Report on Assignment 2 - Querying Knowledge Graphs with SPARQL
- Yash Lucas (12433688)

In this assignment, we explored the creation and execution of both simple and advanced SPARQL queries using the **GraphDB environment**. The project focused on developing and querying a custom **film ontology** and working with the **DBPedia knowledge base**. Below is a detailed breakdown of the work:

Task 1: Film Ontology Implementation

- **Option Selection**: For Job 1, we selected **Option B**, i.e. VS Code which required implementing a custom **film ontology**.
- **Ontology Creation**: The film ontology was designed and implemented using **Visual Studio Code**. Classes, properties, and instances were structured to represent films, directors, genres, studios, and other relationships in RDF format.
- **SPARQL Queries**: Various SPARQL queries were written to retrieve meaningful information from the ontology. These included:
    - Retrieving films and their respective directors.
    - Fetching films by their associated studios.
    - Combining multiple patterns using the `UNION` clause to retrieve films along with their directors and studios in a single query.
- **Environment**: The ontology was programmed on vs code and loaded into **GraphDB**, where all queries were executed and tested for accuracy.

Task 1 :

Q1 : **__SELECT__** Return all actors with their names

All actors and their complete names are retrieved from the dataset using this SPARQL query. This is how it operates:
actor, for ex:Actor: Filters for people of type The namespace ex is an actor.
?actor ex:fullName?name: Retrieves each actor's complete name (?name) (?actor).
All actors are listed in the result, along with their complete names.

```
PREFIX ex: <http://semantics.id/ns/example/film#>
PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema-ns#>

SELECT ?actor ?name
WHERE {
  ?actor a ex:Actor.
  ?actor ex:fullName ?name.
}
```

```
PREFIX ex: <http://semantics.id/ns/example/film#>
PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema-ns#>

SELECT ?actor ?name
WHERE {
  ?actor a ex:Actor.
  ?actor ex:fullName ?name.
}
```

✓ 0.0s

actor	name
<http://semantics.id/ns/example#isabelle_huppert>	Isabelle Huppert
<http://semantics.id/ns/example#annie_girardot>	Annie Girardot
<http://semantics.id/ns/example#juliette_binoche>	Juliette Binoche
<http://semantics.id/ns/example#jean-louis_trintignant>	Jean-Louis Trintignant
<http://semantics.id/ns/example#ralph_fiennes>	Ralph Fiennes
<http://semantics.id/ns/example#william_dafoe>	William Dafoe
ex:harrison_ford	Harrison Ford
ex:ryan_gosling	Ryan Gosling
ex:kathleen_quinlan	Kathleen Quinlan
ex:david_keith	David Keith
ex:will_smith	Will Smith
ex:jeff_goldblum	Jeff Goldblum

> Q2: __ASK__ Is there a film directed by Michael Haneke after 2020?

This SPARQL query checks if there are any films directed by **Michael Haneke** that were released **after the year 2020**. Here's a breakdown:

1. **ASK query**: Returns true or false depending on whether the specified condition is satisfied.
2. **?film mv:hasDirector ex:michael_haneke**: Filters films that have Michael Haneke as their director.
3. **mv:releaseYear ?year**: Retrieves the release year of the filtered films.
4. **FILTER(?year > 2020)**: Ensures only films released after 2020 are considered.

If such films exist, the query will return true; otherwise, it will return false.

```
PREFIX mv: <http://semantics.id/ns/example/film#>
PREFIX ex: <http://semantics.id/ns/example#>

ASK {
  ?film mv:hasDirector ex:michael_haneke ;
        mv:releaseYear ?year .
  FILTER(?year > 2020)
}
```

```
PREFIX mv: <http://semantics.id/ns/example/film#>
PREFIX ex: <http://semantics.id/ns/example#>

ASK {
  ?film mv:hasDirector ex:michael_haneke ;
        mv:releaseYear ?year .
  FILTER(?year > 2020)
}

✓ 0.0s
```

✗ no

> Q3: __DESCRIBE__ Give me all information about the film 'Independence Day' released in 1996

This SPARQL query retrieves detailed information about the film **"Independence Day"** released in **1996** using the DESCRIBE keyword. Here's a brief explanation:

