Movies ontologies

CSE488- Ontologies and the Semantic Web

Team 15 members:

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

Table of Contents

[The Tree Hierarchy 3](#_Toc166536521)

[Object and Data Properties: 5](#_Toc166536522)

[Populating the Ontology: 6](#_Toc166536523)

[Querying the Ontology: 7](#_Toc166536524)

[More SPARQL Queries: 17](#_Toc166536525)

[Manipulating The Ontology in Python 21](#_Toc166536526)

[Ontology1 21](#_Toc166536527)

[Code 21](#_Toc166536528)

[output: 22](#_Toc166536529)

[Ontology2 22](#_Toc166536530)

[Code 22](#_Toc166536531)

[Output 24](#_Toc166536532)

[Ontology3 24](#_Toc166536533)

[Code 24](#_Toc166536534)

[Output 25](#_Toc166536535)

[Ontology4 25](#_Toc166536536)

[Code 25](#_Toc166536537)

[Output 27](#_Toc166536538)

[Ontology5 27](#_Toc166536539)

[Code 27](#_Toc166536540)

[Output 29](#_Toc166536541)

[Ontology6 29](#_Toc166536542)

[Code 29](#_Toc166536543)

[Output 31](#_Toc166536544)

[Application 32](#_Toc166536545)

[Code 32](#_Toc166536546)

[Output 36](#_Toc166536547)

[RDF GRAPH USING PYTHON 37](#_Toc166536548)

[DFD 38](#_Toc166536549)

# The Tree Hierarchy

Classes: Person, Movie, and Genre.

Subclasses: Actor, Director, Writer

A screenshot of a computer

Description automatically generatedOntoGraf:

A diagram of a network

Description automatically generated

# Object and Data Properties:

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

A screenshot of a computer

Description automatically generatedA screenshot of a phone

Description automatically generated

# Populating the Ontology:

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

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Querying the Ontology:

SPARQL query is used for these examples:

1. List the instances of the class Actor

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

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

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

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

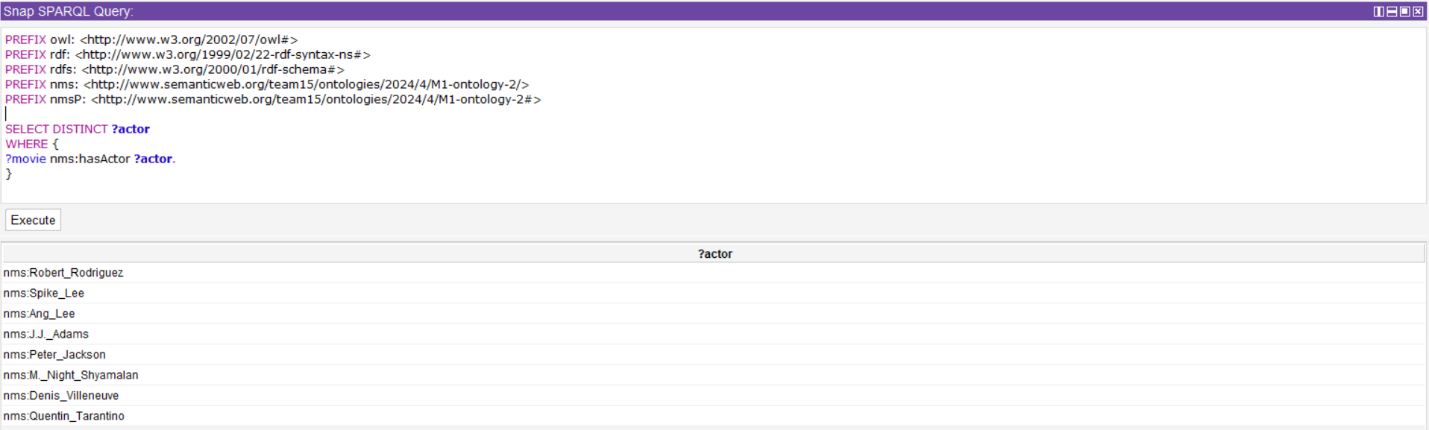
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT DISTINCT **?actor**

WHERE {

?movie nms:hasActor **?actor**.

}



1. List the instances of the class writer

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

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

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

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

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

SELECT DISTINCT **?writer**

WHERE {

**?writer** rdf:type nms:Person.

?movie nms:hasWriter **?writer**.

}

A screenshot of a computer

Description automatically generated

1. List the instances of the class director

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

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

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

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

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

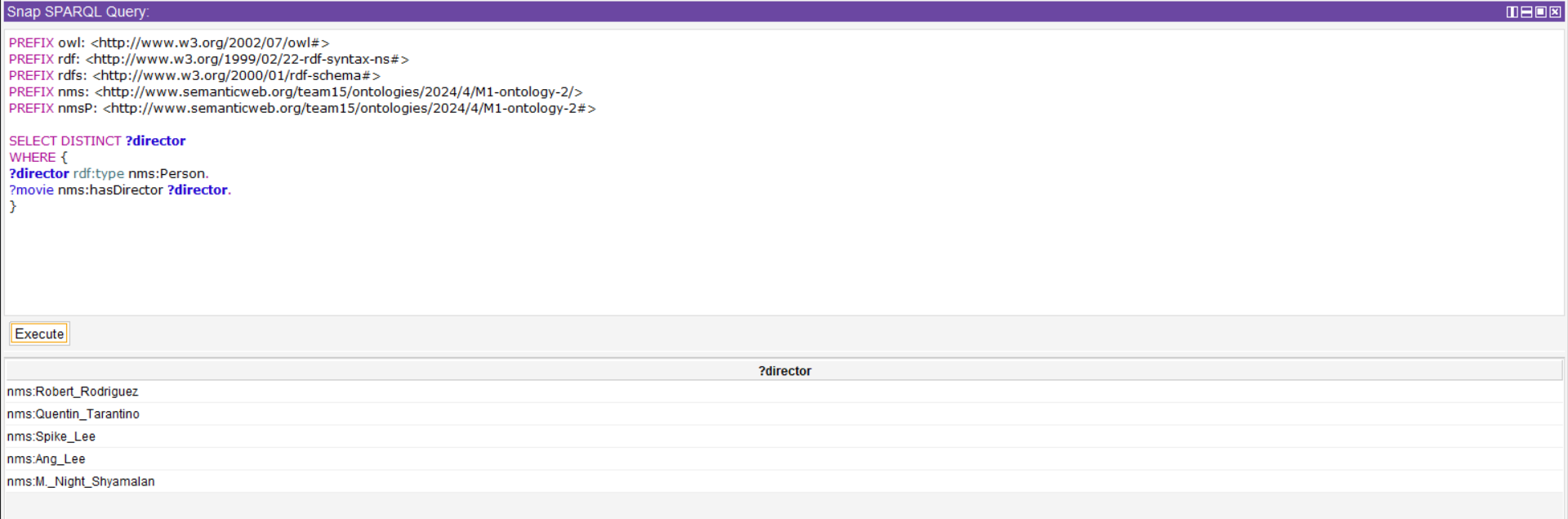
SELECT DISTINCT **?director**

WHERE {

**?director** rdf:type nms:Person.

?movie nms:hasDirector **?director**.

