Mohamad Ahmed Abdelmoniem	19P5170
Ziad Assem Shafik	19P6363
Karim Ashraf	19P6044
Ahmed Mohsen	19P1150
Mohamed Fathy	19P4704

Course: CSE488 - Ontologies and the Semantic Web

Submission date: 13-May-24

Project

Movies Ontology

Contents

Phase 1

Introduction

In the digital age, the film industry has seen a remarkable transformation, driven by rapid technological advances and the increasing accessibility of multimedia content. Amidst this backdrop, the need for sophisticated tools to manage and analyze film-related data has never been greater. Our Film Ontology Project seeks to address this need by developing a comprehensive ontology that serves as a structured framework for representing knowledge within the film industry.

Objective

The primary objective of this project is to create a robust ontology that encapsulates the rich and complex data associated with films. This includes detailed information about movies, such as their titles, genres, release years, and languages, as well as data about the people involved in making these films, like actors, directors, and writers. By formalizing this information in an ontology, we aim to facilitate better data integration, retrieval, and analysis, enabling stakeholders to gain deeper insights into the film industry.

Scope

The scope of the Film Ontology includes:

- **Movie Representation:** Detailed modeling of movie attributes such as genre, title, release year, country of origin, and language.
- **Personnel Classification:** Classification of film industry personnel into actors, directors, and writers, along with their biographical details like age, gender, and nationality.
- **Relationship Mapping:** Establishing relationships between different entities, such as which actors have worked under which directors, or which writers have contributed to particular films.
- **Genre Categorization:** Defining a controlled vocabulary for movie genres to ensure consistency across the dataset.

Approach

Our approach involves using semantic web technologies to build the ontology. We employ the Resource Description Framework (RDF) to describe and interlink data with a defined vocabulary. Furthermore, we use SPARQL (SPARQL Protocol and RDF Query Language) to facilitate complex queries that can fetch nuanced information from our dataset.

Classes

1. Movie

Description: The **Movie** class represents individual films. This class includes properties that describe the film's title, genre, release year, country of production, and language. It serves as a central entity around which much of the film-related data is organized.

2. Person

Description: The **Person** class is a generic class that encompasses all individuals associated with the production of a film. This includes actors, directors, and writers. Attributes such as name, age, gender, and nationality are common to this class, providing basic biographical details about these individuals.

Subclasses of Person:

- **Actor**
  - **Description:** The **Actor** class is a subclass of **Person** that represents individuals who perform roles in films. This class is linked to specific movies through the **isActorOf** relationship.
- **Director**
  - **Description:** The **Director** class, another subclass of **Person**, represents individuals responsible for the artistic and dramatic aspects of a film, guiding the technical crew and actors in the fulfillment of their vision.
- **Writer**
  - **Description:** The **Writer** class includes individuals who write or adapt scripts for films. Writers lay the groundwork for the narrative and dialogue of a movie, making this class vital for understanding the creative origins of film content.

3. Genre

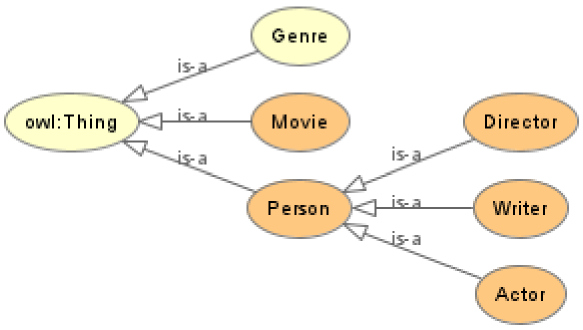
Description: The **Genre** class categorizes movies into specific genres, such as Thriller, Crime, Action, Drama, and Comedy. This classification aids in the organization of movies into thematic or stylistic categories, making it easier for users to find films of a particular type or to analyze cinematic trends within genres.

4. Thriller, Crime, Action, Drama, Comedy (Genre Subclasses)

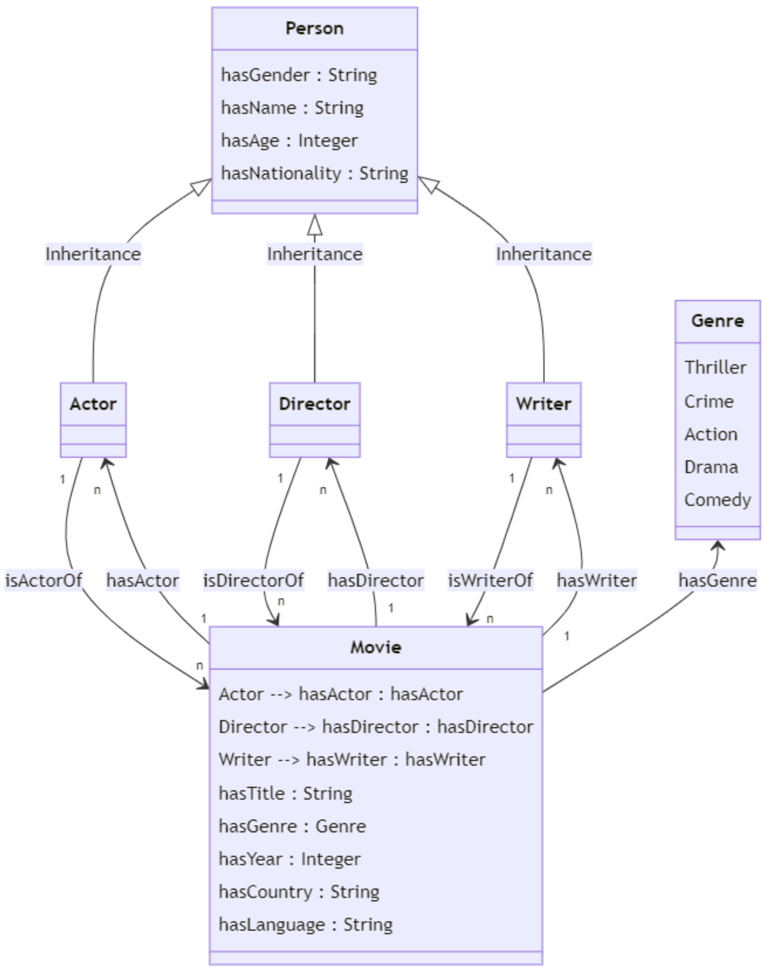
Description: These classes are subclasses of the **Genre** class. Each represents a specific genre category:

- **Thriller:** A genre characterized by suspense, tension, and excitement, often involving complex plots and cliffhangers.
- **Crime:** Films that focus on the lives of criminals or intricate crime-solving stories.
- **Action:** A genre involving a series of fast-paced sequences, often featuring physical feats, extended fight scenes, and chases.
- **Drama:** These films focus on in-depth development of realistic characters and emotional themes.
- **Comedy:** A genre meant to amuse and entertain the audience, characterized by humorous characters, situations, or dialogue.

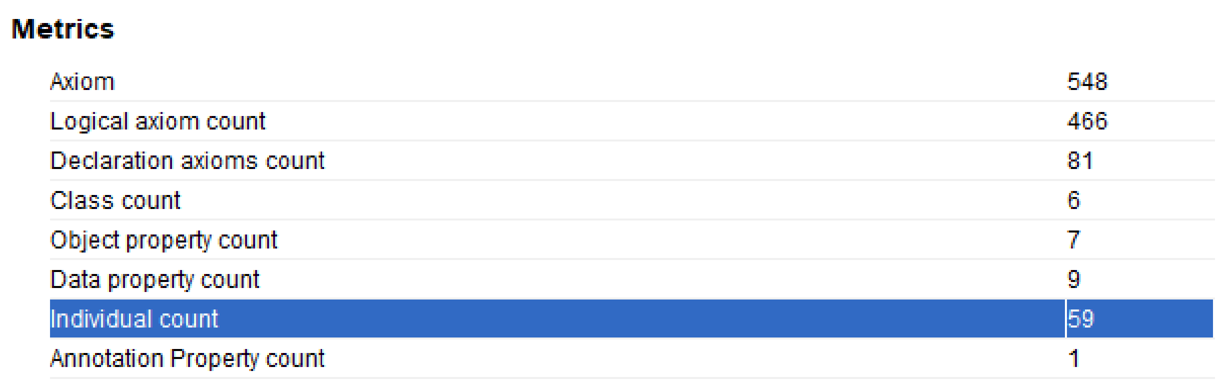
These descriptions define the scope and purpose of each class within your film ontology, providing a structured framework for storing and retrieving comprehensive data about films and the people who make them. This structured approach not only facilitates detailed analysis but also enhances the accessibility and management of film-related information in databases and applications.



The Class Diagram of Our Ontology

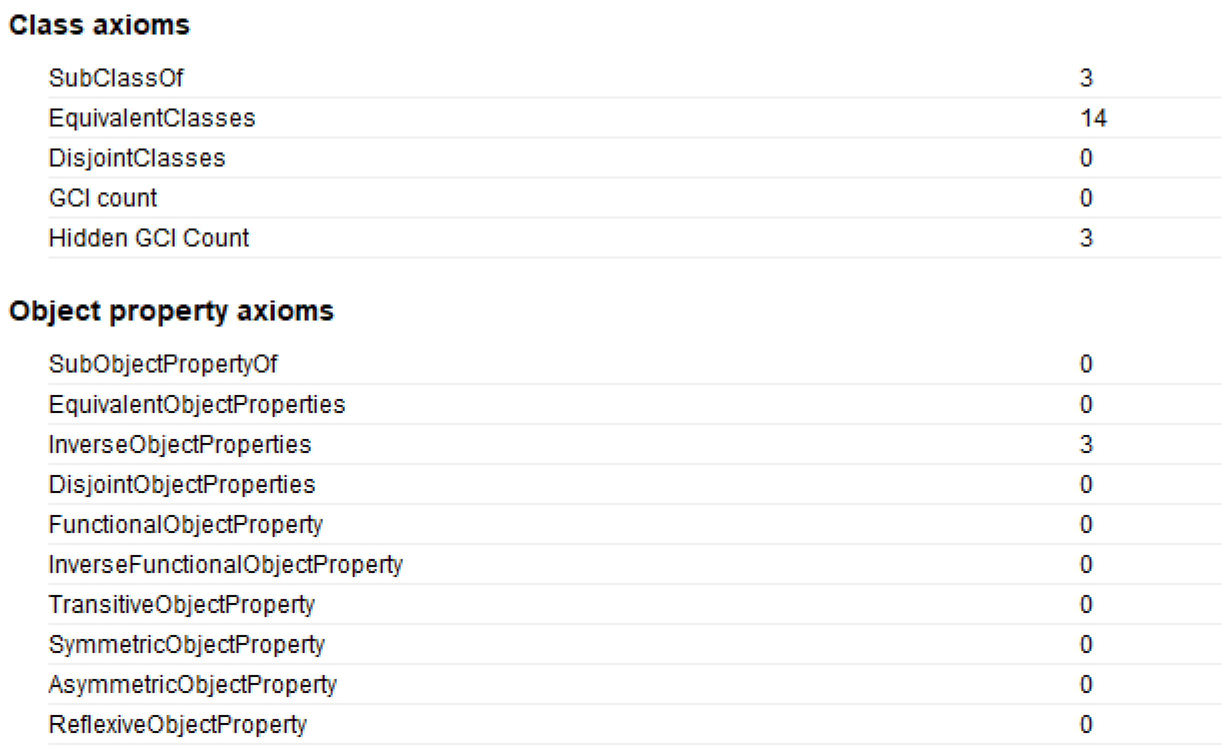


Ontology Information



Metrics

Axiom	548
Logical axiom count	466
Declaration axioms count	81
Class count	6
Object property count	7
Data property count	9
Individual count	59
Annotation Property count	1