1. **DESCRIBE ?film**: Requests all RDF triples associated with the film that matches the criteria, providing a detailed representation of the film resource.
2. **rdfs:label "Independence Day"**: Filters films with the label "Independence Day."
3. **ev:releaseYear 1996**: Ensures the selected film was released in the year 1996.

```
PREFIX ev: <http://semantics.id/ns/example/film#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
describe ?film
where {
  ?film rdfs:label "Independence Day";
  ev:releaseYear 1996.}
```

```
PREFIX ev: <http://semantics.id/ns/example/film#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
describe ?film
where {
  ?film rdfs:label "Independence Day";
  ev:releaseYear 1996.}

✓ 0.0s

@prefix ev: <http://semantics.id/ns/example/film#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdf4j: <http://rdf4j.org/schema/rdf4j#> .
@prefix sesame: <http://www.openrdf.org/schema/sesame#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix fn: <http://www.w3.org/2005/xpath-functions#> .

<http://semantics.id/ns/example#film_10> a ev:Film, ev:Artwork, owl:NamedIndividual;
  rdfs:label "Independence Day";
  ev:hasActor ev:will_smith, ev:jeff_goldblum;
  ev:hasPerformer ev:will_smith, ev:jeff_goldblum;
  ev:hasCrew <http://semantics.id/ns/example#roland_emmerich>, <http://semantics.id/ns/example#dean_devlin>;
  ev:hasDirector <http://semantics.id/ns/example#roland_emmerich>;
  ev:hasFilmStudio <http://semantics.id/ns/example#twentieth_century_fox>;
  ev:hasGenre ev:genre_science_fiction, ev:genre_action;
  ev:hasScriptWriter <http://semantics.id/ns/example#roland_emmerich>, <http://semantics.id/ns/example#dean_devlin>;
  ev:releaseYear 1996 .
```

> Q4: __CONSTRUCT__ Return the directors and script writers who have worked together. You may use :collaboratedWith as the newly constructed property

This SPARQL query uses the CONSTRUCT keyword to generate new RDF triples showing collaborations between directors and scriptwriters.:

1. **CONSTRUCT { ?director mv:collaboratedWith ?scriptWriter . }**: Creates a new triple stating that a director has collaborated with a scriptwriter.
2. **WHERE { ... }**: Specifies the conditions for identifying collaborations:
   - **?film mv:hasDirector ?director ; mv:hasScriptWriter ?scriptWriter .**: Selects films that have both a director and a scriptwriter.
   - **FILTER(?director != ?scriptWriter)**: Ensures the director and scriptwriter are not the same individual.
   - **?director mv:fullName ?directorName . ?scriptWriter mv:fullName ?scriptWriterName .**: Ensures that full names for the director and scriptwriter are available.

```
PREFIX mv: <http://semantics.id/ns/example/film#>
CONSTRUCT {
  ?director mv:collaboratedWith ?scriptWriter .
}
WHERE {
  ?film mv:hasDirector ?director ;
        mv:hasScriptWriter ?scriptWriter .
  FILTER(?director != ?scriptWriter)
  ?director mv:fullName ?directorName .
  ?scriptWriter mv:fullName ?scriptWriterName .
}
```

```
    ?director mv:fullName ?directorName .
    ?scriptWriter mv:fullName ?scriptWriterName .
  }

✓ 0.0s

@prefix mv: <http://semantics.id/ns/example/film#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf4j: <http://rdf4j.org/schema/rdf4j#> .
@prefix sesame: <http://www.openrdf.org/schema/sesame#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix fn: <http://www.w3.org/2005/xpath-functions#> .

<http://semantics.id/ns/example#paul_verhoeven> mv:collaboratedWith <http://semantics.id/ns/example#david_birke> .

<http://semantics.id/ns/example#ridley_scott> mv:collaboratedWith <http://semantics.id/ns/example#hampton_fancher>,
    <http://semantics.id/ns/example#david_peoples> .

<http://semantics.id/ns/example#denis_villeneuve> mv:collaboratedWith <http://semantics.id/ns/example#hampton_fancher>,
    <http://semantics.id/ns/example#michael_green> .

<http://semantics.id/ns/example#robert_mandel> mv:collaboratedWith <http://semantics.id/ns/example#alice_hoffman> .

<http://semantics.id/ns/example#roland_emmerich> mv:collaboratedWith <http://semantics.id/ns/example#dean_devlin> .
```

> Q5: __CONSTRUCT__ Return the directors and films where the director is both director and script writer. You may use :directorandwriterof as the newly constructed property

This SPARQL query identifies individuals who are both the **director** and **scriptwriter** of the same film and generates RDF triples to represent this relationship explicitly.

1. **CONSTRUCT { ?director mv:directorandwriterof ?film . }**:
   - Creates new RDF triples linking directors to the films they directed *and* wrote.
   - The property mv:directorandwriterof is used to represent this relationship.
2. **WHERE { ... }**:
   - **?film mv:hasDirector ?director ; mv:hasScriptWriter ?scriptWriter .**: Selects films with both a director and a scriptwriter.
   - **FILTER(?director = ?scriptWriter)**: Ensures the director and scriptwriter are the same person.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>
CONSTRUCT {
  ?director mv:directorandwriterof ?film .
}
WHERE {
  ?film mv:hasDirector ?director ;
        mv:hasScriptWriter ?scriptWriter .
  FILTER(?director = ?scriptWriter)
}
```

