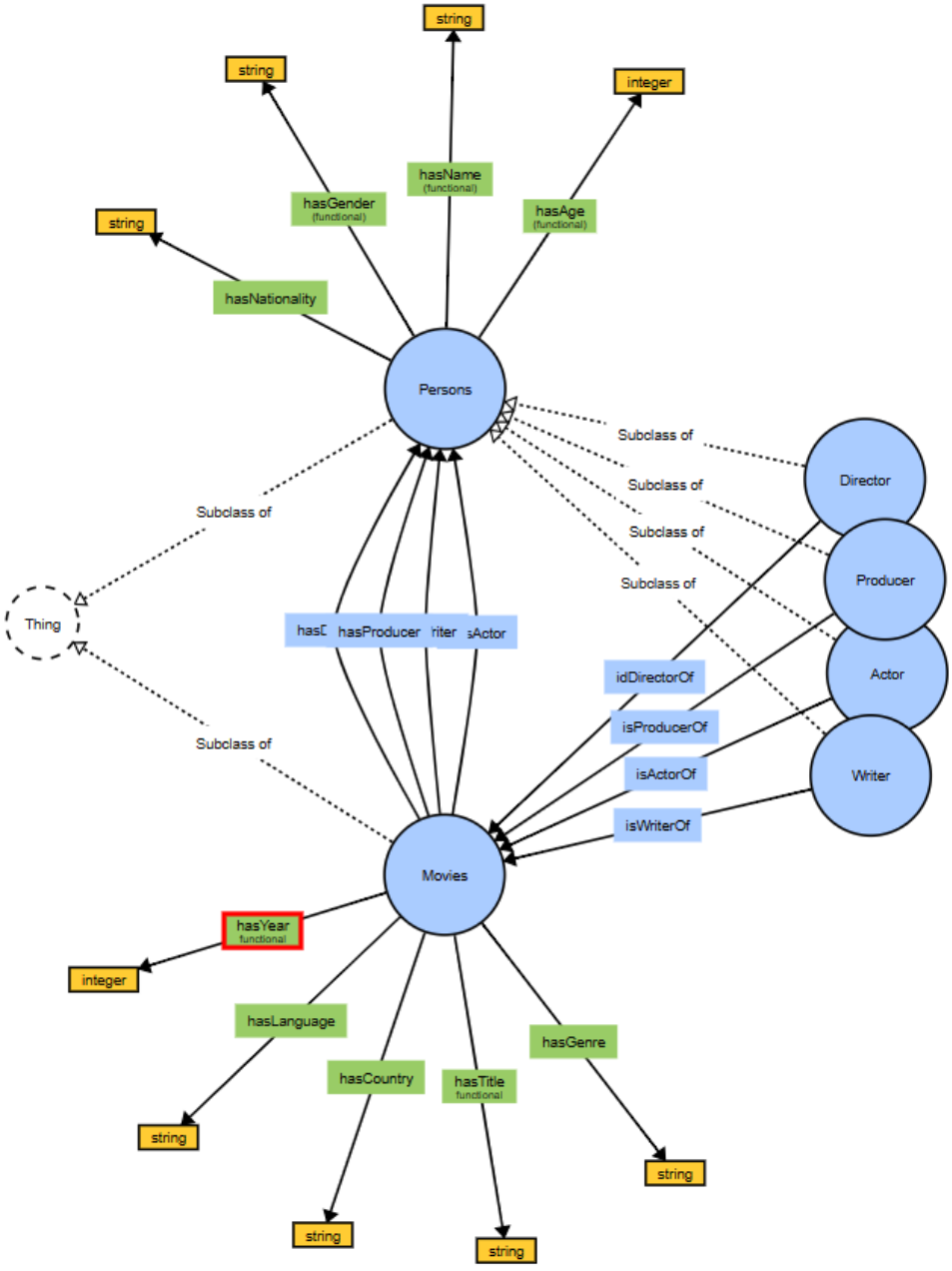
https://service.tib.eu/webvowl/





Graph:

https://service.tib.eu/webvowl/#opts=editorMode=true;#new_ontology1

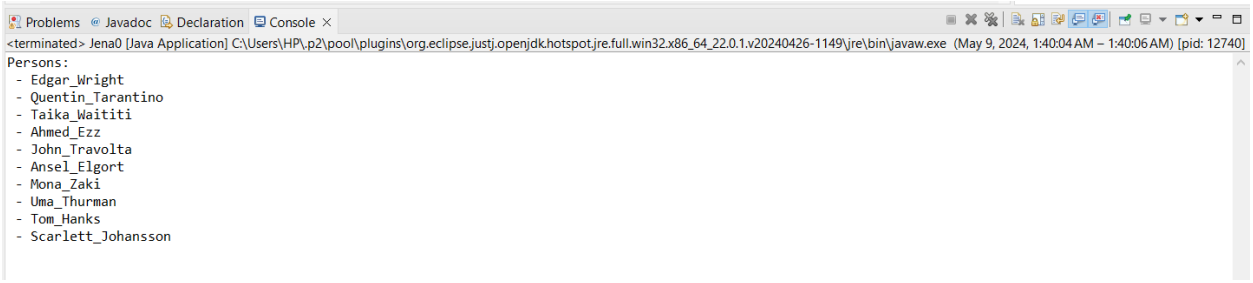


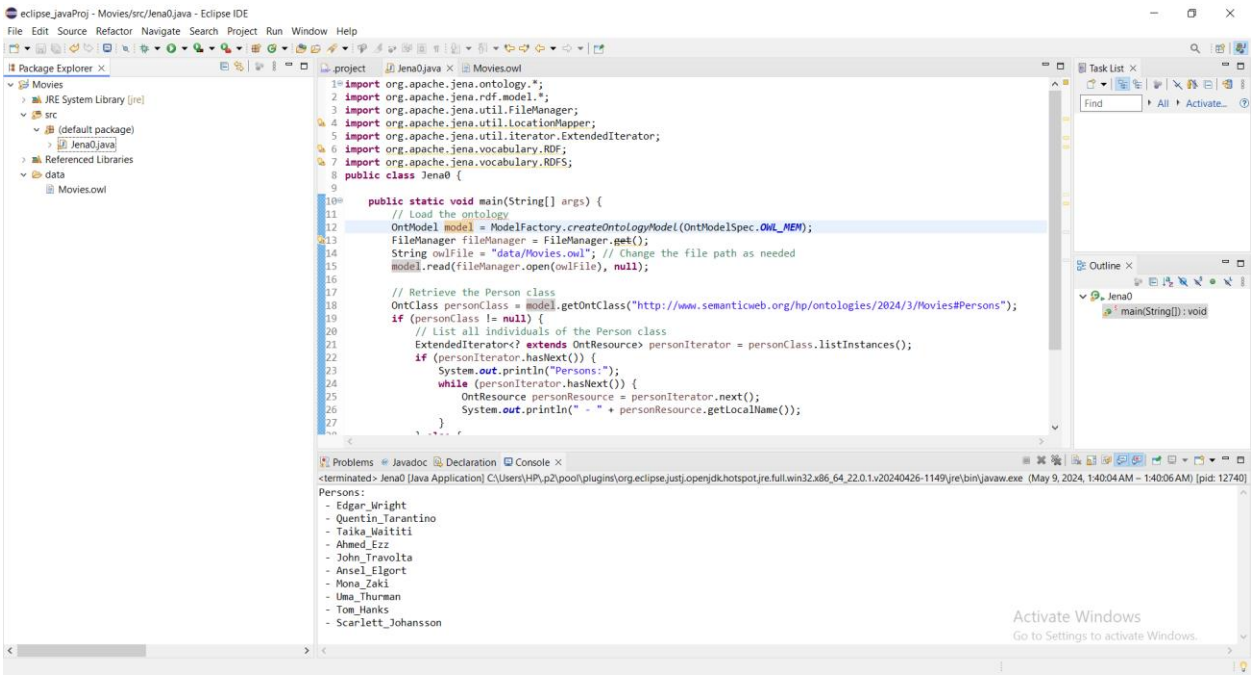
Part IV: Manipulating the ontology using Jena

1. Create a java program (Jena1.java) that loads the ontology and displays all the Persons (without using queries, without inference).

```
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.LocationMapper;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.vocabulary.RDFS;
public class Jena0 {

        public static void main(String[] args) {
                // Load the ontology
    OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
    FileManager fileManager = FileManager.get();
    String owlFile = "data/Movies.owl"; // Change the file path as needed
    model.read(fileManager.open(owlFile), null);

    // Retrieve the Person class
    OntClass personClass =
model.getOntClass("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Persons");
    if (personClass != null) {
      // List all individuals of the Person class
      ExtendedIterator<? extends OntResource> personIterator = personClass.listInstances();
      if (personIterator.hasNext()) {
        System.out.println("Persons:");
```

```
            while (personIterator.hasNext()) {
                OntResource personResource = personIterator.next();
                System.out.println(" - " + personResource.getLocalName());
            }
        } else {
            System.out.println("No individuals of the Person class found.");
        }
    } else {
        System.out.println("Person class not found in the ontology.");
    }
  }
}
```

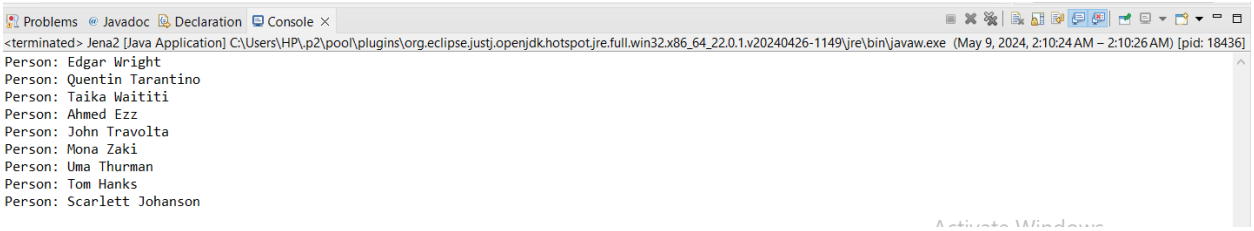


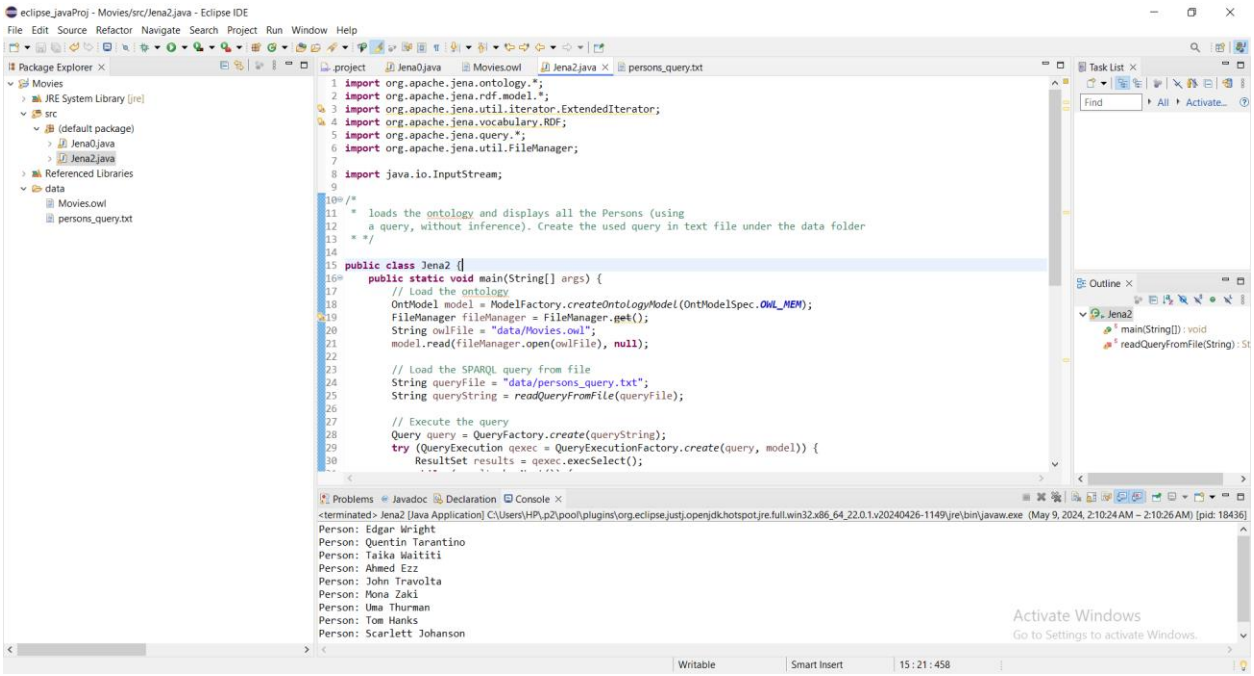


2. Create a java program (Jena2.java) that loads the ontology and displays all the Persons (using a query, without inference). Create the used query in text file under the data folder.

```
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;
import org.apache.jena.query.*;
import org.apache.jena.util.FileManager;

import java.io.InputStream;

/*
 *  loads the ontology and displays all the Persons (using
          a query, without inference). Create the used query in text file under the data folder
 * */

public class Jena2 {
   public static void main(String[] args) {
      // Load the ontology
      OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
      FileManager fileManager = FileManager.get();
      String owlFile = "data/Movies.owl";
      model.read(fileManager.open(owlFile), null);

      // Load the SPARQL query from file
      String queryFile = "data/persons_query.txt";
      String queryString = readQueryFromFile(queryFile);

      // Execute the query
      Query query = QueryFactory.create(queryString);
      try (QueryExecution qexec = QueryExecutionFactory.create(query, model)) {
         ResultSet results = qexec.execSelect();
         while (results.hasNext()) {
            QuerySolution solution = results.next();
            String personName = solution.getLiteral("name").getString();
            System.out.println("Person: " + personName);
         }
      }
   }

   // Helper method to read the query from file
   private static String readQueryFromFile(String queryFile) {
      StringBuilder sb = new StringBuilder();
```

```
            try (InputStream inputStream = FileManager.get().open(queryFile)) {
               if (inputStream != null) {
                  String line;
                  java.io.BufferedReader reader = new java.io.BufferedReader(
                        new java.io.InputStreamReader(inputStream, "UTF-8"));
                  while ((line = reader.readLine()) != null) {
                     sb.append(line).append("\n");
                  }
               }
            } catch (java.io.IOException e) {
               e.printStackTrace();
            }
            return sb.toString();
         }
      }
```

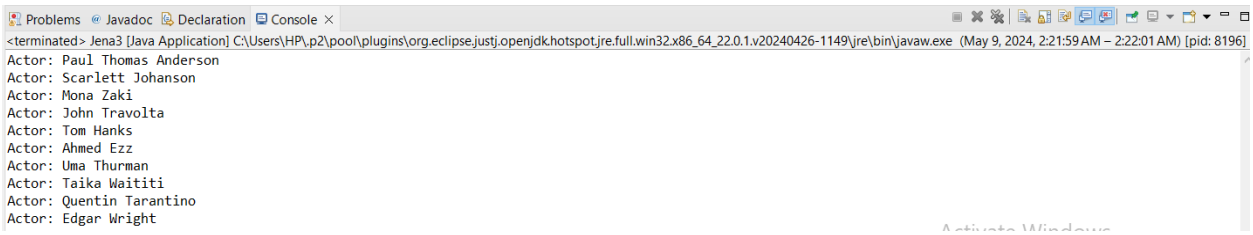


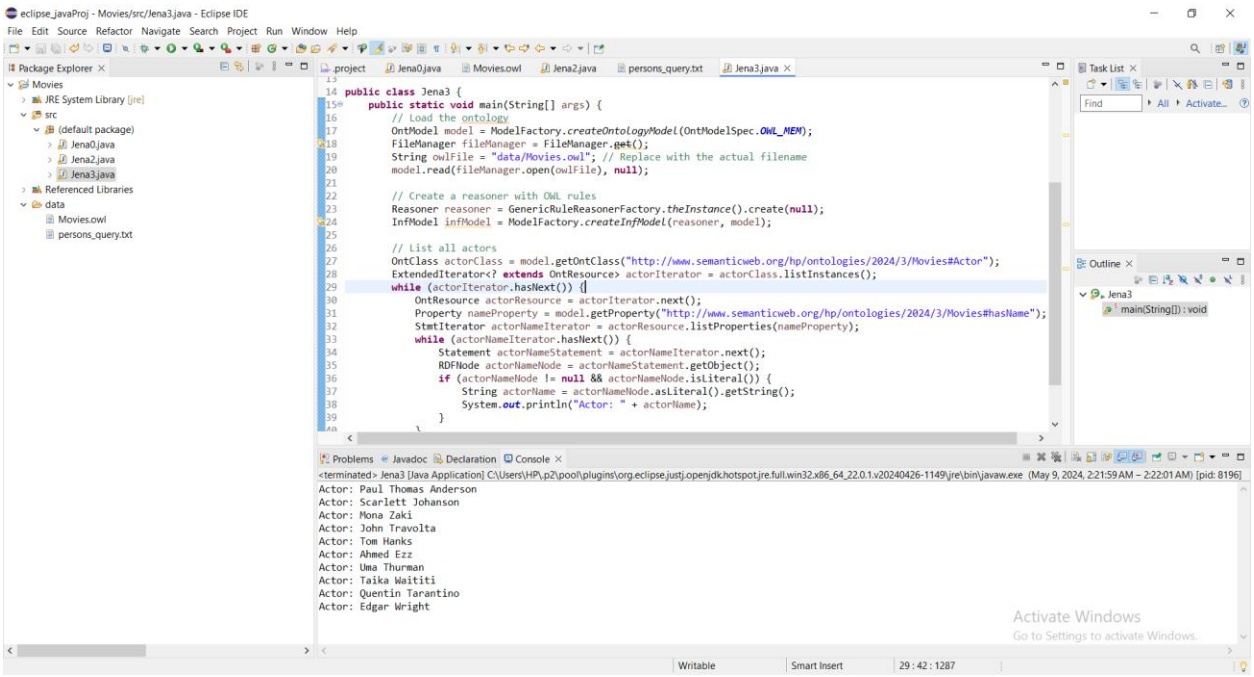


3. Create a java program (Jena3.java) that loads the ontology and displays all the Actors (without using queries, using inference). To load the inferred model, use the JenaEngine.readInferencedModelFromRuleFile method and use owl rules

```
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.rulesys.GenericRuleReasonerFactory;
import org.apache.jena.util.FileManager;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;

/*
 *  loads the ontology and displays all the Actors
        (without using queries, using inference).
 * */

public class Jena3 {
   public static void main(String[] args) {
      // Load the ontology
      OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
      FileManager fileManager = FileManager.get();
      String owlFile = "data/Movies.owl"; // Replace with the actual filename
      model.read(fileManager.open(owlFile), null);

      // Create a reasoner with OWL rules
      Reasoner reasoner = GenericRuleReasonerFactory.theInstance().create(null);
      InfModel infModel = ModelFactory.createInfModel(reasoner, model);

      // List all actors
```

```
        OntClass actorClass =
model.getOntClass("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Actor");

        ExtendedIterator<? extends OntResource> actorIterator = actorClass.listInstances();

        while (actorIterator.hasNext()) {

            OntResource actorResource = actorIterator.next();

            Property nameProperty =
model.getProperty("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasName");

            StmtIterator actorNameIterator = actorResource.listProperties(nameProperty);

            while (actorNameIterator.hasNext()) {

                Statement actorNameStatement = actorNameIterator.next();

                RDFNode actorNameNode = actorNameStatement.getObject();

                if (actorNameNode != null && actorNameNode.isLiteral()) {

                    String actorName = actorNameNode.asLiteral().getString();

                    System.out.println("Actor: " + actorName);

                }

            }

        }

    }

}
```