Class axioms

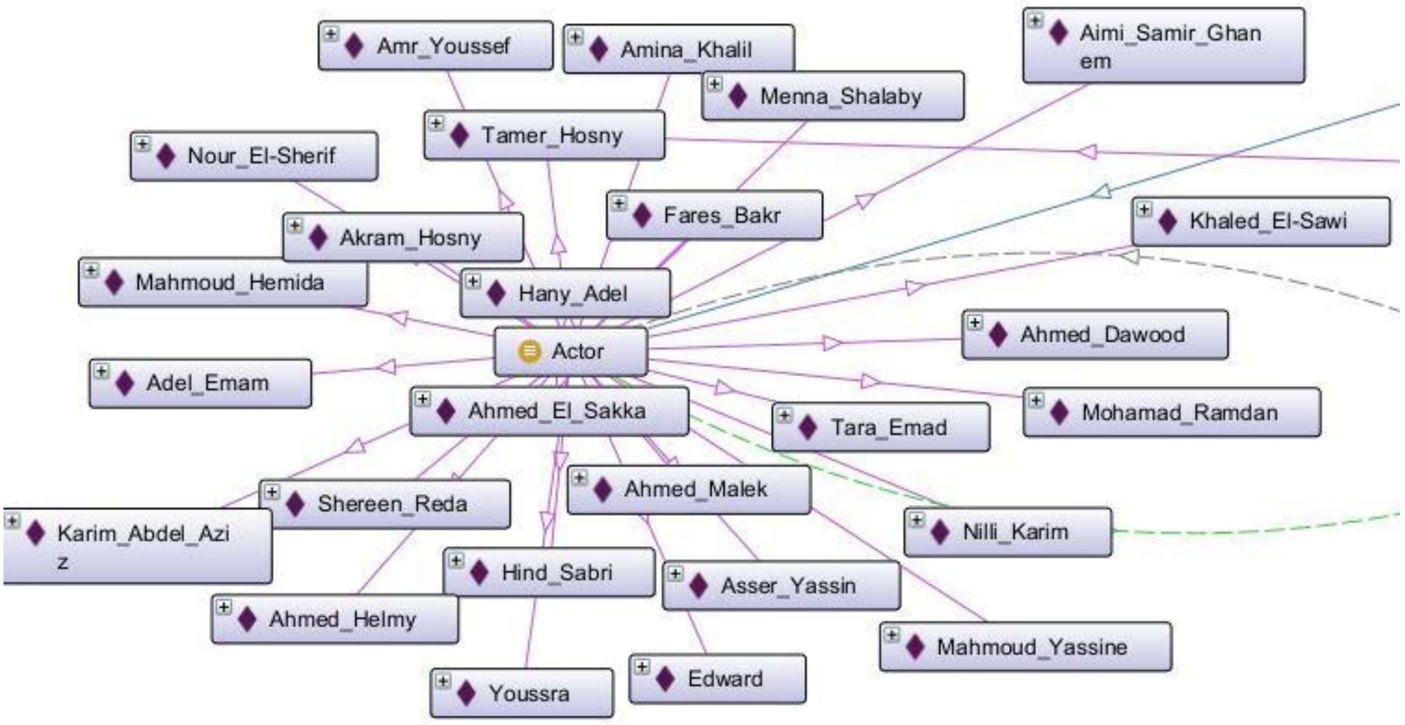
SubClassOf	3
EquivalentClasses	14
DisjointClasses	0
GCI count	0
Hidden GCI Count	3

Object property axioms

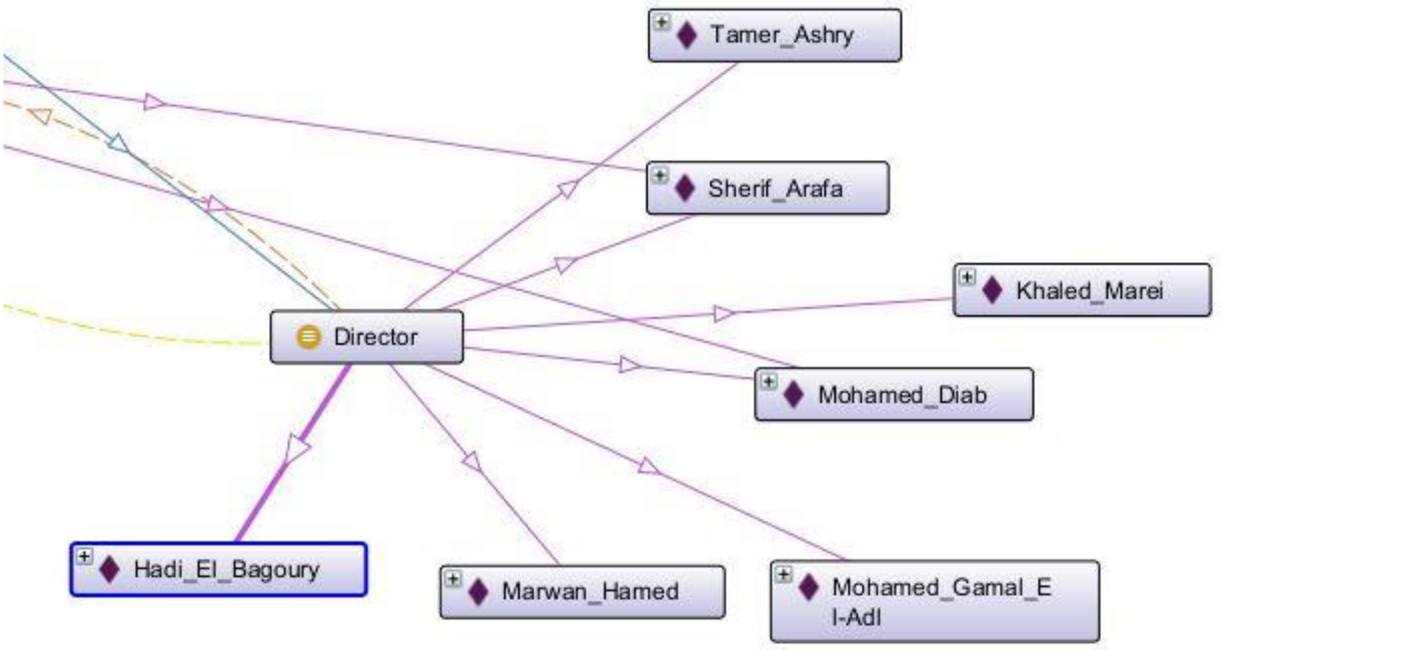
SubObjectPropertyOf	0
EquivalentObjectProperties	0
InverseObjectProperties	3
DisjointObjectProperties	0
FunctionalObjectProperty	0
InverseFunctionalObjectProperty	0
TransitiveObjectProperty	0
SymmetricObjectProperty	0
AsymmetricObjectProperty	0
ReflexiveObjectProperty	0

List of Individuals in the ontology

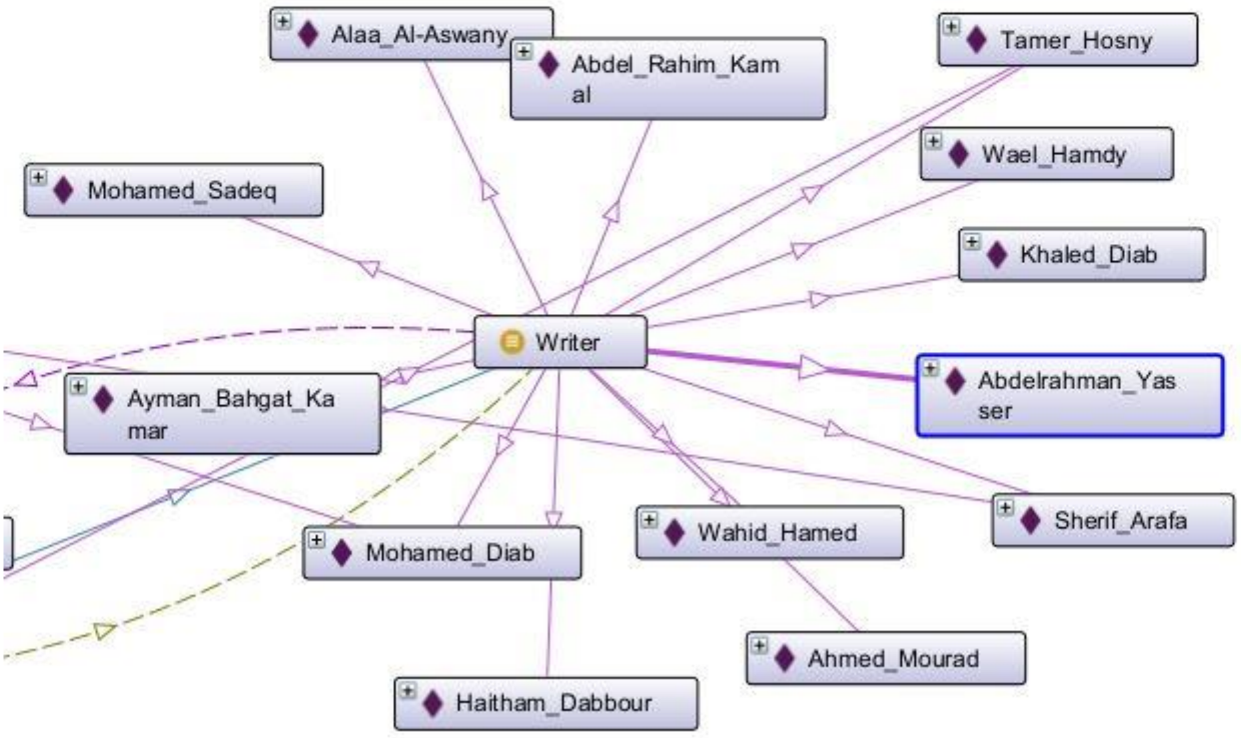
List of Actors



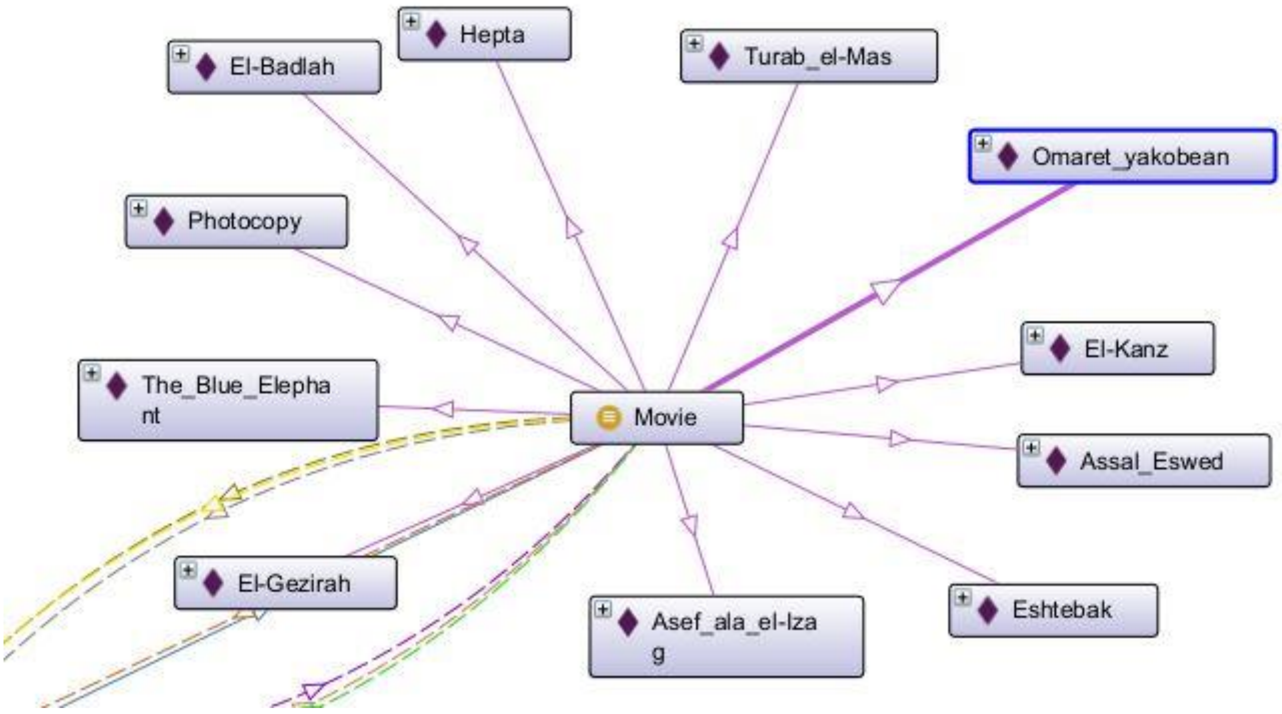
List of Directors



List of Writers



List of movies



Test Cases

Test Case 1: Actor with Two Names

Input: A **Person** entity with dual entries in the **hasName** attribute, either as two separate entries or a non-standard format (e.g., "Ahmed Yasser / Ahmed Hammed").