}



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

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

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

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

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

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

SELECT **?movie** **?director**

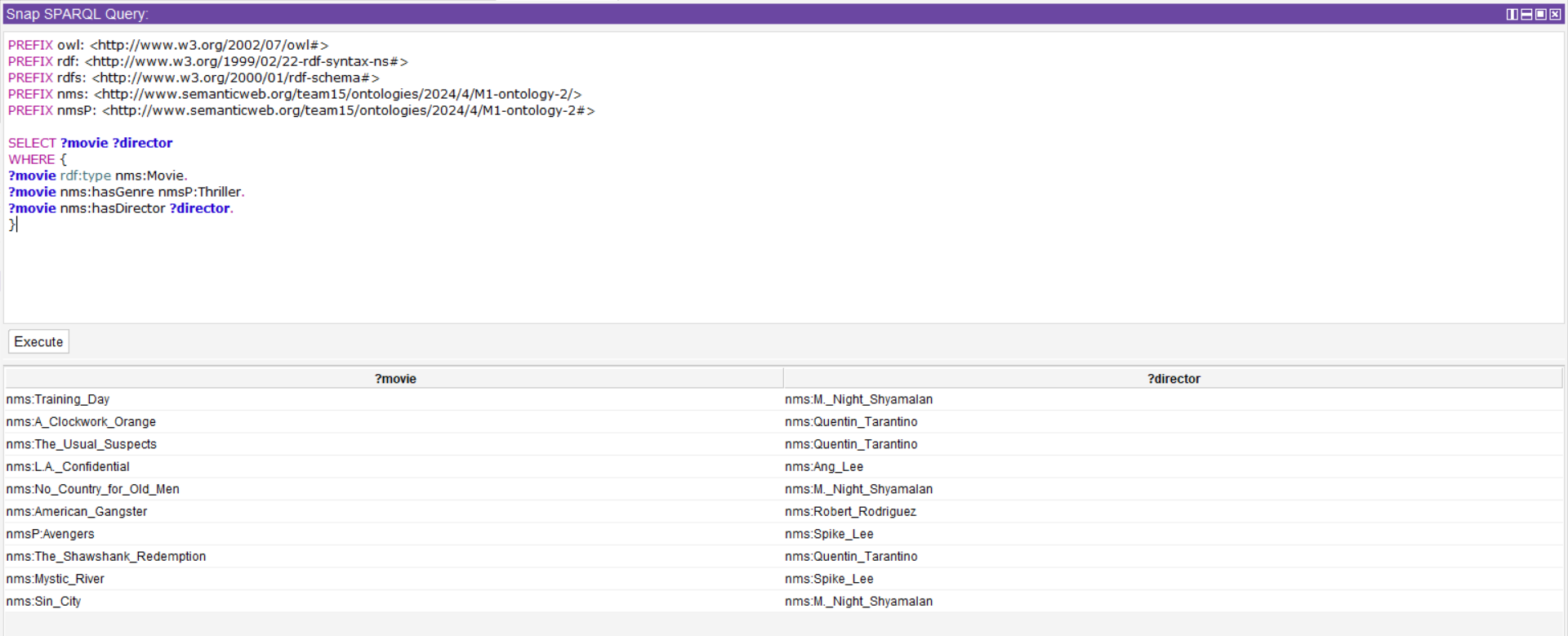
WHERE {

**?movie** rdf:type nms:Movie.

**?movie** nms:hasGenre nmsP:Thriller.

**?movie** nms:hasDirector **?director**.

}



1. List the name of all Crime Thriller movies.

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

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

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

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

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

SELECT **?movie**

WHERE {

**?movie** rdf:type nms:Movie.

**?movie** nms:hasGenre nmsP:Thriller.

**?movie** nms:hasGenre nmsP:Crime.

}

A screenshot of a computer

Description automatically generated

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

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

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

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

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

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

SELECT **?actor**

WHERE {

?movie rdf:type nms:Movie.

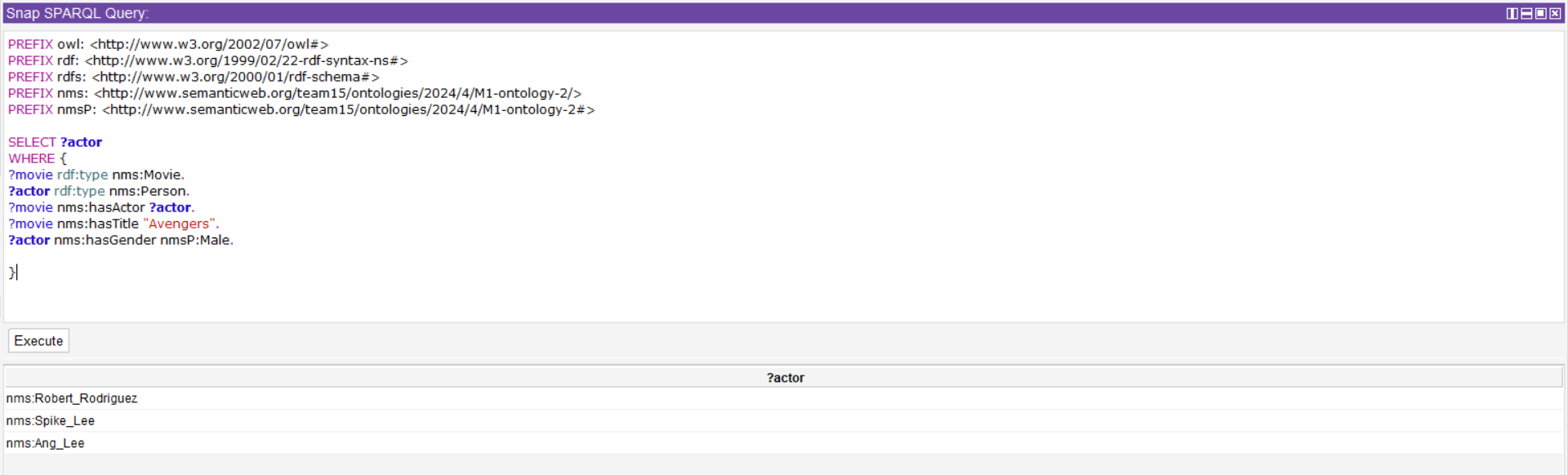
**?actor** rdf:type nms:Person.