```
WHERE {
  ?film mv:hasDirector ?director ;
        mv:hasScriptWriter ?scriptWriter .
  FILTER(?director = ?scriptWriter)
}
```

✓ 0.0s

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mv: <http://semantics.id/ns/example/film#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdf4j: <http://rdf4j.org/schema/rdf4j#> .
@prefix sesame: <http://www.openrdf.org/schema/sesame#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix fn: <http://www.w3.org/2005/xpath-functions#> .

<http://semantics.id/ns/example#michael_haneke> mv:directorandwriterof <http://semantics.id/ns/example#film_1>,
    <http://semantics.id/ns/example#film_2>, <http://semantics.id/ns/example#film_3>,
    <http://semantics.id/ns/example#film_4> .

<http://semantics.id/ns/example#anthony_minghella> mv:directorandwriterof <http://semantics.id/ns/example#film_5> .

<http://semantics.id/ns/example#roland_emmerich> mv:directorandwriterof <http://semantics.id/ns/example#film_10> .
```

> Q6: __FILTER__ Return all films with 'Blade Runner' in their titles

This SPARQL query retrieves all films whose titles contain the phrase **"Blade Runner"**, regardless of case.

1. **SELECT ?film ?title**:
   - The query selects two variables: ?film (the film's URI) and ?title (the film's title).
2. **?film rdfs:label ?title .**:
   - Matches films (?film) with their respective titles (?title) using the rdfs:label property.
3. **FILTER(REGEX(?title, ".*Blade Runner.*", "i"))**:
   - Applies a regular expression filter to check if the title contains the phrase **"Blade Runner"**:
     - .* allows for any characters before or after the phrase.
     - "i" makes the search case-insensitive.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>
SELECT ?film ?title
WHERE {
  ?film rdfs:label ?title .
  FILTER(REGEX(?title, ".*Blade Runner.*", "i"))
}
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>

SELECT ?film ?title
WHERE {
  ?film rdfs:label ?title .
  FILTER(REGEX(?title, ".*Blade Runner.*", "i"))
}
```

✓ 0.0s

film	title
<http://semantics.id/ns/example#film_7>	Blade Runner
<http://semantics.id/ns/example#film_8>	Blade Runner 2049

> Q7: __FILTER__ Return all the names of directors who made any films in 1990 or earlier

This SPARQL query retrieves the **distinct names of directors** who directed films released on or before the year **1990**.

1. **SELECT DISTINCT ?directorName**:
   - Ensures that the output will only include unique (DISTINCT) director names.
2. **?film mv:hasDirector ?director ; mv:releaseYear ?year .**:
   - Matches films (?film) that have a director (?director) and a release year (?year).
3. **FILTER(?year <= 1990)**:
   - Filters the films to include only those released in **1990 or earlier**.
4. **?director mv:fullName ?directorName .**:
   - Retrieves the full name (?directorName) of each director (?director).

```
PREFIX mv: <http://semantics.id/ns/example/film#>
SELECT DISTINCT ?directorName
WHERE {
  ?film mv:hasDirector ?director ;
        mv:releaseYear ?year .
  FILTER(?year <= 1990)

  ?director mv:fullName ?directorName .
}
```

```
WHERE {
  ?film mv:hasDirector ?director ;
        mv:releaseYear ?year .
  FILTER(?year <= 1990)

  ?director mv:fullName ?directorName .
}
```

0.0s Endpoint Connection Cell SPARQL

directorName
Ridley Scott
Robert Mandel

> Q8: __ORDER and GROUP__ Return the actor with number of films they starred in, in descending order

This SPARQL query retrieves a list of actors along with the count of films they have acted in, ordered by the number of films in descending order. Here's a breakdown of the query:

1. **SELECT ?ActorName (COUNT(?film) AS ?filmCount)**:
   - Retrieves the name of each actor (?ActorName) and the count of films (?filmCount) they have acted in.
2. **?film mv:hasActor ?actor .**:
   - Matches films (?film) that have an actor (?actor).
3. **?actor mv:fullName ?ActorName .**:
   - Retrieves the full name of the actor.
4. **GROUP BY ?ActorName**:
   - Groups the results by each actor's name to calculate the total number of films for each actor.
5. **ORDER BY DESC(?filmCount)**:
   - Orders the results by the film count in descending order, so actors with the most films appear at the top.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>
SELECT ?ActorName (COUNT(?film) AS ?filmCount)
WHERE {
  ?film mv:hasActor ?actor .
  ?actor mv:fullName ?ActorName .
}
Group BY ?ActorName
ORDER BY DESC(?filmCount)
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>

SELECT ?ActorName (COUNT(?film) AS ?filmCount)
WHERE {
  ?film mv:hasActor ?actor .
  ?actor mv:fullName ?ActorName .
}
Group BY ?ActorName
ORDER BY DESC(?filmCount)
```

✓ 0.0s Endpoint C

ActorName	filmCount
Isabelle Huppert	4^^xsd:integer
Annie Girardot	2^^xsd:integer
Juliette Binoche	2^^xsd:integer
Jean-Louis Trintignant	2^^xsd:integer
Harrison Ford	2^^xsd:integer
Ralph Fiennes	1^^xsd:integer
William Dafoe	1^^xsd:integer
Ryan Gosling	1^^xsd:integer
Kathleen Quinlan	1^^xsd:integer
David Keith	1^^xsd:integer
Will Smith	1^^xsd:integer
Jeff Goldblum	1^^xsd:integer