Output: The system raises an error

Test Case 2: Movie with Multiple Genres

Input: A **Movie** entity that has multiple genres assigned to it (e.g., both Comedy and Drama).

Output: The ontology correctly Accept movies with multiple genres

Test Case 3: Person as Both Actor and Director of the Same Movie

Input: A **Person** entity linked to a **Movie** entity both as an **Actor** and a **Director**.

Output: The ontology is be able to represent a person who can have multiple roles in the same movie without conflict. The system validate that both relationships (**isActorOf** and **isDirectorOf**) coexist without issues.

Test Case 4: Inconsistent Data Types

Input: A **Movie** entity with a string value in the **hasYear** attribute (e.g., "Two thousand and twenty").

Output: The system enforce datatype constraints and raise an error when the input does not match the expected integer datatype for the year.

Test Case 5: Incomplete Movie Information

Input: A **Movie** entity that is missing mandatory attributes such as **hasTitle** or **hasYear**.

Output: The ontology checks for the completeness of required attributes. If key attributes are missing, the system raises an error or prompt the user to complete the necessary information.

Test Case 6: Person with Non-String Name

Input: A **Person** entity with a numerical value in the **hasName** attribute (e.g., 12345).

Output: The system should enforce the data type constraint that names must be strings. An error is raised for non-string names.

Test Case 7: Movie with All Required Attributes

Input: A **Movie** entity fully populated with valid entries for all attributes (**hasTitle**, **hasYear**, **hasCountry**, **hasLanguage**, **hasGenre**).

Output: The system accept the entry without any errors, confirming that all required attributes are present and correctly formatted. This test confirms the system's ability to handle complete and correct data.

Test Case 8: Valid Gender Input

Input: A **Person** entity with **hasGender** set to "Male," which is within the allowed Enum values.

Output: The system successfully validate and accept the gender input without any issues. This case checks for correct enumeration handling.

Test Case 9: Actor Not Linked to a Movie

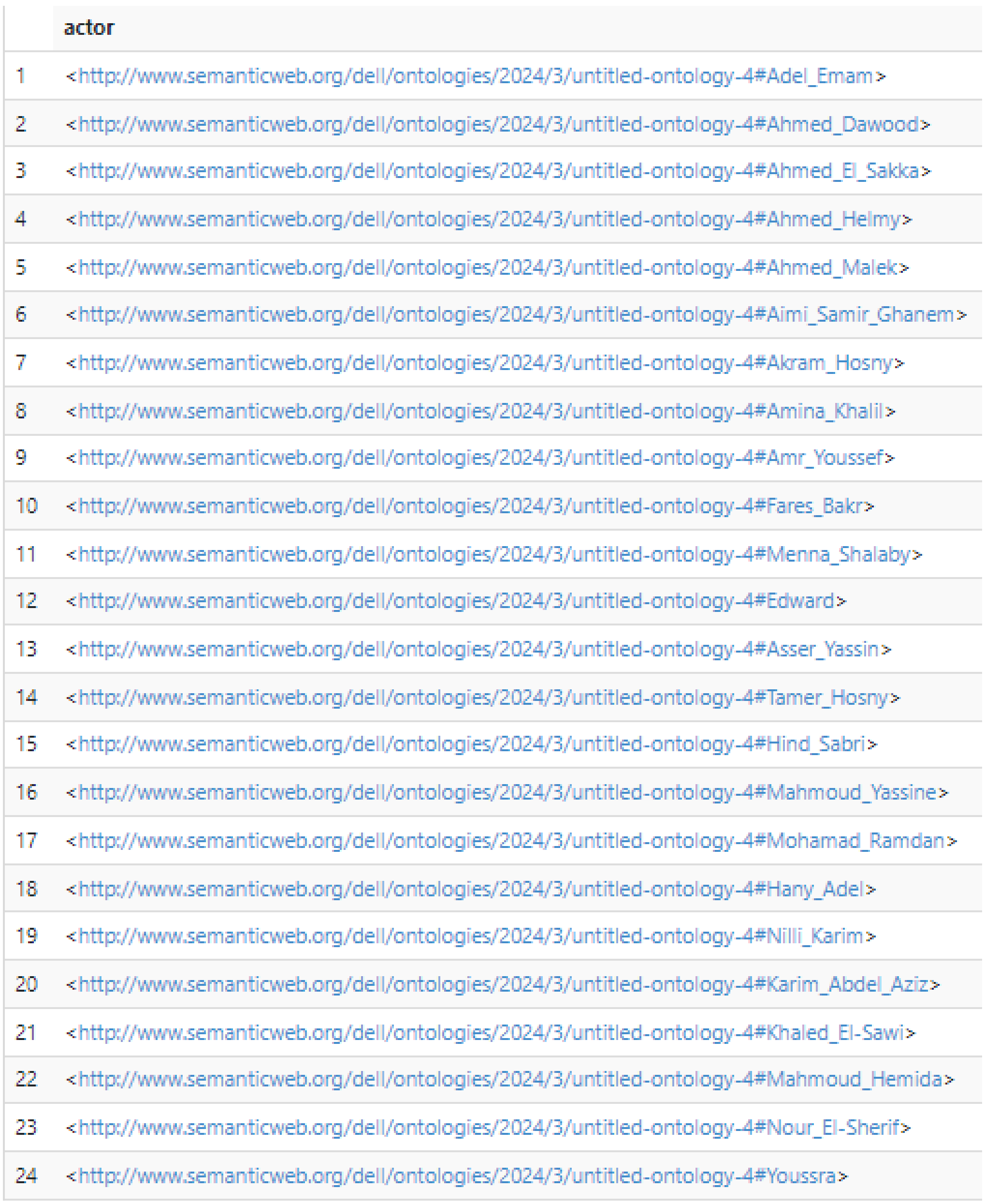
Input: A **Person** entity designated as an **Actor** but not linked to any **Movie** through **isActorOf**.

Output: it raises an error as actor must be **isActorOf** movie at least.

Queries

List the instances of the class Actor:

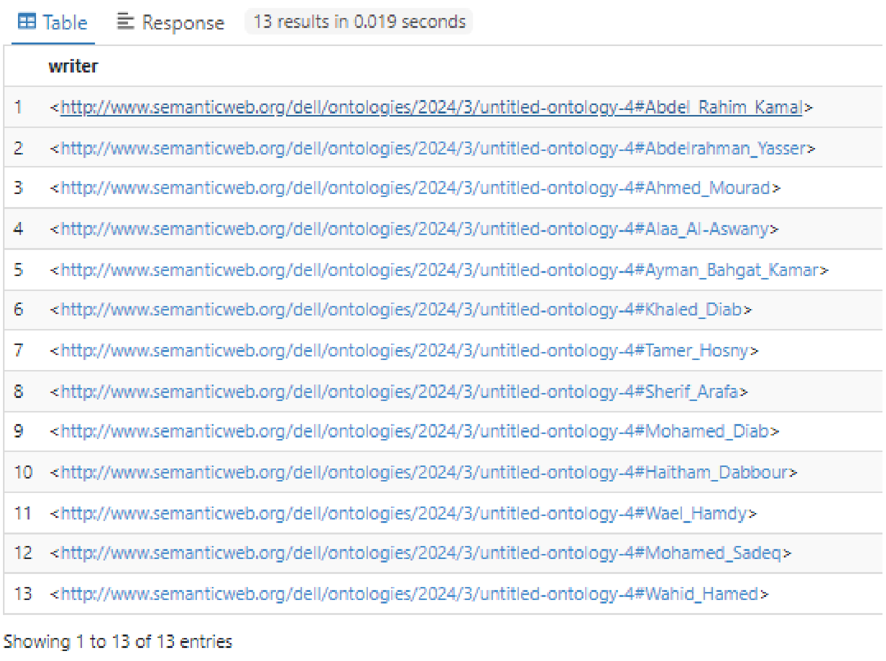
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?actor

WHERE {

  ?actor rdf:type table:Actor .

}
```



	actor
1	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Adel_Emam>
2	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_Dawood>
3	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_El_Sakka>
4	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_Helmy>
5	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_Malek>
6	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Aimi_Samir_Ghanem>
7	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Akram_Hosny>
8	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Amina_Khalil>
9	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Amr_Youssef>
10	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Fares_Bakr>
11	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Menna_Shalaby>
12	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Edward>
13	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Asser_Yassin>
14	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Hosny>
15	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Hind_Sabri>
16	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mahmoud_Yassine>
17	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mohamad_Ramdan>
18	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Hany_Adel>
19	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Nilli_Karim>
20	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Karim_Abdel_Aziz>
21	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Khaled_El-Sawi>
22	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mahmoud_Hemida>
23	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Nour_El-Sherif>
24	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Youssra>

List the instances of the class Writer

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?writer

WHERE {

  ?writer rdf:type table:Writer .

}
```



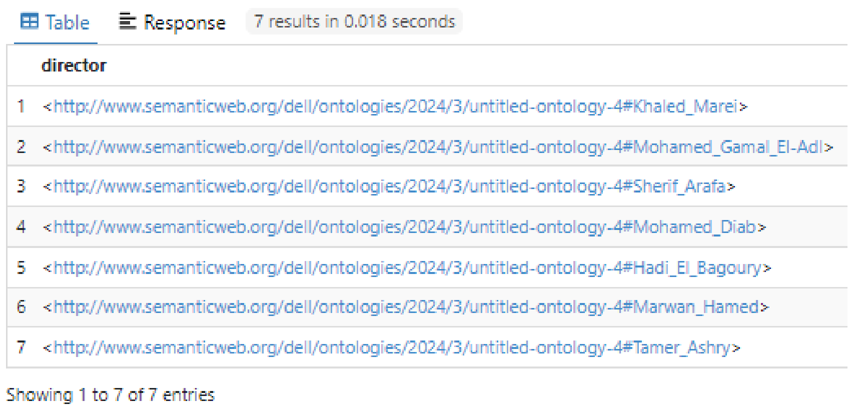
13 results in 0.019 seconds

	writer
1	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Abdel_Rahim_Kamal>
2	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Abdelrahman_Yasser>
3	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_Mourad>
4	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Alaa_Al-Aswany>
5	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ayman_Bahgat_Kamar>
6	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Khaled_Diab>
7	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Hosny>
8	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Sherif_Arafa>
9	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mohamed_Diab>
10	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Haitham_Dabbour>
11	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Wael_Hamdy>
12	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mohamed_Sadeq>
13	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Wahid_Hamed>

Showing 1 to 13 of 13 entries

List the instances of the class Director

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?director

WHERE {

  ?director rdf:type table:Director .