?movie nms:hasActor **?actor**.

?movie nms:hasTitle "Avengers".

**?actor** nms:hasGender nmsP:Male.

}



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

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

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

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

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

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

SELECT (COUNT(?movie) AS **?totalMovies**)

WHERE {

?movie rdf:type nms:Movie.

?movie nms:hasGenre nmsP:Thriller.

?movie nms:hasGenre nmsP:Action.

}

A screenshot of a computer

Description automatically generated

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

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

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

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

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

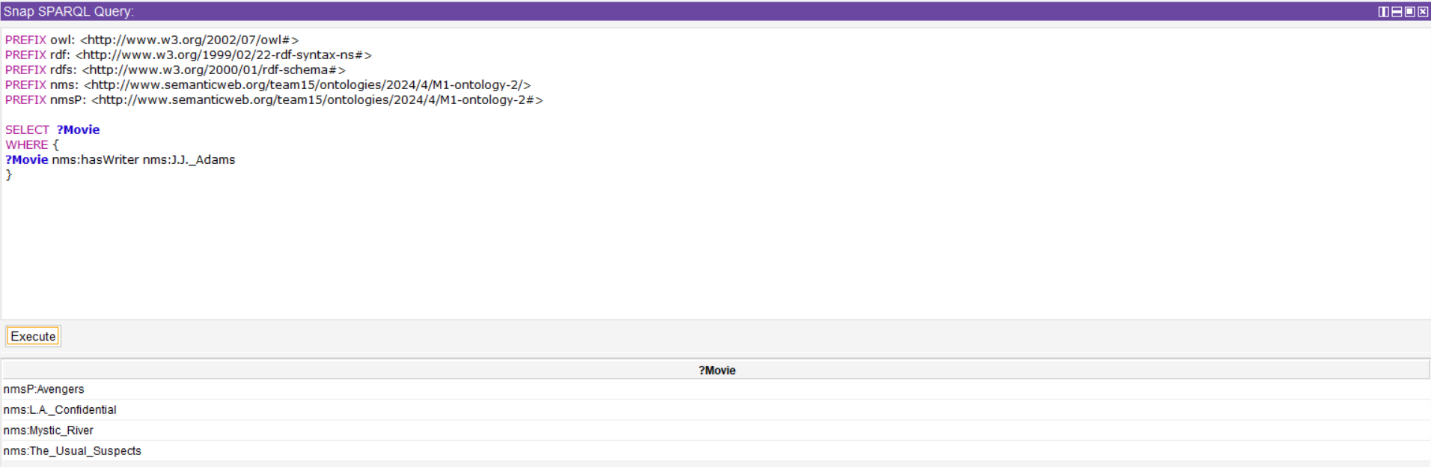
PREFIX nmsP: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#>

SELECT **?Movie**

WHERE {

**?Movie** nms:hasWriter nms:J.J.\_Adams

}



1. Find movies with a certain language.

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

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

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

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

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

SELECT **?AllMovies**

WHERE {

?Movies nms:hasTitle **?AllMovies**.

?Movies nms:hasLanguage "en".

}

A screenshot of a computer

Description automatically generated

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

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

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

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

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

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

SELECT **?actor**

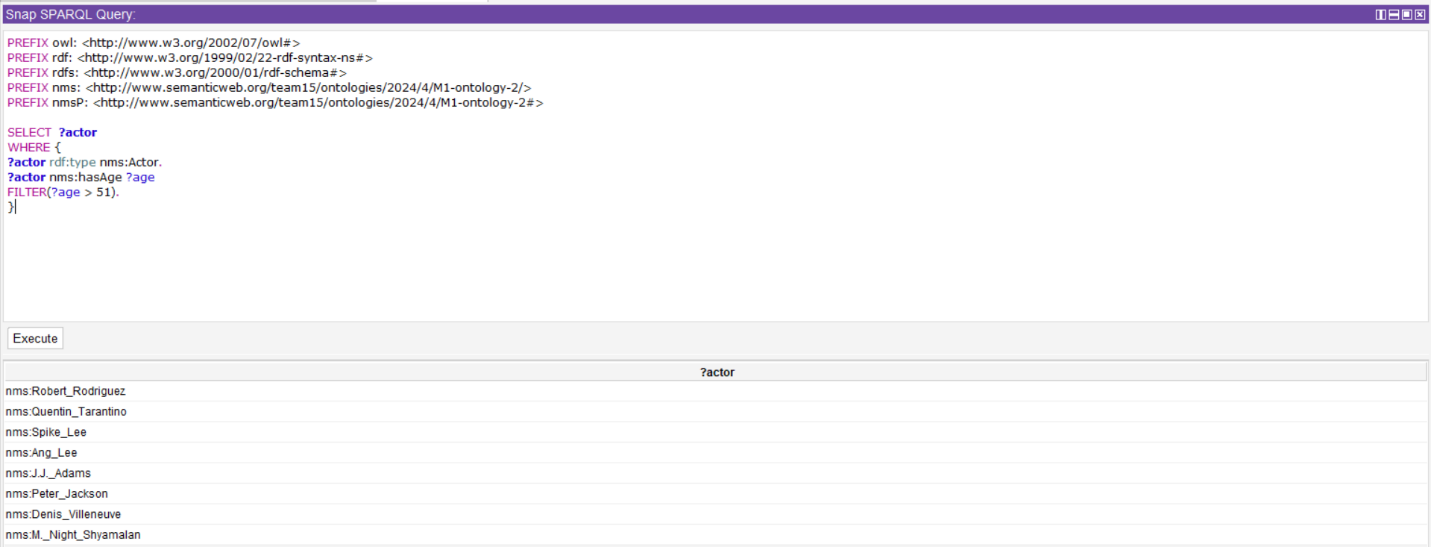
WHERE {

**?actor** rdf:type nms:Actor.

**?actor** nms:hasAge ?age

FILTER(?age > 51).

}



# More SPARQL Queries:

1. Output all the thriller movies, and if one of those thrillers is Action then also print its director.

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

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

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

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

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

SELECT DISTINCT **?movieLabel** **?director**

WHERE {

?movie nms:hasTitle **?movieLabel**.