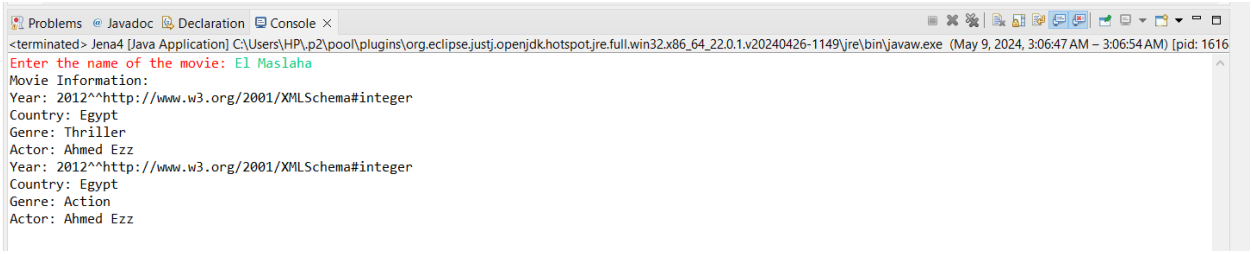
4. Create a java program (Jena4.java) that:
    a. Reads a name of a movie
    b. If it doesn't exist displays an error message
    c. Else, display its year, country, genres and actors

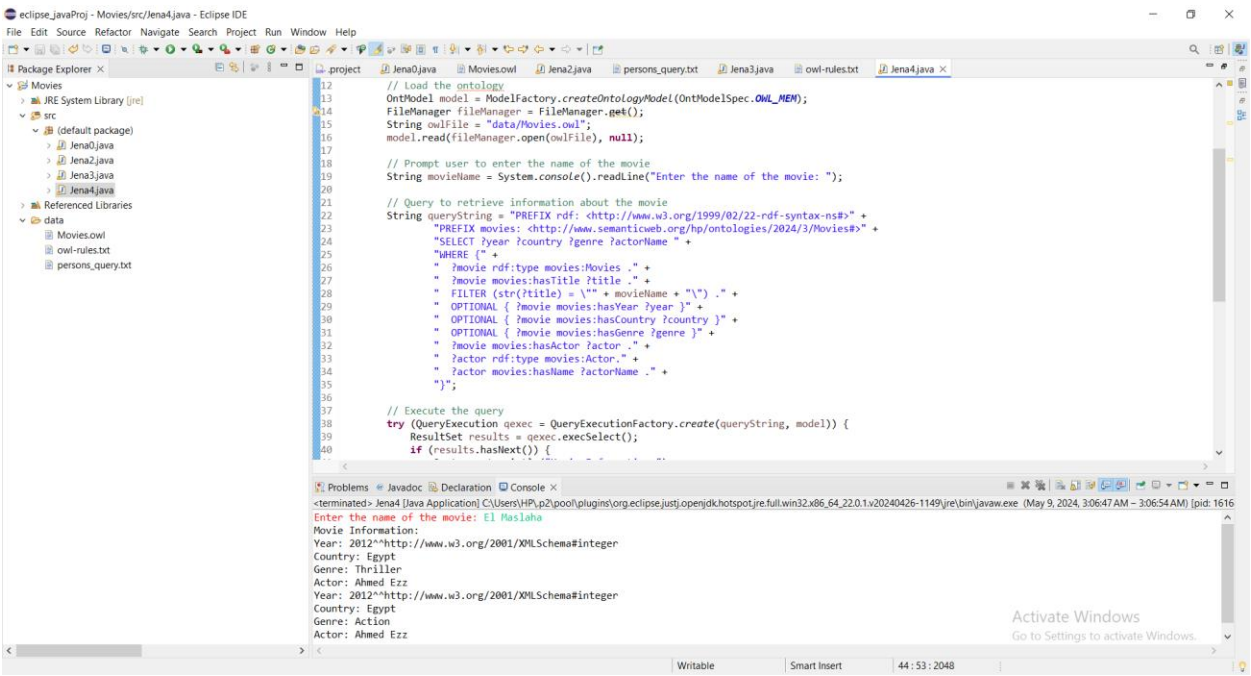
```
import org.apache.jena.ontology.OntModel;

import org.apache.jena.ontology.OntModelSpec;

import org.apache.jena.query.*;

import org.apache.jena.rdf.model.*;

import org.apache.jena.util.FileManager;


import java.util.Iterator;


public class Jena4 {


   public static void main(String[] args) {

      // Load the ontology
```

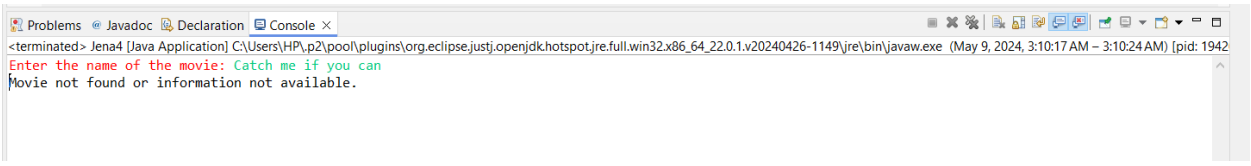
```
    OntModel model =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

    FileManager fileManager = FileManager.get();

     String owlFile = "data/Movies.owl";

     model.read(fileManager.open(owlFile), null);


     // Prompt user to enter the name of the movie

     String movieName = System.console().readLine("Enter the name of the movie:
");


     // Query to retrieve information about the movie

     String queryString = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>" +

          "PREFIX movies:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>" +

          "SELECT ?year ?country ?genre ?actorName " +

          "WHERE {" +

          "  ?movie rdf:type movies:Movies ." +

          "  ?movie movies:hasTitle ?title ." +

          "  FILTER (str(?title) = \"" + movieName + "\") ." +

          "  OPTIONAL { ?movie movies:hasYear ?year }" +

          "  OPTIONAL { ?movie movies:hasCountry ?country }" +

          "  OPTIONAL { ?movie movies:hasGenre ?genre }" +

          "  ?movie movies:hasActor ?actor ." +

          "  ?actor rdf:type movies:Actor." +

          "  ?actor movies:hasName ?actorName ." +

          "}";
```

```
        // Execute the query
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString,
model)) {
            ResultSet results = qexec.execSelect();
            if (results.hasNext()) {
                System.out.println("Movie Information:");
                while (results.hasNext()) {
                    QuerySolution soln = results.nextSolution();
                    RDFNode year = soln.get("year");
                    RDFNode country = soln.get("country");
                    RDFNode genre = soln.get("genre");
                    RDFNode actor = soln.get("actorName");

                    if (year != null) {
                        System.out.println("Year: " + year.toString());
                    }
                    if (country != null) {
                        System.out.println("Country: " + country.toString());
                    }
                    if (genre != null) {
                        System.out.println("Genre: " + genre.toString());
                    }
                    if (actor != null) {
                        System.out.println("Actor: " + actor.toString());
                    }
                }
            } else {
```

```
                System.out.println("Movie not found or information not available.");
            }
        }
    }
}
```







5. Create a java program (Jena5.java) that displays all persons that are actors and directors. Do this using a rule that defines a new class ActorDirector. The rule file must be saved in the data folder.

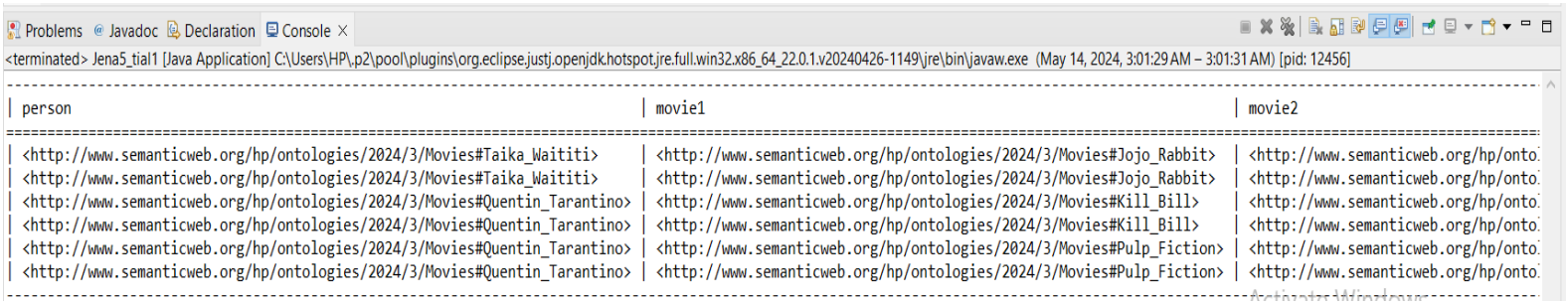
Jena5_tial1.java code:

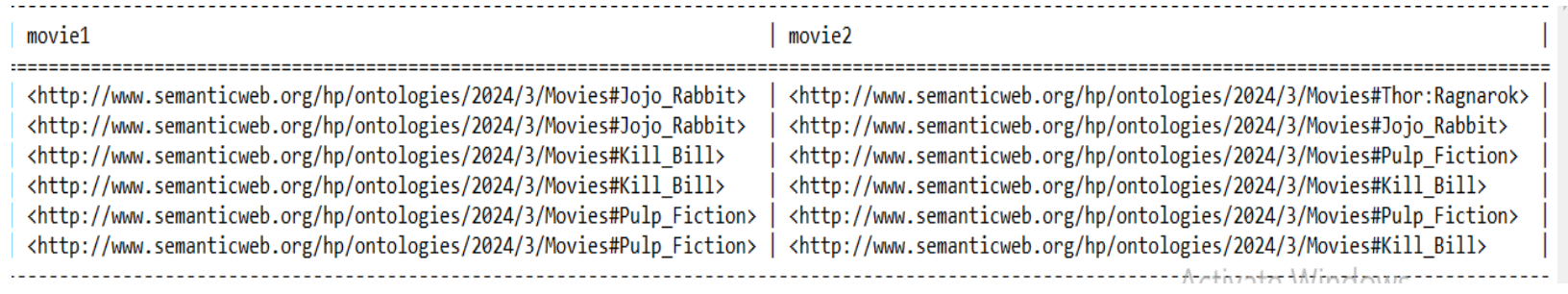
```
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.reasoner.Reasoner;

public class Jena5_tial1 {
        public static void main(String[] args)
  {
                Model model = ModelFactory.createDefaultModel();
                model.read( "data/Movies.owl" );

                Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL(
"data/Jena5_rules.txt" ) );
                InfModel infModel = ModelFactory.createInfModel( reasoner, model );

                readPersonAge(infModel);
  }

        public static void readPersonAge(Model model) {
           String query = "PREFIX ns:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#> " +
                   "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
                   "SELECT ?person ?movie1 ?movie2 " +
                   "WHERE { " +
                   "  ?person rdf:type ns:ActorDirector . " +
                   "  ?person ns:isActorOf ?movie1 . " +
                   "  ?person ns:isDirectorOf ?movie2 . " +
                   "}";
           executeQuery(model, query);
```

```
        }

    // Helper method to execute and print query results
    private static void executeQuery(Model model, String queryString) {
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            ResultSet results = qexec.execSelect();
            ResultSetFormatter.out(System.out, results);
        }
    }
    }
```

6. Jena5_rules.txt:

```
@prefix ns: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

[rule1: (?per ns:isActorOf ?movie1) (?per ns:isDirectorOf ?movie2) -> (?per rdf:type ns:ActorDirector) ]
```

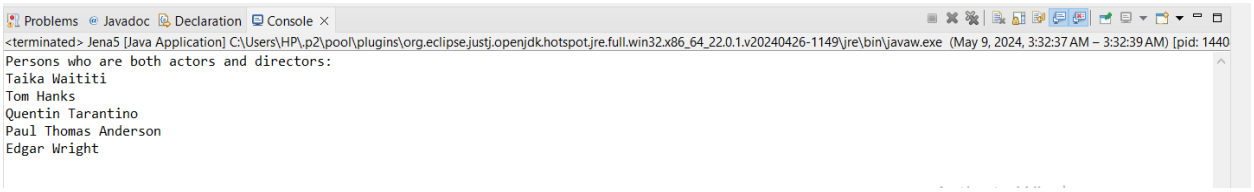





```
import java.io.InputStream;

import org.apache.jena.query.*;

import org.apache.jena.rdf.model.Model;

import org.apache.jena.rdf.model.ModelFactory;

import org.apache.jena.util.FileManager;
```

```
public class Jena5 {
    public static void main(String[] args) {
        // Load the ontology
        Model model = ModelFactory.createDefaultModel();
        FileManager fileManager = FileManager.get();
        String owlFile = "data/Movies.owl"; // Replace with the actual filename
        model.read(fileManager.open(owlFile), null);

        // Read the SPARQL query from file
        String queryFile = "data/actor_rules.txt";
        String queryString = readQueryFromFile(queryFile);

        // Execute the query and create a new model with the constructed triples
        Model resultModel = ModelFactory.createDefaultModel();
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            resultModel = qexec.execConstruct();
        }

        // Extract and display the names of the persons who are both actors and directors
        System.out.println("Persons who are both actors and directors:");

resultModel.listObjectsOfProperty(resultModel.createProperty("http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasName"))
                .forEachRemaining(name -> System.out.println(name.toString()));
    }

    // Helper method to read the query from file
    private static String readQueryFromFile(String queryFile) {
```

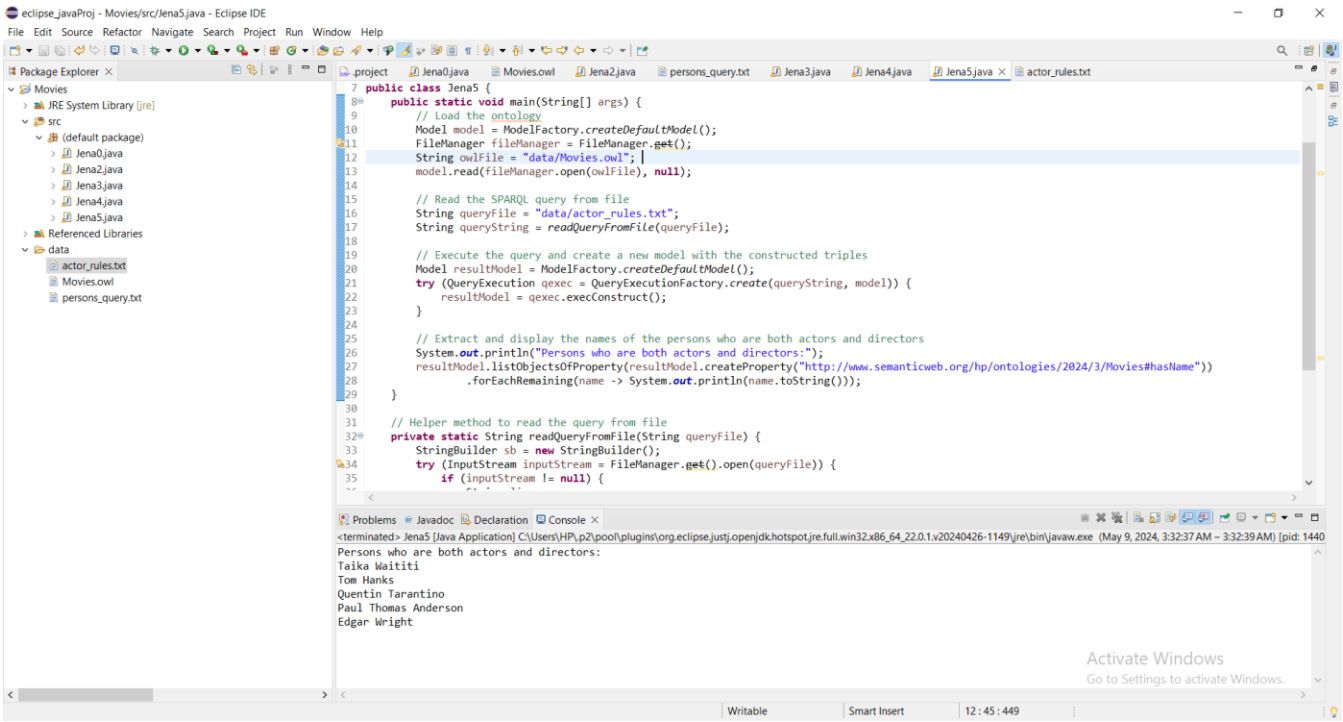
```
        StringBuilder sb = new StringBuilder();
        try (InputStream inputStream = FileManager.get().open(queryFile)) {
           if (inputStream != null) {
              String line;
              java.io.BufferedReader reader = new java.io.BufferedReader(
                    new java.io.InputStreamReader(inputStream, "UTF-8"));
              while ((line = reader.readLine()) != null) {
                 sb.append(line).append("\n");
              }
           }
        } catch (java.io.IOException e) {
           e.printStackTrace();
        }
        return sb.toString();
     }
  }
```