}
```



7 results in 0.018 seconds

	director
1	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Khaled_Marei>
2	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mohamed_Gamal_El-Adl>
3	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Sherif_Arafa>
4	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Mohamed_Diab>
5	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Hadi_El_Bagoury>
6	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Marwan_Hamed>
7	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ashry>

Showing 1 to 7 of 7 entries

List the name of all Drama movies and their directors

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?movieName ?directorName

WHERE {

 ?movie rdf:type table:Movie ;

       table:hasGenre table:Drama ;

       table:hasTitle ?movieName ;

       table:hasDirector ?director .

 ?director table:hasName ?directorName .

}
```

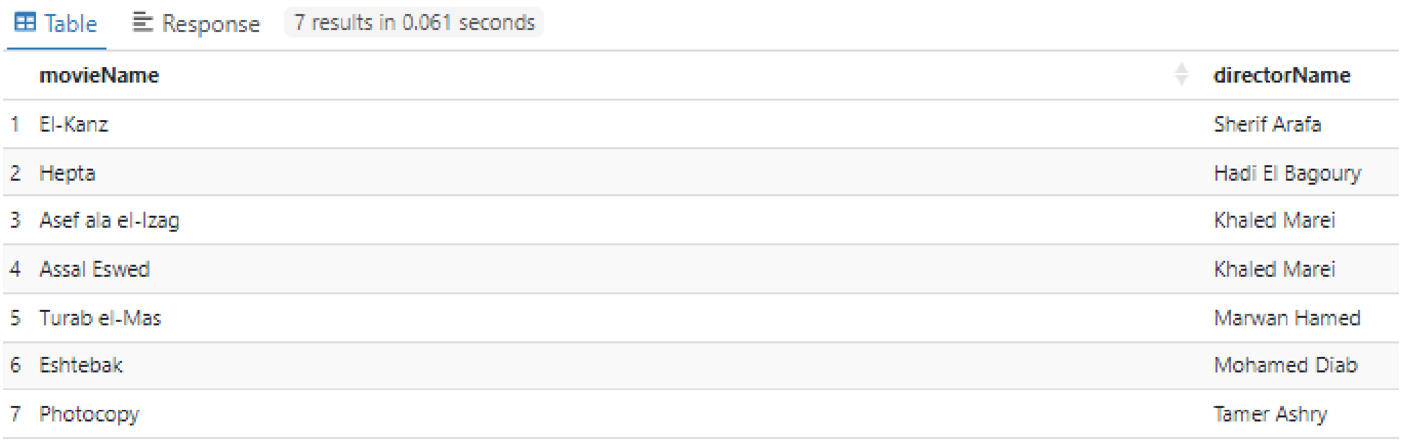


Table Response 7 results in 0.061 seconds

	movieName	directorName
1	El-Kanz	Sherif Arafa
2	Hepta	Hadi El Bagoury
3	Asef ala el-Izag	Khaled Marei
4	Assal Eswed	Khaled Marei
5	Turab el-Mas	Marwan Hamed
6	Eshtebak	Mohamed Diab
7	Photocopy	Tamer Ashry

List the name of all Drama Action movies

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?movieTitle

WHERE {

 ?movie rdf:type table:Movie ;

       table:hasGenre table:Drama ;

       table:hasGenre table:Action ;

       table:hasTitle ?movieTitle .

}}
```

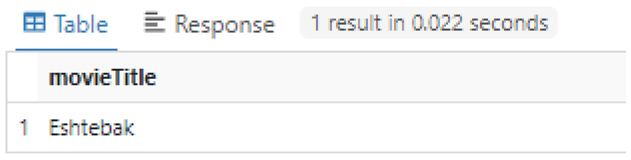


Table Response 1 result in 0.022 seconds

	movieTitle
1	Eshtebak

List the male actors in a specific film

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>
SELECT ?actorName
WHERE {
 ?movie rdf:type table:Movie ;
       table:hasTitle "El-Kanz" .
 ?actor table:isActorOf ?movie ;
       table:hasGender "Male" ;
       table:hasName ?actorName .
}
```

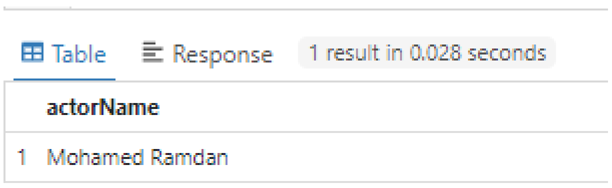


Table Response 1 result in 0.028 seconds

	actorName
1	Mohamed Ramdan

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

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>
SELECT (COUNT(DISTINCT ?movie) as ?numberOfMovies)
WHERE {
 ?movie rdf:type table:Movie ;
       table:hasGenre table:Action ;
       table:hasGenre table:Drama .
}
```

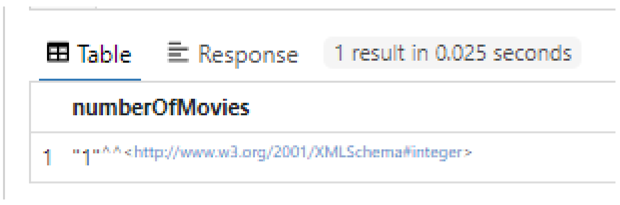


Table Response 1 result in 0.025 seconds

	numberOfMovies
1	"1"^^<http://www.w3.org/2001/XMLSchema#integer>

List all the movies written by a specific writer

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?movieTitle

WHERE {

  ?writer rdf:type table:Writer ;

         table:hasName "Sherif Arafa" .

  ?movie table:hasWriter ?writer ;

        table:hasTitle ?movieTitle .

}
```

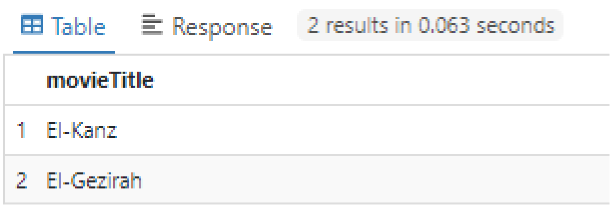


Table Response 2 results in 0.063 seconds

	movieTitle
1	El-Kanz
2	El-Gezirah

Find movies with a certain language

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?movieTitle

WHERE {

  ?movie rdf:type table:Movie ;

        table:hasLanguage "Arabic" ;

        table:hasTitle ?movieTitle .

}
```

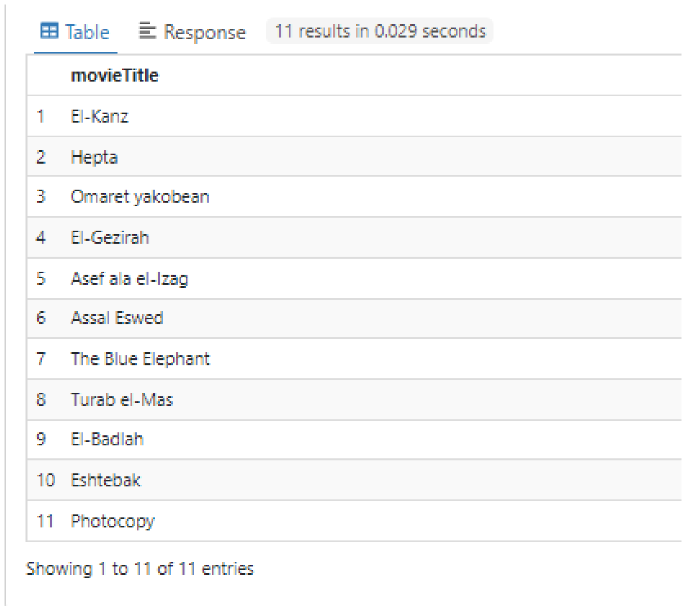


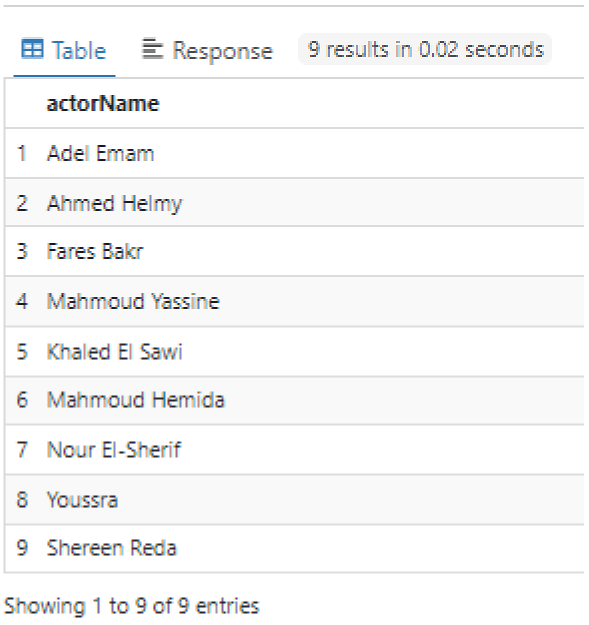
Table Response 11 results in 0.029 seconds

	movieTitle
1	El-Kanz
2	Hepta
3	Omaret yakobean
4	El-Gezirah
5	Asef ala el-Izag
6	Assal Eswed
7	The Blue Elephant
8	Turab el-Mas
9	El-Badlah
10	Eshtebak
11	Photocopy

Showing 1 to 11 of 11 entries

List the name of Actors older than 51 years

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?actorName

WHERE {

 ?actor rdf:type table:Actor ;

       table:hasAge ?age ;

       table:hasName ?actorName .

 FILTER (?age > 51)

}
```