?movie nms:hasGenre nmsP:Thriller.

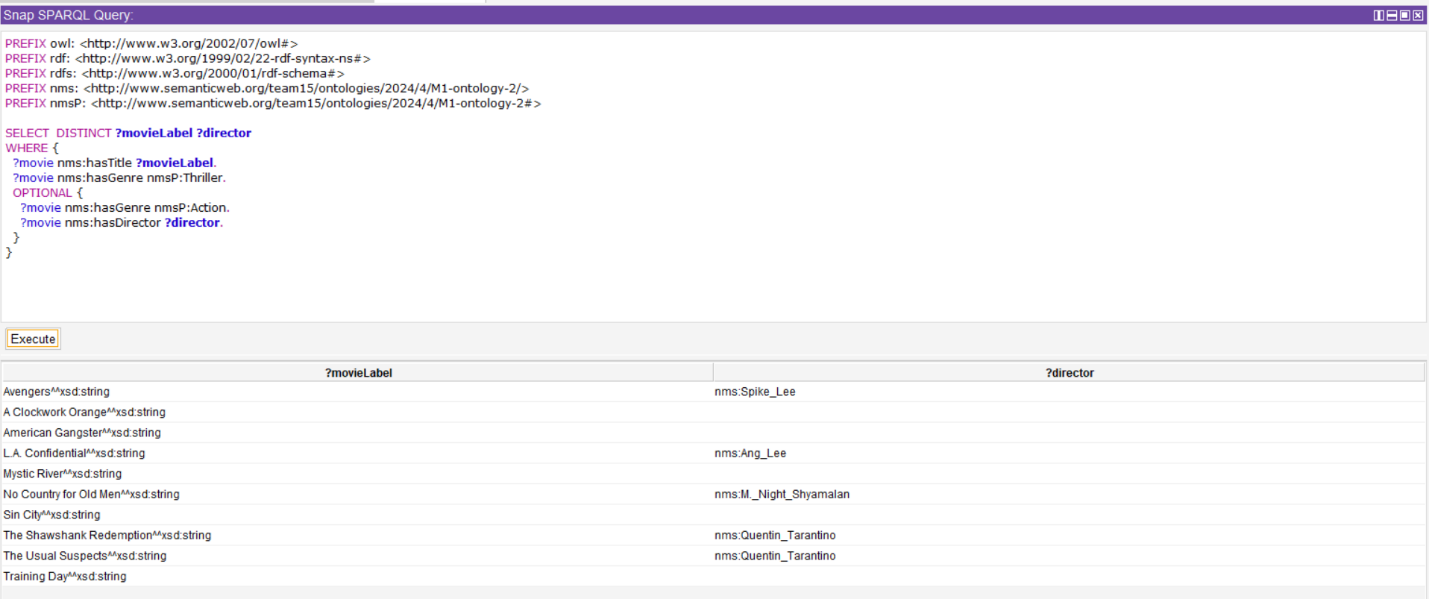
OPTIONAL {

?movie nms:hasGenre nmsP:Action.

?movie nms:hasDirector **?director**.

}

}



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

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

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

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

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

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

SELECT **?movieLabel** **?director**

WHERE {

?movie nms:hasTitle **?movieLabel**.

?movie nms:hasGenre nmsP:Action.

?movie nms:hasDirector **?director**.

{

?movie nms:hasGenre nmsP:Action.

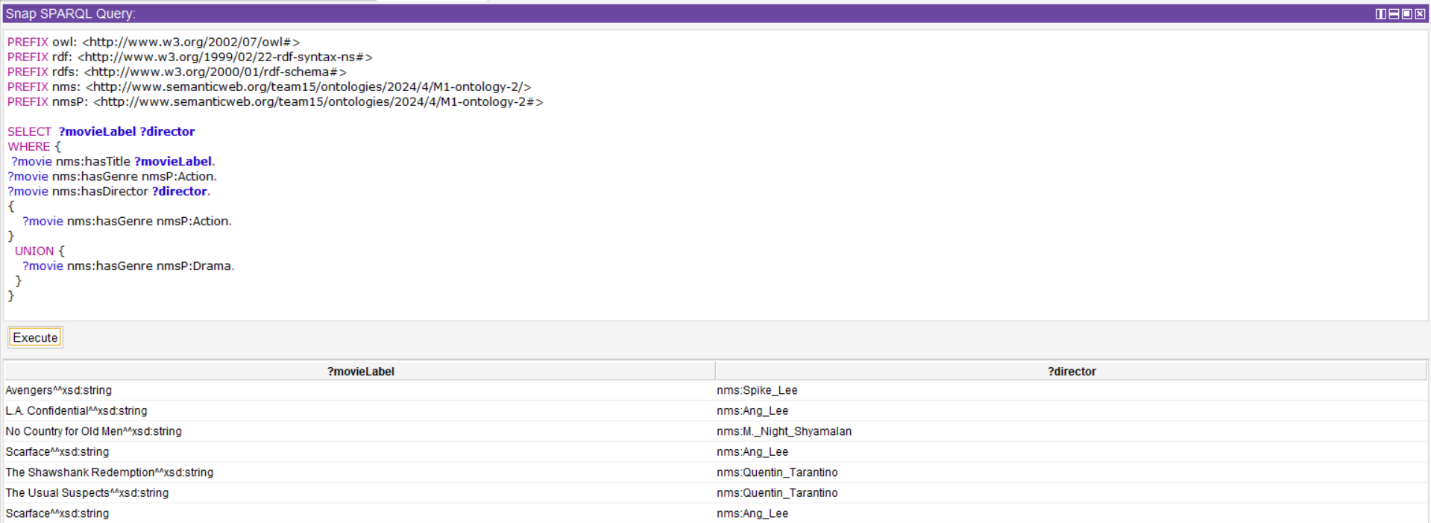
}

UNION {

?movie nms:hasGenre nmsP:Drama.

}

}



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

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

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

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

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

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

SELECT **?movieLabel** **?releaseYear** **?director**

WHERE {

{

?movie nms:hasTitle **?movieLabel**;

nms:hasGenre nmsP:Action;

nms:hasDirector **?director**;

nms:hasYear **?releaseYear**.