> Q9: __ORDER and GROUP__ Return the number of actors in each film, in ascending order of their release year

This SPARQL query retrieves information about films, their release years, and the count of actors associated with each film, ordered by the release year in ascending order. Here's a breakdown:

1. **SELECT ?film ?releaseYear (COUNT(?actor) AS ?Actor_Count)**:
   - Retrieves the film's URI (?film), its release year (?releaseYear), and the total number of actors (?Actor_Count) in each film.
2. **?film rdfs:label ?title ; mv:hasActor ?actor ; mv:releaseYear ?releaseYear .**:
   - Matches films with their titles (?title), associated actors (?actor), and release years (?releaseYear).
3. **GROUP BY ?film ?title ?releaseYear**:
   - Groups the results by film URI, title, and release year to calculate the actor count for each film.
4. **ORDER BY ASC(?releaseYear)**:
   - Orders the results by the release year in ascending order, so older films appear first.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>
SELECT ?film ?releaseYear(COUNT(?actor) AS ?Actor_Count)
WHERE {
  ?film rdfs:label ?title ;
  mv:hasActor ?actor ;
   mv:releaseYear ?releaseYear.
}
GROUP BY ?film ?title ?releaseYear
ORDER BY ASC(?releaseYear)
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>

SELECT ?film ?releaseYear(COUNT(?actor) AS ?Actor_Count)
WHERE {
  ?film rdfs:label ?title ;
  mv:hasActor ?actor ;
   mv:releaseYear ?releaseYear.
}
GROUP BY ?film ?title ?releaseYear
ORDER BY ASC(?releaseYear)
```

[34] ✓ 0.0s Endpoint Connection Cell SP

film	releaseYear	Actor_Count
<http://semantics.id/ns/example#film_7>	1982^^xsd:integer	1^^xsd:integer
<http://semantics.id/ns/example#film_9>	1983^^xsd:integer	2^^xsd:integer
<http://semantics.id/ns/example#film_5>	1996^^xsd:integer	3^^xsd:integer
<http://semantics.id/ns/example#film_10>	1996^^xsd:integer	2^^xsd:integer
<http://semantics.id/ns/example#film_1>	2001^^xsd:integer	2^^xsd:integer
<http://semantics.id/ns/example#film_2>	2005^^xsd:integer	2^^xsd:integer
<http://semantics.id/ns/example#film_3>	2012^^xsd:integer	2^^xsd:integer
<http://semantics.id/ns/example#film_6>	2016^^xsd:integer	1^^xsd:integer
<http://semantics.id/ns/example#film_4>	2017^^xsd:integer	2^^xsd:integer
<http://semantics.id/ns/example#film_8>	2017^^xsd:integer	2^^xsd:integer

> Q10: __UNION__ Return a combined list of films and their directors, and films and their film studios

This SPARQL query retrieves a distinct list of films along with either their **director's name** or **studio name** (or both, if available). It uses the UNION operator to combine results from two different patterns.

1. **SELECT DISTINCT ?film ?directorName ?studioname**:
   - Retrieves the film URI (?film), the director's name (?directorName), and the studio name (?studioname) without duplicates due to the DISTINCT keyword.
2. **First Pattern**:
   - **?film mv:hasDirector ?director .**: Matches films that have a director.
   - **?director mv:fullName ?directorName .**: Retrieves the director's full name.
3. **UNION**:
   - Combines results from the first pattern with the second pattern.
4. **Second Pattern**:
   - **?film mv:hasFilmStudio ?studio .**: Matches films that have an associated film studio.
   - **?studio rdfs:label ?studioname .**: Retrieves the studio name.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>

SELECT DISTINCT ?film ?directorName ?studioname
WHERE {
  {
    ?film mv:hasDirector ?director .
    ?director mv:fullName ?directorName .
  }
  UNION
  {
    ?film mv:hasFilmStudio ?studio .
    ?studio rdfs:label ?studioname .
  }
}
```

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX mv: <http://semantics.id/ns/example/film#>