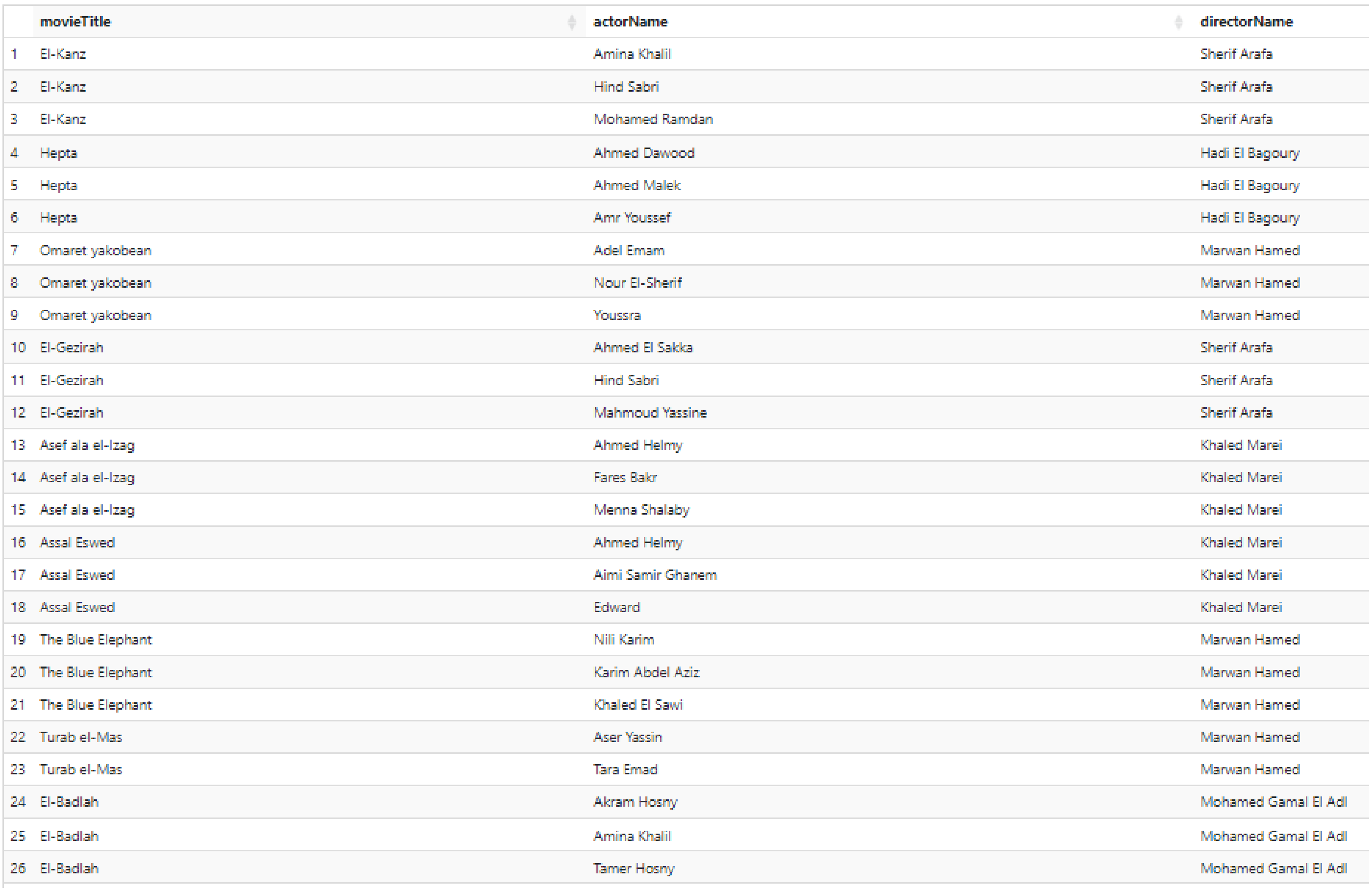


	actorName
1	Adel Emam
2	Ahmed Helmy
3	Fares Bakr
4	Mahmoud Yassine
5	Khaled El Sawi
6	Mahmoud Hemida
7	Nour El-Sherif
8	Youssra
9	Shereen Reda

Showing 1 to 9 of 9 entries

This query selects all movies and optionally includes the names of actors and directors. If a movie does not have an associated actor or director, it will still appear in the results with NULL values for the missing information.

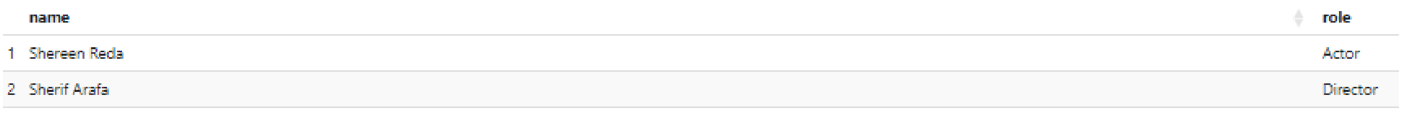
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?movieTitle ?actorName ?directorName

WHERE {

 ?movie rdf:type table:Movie ;

      table:hasTitle ?movieTitle .

 OPTIONAL { ?movie table:hasActor ?actor . ?actor table:hasName ?actorName . }

 OPTIONAL { ?movie table:hasDirector ?director . ?director table:hasName ?directorName . }

}
```



	movieTitle	actorName	directorName
1	El-Kanz	Amina Khalil	Sherif Arafa
2	El-Kanz	Hind Sabri	Sherif Arafa
3	El-Kanz	Mohamed Ramdan	Sherif Arafa
4	Hepta	Ahmed Dawood	Hadi El Bagoury
5	Hepta	Ahmed Malek	Hadi El Bagoury
6	Hepta	Amr Youssef	Hadi El Bagoury
7	Omaret yakobean	Adel Emam	Marwan Hamed
8	Omaret yakobean	Nour El-Sherif	Marwan Hamed
9	Omaret yakobean	Youssra	Marwan Hamed
10	El-Gezirah	Ahmed El Sakka	Sherif Arafa
11	El-Gezirah	Hind Sabri	Sherif Arafa
12	El-Gezirah	Mahmoud Yassine	Sherif Arafa
13	Asef ala el-Izag	Ahmed Helmy	Khaled Marei
14	Asef ala el-Izag	Fares Bakr	Khaled Marei
15	Asef ala el-Izag	Menna Shalaby	Khaled Marei
16	Assal Eswed	Ahmed Helmy	Khaled Marei
17	Assal Eswed	Aimi Samir Ghanem	Khaled Marei
18	Assal Eswed	Edward	Khaled Marei
19	The Blue Elephant	Nili Karim	Marwan Hamed
20	The Blue Elephant	Karim Abdel Aziz	Marwan Hamed
21	The Blue Elephant	Khaled El Sawi	Marwan Hamed
22	Turab el-Mas	Aser Yassin	Marwan Hamed
23	Turab el-Mas	Tara Emad	Marwan Hamed
24	El-Badlah	Akram Hosny	Mohamed Gamal El Adl
25	El-Badlah	Amina Khalil	Mohamed Gamal El Adl
26	El-Badlah	Tamer Hosny	Mohamed Gamal El Adl

This query looks for persons who are either actors or directors (alternatives) and whose name starts with 'S' (conjunction). The **UNION** operator is used to combine the alternatives, and each alternative is linked with the conjunction condition on the name.

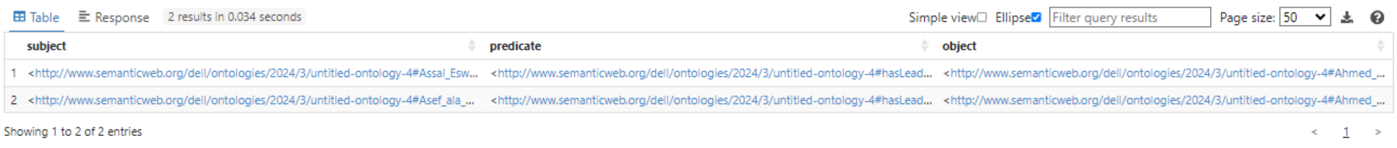
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?name ?role

WHERE {

 {

   ?person rdf:type table:Actor .

   BIND("Actor" AS ?role)

 } UNION {

   ?person rdf:type table:Director .

   BIND("Director" AS ?role)

 }

 ?person table:hasName ?name .

 FILTER (regex(?name, "^S", "i"))

}
```



	name	role
1	Shereen Reda	Actor
2	Sherif Arafa	Director

This **CONSTRUCT** query creates a new set of triples based on the condition that a movie has an actor named Ahmed Helmy. In the new triples, **table:hasLeadActor** is a made-up property that associates movies with their lead actors.

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

CONSTRUCT {

 ?movie table:hasLeadActor ?actor .

}

WHERE {

 ?movie rdf:type table:Movie ; table:hasActor ?actor .

 ?actor table:hasName "Ahmed Helmy" .}
```



2 results in 0.034 seconds

	subject	predicate	object
1	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Assal_Esw...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#hasLead...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_...
2	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Asef_ala_...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#hasLead...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Ahmed_...

Showing 1 to 2 of 2 entries

This **ASK** query checks if there is at least one movie with the title "Pulp Fiction". It returns a boolean result: **true** if such a movie exists, **false** otherwise.

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

ASK {

  ?movie rdf:type table:Movie .

  ?movie table:hasTitle "Hepta" .

}
```

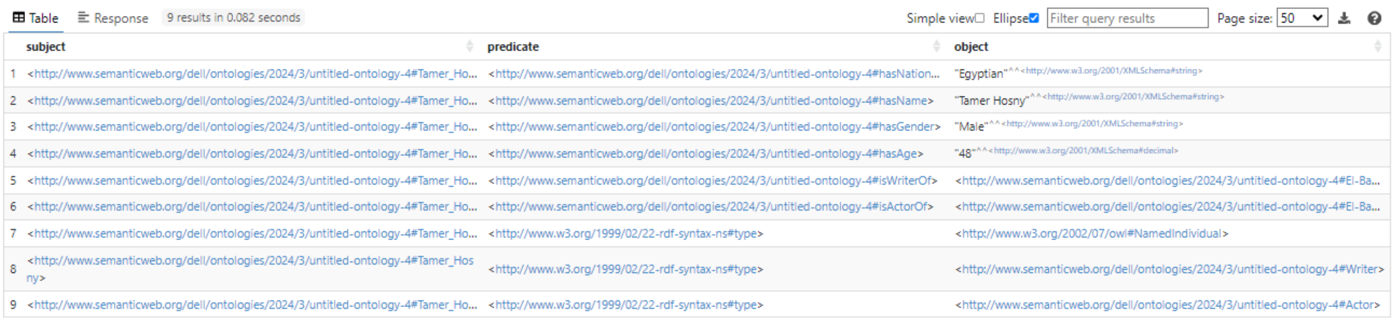
Table
Response

Response in 0.026 seconds

✓ True

This **DESCRIBE** query retrieves all information available about the actor named "Tamer Hosny".

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

DESCRIBE ?actor

WHERE {

  ?actor rdf:type table:Actor ;

        table:hasName "Tamer Hosny" .

}
```



9 results in 0.082 seconds

	subject	predicate	object
1	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#hasNation...	"Egyptian"^^<http://www.w3.org/2001/XMLSchema#string>
2	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#hasName>	"Tamer Hosny"^^<http://www.w3.org/2001/XMLSchema#string>
3	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#hasGender>	"Male"^^<http://www.w3.org/2001/XMLSchema#string>
4	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#hasAge>	"48"^^<http://www.w3.org/2001/XMLSchema#decimal>
5	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#isWriterOf>	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#El-Ba...
6	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#isActorOf>	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#El-Ba...
7	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#NamedIndividual>
8	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Hosny>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Writer>
9	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Tamer_Ho...	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Actor>

Count how many movies each director has directed

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?directorName (COUNT(?movie) AS ?numberOfMovies)

WHERE {

  ?director rdf:type table:Director ;

          table:hasName ?directorName .

  ?movie table:hasDirector ?director .

}

GROUP BY ?directorName
```

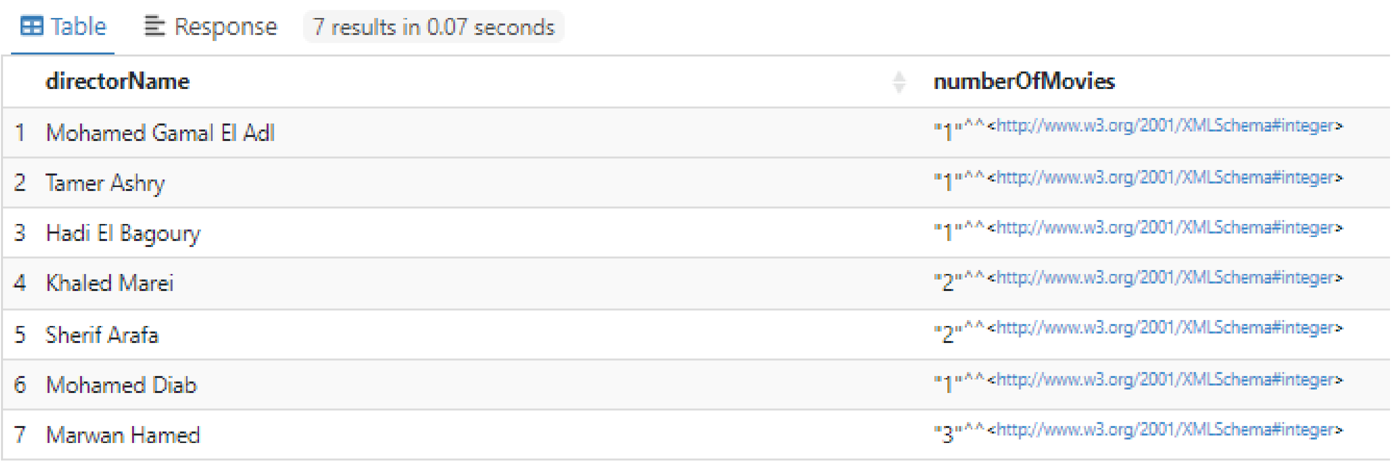
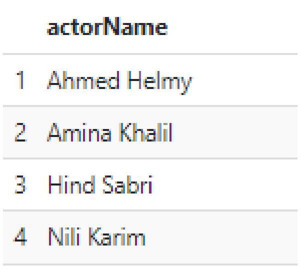