}

UNION {

?movie nms:hasTitle **?movieLabel**;

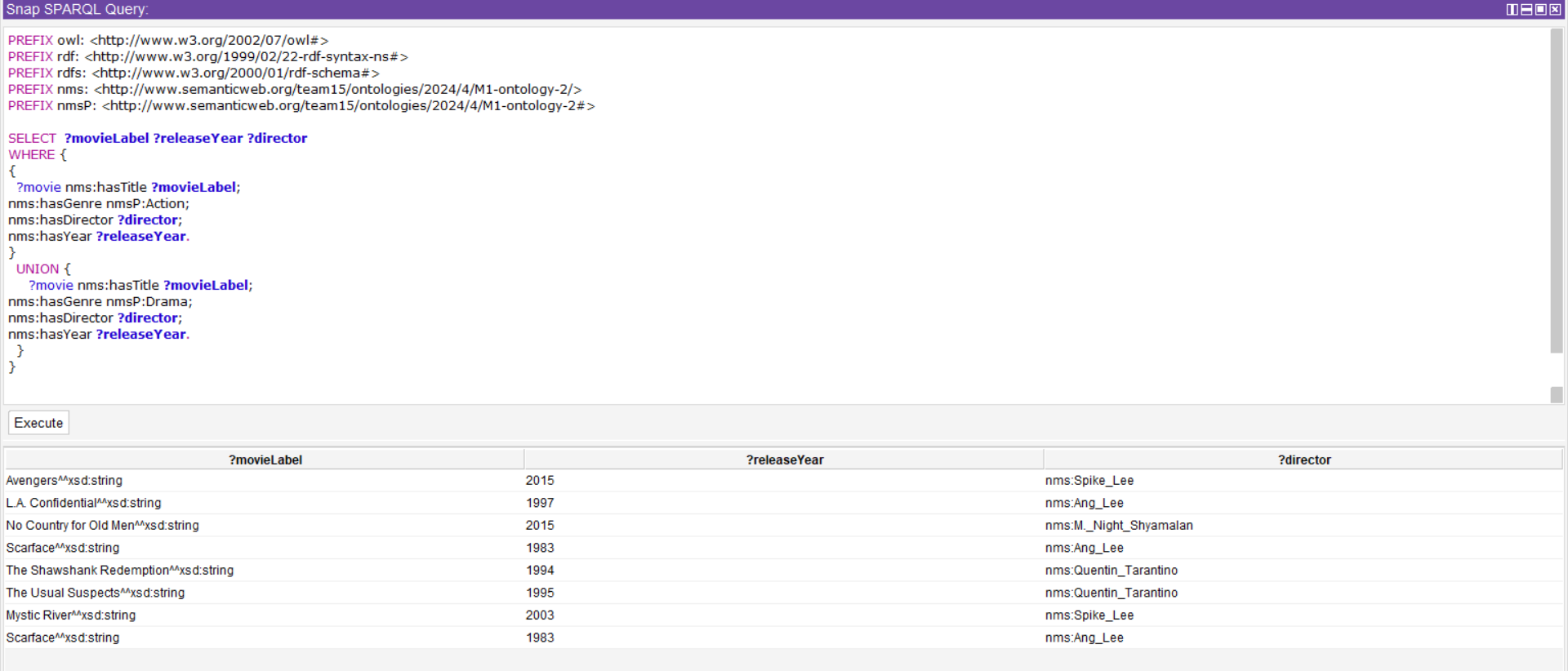
nms:hasGenre nmsP:Drama;

nms:hasDirector **?director**;

nms:hasYear **?releaseYear**.

}

}



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

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

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

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

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

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

SELECT **?movieLabel** **?releaseYear**

WHERE {

?movie rdf:type nms:Movie.

?movie nms:hasTitle **?movieLabel**.

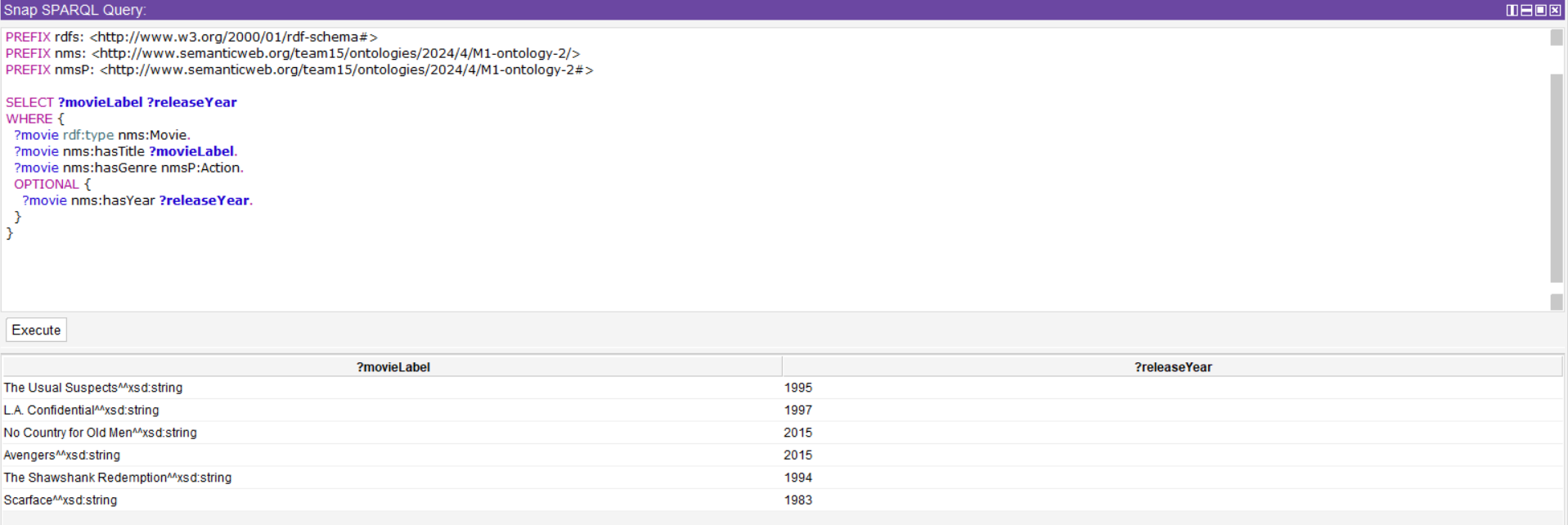
?movie nms:hasGenre nmsP:Action.

OPTIONAL {

?movie nms:hasYear **?releaseYear**.