SELECT DISTINCT ?film ?directorName ?studioname
WHERE {
  {
    ?film mv:hasDirector ?director .
    ?director mv:fullName ?directorName .
  }
  UNION
  {
    ?film mv:hasFilmStudio ?studio .
    ?studio rdfs:label ?studioname .
  }
}
```

✓ 0.0s Endpoint Connection Ce

film	directorName	studioname
<http://semantics.id/ns/example#film_1>	Michael Haneke	
<http://semantics.id/ns/example#film_2>	Michael Haneke	
<http://semantics.id/ns/example#film_3>	Michael Haneke	
<http://semantics.id/ns/example#film_4>	Michael Haneke	
<http://semantics.id/ns/example#film_5>	Anthony Minghella	
<http://semantics.id/ns/example#film_6>	Paul Verhoeven	
<http://semantics.id/ns/example#film_7>	Ridley Scott	
<http://semantics.id/ns/example#film_8>	Denis Villeneuve	
<http://semantics.id/ns/example#film_9>	Robert Mandel	
<http://semantics.id/ns/example#film_10>	Roland Emmerich	
<http://semantics.id/ns/example#film_1>		MK2
<http://semantics.id/ns/example#film_3>		Les Films du Losange
<http://semantics.id/ns/example#film_4>		Les Films du Losange
<http://semantics.id/ns/example#film_5>		Miramax Films
<http://semantics.id/ns/example#film_6>		SBS Productions
<http://semantics.id/ns/example#film_7>		Warner Bros.
<http://semantics.id/ns/example#film_8>		Warner Bros.
<http://semantics.id/ns/example#film_9>		Warner Bros.
<http://semantics.id/ns/example#film_10>		20th Century Fox

Task 2:

In this task, we explored DBPedia's SPARQL endpoint to perform three types of queries. The first involved extracting data about actors from *Star Wars IV: A New Hope* and sorting them by age, focusing on relationship and attribute retrieval. The second query used an ASK format to check if Steven Spielberg and Tom Hanks co-directed a movie, highlighting the use of Boolean logic in SPARQL. The third query combined filtering conditions to count movies based on writer and actor attributes, demonstrating how to handle multiple criteria and date ranges. Throughout, we leveraged DBPedia's ontology (dbo) and resource (dbr) namespaces for accurate data access and manipulation.

> Q11: List the names of all Actors who starred in the movie Star Wars IV: A New Hope and order by their age

This search returns the birthdates of the actors who starred in Star Wars IV: A New Hope:

- dbo:wikiPageWikiLink connects the film (dbr:Star Wars IV:_A_New_Hope) to relevant elements.
- uses dbo:starring to filter entities associated with actors.
- returns the English names of the actors (rdfs:label) together with their birth dates (dbo:birthDate).
- FILTER(LANG(?actorName) = "en" is used to filter results to English labels.
- Birthdate is used to arrange the results.

```
# [endpoint=https://dbpedia.org/sparql]
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT DISTINCT ?actorName ?birthDate
WHERE {
  dbr:Star_Wars_IV:_A_New_Hope dbo:wikiPageWikiLink ?entity .
  ?entity dbo:starring ?actor .
  ?actor rdfs:label ?actorName .
  ?actor dbo:birthDate ?birthDate .
  FILTER(LANG(?actorName) = "en")
}
ORDER BY ?birthDate
```

```
# [endpoint=https://dbpedia.org/sparql]
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT DISTINCT ?actorName ?birthDate
WHERE {
  dbr:Star_Wars_IV:_A_New_Hope dbo:wikiPageWikiLink ?entity .
  ?entity dbo:starring ?actor .
  ?actor rdfs:label ?actorName .
  ?actor dbo:birthDate ?birthDate .
  OPTIONAL { ?actor dbo:deathDate ?deathDate . }
  FILTER(LANG(?actorName) = "en")

}
ORDER BY ?birthDate
```

✓ 0.1s Cell Comment Cell SPAR

actorName	birthDate
Peter Cushing@en	1913-05-26^^xsd:date
Alec Guinness@en	1914-04-02^^xsd:date
Harrison Ford@en	1942-07-13^^xsd:date
Mark Hamill@en	1951-09-25^^xsd:date
Carrie Fisher@en	1956-10-21^^xsd:date

> Q12: ASK Is there a movie that Steven Spielberg and Tom Hanks both directed?

ASK Query:
ASK is used to ask a question that returns either `true` or `false` depending on whether the pattern in the query matches any data in the dataset.

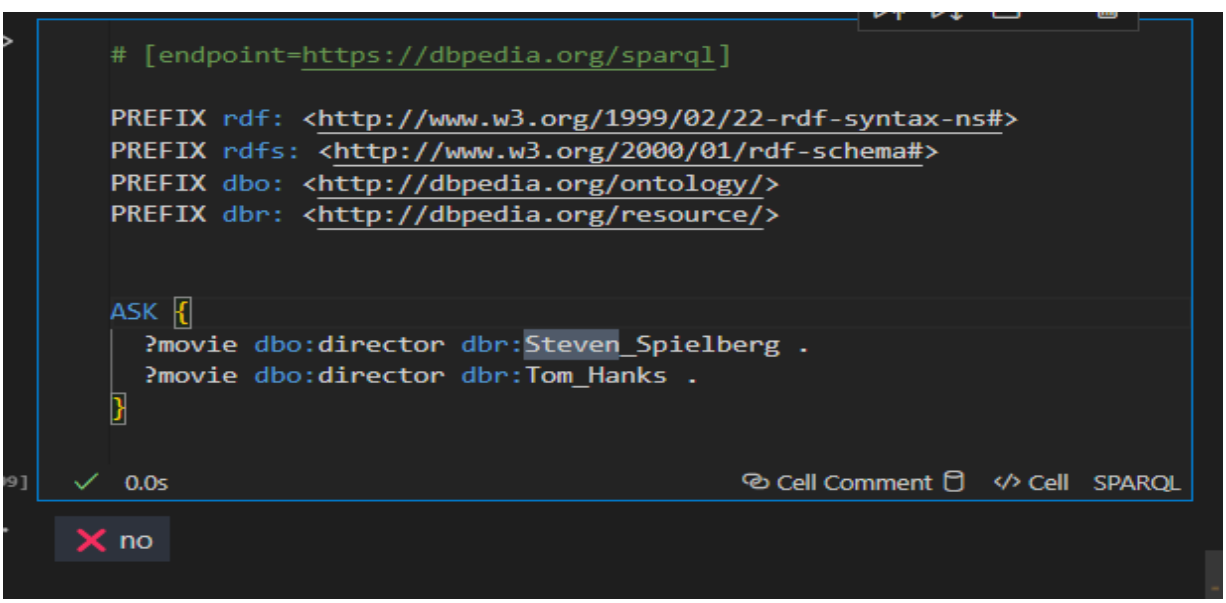
Pattern:
The query is looking for movies (`?movie`) that have both **Steven Spielberg** (`dbr:Steven_Spielberg`) and **Tom Hanks** (`dbr:Tom_Hanks`) as directors:

```
?movie dbo:director dbr:Steven_Spielberg .
?movie dbo:director dbr:Tom_Hanks .
```

Result:
If the conditions are true (i.e., there exists a movie that has both Steven Spielberg and Tom Hanks as directors), the result will be `true`. Otherwise, it will be `false`.

```
# [endpoint=https://dbpedia.org/sparql]

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
ASK {
  ?movie dbo:director dbr:Steven_Spielberg .
  ?movie dbo:director dbr:Tom_Hanks .
}
```



> Q13: Count the number of movies released after 1970 with at least one writer with the first name "Alex" and the number of movies starring an actor with the first name "Leo" released before or in 1970. The result of the query should be the sum of the two amounts.

This SPARQL query uses two criteria to determine how many movies there are in DBpedia overall:

Movies Released by "Alex" and Released After 1970:
Movies (?movie1) released after 1970 are counted in the first subquery (YEAR(?date1) > 1970).
Movies authored by authors whose names begin with "Alex" in English are filtered out (FILTER(STRSTARTS(LCASE(?firstName), "alex") AND LANG(?firstName) = "en")).

Films Starring the Actor "Leo" That Were Released on or Before 1970:
Movies (?movie2) released on or before 1970 are counted in the second subquery (YEAR(?date2) <= 1970).
It searches for films having actors whose names begin with "Leo" in English.
Last Count:

To get the overall number of pertinent films (?totalMovies), the outer query adds the counts from the two subqueries (?after1970Count and?before1970Count).

```
# [endpoint=https://dbpedia.org/sparql]
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT (SUM(?after1970Count + ?before1970Count) AS ?totalMovies)
WHERE {
  # Count movies released after 1970 with a writer named "Alex"
  {
    SELECT (COUNT(?movie1) AS ?after1970Count)
    WHERE {
      ?movie1 rdf:type dbo:Film .
      ?movie1 dbo:releaseDate ?date1 .
      FILTER(YEAR(?date1) > 1970) .
      ?movie1 dbo:writer ?writer .
      ?writer rdfs:label ?firstName .
      FILTER(STRSTARTS(LCASE(?firstName), "alex") AND LANG(?firstName) =
"en")
    }
  }
```

```
  # Count movies released on or before 1970 with an actor named "Leo"
  {
    SELECT (COUNT(DISTINCT ?movie2) AS ?before1970Count)
    WHERE {
      ?movie2 rdf:type dbo:Film .
      ?movie2 dbo:releaseDate ?date2 .
      FILTER(YEAR(?date2) <= 1970) .
      ?movie2 dbo:starring ?actor .
      ?actor rdfs:label ?firstNameActor .
      FILTER(STRSTARTS(LCASE(?firstNameActor), "leo") AND
LANG(?firstNameActor) = "en")
    }
  }
}
```