- **actor_rules.txt file:**

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX movies: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

CONSTRUCT {
   ?person movies:hasName ?name .
}
```

```
WHERE {
   ?person rdf:type movies:Actor .
   ?person rdf:type movies:Director .
   ?person movies:hasName ?name .
}
```



7. Specify 3 different rules and implement them in a java program (Jena6.java)

- Jena6_trial1 code:

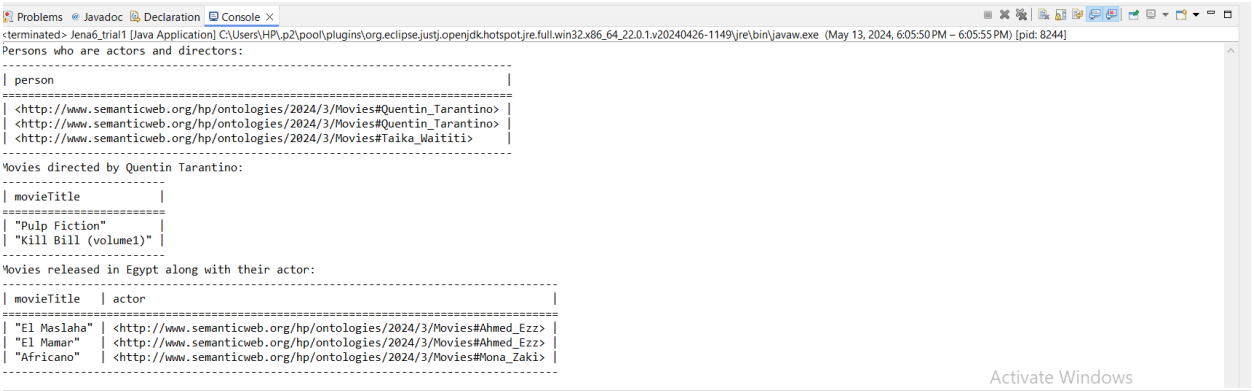
```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.rulesys.*;
import org.apache.jena.vocabulary.RDF;
```

```
public class Jena6_trial1 {
   public static void main(String[] args) {
      // Load the ontology
      Model model = ModelFactory.createDefaultModel();
      FileManager fileManager = FileManager.get();
      String owlFile = "data/Movies.owl"; // Replace with the actual filename
      model.read(fileManager.open(owlFile), null);

      // Define rules
      String rule1 = "[ruleHasTwoRoles: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?person) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasDirector> ?person) ->
(?person <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles>
?movie)]";
      String rule2 = "[ruleMoviesDirectedByQT: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasDirector>
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Quentin_Tarantino>) ->
(?movie <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#DirectedByQT>
?movie)]";
      String rule3 = "[ruleMoviesInEgypt: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasCountry>
\"Egypt\"^^xsd:string) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?actor) -> (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ActorName> ?actor)]";
      String rule4 = "[ruleMoviesInEgypt: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasGenre>
\"Thriller\"^^xsd:string) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?actor) -> (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerActor> ?actor)]";
      String rule5 = "[ruleMoviesInEgypt: (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasGenre>
\"Romance\"^^xsd:string) (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasActor> ?actor) -> (?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#RomanceActor> ?actor)]";


      // Create a reasoner with the rules
      Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rule1 + " " + rule2 + " " +
rule3+ " "+rule4+ " " + rule5));

      // Apply the reasoner to the model
      InfModel infModel = ModelFactory.createInfModel(reasoner, model);
```

```
        // Execute queries
        askForTwoRoles(infModel);
        askMoviesDirectedByQuentinTarantino(infModel);
        askMoviesInCountry(infModel, "Egypt");
        askMoviesInGenre(infModel, "Thriller");
        askMoviesInGenreRomance(infModel, "Romance");
    }

    // Rule 1: Persons Who Are Actors And Directors
    private static void askForTwoRoles(Model model) {
        System.out.println("Persons who are actors and directors:");
        String queryString = "SELECT ?person WHERE {?person
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#HasTwoRoles> ?movie}";
        executeAndPrintQuery(model, queryString);
    }

    // Rule 2: Movies Directed by Quentin Tarantino
    private static void askMoviesDirectedByQuentinTarantino(Model model) {
        System.out.println("Movies directed by Quentin Tarantino:");
        String queryString = "SELECT ?movieTitle WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#DirectedByQT> ?movie .
?movie <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle>
?movieTitle}";
        executeAndPrintQuery(model, queryString);
    }

    // Rule 3: Movies Released in a Specific Country with Directors' Names
    private static void askMoviesInCountry(Model model, String country) {
        System.out.println("Movies released in " + country + " along with their actor:");
        String queryString = "SELECT ?movieTitle ?actor WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ActorName> ?actor . ?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle> ?movieTitle}";
        executeAndPrintQuery(model, queryString);
    }

    // Rule 4: Movies with Thriller genre
    private static void askMoviesInGenre(Model model, String genre) {
            System.out.println("Movies have thriller genre " + genre + " along with their actor:");
        String queryString = "SELECT ?movieTitle ?actor WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#ThrillerActor> ?actor . ?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle> ?movieTitle}";
        executeAndPrintQuery(model, queryString);
    }
 // Rule 4: Movies with Romance genre
```

```
    private static void askMoviesInGenreRomance(Model model, String genre1) {
            System.out.println("Movies have thriller genre " + genre1 + " along with their actor:");
        String queryString = "SELECT ?movieTitle ?actor WHERE {?movie
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#RomanceActor> ?actor .
?movie <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#hasTitle>
?movieTitle}";
        executeAndPrintQuery(model, queryString);
    }


    // Helper method to execute and print query results
    private static void executeAndPrintQuery(Model model, String queryString) {
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            ResultSet results = qexec.execSelect();
            ResultSetFormatter.out(System.out, results);
        }
    }
}
```

- Output:



- Created Jena6_rules.java using inferred rules:
- The rule finds actors that are directors at the same time and display the movies they acted and directed in with new property called “HasTwoRoles”

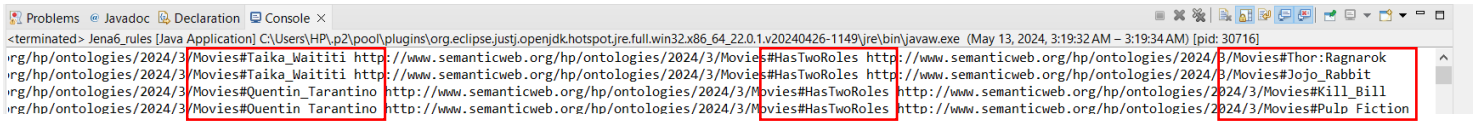
```
import org.apache.jena.ontology.OntModel;
```

```
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.*;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model.StmtIterator;
import org.apache.jena.reasoner.Reasoner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Iterator;

public class Jena6_rules {
        public static void main(String[] args)
  {
    Model model = ModelFactory.createDefaultModel();
    model.read( "data/Movies.owl" );

    Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL( "data/rul1.txt" ) );

    InfModel infModel = ModelFactory.createInfModel( reasoner, model );
    StmtIterator it = infModel.listStatements();

    while ( it.hasNext() )
    {
      Statement stmt = it.nextStatement();

      Resource subject = stmt.getSubject();
      Property predicate = stmt.getPredicate();
      RDFNode object = stmt.getObject();
      System.out.println( subject.toString() + " " + predicate.toString() + " " + object.toString()
);
    }
  }
}
```

- rul1.txt:

```
@prefix ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>

[ruleHasTwoRoles: (?s ont:hasActor ?c) (?b ont:hasDirector ?c) -> (?c ont:HasTwoRoles ?s) (?c ont:HasTwoRoles ?b)]
```

- Output:



- Jena7 is to display actors below 51 using rule:
- Jena7 code:

```
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.*;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model.StmtIterator;
import org.apache.jena.reasoner.Reasoner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Iterator;

public class Jena7 {
        public static void main(String[] args)
  {
                Model model = ModelFactory.createDefaultModel();
                model.read( "data/Movies.owl" );

                Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL(
"data/Jena7_rules.txt" ) );
                InfModel infModel = ModelFactory.createInfModel( reasoner, model );
```

```
                readPersonAge(infModel);
    }

        public static void readPersonAge(Model model) {
            String query = "PREFIX ns:
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#> " +
                    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
                    "SELECT ?person ?age " +
                    "WHERE { " +
                    "  ?person rdf:type ns:PersonAge . " +
                    "  ?person ns:hasAge ?age . " +
                    "}";
            executeQuery(model, query);
        }

// Helper method to execute and print query results
private static void executeQuery(Model model, String queryString) {
    try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
        ResultSet results = qexec.execSelect();
        ResultSetFormatter.out(System.out, results);
    }
}
}
```

- Jena7_rule.txt:

```
@prefix ns: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

[rule1: (?per rdf:type ns:Persons) (?per ns:hasAge ?age) lessThan(?age, 51)-> (?per
rdf:type ns:PersonAge) ]
```

- **Output:**

Problems Javadoc Declaration Console ×

<terminated> Jena7 [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe

person	age
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Edgar_Wright>	48
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Taika_Waititi>	47
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Mona_Zaki>	47
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Uma_Thurman>	43
<http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Scarlett_Johansson>	39

Test Cases:

https://jena.apache.org/documentation/inference/

A suite of test cases designed to evaluate the correctness and completeness of the ontology. These test cases may cover various aspects such as ontology consistency, inference correctness, and query performance.

Inference Correctness:

Inference correctness focuses on assessing whether the ontology's reasoning mechanisms produce accurate and valid results. Test cases in this category evaluate the ontology's ability to infer new knowledge based on existing data and logical rules. These tests often involve scenarios where the ontology needs to deduce implicit information or make logical conclusions from explicit assertions. Examples include verifying the correctness of inferred relationships between entities or validating the outcomes of logical entailment based on ontology axioms.

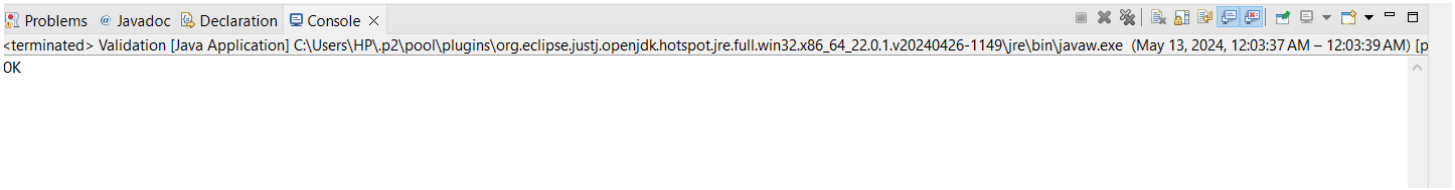
The most common reasoner operation which can't be exposed through additional triples in the inference model is that of validation. Typically the ontology languages used with the semantic web allow constraints to be expressed, the validation interface is used to detect when such constraints are violated by some data set.

A simple but typical example is that of datatype ranges in RDFS. RDFS allows us to specify the range of a property as lying within the value space of some datatype. If an RDF statement asserts an object value for that property which lies outside the given value space there is an inconsistency.

To test for inconsistencies with a data set using a reasoner we use the `InfModel.validate()` interface. This performs a global check across the schema and instance data looking for inconsistencies. The result is a `ValidityReport` object which comprises a simple pass/fail flag (`ValidityReport.isValid()`) together with a list of specific reports (instances of the `ValidityReport.Report` interface) which detail any detected inconsistencies. At a minimum the individual reports should be printable descriptions of the problem but they can also contain an arbitrary reasoner-specific object which can be used to pass additional information which can be used for programmatic handling of the violations.

```
import java.util.Iterator;
import org.apache.jena.ontology.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.ValidityReport;
import org.apache.jena.reasoner.rulesys.GenericRuleReasonerFactory;
import org.apache.jena.riot.RDFDataMgr;
```

```java
import org.apache.jena.util.FileManager;
import org.apache.jena.util.iterator.ExtendedIterator;
import org.apache.jena.vocabulary.RDF;