}

}



# Manipulating The Ontology in Python

## Ontology1

### Code

*from* rdflib *import* Graph

*from* rdflib.tools.rdf2dot *import* rdf2dot

def load\_ontology(file\_path):

*# Load the TTL file into an RDF graph*

    g = Graph()

    g.parse(file\_path, format='ttl')

*return* g

def get\_persons(graph):

    persons = set()

*# Iterate through all triples in the graph*

*for* subj, pred, obj *in* graph:

*# Check if the predicate is 'rdf:type' and the object is 'Person'*

*if* pred.endswith('type') and (str(obj).endswith('Actor') or str(obj).endswith('Writer') or str(obj).endswith('Director')):

            persons.add(subj)

*return* persons

def display\_persons(file\_path):

*# Load the ontology*

    graph = load\_ontology(file\_path)

*# Get all the Persons from the ontology*

    persons = get\_persons(graph)

*# Display the Persons*

    print("Persons Without Using Inference and Query:")

*for* person *in* persons:

        print(person)

def visualize\_graph(file\_path, output\_file):

*# Load the ontology*

    graph = load\_ontology(file\_path)

*# Create a stream to write DOT data*

*with* open(output\_file, 'w') *as* f:

*# Convert RDF graph to DOT format and write to the stream*

        rdf2dot(graph, f)

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    ttl\_file = "Ontology\_phase1\_team15.ttl"  *# Path to your TTL file*

    output\_dot\_file = "graph.dot"  *# Output DOT file*

    display\_persons(ttl\_file)

    visualize\_graph(ttl\_file, output\_dot\_file)

### output:

A screen shot of a computer

Description automatically generated

## Ontology2

### Code

*from* rdflib *import* Graph, Namespace

*from* rdflib.plugins.sparql *import* prepareQuery

*# Define namespaces*

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

RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

def load\_ontology(file\_path):

*# Load the TTL file into an RDF graph*

    g = Graph()

    g.parse(file\_path, format='ttl')

*return* g

def read\_query\_from\_file(query\_file):

*# Read the SPARQL query from the file*

*with* open(query\_file, 'r') *as* f:

        query\_text = f.read()

*return* query\_text

def get\_persons\_with\_query(graph, query\_text):

    persons = set()

*# Prepare a SPARQL query from the text*

    query = prepareQuery(query\_text, initNs={"rdf": RDF, "base": BASE})

*# Execute the query and collect the results*

    results = graph.query(query)

*for* row *in* results:

        persons.add(row.person)

*return* persons

def display\_persons\_with\_query(file\_path, query\_file):

*# Load the ontology*

    graph = load\_ontology(file\_path)

*# Read the SPARQL query from the file*

    query\_text = read\_query\_from\_file(query\_file)

*# Get all the Persons who are actors, directors, or writers using SPARQL query*

    persons = get\_persons\_with\_query(graph, query\_text)

*# Display the Persons*

    print("Persons:")

*for* person *in* persons:

        print(person)

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    ttl\_file = "Ontology\_phase1\_team15.ttl"  *# Path to your TTL file*

    query\_file = "query.txt"    *# Path to your SPARQL query text file*

    display\_persons\_with\_query(ttl\_file, query\_file)

query file

SELECT ?person

WHERE {

{

?person rdf:type base:Actor .

}

UNION

{

?person rdf:type base:Director .

}

UNION

{

?person rdf:type base:Writer .

}

}

### Output

A screen shot of a computer

Description automatically generated

## Ontology3

### Code

*from* rdflib *import* Graph, RDF, RDFS, OWL, Namespace

*from* owlrl *import* DeductiveClosure, RDFS\_Semantics

def load\_ontology(file\_path):

*# Load the TTL file into an RDF graph*

    g = Graph()

    g.parse(file\_path, format='ttl')

*# Apply Deductive Closure reasoning with RDFS semantics to the graph*

    DeductiveClosure(RDFS\_Semantics).expand(g)

*return* g

def get\_actors(graph, ns):

    actors = set()

*# Iterate through all inferred triples in the graph*

*for* subj, pred, obj *in* graph:

*# Check if the subject is an Actor*

*if* (subj, RDF.type, ns.Actor) in graph:

            actors.add(subj)

*return* actors

def display\_actors(file\_path):

*# Load the ontology and apply Deductive Closure reasoning*

    graph = load\_ontology(file\_path)

*# Define namespace*

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

*# Get all the Actors from the ontology*

    actors = get\_actors(graph, ns)

*# Display the Actors*

    print("Actors Using Inference without query")

*for* actor *in* actors:

        print(actor)

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    ttl\_file = "Ontology\_phase1\_team15.ttl"  *# Path to your TTL file*

    display\_actors(ttl\_file)

### Output

A screen shot of a computer

Description automatically generated

## Ontology4

### Code

*from* rdflib *import* Graph, RDF, RDFS, OWL, Namespace, Literal

*from* owlrl *import* DeductiveClosure, RDFS\_Semantics

def load\_ontology(file\_path):

*# Load the TTL file into an RDF graph*

    g = Graph()

    g.parse(file\_path, format='ttl')

*# Apply Deductive Closure reasoning with RDFS semantics to the graph*

    DeductiveClosure(RDFS\_Semantics).expand(g)

*return* g

def get\_movie\_info(graph, ns, movie\_name):

    movie\_info = {}

*# Iterate through all inferred triples in the graph*

*for* subj, pred, obj *in* graph:

*# Check if the subject is an instance of Movie with the given name*

*if* (subj, RDF.type, ns.Movie) in graph and (subj, ns.hasTitle, Literal(movie\_name)) in graph:

*# Get movie year, country, genres, and actors*

            year = graph.value(subj, ns.hasYear)

            country = graph.value(subj, ns.hasCountry)

            genres = [genre *for* genre *in* graph.objects(subj, ns.hasGenre)]

            actors = [actor *for* actor *in* graph.objects(subj, ns.hasActor)]

            movie\_info = {

                'Year': year,

                'Country': country,

                'Genres': genres,

                'Actors': actors

            }

*break*

*return* movie\_info

def display\_movie\_info(file\_path, movie\_name):

*# Load the ontology and apply Deductive Closure reasoning*

    graph = load\_ontology(file\_path)

*# Define namespace*

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

*# Get movie information*

    movie\_info = get\_movie\_info(graph, ns, movie\_name)

*# Display movie information*

*if* movie\_info:

        print("Movie Information:")

        print("Name:", movie\_name)

        print("Year:", movie\_info.get('Year'))

        print("Country:", movie\_info.get('Country'))

        print("Genres:", ', '.join(movie\_info.get('Genres')))

        print("Actors:", ', '.join(movie\_info.get('Actors')))

*else*:

        print("Error: Movie '{}' not found.".format(movie\_name))

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    ttl\_file = "Ontology\_phase1\_team15.ttl"  *# Path to your TTL file*

    movie\_name = input("Enter the name of the movie: ")

    display\_movie\_info(ttl\_file, movie\_name)