```
# [endpoint=https://dbpedia.org/sparql]
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT (SUM(?after1970Count + ?before1970Count) AS ?totalMovies)
WHERE {
  # Count movies released after 1970 with a writer named "Alex"
  {
    SELECT (COUNT(?movie1) AS ?after1970Count)
    WHERE {
      ?movie1 rdf:type dbo:Film .
      ?movie1 dbo:releaseDate ?date1 .
      FILTER(YEAR(?date1) > 1970) .
      ?movie1 dbo:writer ?writer .
      ?writer rdfs:label ?firstName .
      FILTER(STRSTARTS(LCASE(?firstName), "alex") AND LANG(?firstName) = "en")
    }
  }
  # Count movies released on or before 1970 with an actor named "Leo"
  {
    SELECT (COUNT(DISTINCT ?movie2) AS ?before1970Count)
    WHERE {
      ?movie2 rdf:type dbo:Film .
      ?movie2 dbo:releaseDate ?date2 .
      FILTER(YEAR(?date2) <= 1970) .
      ?movie2 dbo:starring ?actor .
      ?actor rdfs:label ?firstNameActor .
      FILTER(STRSTARTS(LCASE(?firstNameActor), "leo") AND LANG(?firstNameActor) = "en")
    }
  }
}
```

✓ 7.2s Cell Comment Cell SPARQL

totalMovies
103^^xsd:integer

Task 3:

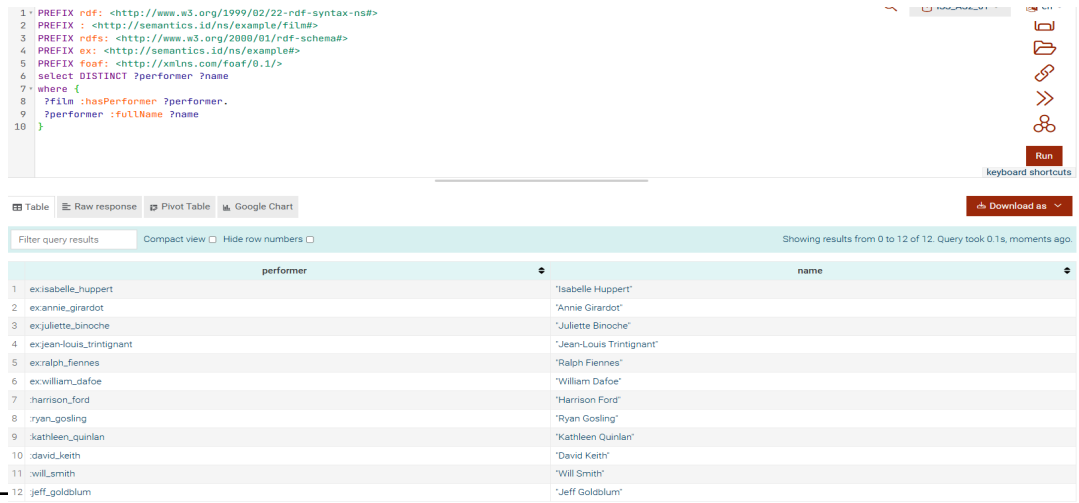
In this task, three custom SPARQL queries (Q14, Q15, and Q16) were created to demonstrate how enabling inference in the triple store affects query results. Each query was designed to be influenced by different entailment patterns, such as RDFS property domains, sub-properties, and class hierarchies. The task involved comparing query results with and without reasoning enabled, highlighting how reasoning impacts data retrieval. The explanation focused on the specific entailment pattern that affected each query's outcome.

> Query Q14: Select all the performers from the movie with their names

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://semantics.id/ns/example/film#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://semantics.id/ns/example#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
select DISTINCT ?performer ?name
where {
 ?film :hasPerformer ?performer.
 ?performer :fullName ?name
}
```

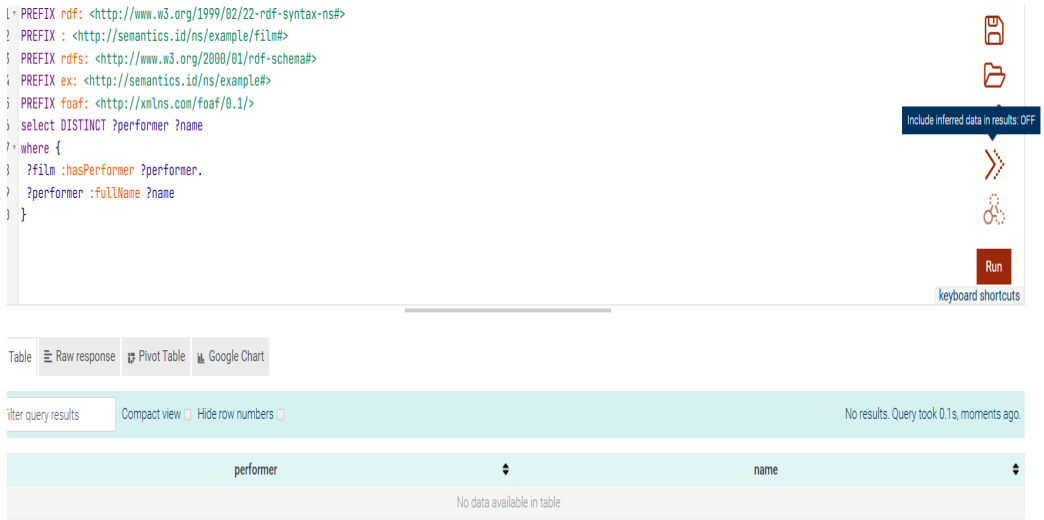
When using RDFS entailment, subclass relationships (e.g., `:Actor rdfs:subClassOf :Performer`) allow SPARQL queries to infer that all instances of `:Actor` are also instances of `:Performer`. With inference enabled, querying for `:Performer` returns all performers, including actors, because the system recognizes the class hierarchy. However, without inference, the SPARQL engine only retrieves individuals explicitly declared as `:Performer`, ignoring those declared as `:Actor`. This difference occurs because, without reasoning, the system cannot automatically deduce subclass relationships, leading to fewer results in the query output.