public class Validation {
        public static void main(String[] args) {
        //Model model = ModelFactory.createDefaultModel();
  //FileManager fileManager = FileManager.get();
  String fname = "data/Movies.owl";
 // model.read(fileManager.open(owlFile), null);
        Model data = RDFDataMgr.loadModel(fname);
        InfModel infmodel = ModelFactory.createRDFSModel(data);
        ValidityReport validity = infmodel.validate();
        if (validity.isValid()) {
           System.out.println("OK");
        } else {
           System.out.println("Conflicts");
           for (Iterator i = validity.getReports(); i.hasNext(); ) {
              System.out.println(" - " + i.next());
           }
        }
         }
```

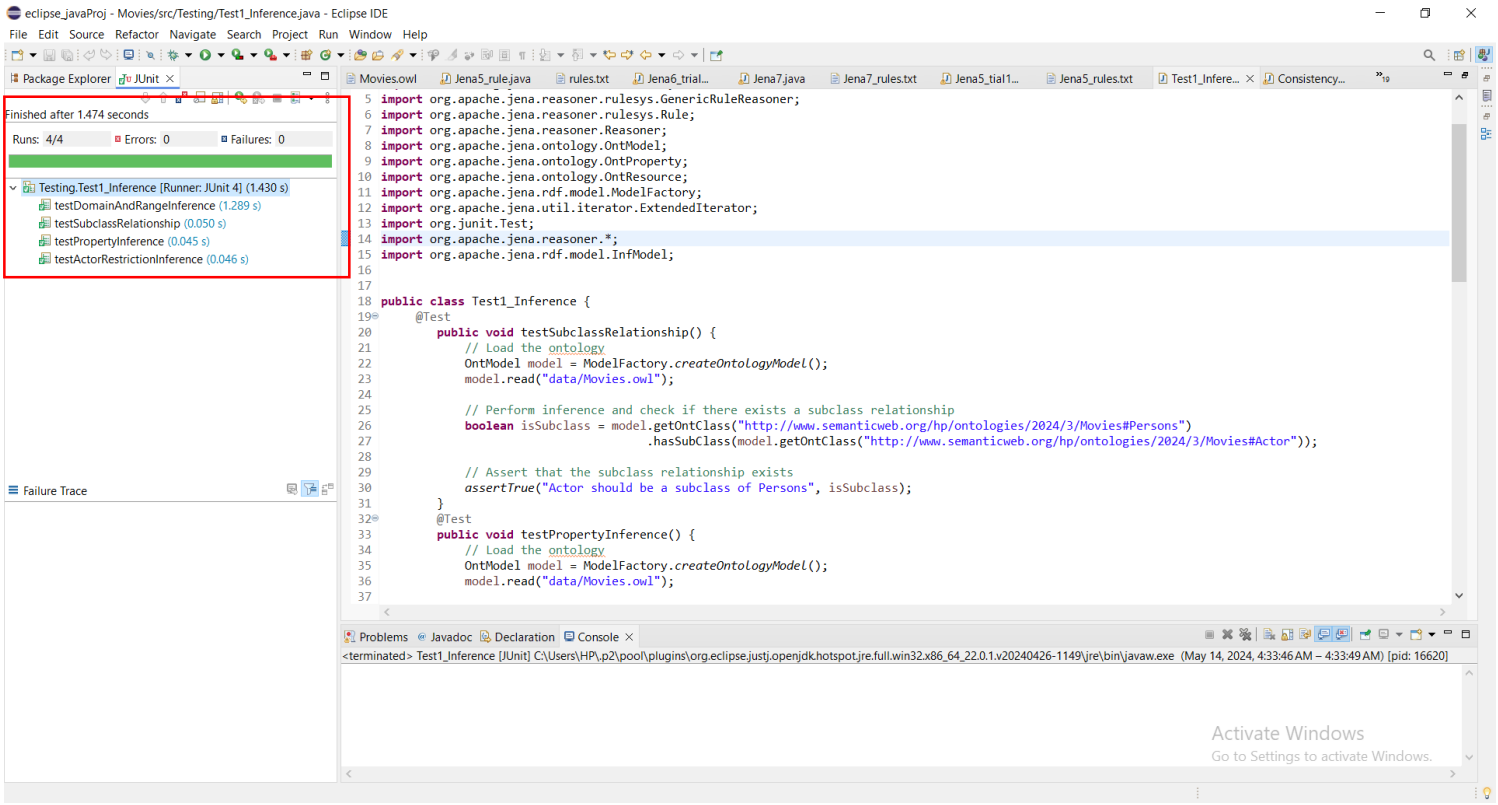


- Possible outcome if there is inconsistencies:

The file `testing/reasoners/rdfs/dttest2.nt` declares a property `bar` with range `xsd:integer` and attaches a `bar` value to some resource with the value `"25.5"^^xsd:decimal`. If we run the above sample code on this file we see:

> *Conflicts*
> *- Error (dtRange): Property http://www.hpl.hp.com/semweb/2003/eg#bar has a typed range Datatype[http://www.w3.org/2001/XMLSchema#integer -> class java.math.BigInteger]that is not compatible with 25.5:http://www.w3.org/2001/XMLSchema#decimal*

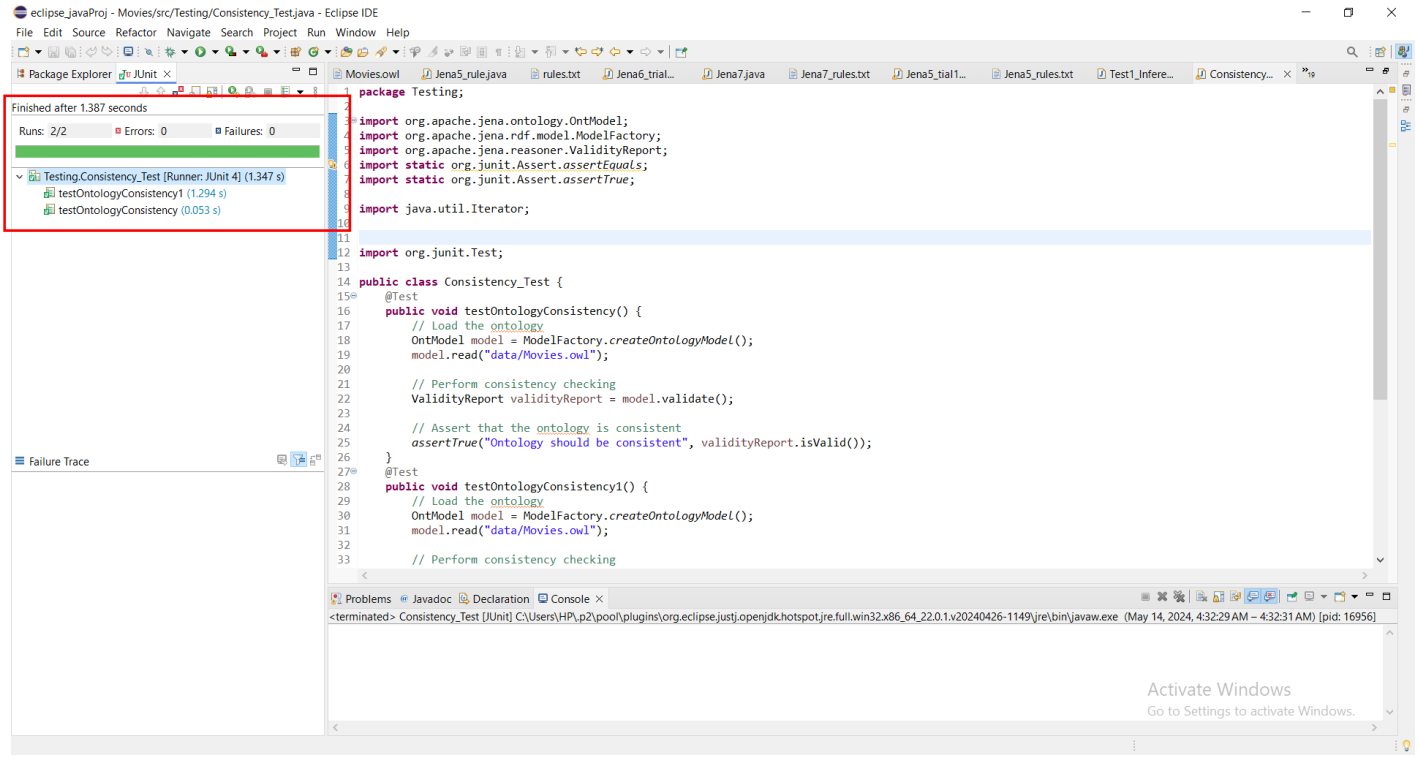
- Made a test suite using JUNIT Tes1_Inference:



Ontology Consistency:

This aspect ensures that the ontology maintains internal coherence and integrity. Test cases under this category typically involve verifying that the relationships, properties, and constraints defined within the ontology are consistent and adhere to specified rules and axioms. For instance, test cases may check for inconsistencies in data representation, such as conflicting statements about the same entity, or violations of defined constraints.

- Created Test suite called Consistency_Test:



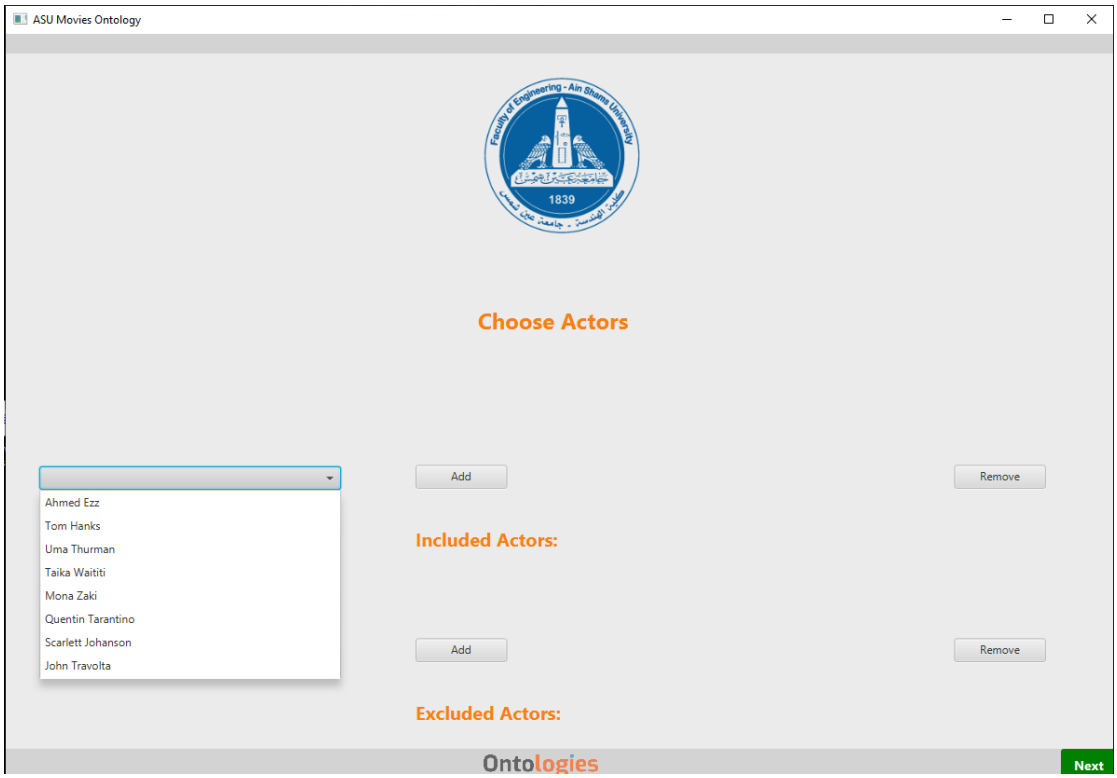
Query Performance:

Query performance evaluates the efficiency and effectiveness of querying operations on the ontology. Test cases under this aspect assess the ontology's responsiveness and scalability when processing queries of varying complexity and size. This includes measuring the speed of query execution, evaluating resource utilization (such as memory and CPU usage), and assessing the ontology's ability to handle concurrent or parallel queries. Test scenarios may involve benchmarking query response times, stress testing the system with a large volume of queries,and analyzing performance metrics under different load conditions.

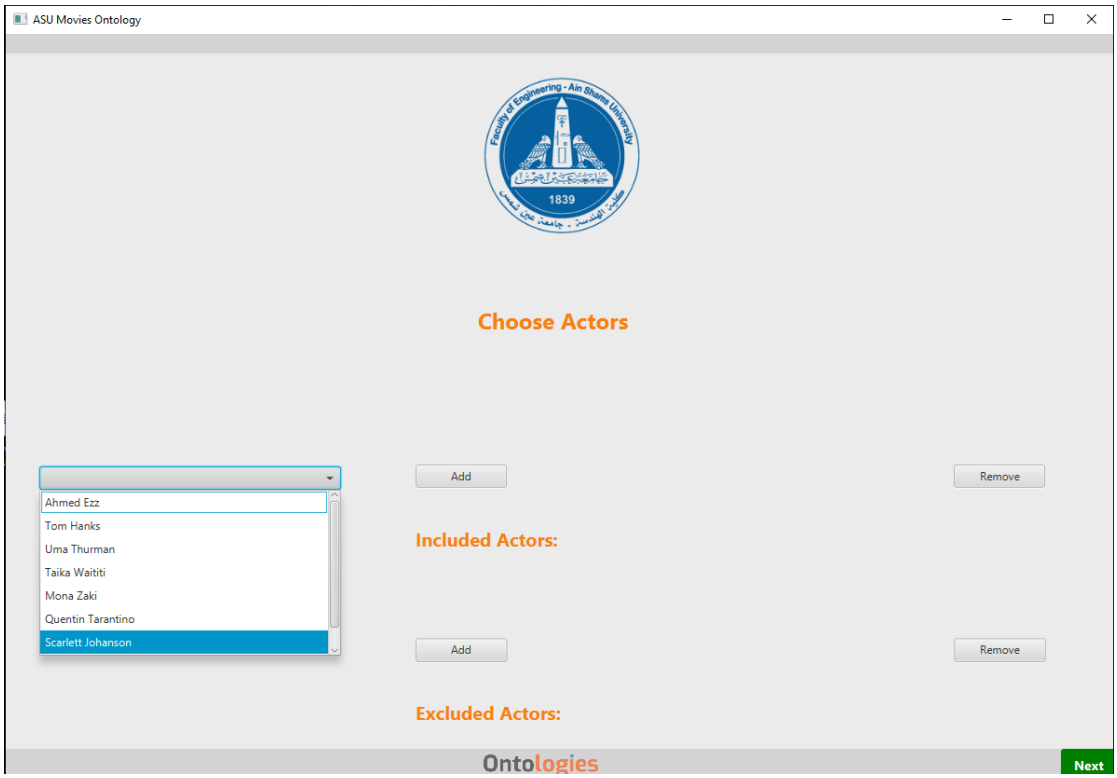
Snapshots of the Interface:

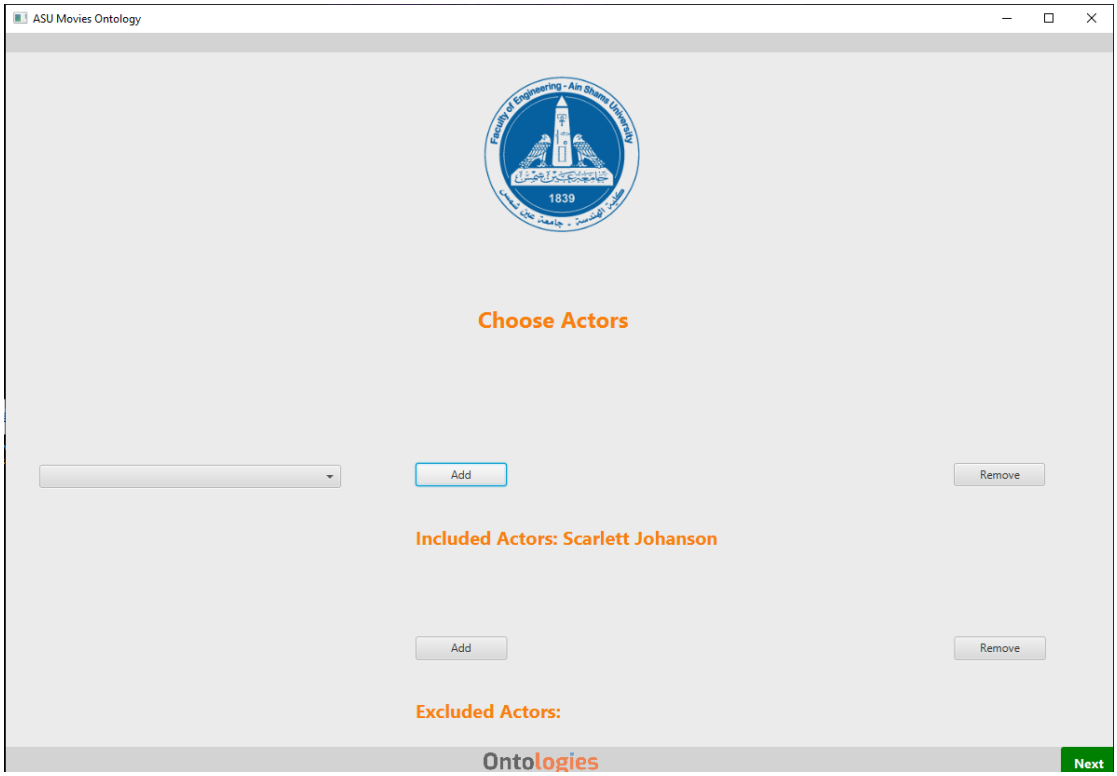
1st: Choose the actors to be included\excluded:

- The application queries the ontology for all actor names allowing the user to choose the actors to be included\excluded through an interactive GUI .

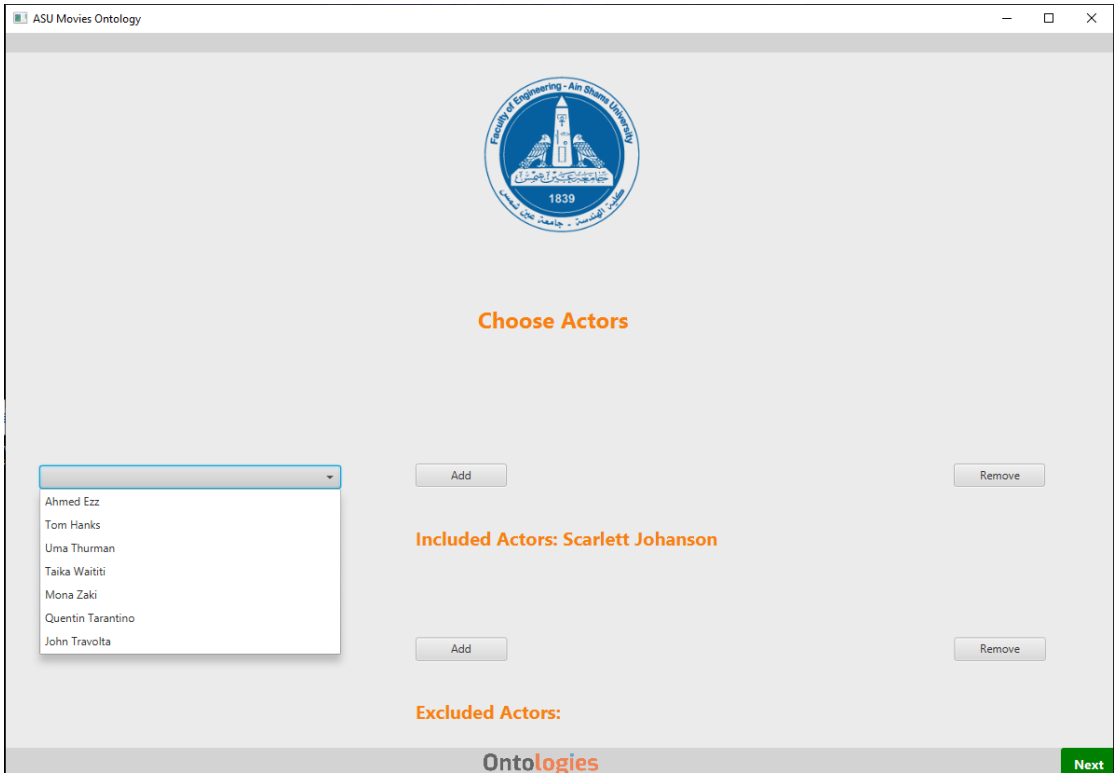


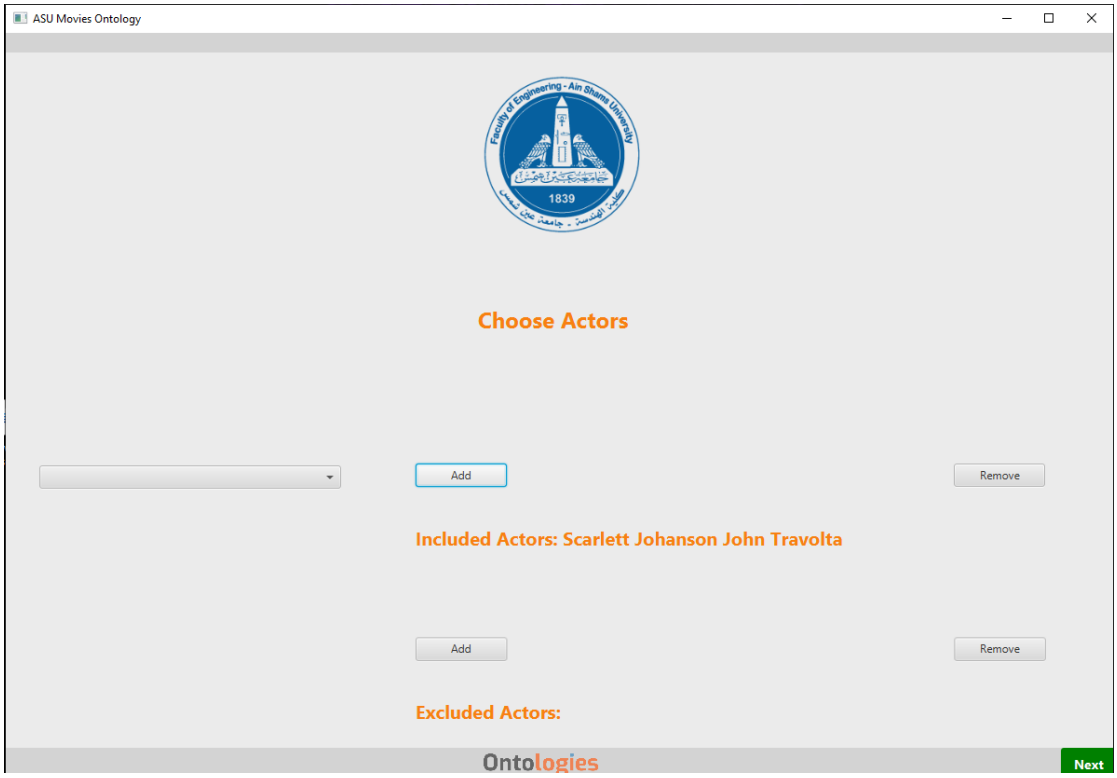
- Include “Scarlett Johansen”



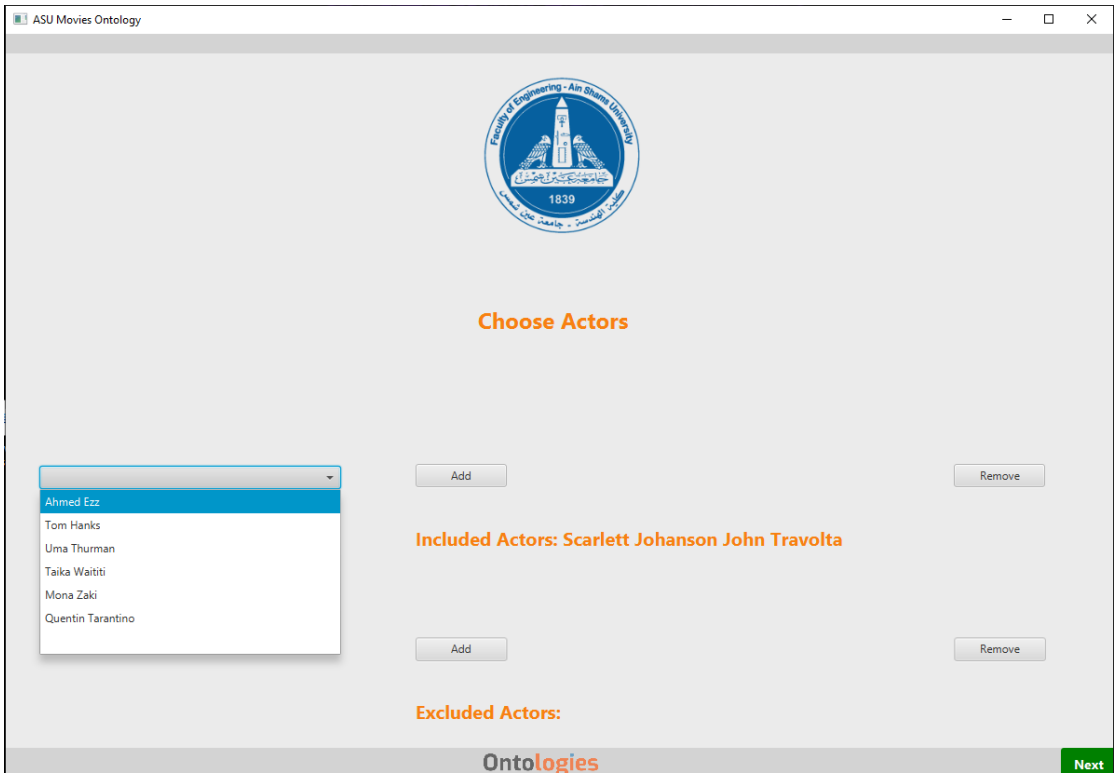


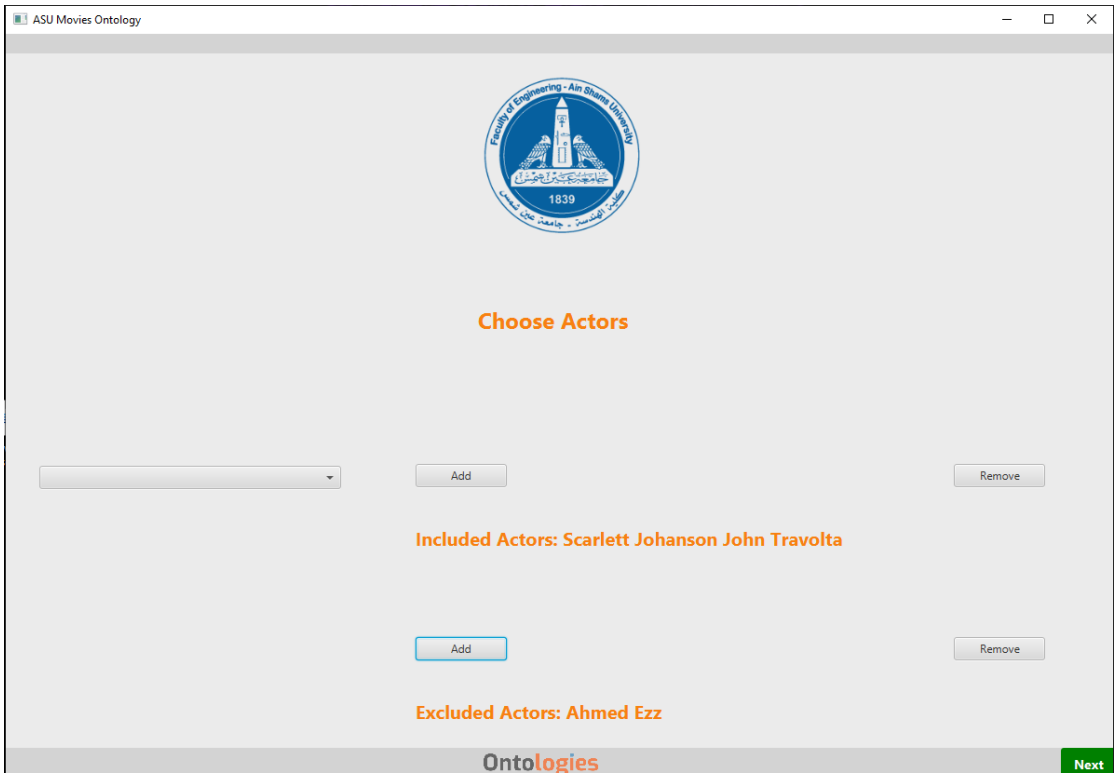
- Include “John Travolta”





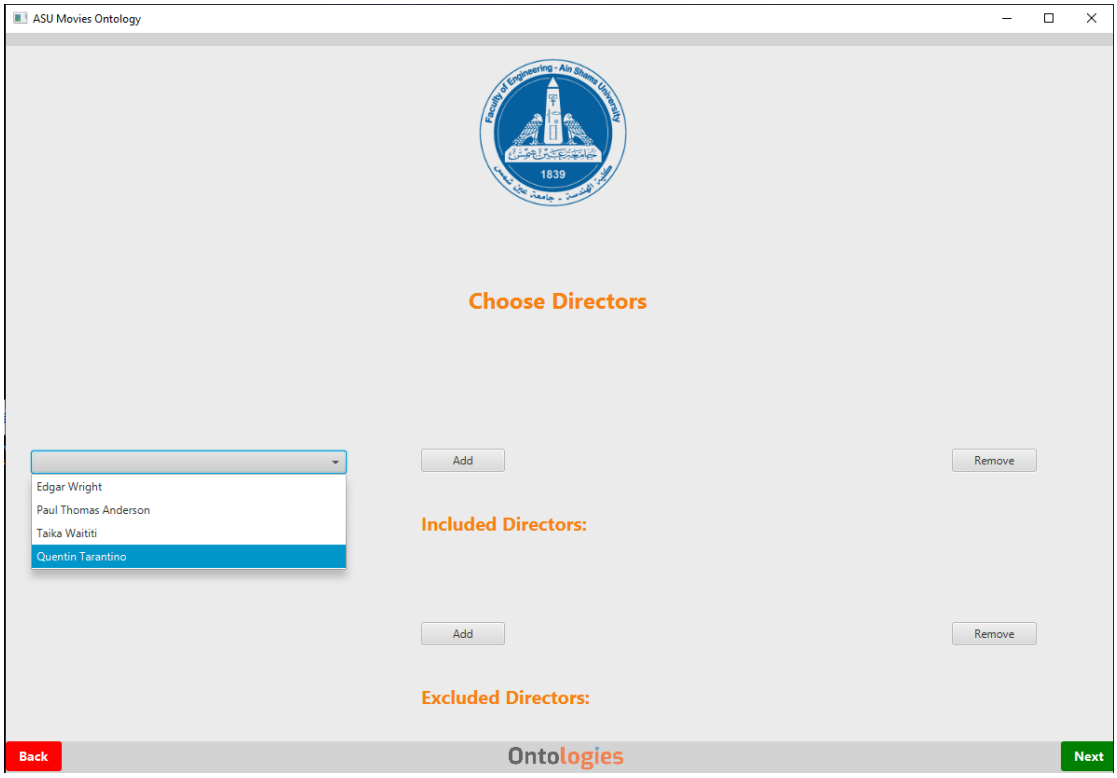
- Exclude “Ahmed Ezz”

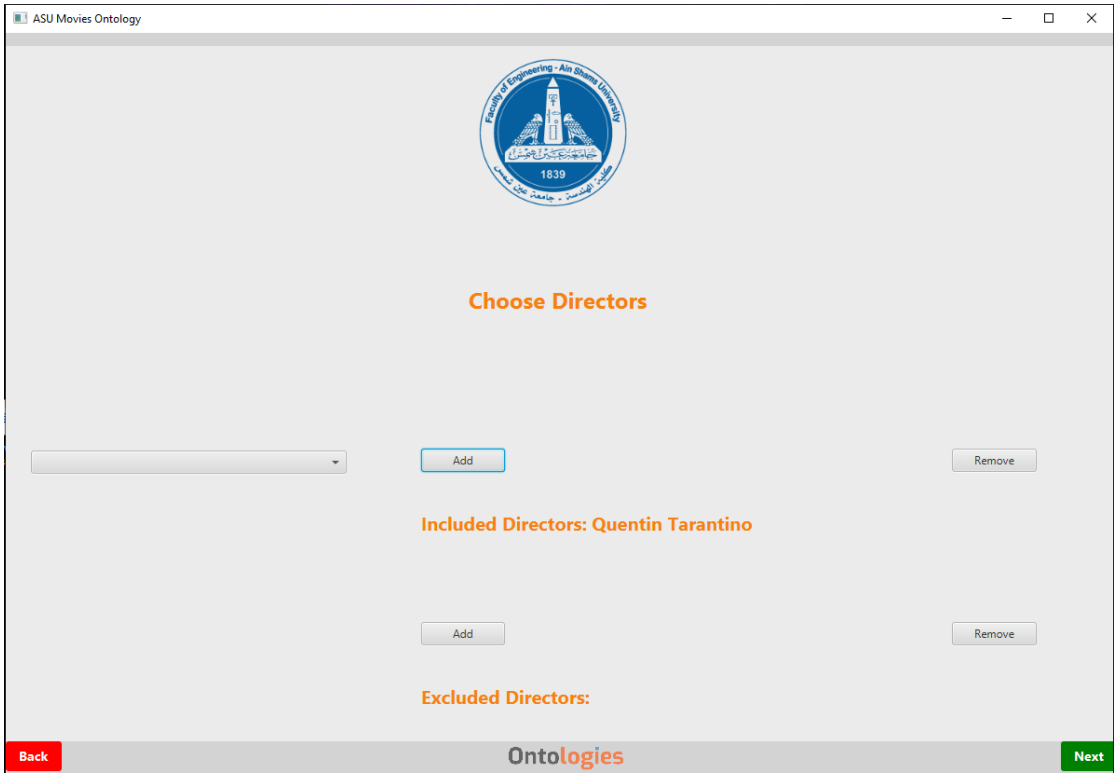




2nd: Press Next and include/exclude directors:

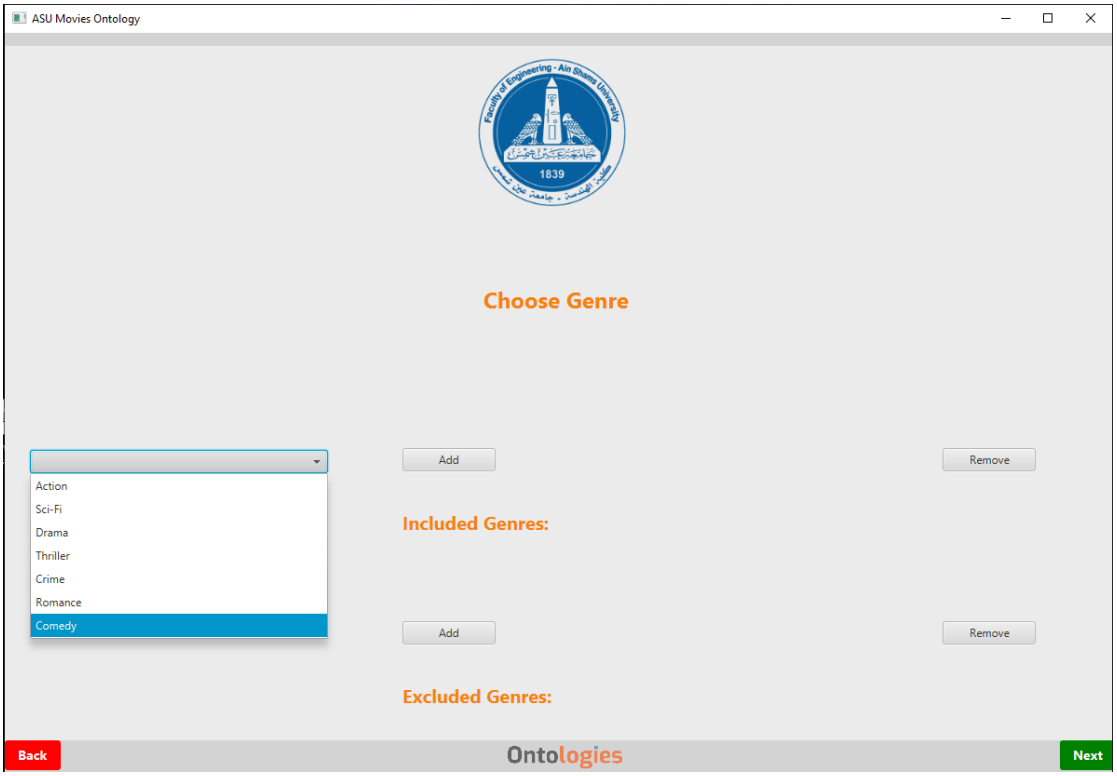
- Include “Quintin Tarentino”

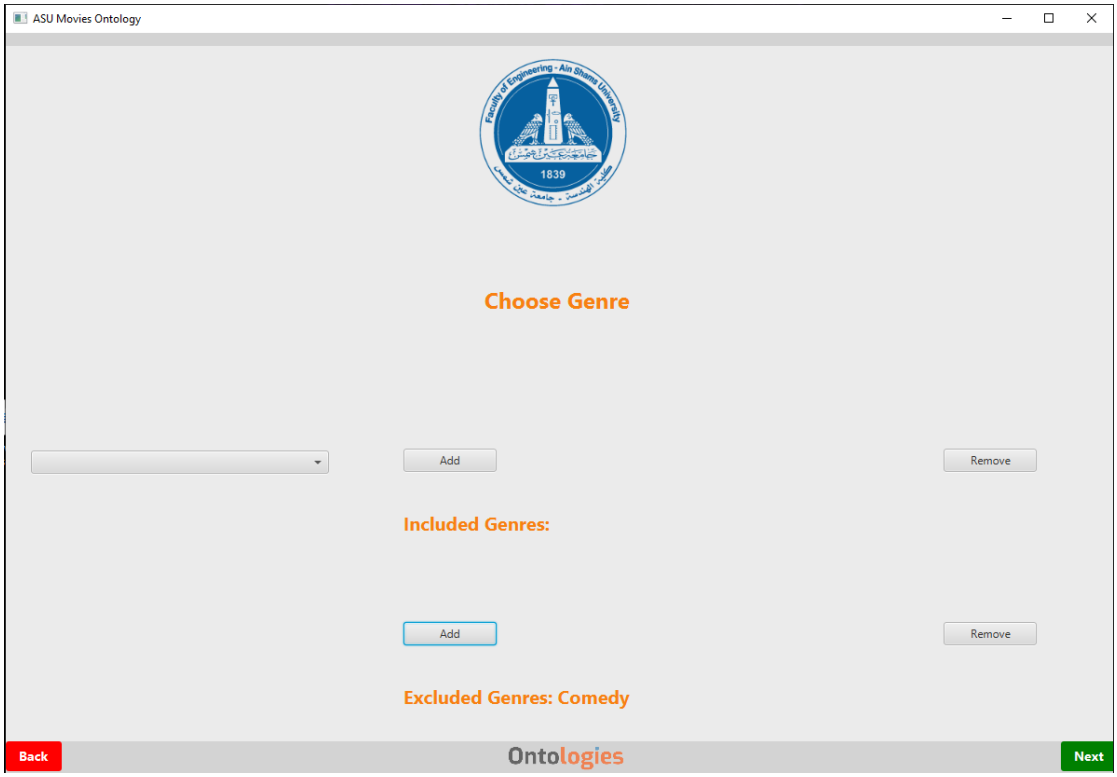




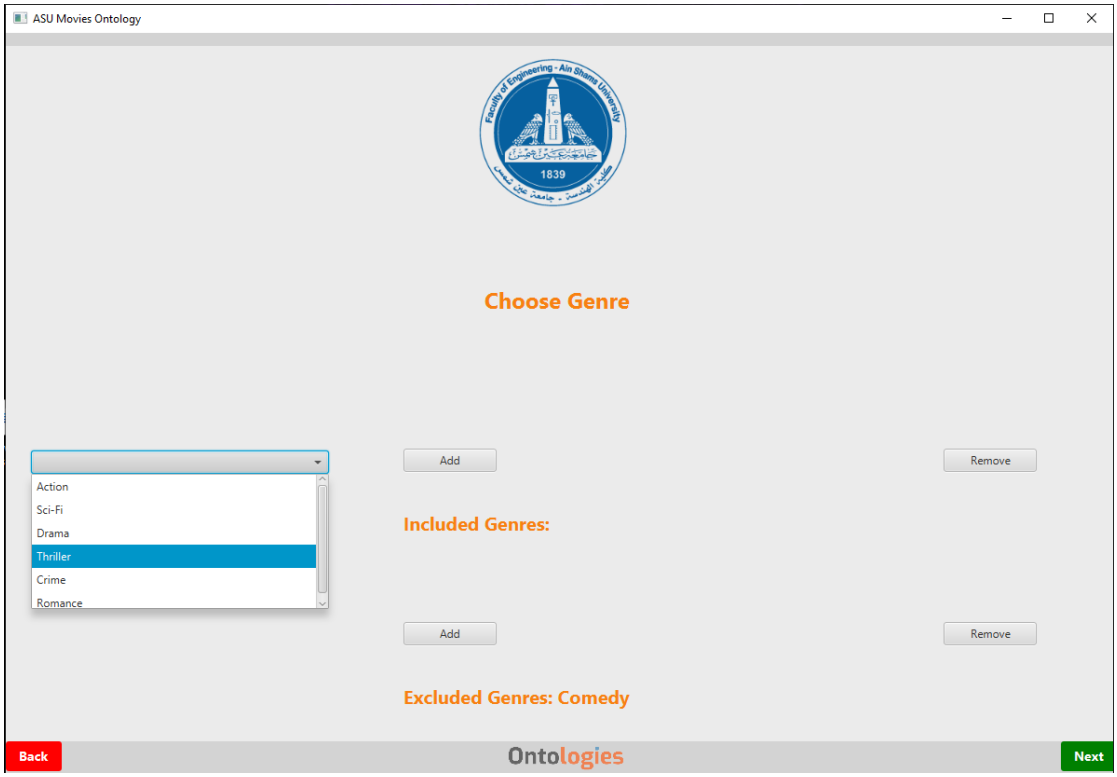
3rd: Press Next and include/exclude genres:

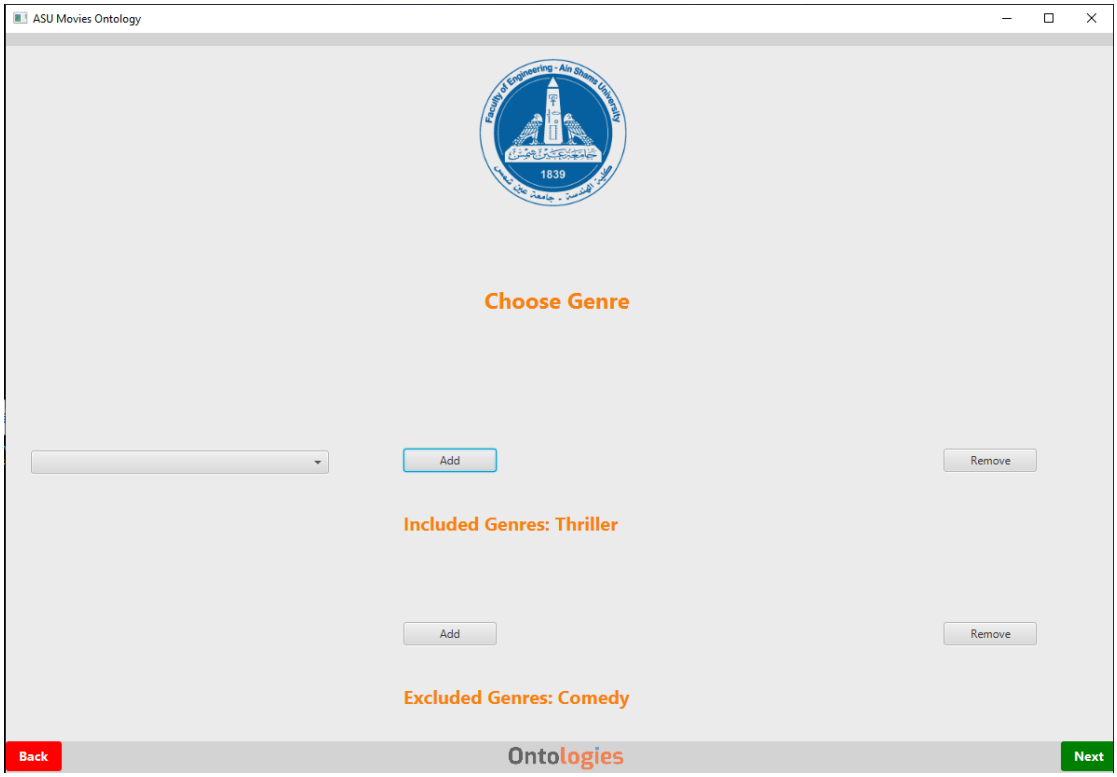
- Exclude “Comedy”





- Include “Thriller”





4th: Press Next and Search to display the query result:

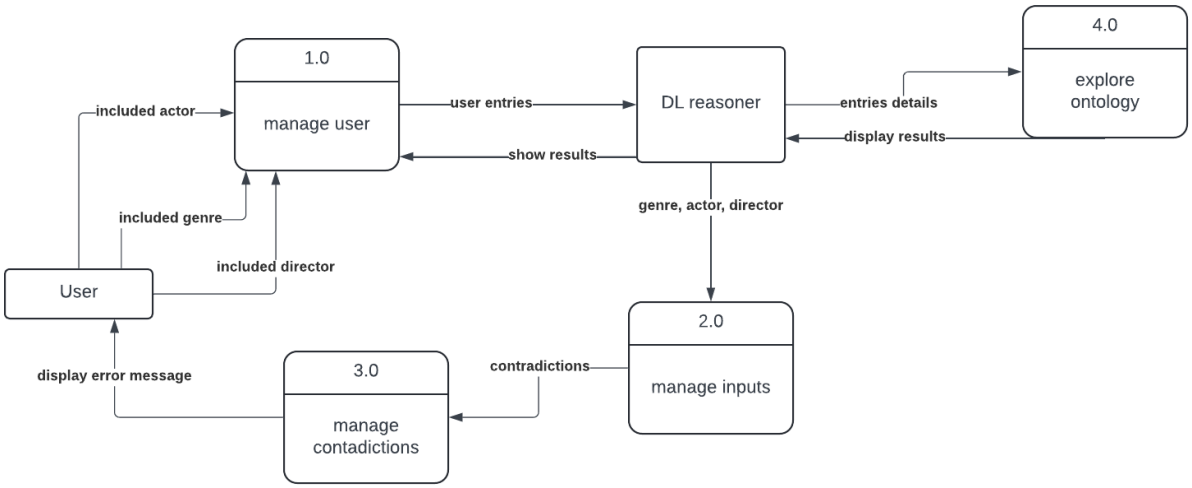
Bonus: What does the query string look like?

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX eg: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT DISTINCT ?movieTitle ?releaseYear
WHERE {
  ?movie rdf:type eg:Movies .
  ?movie eg:hasTitle ?movieTitle .
  ?movie eg:hasYear ?releaseYear .
  ?movie eg:hasActor ?actor .
  ?movie eg:hasDirector ?director .
  ?movie eg:hasGenre ?genre .
 FILTER( ?genre = "Thriller" )
 FILTER(!( ?genre = "Comedy" ))
  ?director eg:hasName ?directorName .
 FILTER( ?directorName = "Quentin Tarantino" )
  ?actor eg:hasName ?actorName .
 FILTER( ?actorName = "Scarlett Johanson" || ?actorName = "John Travolta" )
  ?actor eg:hasName ?actorName .
 FILTER(!( ?actorName = "Ahmed Ezz" ))
}
```