### Output

A screen shot of a computer

Description automatically generated

## Ontology5

### Code

*from* rdflib *import* Graph, RDF, Namespace

*from* rdflib.plugins.sparql *import* prepareQuery

*from* owlrl *import* DeductiveClosure, RDFS\_Semantics, OWLRL\_Semantics

*# Define namespaces*

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

RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

def load\_ontology(file\_path, rule\_path):

*# Load the TTL file into an RDF graph*

    g = Graph()

    g.parse(file\_path, format='ttl')

    g.parse(rule\_path, format='ttl')

*# Apply Deductive Closure reasoning with RDFS semantics to the graph*

    DeductiveClosure(OWLRL\_Semantics).expand(g)

*return* g

def get\_actor\_directors(graph):

    persons = set()

*# Prepare a SPARQL query from the text*

    query = prepareQuery("""

        SELECT DISTINCT ?actorDirector

        WHERE {

            ?actorDirector rdf:type base:ActorDirector .

        }

        """, initNs={"rdf": RDF, "base": BASE})

*# Execute the query and collect the results*

    results = graph.query(query)

*for* row *in* results:

        persons.add(row["actorDirector"])

*return* persons

def display\_actor\_directors(file\_path, rule\_path):

*# Load the ontology*

    graph = load\_ontology(file\_path, rule\_path)

*# Get all the actor-directors from the ontology*

    actor\_directors = get\_actor\_directors(graph)

*# Display the actor-directors*

    print("Actor-Directors:")

*for* actor\_director *in* actor\_directors:

        print(actor\_director)

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    ttl\_file = "Ontology\_phase1\_team15.ttl"  *# Path to your TTL file*

    rule\_file = "rule.ttl"

    display\_actor\_directors(ttl\_file, rule\_file)

Rule File  
@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix : <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/> .

@prefix ns: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#> .

# Define ActorDirector as an intersection of Actor and Director

ns:ActorDirector a owl:Class ;

owl:equivalentClass [

a owl:Class ;

owl:intersectionOf ( :Actor :Director )

] .

### Output

A screen shot of a computer code

Description automatically generated

## Ontology6

### Code

*from* rdflib *import* Graph, RDF, Namespace, OWL

*from* owlrl *import* DeductiveClosure, RDFS\_Semantics, OWLRL\_Semantics

*# Define namespaces*

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

RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

def load\_ontology(file\_path, rule\_path):

*# Load the TTL file into an RDF graph*

    g = Graph()

    g.parse(file\_path, format='ttl')

    g.parse(rule\_path, format='ttl')

*# Apply Deductive Closure reasoning with RDFS semantics to the graph*

    DeductiveClosure(OWLRL\_Semantics).expand(g)

*return* g

def get\_actor\_writer(graph):

    actor\_writers = set()

*# Iterate through all inferred triples in the graph*

*for* subj, pred, obj *in* graph:

*# Check if the subject is an ActorWriter*

*if* (subj, RDF.type, BASE.ActorWriter) in graph:

            actor\_writers.add(subj)

*return* actor\_writers

def display\_actor\_writer(file\_path, rule\_path):

*# Load the ontology*

    graph = load\_ontology(file\_path, rule\_path)

*# Get all the actor-directors from the ontology*

    actor\_writers = get\_actor\_writer(graph)

*# Display the actor-directors*

    print("ActorWriter:")

*for* actor\_writer *in* actor\_writers:

        print(actor\_writer)

def get\_director\_writer(graph):

    director\_writers = set()

*# Iterate through all inferred triples in the graph*

*for* subj, pred, obj *in* graph:

*# Check if the subject is an ActorWriter*

*if* (subj, RDF.type, BASE.WriterDirector) in graph:

            director\_writers.add(subj)

*return* director\_writers

def display\_director\_writer(file\_path, rule\_path):

*# Load the ontology*

    graph = load\_ontology(file\_path, rule\_path)

*# Get all the actor-directors from the ontology*

    director\_writers = get\_director\_writer(graph)

*# Display the actor-directors*

    print("DirectorWriter:")

*for* director\_writer *in* director\_writers:

        print(director\_writer)

def get\_director\_writer\_actor(graph):

    director\_writers\_actor = set()

*# Iterate through all inferred triples in the graph*

*for* subj, pred, obj *in* graph:

*# Check if the subject is an ActorWriter*

*if* (subj, RDF.type, BASE.ActorWriterDirector) in graph:

            director\_writers\_actor.add(subj)

*return* director\_writers\_actor

def display\_director\_writer\_actor(file\_path, rule\_path):

*# Load the ontology*

    graph = load\_ontology(file\_path, rule\_path)

*# Get all the actor-directors from the ontology*

    director\_writers\_actor = get\_director\_writer\_actor(graph)

*# Display the actor-directors*

    print("DirectorWriterActor:")

*for* director\_writer\_actor *in* director\_writers\_actor:

        print(director\_writer\_actor)

*# Example usage*

*if* \_\_name\_\_ == "\_\_main\_\_":

    ttl\_file = "Ontology\_phase1\_team15.ttl"  *# Path to your TTL file*

    rule\_file = "rule.ttl"

    display\_actor\_writer(ttl\_file, rule\_file)

    display\_director\_writer(ttl\_file, rule\_file)

    display\_director\_writer\_actor(ttl\_file,rule\_file)

Rule File  
  
Rule File  
@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix : <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2/> .

@prefix ns: <http://www.semanticweb.org/team15/ontologies/2024/4/M1-ontology-2#> .

# Define ActorWriter as an intersection of Actor and Writer

ns:ActorWriter a owl:Class ;

owl:equivalentClass [

a owl:Class ;

owl:intersectionOf ( :Actor :Writer )

] .

# Define WriterDirector as an intersection of Actor and Writer

ns:WriterDirector a owl:Class ;

owl:equivalentClass [

a owl:Class ;

owl:intersectionOf ( :Director :Writer )

] .

# Define ActorWriterDirector as an intersection of Actor and Writer and Director

ns:ActorWriterDirector a owl:Class ;

owl:equivalentClass [

a owl:Class ;

owl:intersectionOf ( :Actor :Director :Writer )

] .