Table   Response   7 results in 0.07 seconds

	directorName	numberOfMovies
1	Mohamed Gamal El Adl	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
2	Tamer Ashry	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
3	Hadi El Bagoury	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
4	Khaled Marei	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
5	Sherif Arafa	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
6	Mohamed Diab	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
7	Marwan Hamed	"3"^^<http://www.w3.org/2001/XMLSchema#integer>

Find actors who have acted in more than 1 movie

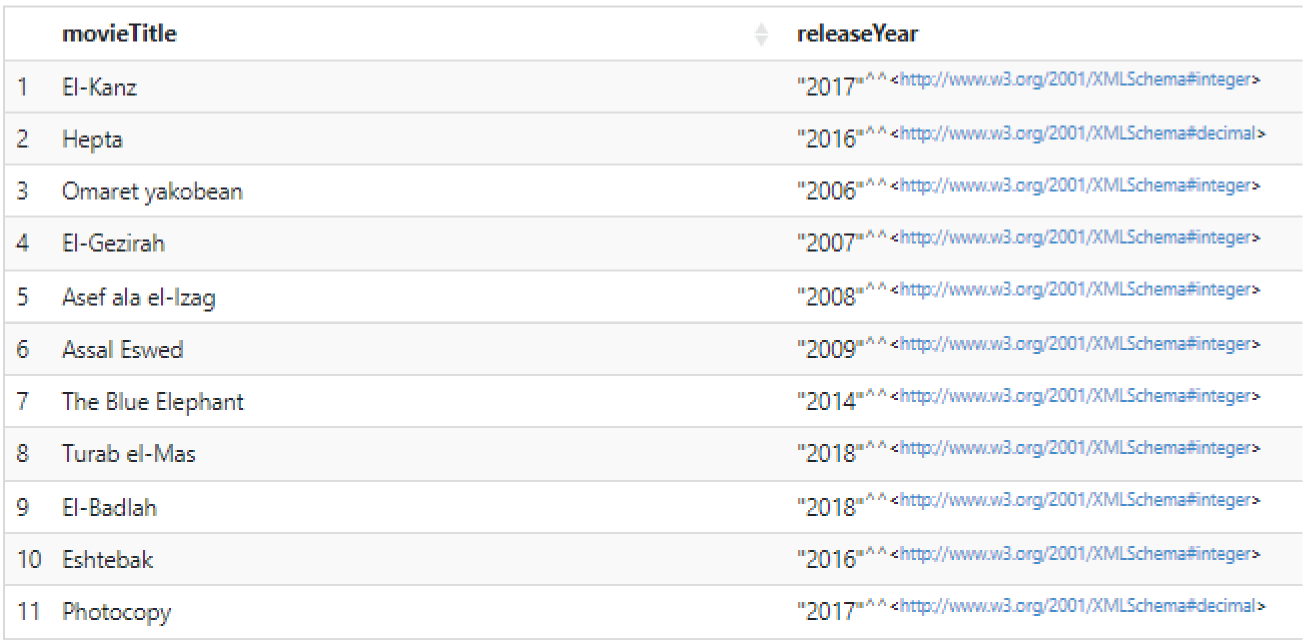
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>

SELECT ?actorName

WHERE {

 ?actor rdf:type table:Actor ;

       table:hasName ?actorName .

 {

  SELECT ?actor (COUNT(?movie) AS ?count)

  WHERE {

   ?movie table:hasActor ?actor .

  }

  GROUP BY ?actor

  HAVING (?count > 1)

 }

}
```



	actorName
1	Ahmed Helmy
2	Amina Khalil
3	Hind Sabri
4	Nili Karim

Find movies released after the year 2000

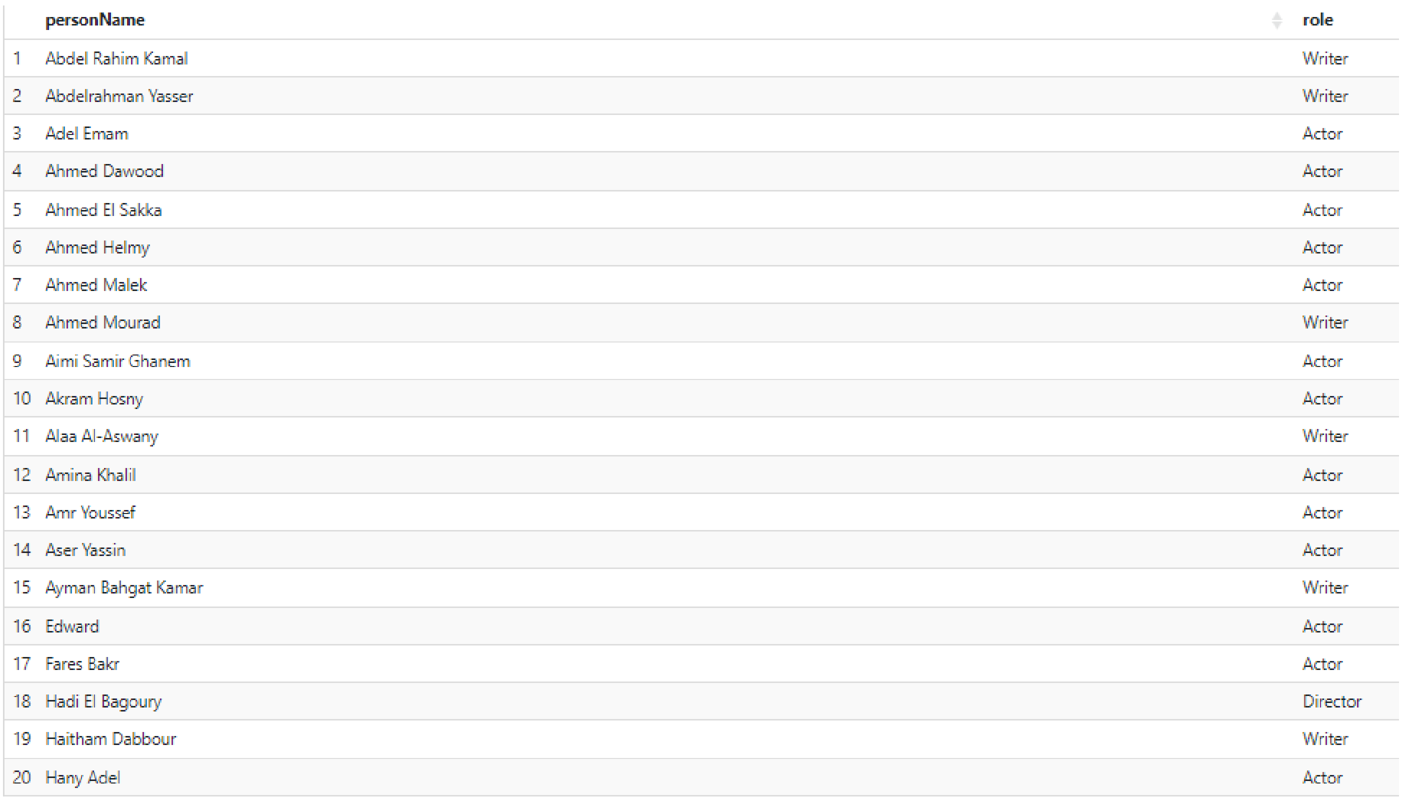
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?movieTitle ?releaseYear

WHERE {

 ?movie rdf:type table:Movie ;

       table:hasTitle ?movieTitle ;

       table:hasYear ?releaseYear .

 FILTER (?releaseYear > 2000)

}
```



	movieTitle	releaseYear
1	El-Kanz	"2017"^^<http://www.w3.org/2001/XMLSchema#integer>
2	Hepta	"2016"^^<http://www.w3.org/2001/XMLSchema#decimal>
3	Omaret yakobean	"2006"^^<http://www.w3.org/2001/XMLSchema#integer>
4	El-Gezirah	"2007"^^<http://www.w3.org/2001/XMLSchema#integer>
5	Asef ala el-Izag	"2008"^^<http://www.w3.org/2001/XMLSchema#integer>
6	Assal Eswed	"2009"^^<http://www.w3.org/2001/XMLSchema#integer>
7	The Blue Elephant	"2014"^^<http://www.w3.org/2001/XMLSchema#integer>
8	Turab el-Mas	"2018"^^<http://www.w3.org/2001/XMLSchema#integer>
9	El-Badlah	"2018"^^<http://www.w3.org/2001/XMLSchema#integer>
10	Eshtebak	"2016"^^<http://www.w3.org/2001/XMLSchema#integer>
11	Photocopy	"2017"^^<http://www.w3.org/2001/XMLSchema#decimal>

List all people and their roles in the film industry

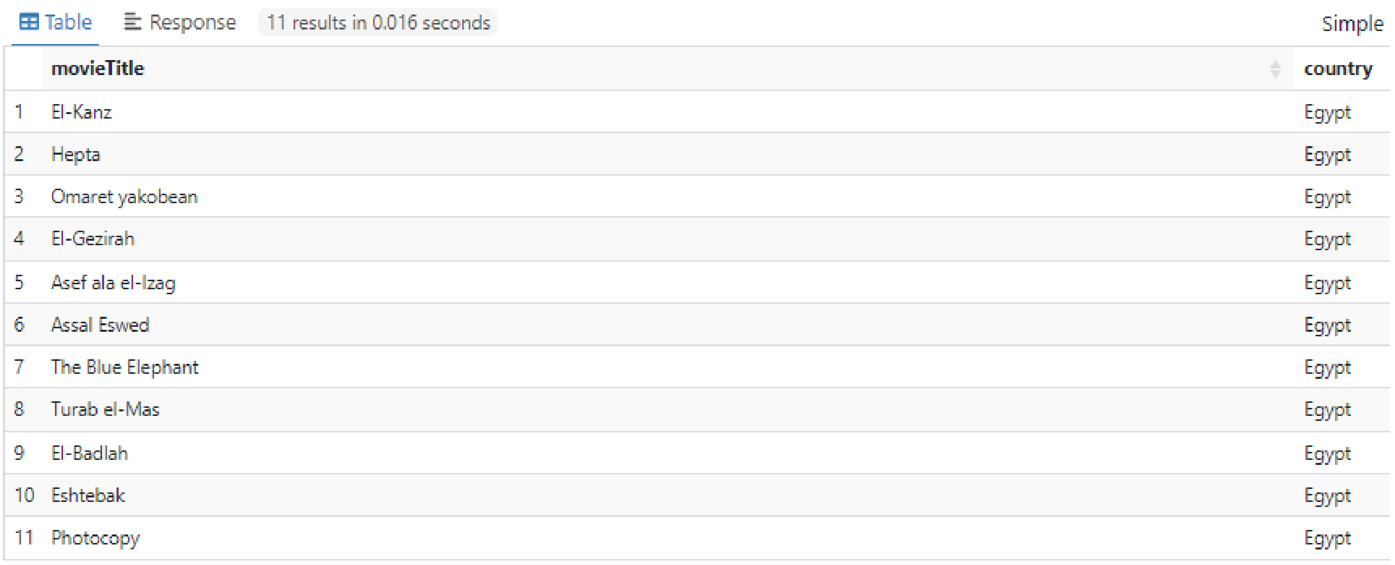
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?personName ?role

WHERE {

 ?person table:hasName ?personName .