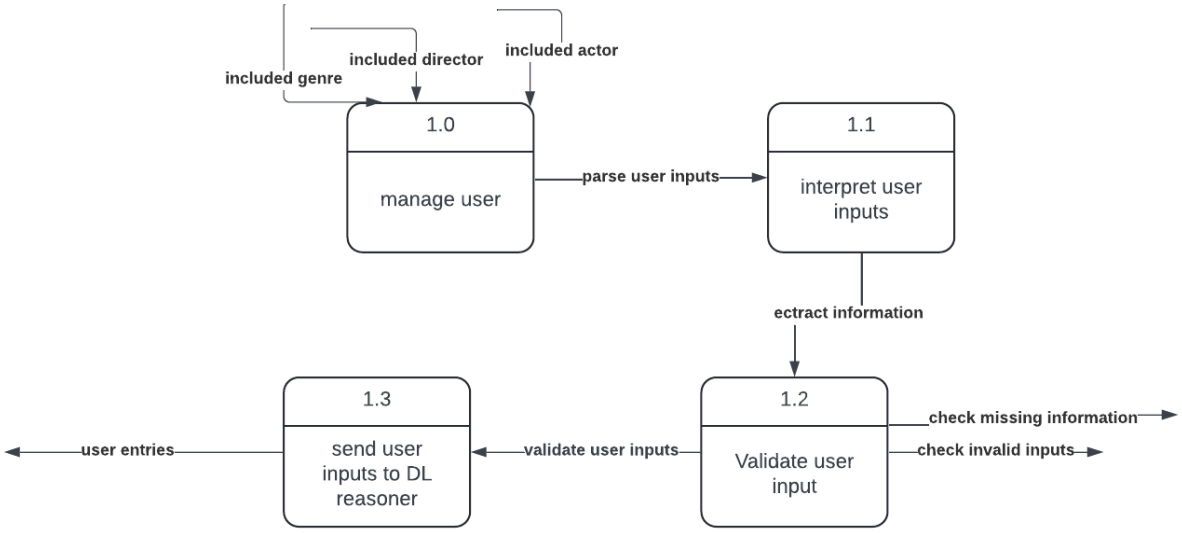
DFD (Data Flow Diagram):