### Output

A screen shot of a computer

Description automatically generated

# Application

## Code

*from* rdflib *import* Graph, URIRef, Namespace

*from* rdflib.namespace *import* RDF, RDFS, XSD

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

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

*# Define a Movie class to represent each movie*

class Movie:

    def \_\_init\_\_(self, title, actors, director, genres, year):

*self*.title = title

*self*.actors = actors

*self*.director = director

*self*.genres = genres

*self*.year = year

*# Define a class to manage the movie database*

class MovieDatabase:

    def \_\_init\_\_(self):

*self*.movies = []

    def load\_from\_ttl(self, ttl\_file):

        g = Graph()

        g.parse(ttl\_file, format="ttl")

*# Iterate over all triples in the graph*

*for* subj, pred, obj *in* g:

*# Check if the subject is a movie*

*if* (subj, RDF.type, BASE.Movie) in g:

*# Initialize variables to store movie details*

                title = None

                actors = set()

                director = None

                genres = set()

                year = None

*# Extract movie details from triples*

*if* (subj, BASE.hasTitle, None) in g:

                    title = str(g.value(subj, BASE.hasTitle))

*if* (subj, BASE.hasActor, None) in g:

                    actors.update(str(actor) *for* actor *in* g.objects(subj, BASE.hasActor))

*if* (subj, BASE.hasDirector, None) in g:

                    director = str(g.value(subj, BASE.hasDirector))

*if* (subj, BASE.hasGenre, None) in g:

                    genres.update(str(genre) *for* genre *in* g.objects(subj, BASE.hasGenre))

*if* (subj, BASE.hasYear, None) in g:

                    year = int(g.value(subj, BASE.hasYear))

*# Create a Movie object and append it to the list of movies*

                movie = Movie(title, list(actors), director, list(genres), year)

*self*.movies.append(movie)

    def search\_movies(self, included\_actors=[], included\_directors=[], included\_genres=[], excluded\_actors=[], excluded\_directors=[], excluded\_genres=[]):

        result = []

*for* movie *in* *self*.movies:

*if* (not included\_actors or set(movie.actors).intersection(included\_actors)) \

                    and (not included\_directors or movie.director in included\_directors) \

                    and (not included\_genres or set(movie.genres).intersection(included\_genres)) \

                    and (not excluded\_actors or not set(movie.actors).intersection(excluded\_actors)) \

                    and (not excluded\_directors or movie.director not in excluded\_directors) \

                    and (not excluded\_genres or not set(movie.genres).intersection(excluded\_genres)):

                result.append(movie)

*return* result

def prompt\_and\_process\_input(prompt):

*"""*

*Prompt the user for input and process it into URIs.*

*"""*

    names = input(prompt).split(',')

*if* not names:

*return* []

*else*:

*return* [BASE + name.strip() *for* name *in* names]

def prompt\_and\_process\_input\_for\_genres(prompt):

*"""*

*Prompt the user for input and process it into URIs.*

*"""*

    names = input(prompt).split(',')

*if* not names:

*return* []

*else*:

*return* [BASE2 + name.strip() *for* name *in* names]

*# Main function*

def main():

*# Create a movie database*

    movie\_database = MovieDatabase()

*# Load movie data from TTL file*

    movie\_database.load\_from\_ttl("Ontology\_phase1\_team15.ttl")

*# Take input from User*

    included\_actor\_uris = prompt\_and\_process\_input("Actor names to include separated by commas: ")

    excluded\_actor\_uris = prompt\_and\_process\_input("Actor names to exclude separated by commas: ")

    included\_genre\_uris = prompt\_and\_process\_input\_for\_genres("Genre types to include separated by commas: ")

    excluded\_genre\_uris = prompt\_and\_process\_input\_for\_genres("Genre types to exclude separated by commas: ")

    included\_director\_uris = prompt\_and\_process\_input("Director names to include separated by commas: ")

    excluded\_director\_uris = prompt\_and\_process\_input("Director names to exclude separated by commas: ")

*# Check if any of the lists are empty and replace them with []*

    included\_actor\_uris = included\_actor\_uris *if* included\_actor\_uris!=[BASE] *else* []

    excluded\_actor\_uris = excluded\_actor\_uris *if* excluded\_actor\_uris!=[BASE] *else* []

    included\_director\_uris = included\_director\_uris *if* included\_director\_uris!=[BASE] *else* []

    excluded\_director\_uris = excluded\_director\_uris *if* excluded\_director\_uris!=[BASE] *else* []

    included\_genre\_uris = included\_genre\_uris *if* included\_genre\_uris!=[BASE2] *else* []

    excluded\_genre\_uris = excluded\_genre\_uris *if* excluded\_genre\_uris!=[BASE2] *else* []

*# Search for movies based on criteria*

    found\_movies = movie\_database.search\_movies(

                                                included\_actors=included\_actor\_uris,

                                                included\_genres=included\_genre\_uris,

                                                included\_directors=included\_director\_uris,

                                                excluded\_actors=excluded\_actor\_uris,

                                                excluded\_genres=excluded\_genre\_uris,

                                                excluded\_directors=excluded\_director\_uris,

                                            )

*# Display the found movies*

    print("------------------------------------------------------------------")

*if* found\_movies:

        print("Found movies:")

        printed\_titles = set()  *# Set to store encountered titles*

*for* movie *in* found\_movies:

*if* movie.title not in printed\_titles:  *# Check if title is not already printed*

                print("Movie Title:", movie.title)

                print("Movie Release Year:", movie.year)

                print("Movie Genre:", ", ".join(movie.genres))

                print("Movie Director Name:", movie.director)

                print("Movie Actor Name:", ", ".join(movie.actors))

                print("----------")

                printed\_titles.add(movie.title)  *# Add title to printed set*

*else*:

        print("There are no movies with this info.")

*if* \_\_name\_\_ == "\_\_main\_\_":

    main()

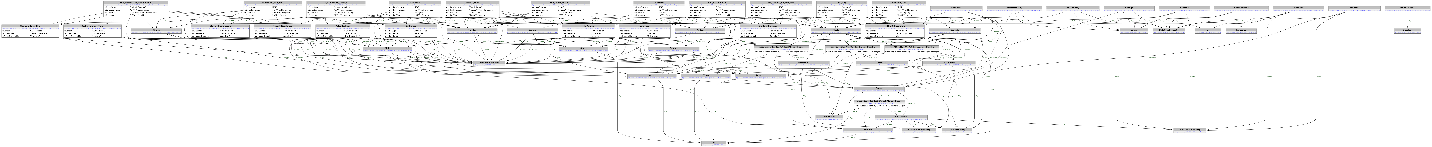
## Output

A screen shot of a computer program

Description automatically generated A screen shot of a computer

Description automatically generated

# RDF GRAPH USING PYTHON



# DFD

A diagram of a computer

Description automatically generated