 {

  { ?person rdf:type table:Actor . BIND("Actor" AS ?role) }

  UNION

  { ?person rdf:type table:Director . BIND("Director" AS ?role) }

  UNION

  { ?person rdf:type table:Writer . BIND("Writer" AS ?role) }

 }

}
```



	personName	role
1	Abdel Rahim Kamal	Writer
2	Abdelrahman Yasser	Writer
3	Adel Emam	Actor
4	Ahmed Dawood	Actor
5	Ahmed El Sakka	Actor
6	Ahmed Helmy	Actor
7	Ahmed Malek	Actor
8	Ahmed Mourad	Writer
9	Aimi Samir Ghanem	Actor
10	Akram Hosny	Actor
11	Alaa Al-Aswany	Writer
12	Amina Khalil	Actor
13	Amr Youssef	Actor
14	Aser Yassin	Actor
15	Ayman Bahgat Kamar	Writer
16	Edward	Actor
17	Fares Bakr	Actor
18	Hadi El Bagoury	Director
19	Haitham Dabbour	Writer
20	Hany Adel	Actor

List movies and their country of origin

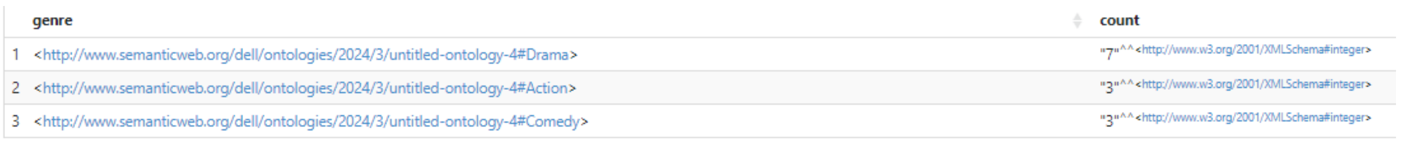
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?movieTitle ?country

WHERE {

  ?movie rdf:type table:Movie ;

        table:hasTitle ?movieTitle ;

        table:hasCountry ?country .

}
```



	movieTitle	country
1	El-Kanz	Egypt
2	Hepta	Egypt
3	Omaret yakobean	Egypt
4	El-Gezirah	Egypt
5	Asef ala el-Izag	Egypt
6	Assal Eswed	Egypt
7	The Blue Elephant	Egypt
8	Turab el-Mas	Egypt
9	El-Badlah	Egypt
10	Eshtebak	Egypt
11	Photocopy	Egypt

Count the number of movies for each genre

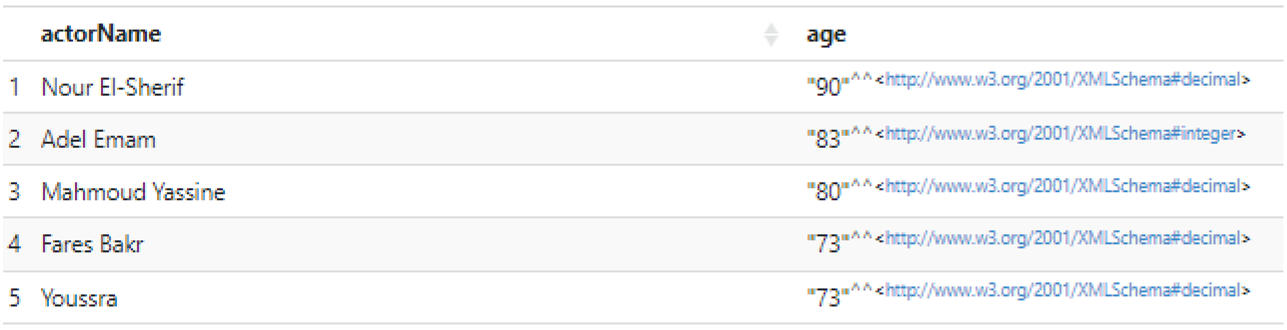
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?genre (COUNT(?movie) AS ?count)

WHERE {

  ?movie table:hasGenre ?genre .

}

GROUP BY ?genre
```



	genre	count
1	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Drama>	"7"^^<http://www.w3.org/2001/XMLSchema#integer>
2	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Action>	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
3	<http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#Comedy>	"3"^^<http://www.w3.org/2001/XMLSchema#integer>

Find the oldest actors

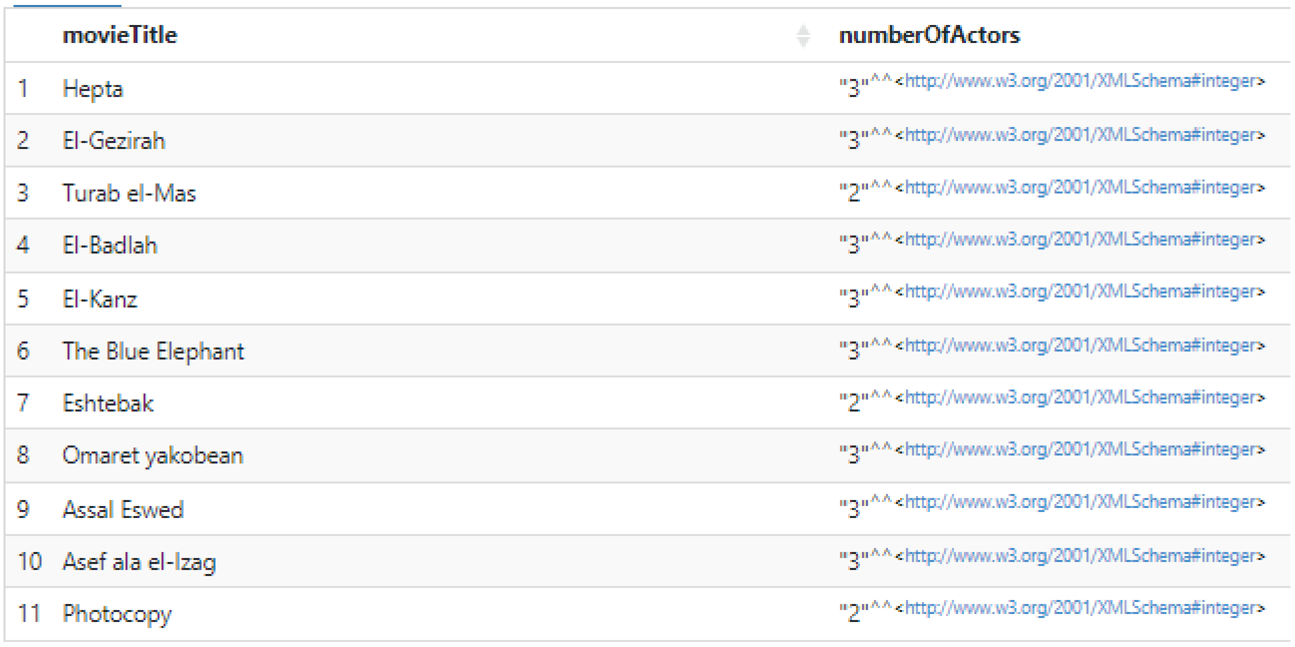
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?actorName ?age

WHERE {

 ?actor rdf:type table:Actor ;

     table:hasName ?actorName ;

     table:hasAge ?age .

}

ORDER BY DESC(?age)

LIMIT 5
```



	actorName	age
1	Nour El-Sherif	"90"^^<http://www.w3.org/2001/XMLSchema#decimal>
2	Adel Emam	"83"^^<http://www.w3.org/2001/XMLSchema#integer>
3	Mahmoud Yassine	"80"^^<http://www.w3.org/2001/XMLSchema#decimal>
4	Fares Bakr	"73"^^<http://www.w3.org/2001/XMLSchema#decimal>
5	Youssra	"73"^^<http://www.w3.org/2001/XMLSchema#decimal>

List all movies and the number of actors in each

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

PREFIX table: <http://www.semanticweb.org/dell/ontologies/2024/3/untitled-ontology-4#>


SELECT ?movieTitle (COUNT(?actor) AS ?numberOfActors)

WHERE {

 ?movie rdf:type table:Movie ;

       table:hasTitle ?movieTitle .

 ?movie table:hasActor ?actor .