A diagram illustrating the flow of data within the system, showing how information is input, processed, and output.

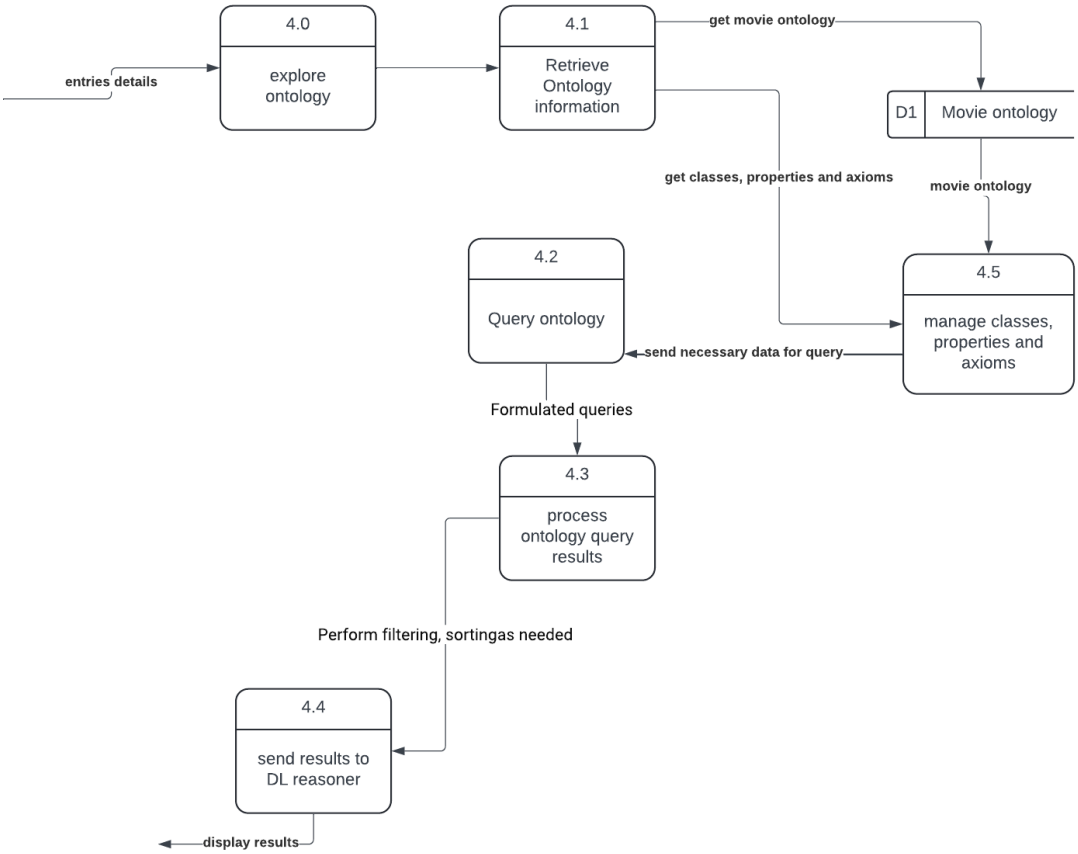
The whole system:



Manage user component:

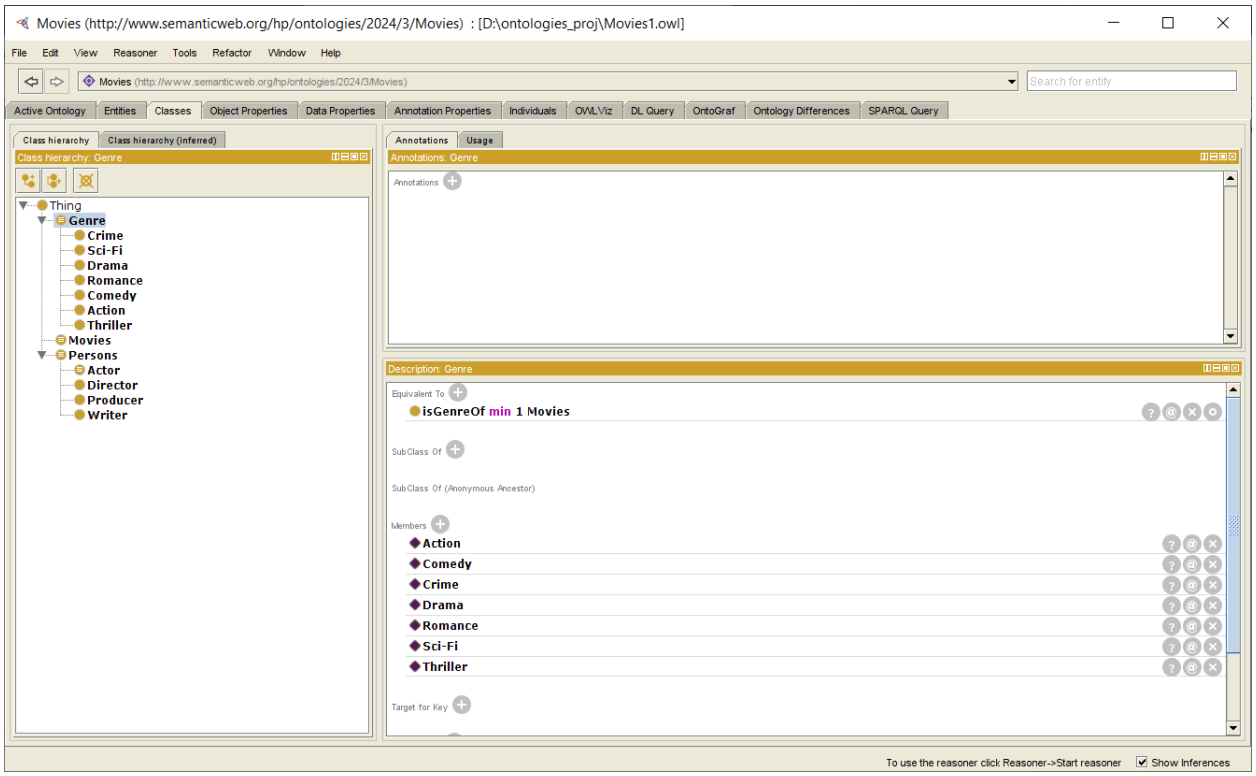


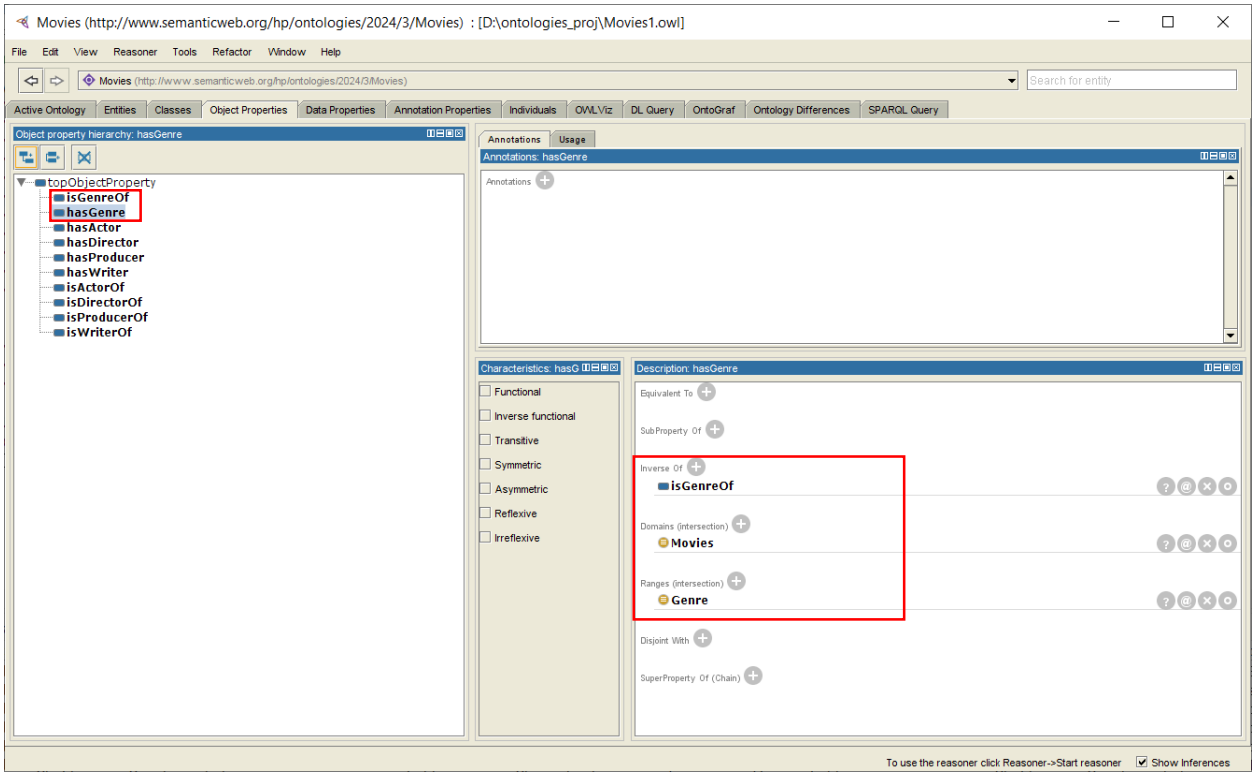
Explore ontology Component:

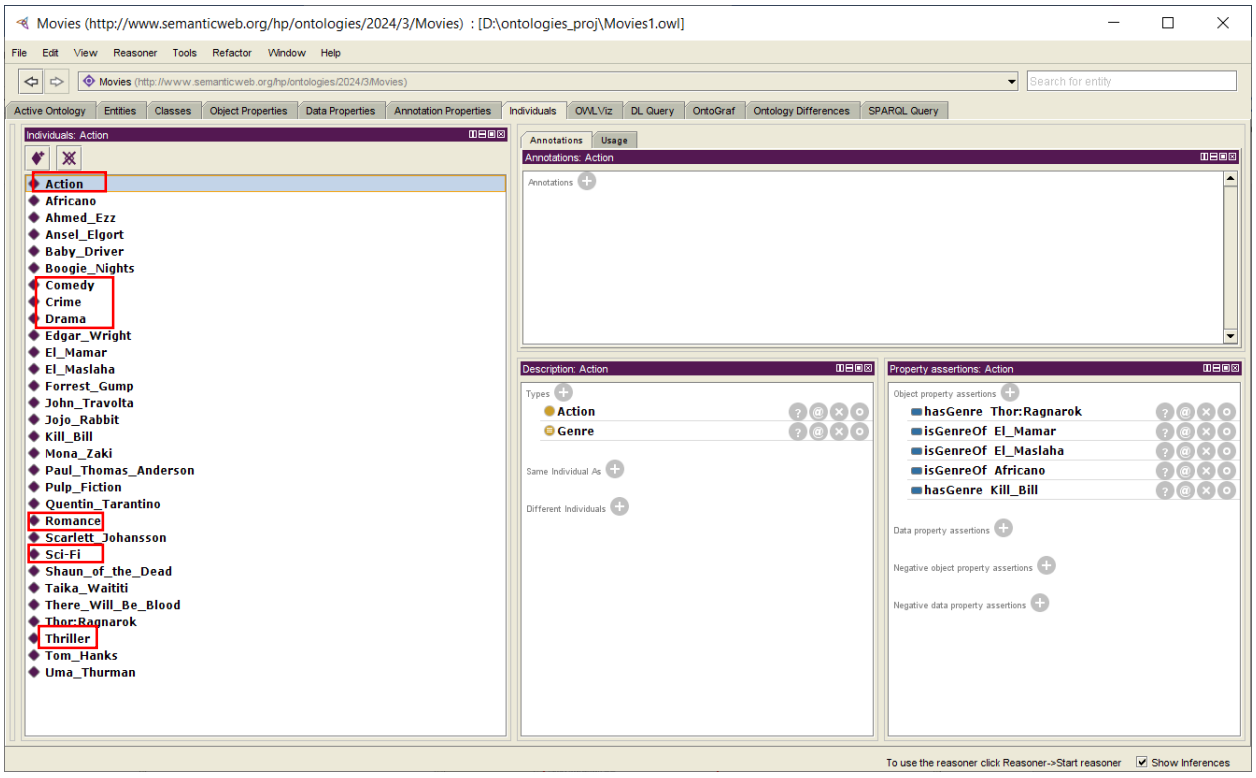


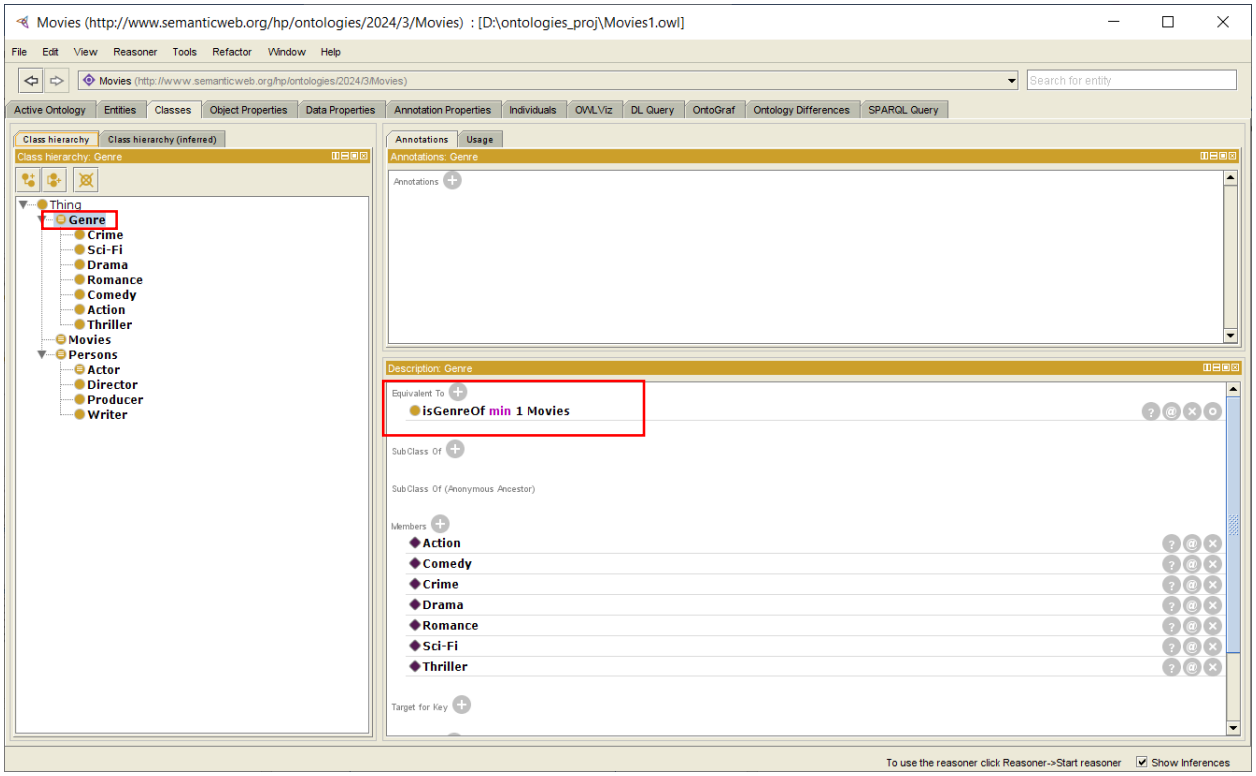
Additional Part:

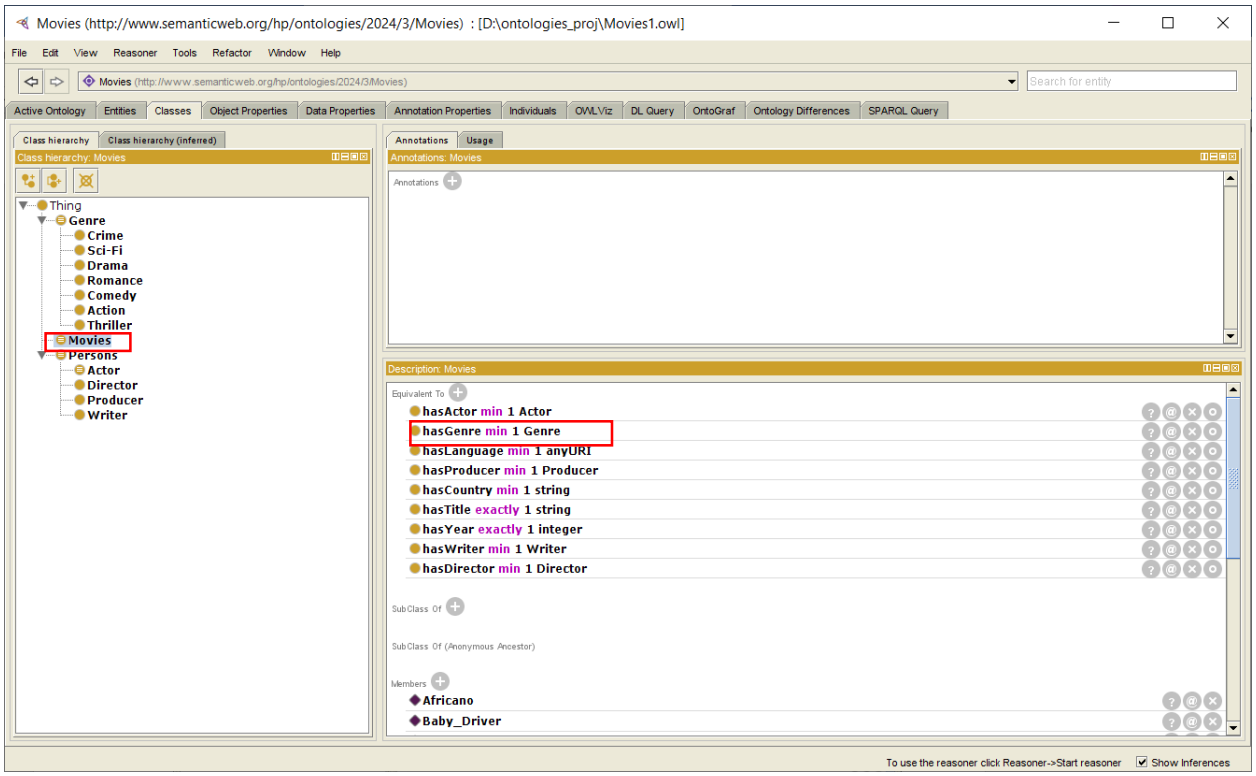
- Concerning this part we made an adjustment: added Genre Class to the ontology and Action, Crime, Comedy, Drama, Romance, Thriller and Sci-Fi as subclasses
- Made it disjoint with Persons and Movies
- Created individuals of: Action, Crime, Comedy, Drama, Romance, Thriller and Sci-Fi.
- Created object property hasGenre, isGenreOf to connect between Movies and Genre classes

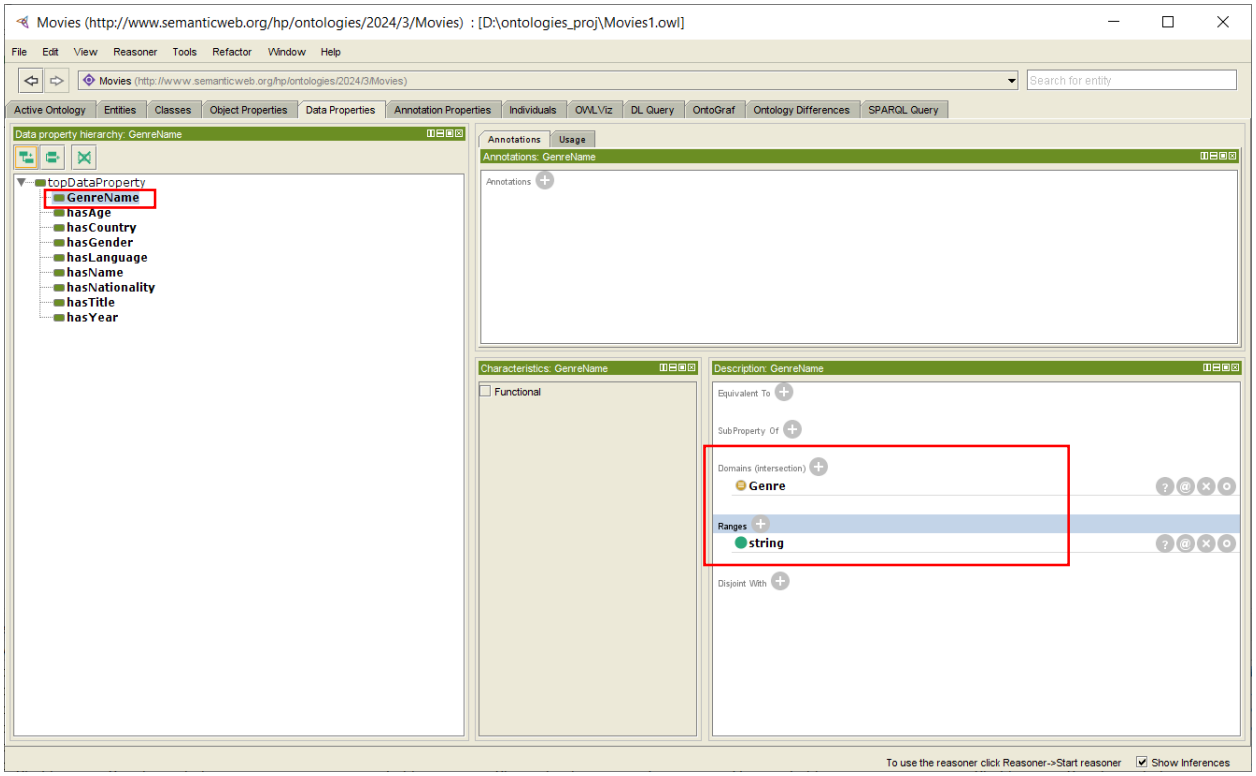


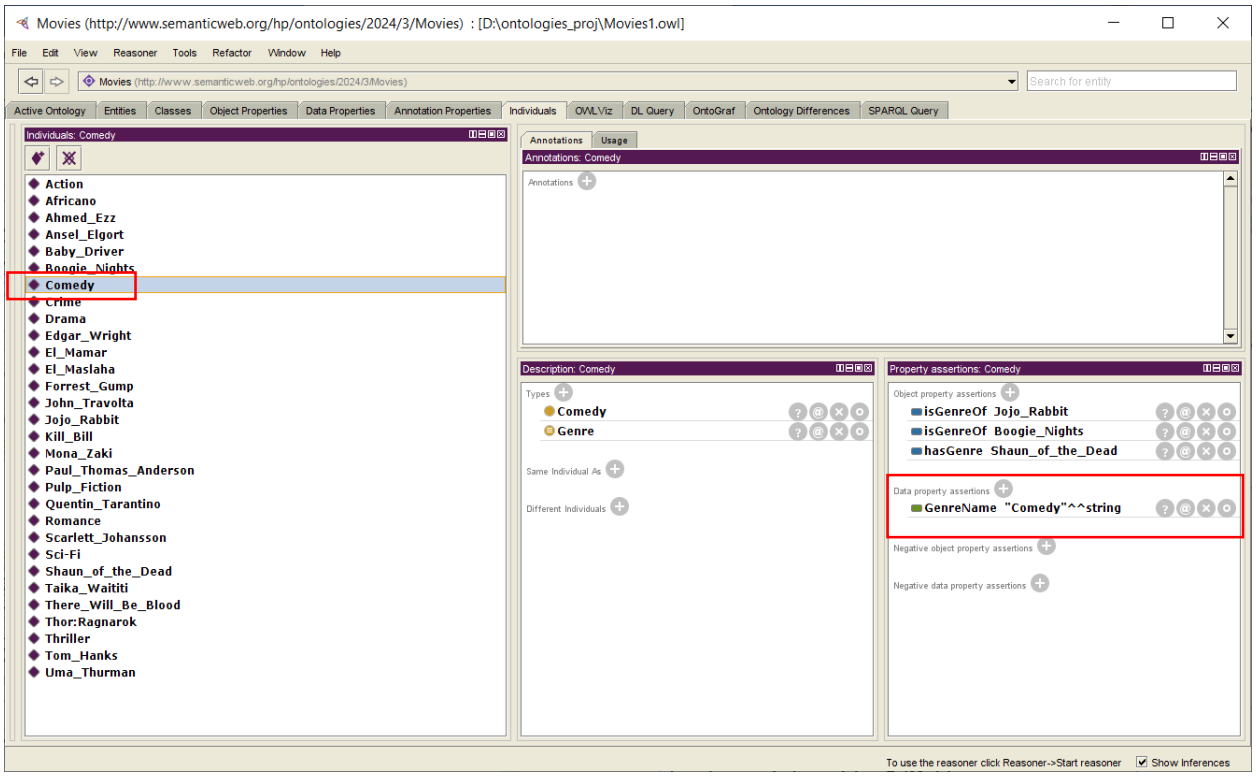

Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies1.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Object property hierarchy: hasGenre
topObjectProperty
isGenreOf
hasGenre
hasActor
hasDirector
hasProducer
hasWriter
isActorOf
isDirectorOf
isProducerOf
isWriterOf
Annotations Usage
Annotations: hasGenre
Characteristics: hasG
Functional
Inverse functional
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: hasGenre
Equivalent To
SubProperty Of
Inverse Of
isGenreOf
Domains (intersection)
Movies
Ranges (intersection)
Genre
Disjoint With
SuperProperty Of (Chain)
To use the reasoner click Reasoner->Start reasoner
Show Inferences


Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies1.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Individuals: Action
Action
Africano
Ahmed_Ezz
Ansel_Elgort
Baby_Driver
Boogie_Nights
Comedy
Crime
Drama
Edgar_Wright
El_Mamar
El_Maslaha
Forrest_Gump
John_Travolta
Jojo_Rabbit
Kill_Bill
Mona_Zaki
Paul_Thomas_Anderson
Pulp_Fiction
Quentin_Tarantino
Romance
Scarlett_Johansson
Sci-Fi
Shaun_of_the_Dead
Taika_Waititi
There_Will_Be_Blood
Thor:Ragnarok
Thriller
Tom_Hanks
Uma_Thurman
Annotations Usage
Annotations: Action
Description: Action
Types
Action
Genre
Same Individual As
Different Individuals
Property assertions: Action
Object property assertions
hasGenre Thor:Ragnarok
isGenreOf El_Mamar
isGenreOf El_Maslaha
isGenreOf Africano
hasGenre Kill_Bill
Data property assertions
Negative object property assertions
Negative data property assertions
To use the reasoner click Reasoner->Start reasoner
Show Inferences


Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies1.owl]
File Edit View Reasoner Tools Refactor Window Help
Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies)
Search for entity
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Genre
Thing
Genre
Crime
Sci-Fi
Drama
Romance
Comedy
Action
Thriller
Movies
Persons
Actor
Director
Producer
Writer
Annotations Usage
Annotations: Genre
Annotations
Description: Genre
Equivalent To
isGenreOf min 1 Movies
SubClass Of
SubClass Of (Anonymous Ancestor)
Members
Action
Comedy
Crime
Drama
Romance
Sci-Fi
Thriller
Target for Key
To use the reasoner click Reasoner->Start reasoner
Show Inferences


Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies1.owl]
File Edit View Reasoner Tools Refactor Window Help
Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies)
Search for entity
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Movies
Thing
Genre
Crime
Sci-Fi
Drama
Romance
Comedy
Action
Thriller
Movies
Persons
Actor
Director
Producer
Writer
Annotations Usage
Annotations: Movies
Annotations
Description: Movies
Equivalent To
hasActor min 1 Actor
hasGenre min 1 Genre
hasLanguage min 1 anyURI
hasProducer min 1 Producer
hasCountry min 1 string
hasTitle exactly 1 string
hasYear exactly 1 integer
hasWriter min 1 Writer
hasDirector min 1 Director
SubClass Of
SubClass Of (Anonymous Ancestor)
Members
Africano
Baby_Driver
To use the reasoner click Reasoner->Start reasoner
Show Inferences


Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies1.owl]
File Edit View Reasoner Tools Refactor Window Help
Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies)
Search for entity
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Data property hierarchy: GenreName
topDataProperty
GenreName
hasAge
hasCountry
hasGender
hasLanguage
hasName
hasNationality
hasTitle
hasYear
Annotations Usage
Annotations: GenreName
Annotations
Characteristics: GenreName
Functional
Description: GenreName
Equivalent To
SubProperty Of
Domains (intersection)
Genre
Ranges
string
Disjoint With
To use the reasoner click Reasoner->Start reasoner
Show Inferences


Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies) : [D:\ontologies_proj\Movies1.owl]
File Edit View Reasoner Tools Refactor Window Help
Movies (http://www.semanticweb.org/hp/ontologies/2024/3/Movies)
Search for entity
Active Ontology Entities Classes Object Properties Data Properties Annotation Properties Individuals OWLViz DL Query OntoGraf Ontology Differences SPARQL Query
Individuals: Comedy
Action
Africano
Ahmed_Ezz
Ansel_Elgort
Baby_Driver
Boogie_Nights
Comedy
Crime
Drama
Edgar_Wright
El_Mamar
El_Maslaha
Forrest_Gump
John_Travolta
Jojo_Rabbit
Kill_Bill
Mona_Zaki
Paul_Thomas_Anderson
Pulp_Fiction
Quentin_Tarantino
Romance
Scarlett_Johansson
Sci-Fi
Shaun_of_the_Dead
Taika_Waititi
There_Will_Be_Blood
Thor:Ragnarok
Thriller
Tom_Hanks
Uma_Thurman
Annotations Usage
Annotations: Comedy
Annotations
Description: Comedy
Types
Comedy
Genre
Same Individual As
Different Individuals
Property assertions: Comedy
Object property assertions
isGenreOf Jojo_Rabbit
isGenreOf Boogie_Nights
hasGenre Shaun_of_the_Dead
Data property assertions
GenreName "Comedy"^^string
Negative object property assertions
Negative data property assertions
To use the reasoner click Reasoner->Start reasoner
Show Inferences

- All rul1.txt:

```
@prefix ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
[ruleHasTwoRoles: (?s ont:hasActor ?c) (?b ont:hasDirector ?c) -> (?c
ont:ActorAndWriter ?s) (?c ont:ActorAndDirector ?b)]
[ruleActorAndWriter: (?s ont:hasActor ?c) (?b ont:hasWriter ?c) -> (?c
ont:ActorAndWriter  ?s) (?c ont:ActorAndWriter ?b)]
[ruleGenreClassification: (?movie ont:hasGenre ?genre), (?genre rdf:type
ont:Thriller) -> (?movie rdf:type ont:ThrillerMovie)]
[ruleGenreClassification1: (?movie ont:hasGenre ?genre), (?genre rdf:type
ont:Romance) -> (?movie rdf:type ont:RomanceMovie)]
```

- Jena6_rules afte modifications:

```
import org.apache.jena.ontology.OntModel;
import org.apache.jena.ontology.OntModelSpec;
import org.apache.jena.rdf.model.*;
import org.apache.jena.reasoner.*;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
```

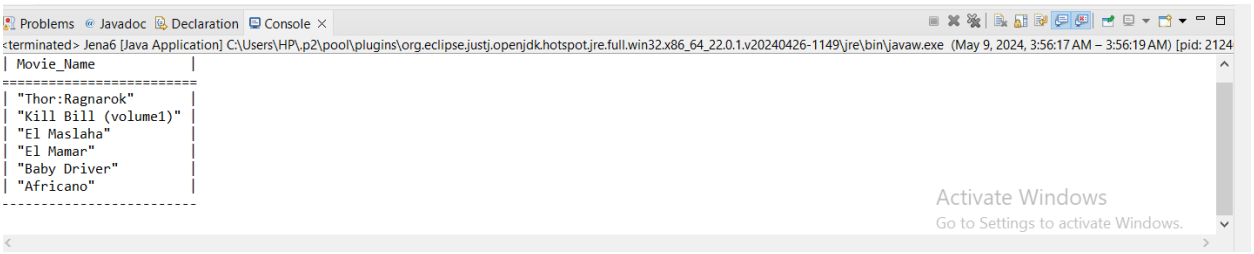
```
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.rdf.model.Property;
import org.apache.jena.rdf.model.RDFNode;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.rdf.model.Statement;
import org.apache.jena.rdf.model.StmtIterator;
import org.apache.jena.reasoner.Reasoner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

```
import java.util.Iterator;

public class Jena6_rules {

        public static void main(String[] args)

  {

    Model model = ModelFactory.createDefaultModel();

    model.read( "data/Movies1.owl" );

    Reasoner reasoner = new GenericRuleReasoner( Rule.rulesFromURL( "data/rul1.txt" ) );

    InfModel infModel = ModelFactory.createInfModel( reasoner, model );

    StmtIterator it = infModel.listStatements();

    while ( it.hasNext() )

    {

      Statement stmt = it.nextStatement();

      Resource subject = stmt.getSubject();

      Property predicate = stmt.getPredicate();

      RDFNode object = stmt.getObject();

      System.out.println( subject.toString() + " " + predicate.toString() + " " + object.toString()
);

    }

  }

}
```

- Output:

a. Movie with specific genre

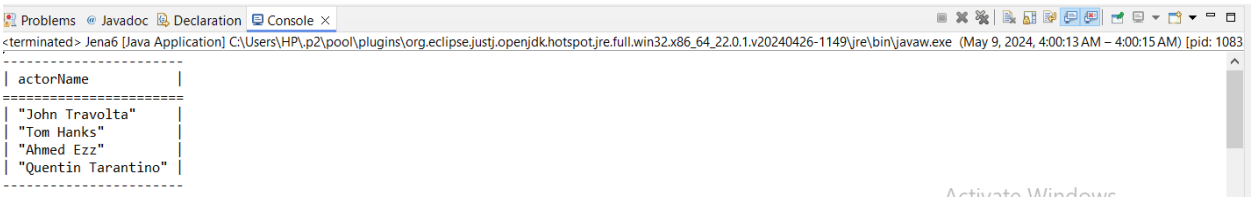
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?Movie_Name
WHERE {
?movie rdf:type ont:Movies.
?movie ont:hasTitle ?Movie_Name.
?movie ont:hasGenre ?genre.
FILTER(?genre = "Action")
}
```



b. Actors with age bigger than 51

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
SELECT ?actorName
```

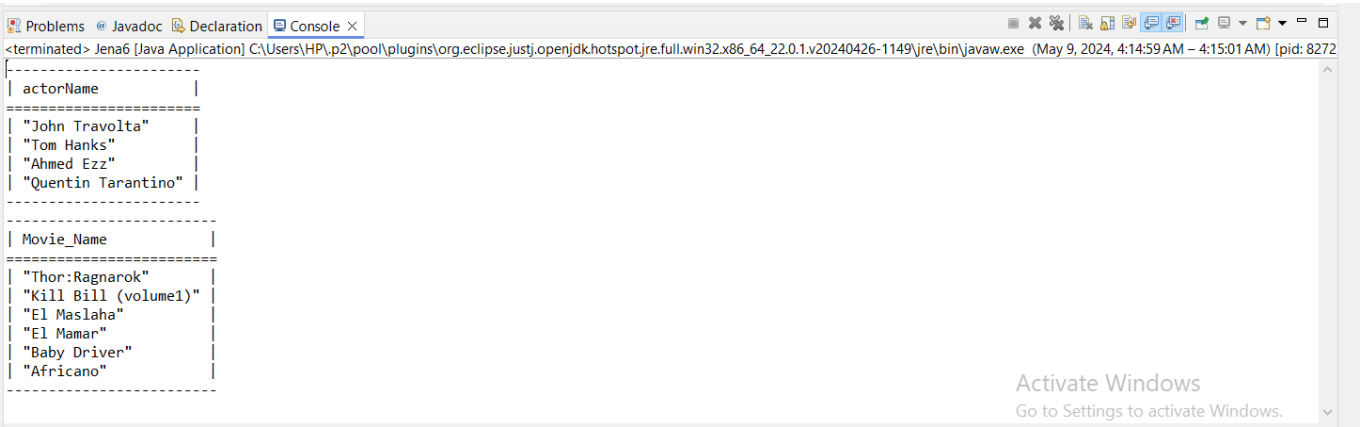
```
WHERE {


?actor rdf:type ont:Actor.

?actor ont:hasAge ?age.

?actor ont:hasName ?actorName.

FILTER(?age > 51)

}
```



- Jena6 code:

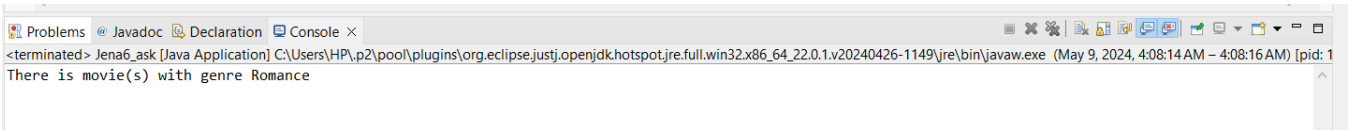
```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
public class Jena6 {
   public static void main(String[] args) {
      // Load the ontology
      Model model = ModelFactory.createDefaultModel();
      FileManager fileManager = FileManager.get();
      String owlFile = "data/Movies.owl"; // Replace with the actual filename
      model.read(fileManager.open(owlFile), null);

      // Execute and display results for Rule 1
      executeRule(model, "data/age_rule.txt");

      // Execute and display results for Rule 2
      executeRule(model, "data/genre_rule.txt");
   }
```

```
    private static void executeRule(Model model, String ruleFile) {
        // Read the SPARQL query from file
        String queryString = readQueryFromFile(ruleFile);

        // Execute the query
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            ResultSet results = qexec.execSelect();

            // Print the results
            ResultSetFormatter.out(results);
        }
    }

    // Helper method to read the query from file
    private static String readQueryFromFile(String queryFile) {
        StringBuilder sb = new StringBuilder();
        try (InputStream inputStream = FileManager.get().open(queryFile)) {
            if (inputStream != null) {
                String line;
                BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
"UTF-8"));
                while ((line = reader.readLine()) != null) {
                    sb.append(line).append("\n");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return sb.toString();
    }
}
```



c. ASK query of specific genre (Romance)

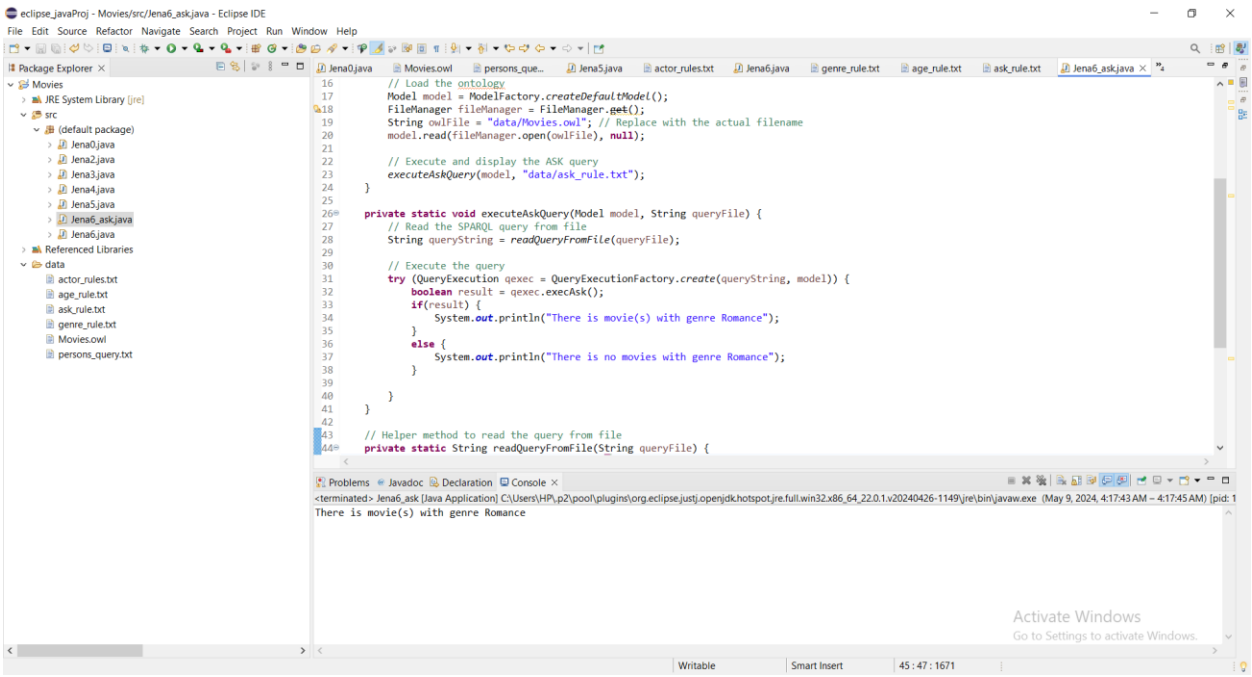
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ont: <http://www.semanticweb.org/hp/ontologies/2024/3/Movies#>
ASK
WHERE {
  ?movie rdf:type ont:Movies.
  ?movie ont:hasTitle ?title.
  ?movie ont:hasGenre ?genre.
  FILTER(?genre = "Romance") }
```



- Jena6_ask code:

```
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
public class Jena6_ask {
   public static void main(String[] args) {
      // Load the ontology
      Model model = ModelFactory.createDefaultModel();
      FileManager fileManager = FileManager.get();
      String owlFile = "data/Movies.owl"; // Replace with the actual filename
      model.read(fileManager.open(owlFile), null);
      // Execute and display the ASK query
      executeAskQuery(model, "data/ask_rule.txt");
```

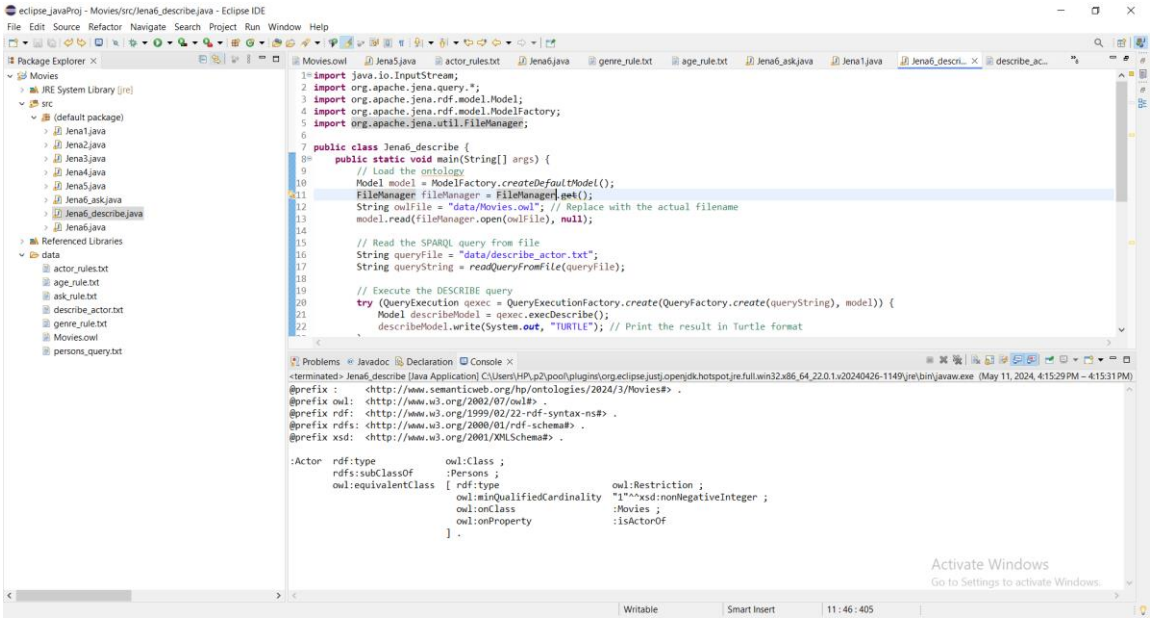
```
    private static void executeAskQuery(Model model, String queryFile) {
        // Read the SPARQL query from file
        String queryString = readQueryFromFile(queryFile);

        // Execute the query
        try (QueryExecution qexec = QueryExecutionFactory.create(queryString, model)) {
            boolean result = qexec.execAsk();
            if(result) {
              System.out.println("There is movie(s) with genre Romance");
            }
            else {
              System.out.println("There is no movies with genre Romance");
            }

        }
    }

    // Helper method to read the query from file
    private static String readQueryFromFile(String queryFile) {
        StringBuilder sb = new StringBuilder();
        try (InputStream inputStream = FileManager.get().open(queryFile)) {
            if (inputStream != null) {
                String line;
                BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
"UTF-8"));
                while ((line = reader.readLine()) != null) {
                    sb.append(line).append("\n");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return sb.toString();
    }
}
```



- Jena6_describe code:

```
import java.io.InputStream;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.util.FileManager;


public class Jena6_describe {
   public static void main(String[] args) {
      // Load the ontology
Model model = ModelFactory.createDefaultModel();
      FileManager fileManager = FileManager.get();
      String owlFile = "data/Movies.owl"; // Replace with the actual filename
      model.read(fileManager.open(owlFile), null);
```

```
    // Read the SPARQL query from file
    String queryFile = "data/describe_actor.txt";
    String queryString = readQueryFromFile(queryFile);
    // Execute the DESCRIBE query
    try (QueryExecution qexec = QueryExecutionFactory.create(QueryFactory.create(queryString),
model)) {
      Model describeModel = qexec.execDescribe();
      describeModel.write(System.out, "TURTLE"); // Print the result in Turtle format
    }
  }
  // Helper method to read the query from file
  private static String readQueryFromFile(String queryFile) {
    StringBuilder sb = new StringBuilder();
    try (InputStream inputStream = FileManager.get().open(queryFile)) {
      if (inputStream != null) {
        String line;
        java.io.BufferedReader reader = new java.io.BufferedReader(
            new java.io.InputStreamReader(inputStream, "UTF-8"));
        while ((line = reader.readLine()) != null) {
          sb.append(line).append("\n");
        }
      }
    } catch (java.io.IOException e) {
      e.printStackTrace();
    }
    return sb.toString();
  }
}
```



- describe_actor.txt:

```
DESCRIBE http://www.semanticweb.org/hp/ontologies/2024/3/Movies#Actor
```