}

GROUP BY ?movieTitle
```



	movieTitle	numberOfActors
1	Hepta	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
2	El-Gezirah	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
3	Turab el-Mas	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
4	El-Badlah	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
5	El-Kanz	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
6	The Blue Elephant	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
7	Eshtebak	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
8	Omaret yakobean	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
9	Assal Eswed	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
10	Asef ala el-Izag	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
11	Photocopy	"2"^^<http://www.w3.org/2001/XMLSchema#integer>

Phase 2

Manipulating the ontology

1. **Q1**: Displays all persons by inference without using a SPARQL query.
2. **Q2**: Uses a SPARQL query to display all persons and outlines the approach to write this query to a text file.
3. **Q3**: Lists all actors using inference from the RDF graph, not utilizing direct queries.
4. **Q4**: Provides functionality to input a movie name and display its details (year, country, genres, and actors) if the movie exists.
5. **Q5**: Explores relations involving actors who are also writers, implementing rules from an external file.
6. **Q6**: Focuses on specific rules related to movies and actors, including calculating movie counts and classifying actors as old or young based on criteria.

Pizza Finder Application

Welcome to the Film Finder application. This application leverages the power of the Semantic Web, specifically utilizing RDF (Resource Description Framework) and SPARQL, to provide a dynamic and flexible way to search for films based on a variety of criteria. Built upon an ontology designed to represent intricate relationships within the film industry, this application offers users the ability to include or exclude films in their search results based on actors, directors, genres, and more.

Purpose

The primary purpose of the Film Finder is to demonstrate how semantic technologies can be applied to create sophisticated query capabilities in an application that is both user-friendly and efficient. By using RDF and SPARQL, the application can perform complex queries that take into account multiple attributes and relationships, providing tailored results that match the specific preferences or requirements of the user.

Application Structure

The Film Finder application consists of several key components:

1. **RDF Graph**: At the core of the application is an RDF graph that stores all data about films, actors, directors, genres, and other relevant entities. This graph is constructed and queried using SPARQL to retrieve information based on user-defined filters.
2. **Ontology**: The ontology defines the structure of the RDF data and includes classes and properties that represent the various elements and relationships in the film domain. This ontology is crucial for ensuring that the data is well-organized and that the queries are semantically meaningful.
3. **User Interface**: Built with Tkinter, the GUI provides a simple and intuitive interface for users to specify their search criteria. Users can input names of actors, directors, or genres to include or exclude from their search, and the results are displayed in a readable format.
4. **Query Functionality**: The application contains a sophisticated query system that constructs and executes SPARQL queries based on user inputs. This system includes error handling and optimizations to ensure that queries are not only accurate but also perform efficiently.
5. **Integration and Testing**: The integration of the RDF backend with the Tkinter frontend is meticulously handled to provide a seamless user experience. Additionally, the application includes test cases to ensure that all components work correctly and that the ontology supports the required queries.

Graphical User Interface (GUI)

The Graphical User Interface (GUI) of the Film Finder application is designed to provide an intuitive and user-friendly platform for users to search for films based on specific criteria such as actors, directors, and genres. This document describes the layout, components, and functionalities of the GUI.

GUI Layout and Components

Main Window

- **Title**: The window is titled "Film Finder," clearly indicating the purpose of the application.

Input Fields

1. **Include Actors (comma-separated)**:
    - **Description**: Allows users to enter the names of actors to include in the search.
2. **Exclude Actors (comma-separated)**:
    - **Description**: Allows users to specify actor names whose associated films should be excluded from the results.
3. **Include Directors (comma-separated)**:
    - **Description**: Permits the inclusion of films directed by the specified directors.
4. **Exclude Directors (comma-separated)**:
    - **Description**: Enables users to exclude films directed by specified directors from the search results.
5. **Include Genres (comma-separated)**:
    - **Description**: Allows users to filter films by including specific genres.
6. **Exclude Genres (comma-separated)**:
    - **Description**: Users can exclude films of specific genres from their search results.

Buttons

- **Find Films**:
    - **Description**: Initiates the search based on the specified criteria.
    - **Function**: Calls the **filter_movies()** function which processes the input and updates the results display.

Results Display

- **Scrolled Text Widget**:
    - **Description**: Displays the list of films that match the search criteria or a message if no films are found.

Functional Description

The Film Finder GUI operates by allowing users to input their search criteria across multiple dimensions (actors, directors, genres). The system uses these inputs to construct a SPARQL query dynamically, querying an underlying RDF graph populated with film data.

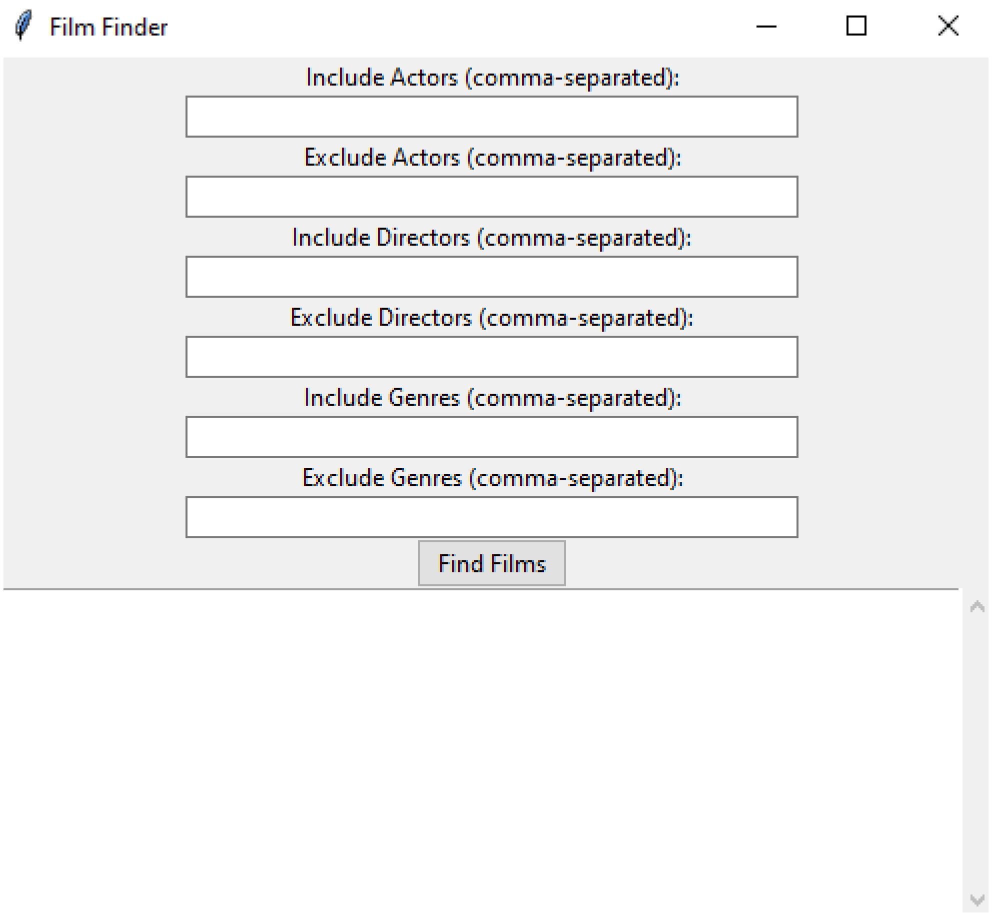
1. **Input Processing**:
   - Users can enter names or titles separated by commas in the respective fields.
   - The inputs are stripped of extra spaces and processed to form part of the SPARQL query filters.
2. **Query Execution**:
   - Upon clicking the "Find Films" button, the **filter_movies()** function is executed.
   - This function constructs and runs the SPARQL query using the user-provided criteria.
   - Results are fetched and displayed in the Scrolled Text Widget.
3. **Result Handling**:
   - If the query finds matching films, their titles are listed in the results area.
   - If no matches are found, a "No movies found" message is displayed.

Interaction Flow

1. **User enters search criteria** into the various input fields.
2. **User clicks "Find Films" button** to submit the query.
3. **Application displays results** in the scrolled text widget, either listing the film titles that match the query or indicating that no matches were found.

Error Handling

- **Input Validation**: Ensures that the text fields do not process entries as valid criteria if only spaces are entered.
- **SPARQL Query Errors**: Any errors in query execution due to malformed inputs or backend issues are caught and logged, though the user is simply shown "No movies found."


Film Finder
Include Actors (comma-separated):
Exclude Actors (comma-separated):
Include Directors (comma-separated):
Exclude Directors (comma-separated):
Include Genres (comma-separated):
Exclude Genres (comma-separated):
Find Films

Data Flow Diagram (DFD)

This Data Flow Diagram (DFD) illustrates the flow of data within the Film Finder project. It maps out how data is processed and exchanged between different processes and data stores within the system, facilitating the querying and retrieval of movie data based on user-specified filters.

Processes and Data Stores

1. **PROJECT**
   - **Function:** Acts as the central process coordinating the major activities of the Film Finder application.
   - **Data Flows:**
     - Receives user input data from the **UI**.
     - Sends commands to **RDF_GRAPH** to fetch or update RDF data.
2. **RDF_GRAPH**
   - **Function:** Manages the RDF data interactions.
   - **Data Flows:**
     - Retrieves RDF data from the **FILE**.
     - Processes and sends query results back to the **PROJECT**.
3. **FILE**
   - **Function:** Acts as the primary data store for the RDF data.
   - **Data Flows:**
     - Stores the RDF file, facilitating data access by the **RDF_GRAPH**.
4. **UI**
   - **Function:** Provides the graphical user interface through which the **USER** interacts with the system.
   - **Data Flows:**
     - Collects input from the **USER**.
     - Displays query results received from the **PROJECT**.
5. **USER**
   - **Function:** The end user who interacts with the Film Finder system.
   - **Data Flows:**
     - Submits filter criteria through the **UI**.
     - Receives filtered movie results via the **UI**.
6. **QUERY**
   - **Function:** Represents the process of querying the RDF data based on user inputs.
   - **Data Flows:**
     - Receives input parameters from the **PROJECT**.
     - Sends results data back to the **PROJECT** after processing.

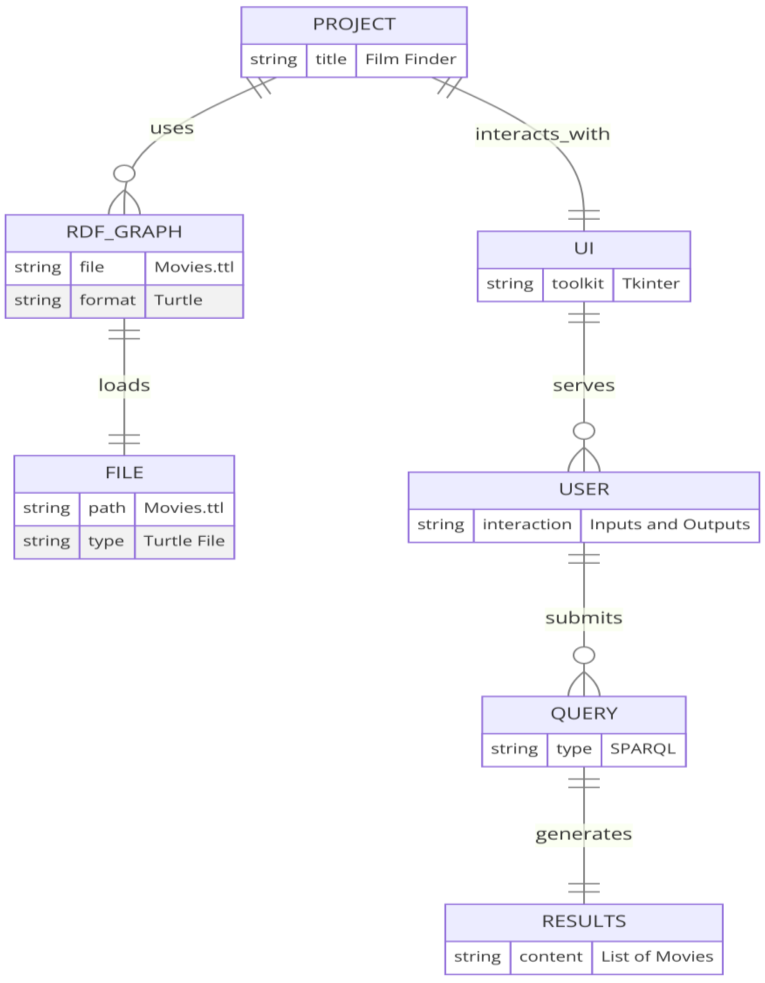
7. **RESULTS**

- **Function:** The output generated from the **QUERY** process.
- **Data Flows:**
    - Delivers a list of movies to the **UI** for display to the **USER**.

Functionality

- The **PROJECT** acts as the central coordinator, interfacing with the **UI** and managing data flow to and from the **RDF_GRAPH**.
- The **RDF_GRAPH** processes RDF data stored in the **FILE**, executes queries, and returns results.
- The **UI** serves as the interface for **USER** interaction, collecting input for queries and displaying results.
- The **USER** interacts with the system through the **UI**, initiating queries with specific filters.
- The **QUERY** process encapsulates the logic to filter and retrieve movie data, sending back the results to be displayed.

This DFD provides a comprehensive view of how data is processed and transferred within the Film Finder application, detailing the interactions between user inputs, system processes, and data storage.


PROJECT
string | title | Film Finder
uses
interacts_with
RDF_GRAPH
string | file | Movies.ttl
string | format | Turtle
UI
string | toolkit | Tkinter
loads
serves
FILE
string | path | Movies.ttl
string | type | Turtle File
USER
string | interaction | Inputs and Outputs
submits
QUERY
string | type | SPARQL
generates
RESULTS
string | content | List of Movies

Github

This repository contains the RDF and OWL files for the Movies Ontology project, along with the source code for Phase 4 and 5 of the project.

RDF File

The RDF file (**movies.rdf**) contains the data represented in RDF (Resource Description Framework) format. RDF is used to describe resources on the web, making it easier to share data between different applications.

OWL File

The OWL file (**movies.owl**) contains the ontology schema defined using the Web Ontology Language (OWL). OWL is used to formally describe the classes, properties, and relationships within a domain, in this case, the domain of movies.

Source Code

The source code for Phase 4 and 5 of the project is available in the **src** directory. Phase 4 and 5 likely refer to specific stages or iterations of development, each adding new features or functionality to the ontology.

For more information about the Movies Ontology project, please visit the GitHub repository.