Inference ON



	performer	name
1	ex:isabelle_huppert	"Isabelle Huppert"
2	ex:annie_girardot	"Annie Girardot"
3	ex:juliette_binoche	"Juliette Binoche"
4	ex:jean-louis_trintignant	"Jean-Louis Trintignant"
5	ex:ralph_fiennes	"Ralph Fiennes"
6	ex:william_dafoe	"William Dafoe"
7	:harrison_ford	"Harrison Ford"
8	:ryan_gosling	"Ryan Gosling"
9	:kathleen_quinlan	"Kathleen Quinlan"
10	:david_keith	"David Keith"
11	:will_smith	"Will Smith"
12	:jeff_goldblum	"Jeff Goldblum"

Inference OFF



> Query Q15: Select all people (actor, directors, writers) with names and date of birth

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://semantics.id/ns/example/film#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://semantics.id/ns/example#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person ?name ?birthdate
WHERE {
  ?person a foaf:Person.
  ?person :fullName ?name.
  OPTIONAL { ?person :dateOfBirth ?birthdate. }
}
```

The query in question utilizes an RDFS entailment pattern related to sub-properties. In this case, the property `:hasCrew` has three sub-properties: `:hasDirector`, `:hasScriptWriter`, and `:hasComposer`. When inference is enabled, the SPARQL engine recognizes the sub-property relationships, allowing all results, including those associated with the sub-properties, to be retrieved under the `:hasCrew` property. However, without inference, the system does not recognize this hierarchy and only retrieves results explicitly associated with `:hasCrew`, leading to incomplete output. This discrepancy highlights the critical role of inference in leveraging RDFS-defined relationships for comprehensive query results.

Inference ON

```
PREFIX ex: <http://semantics.id/ns/example#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person ?name ?birthdate
WHERE {
  ?person a foaf:Person.
  ?person :fullName ?name.
  OPTIONAL { ?person :dateOfBirth ?birthdate. }
}
```

Press Alt+Enter to autocomplete

Table | Raw response | Pivot Table | Google Chart

Filter query results | Compact view | Hide row numbers

Showing results from 0 to 27 of 27. Query took 0.1s, mo

	person	name	birthdate
1	http://semantics.id/ns/example#isabelle_huppert	"Isabelle Huppert"	"1953-03-16"^^xsd:date
2	http://semantics.id/ns/example#annie_girardot	"Annie Girardot"	"1931-10-25"^^xsd:date
3	http://semantics.id/ns/example#juliette_binoche	"Juliette Binoche"	"1964-03-09"^^xsd:date
4	http://semantics.id/ns/example#jean-louis_trintignant	"Jean-Louis Trintignant"	"1930-12-11"^^xsd:date
5	http://semantics.id/ns/example#ralph_fiennes	"Ralph Fiennes"	"1962-12-22"^^xsd:date
6	http://semantics.id/ns/example#william_defoe	"William Defoe"	"1955-07-22"^^xsd:date
7	http://semantics.id/ns/example#michael_haneke	"Michael Haneke"	
8	http://semantics.id/ns/example#anthony_minghella	"Anthony Minghella"	
9	http://semantics.id/ns/example#paul_verhoeven	"Paul Verhoeven"	
10	http://semantics.id/ns/example#david_birke	"David Birke"	
11	:harrison_ford	"Harrison Ford"	"1942-07-13"^^xsd:date
12	http://semantics.id/ns/example#ridley_scott	"Ridley Scott"	
13	http://semantics.id/ns/example#hampton_fancher	"Hampton Fancher"	
14	http://semantics.id/ns/example#david_peoples	"David Peoples"	
15	:ryan_gosling	"Ryan Gosling"	"1980-11-12"^^xsd:date
16	http://semantics.id/ns/example#denis_villeneuve	"Denis Villeneuve"	
17	http://semantics.id/ns/example#michael_green	"Michael Green"	
18	:kathleen_quinlan	"Kathleen Quinlan"	"1954-11-19"^^xsd:date
19	:david_keith	"David Keith"	"1954-05-08"^^xsd:date
20	http://semantics.id/ns/example#robert_mandel	"Robert Mandel"	
21	http://semantics.id/ns/example#alice_hoffman	"Alice Hoffman"	
22	:will_smith	"Will Smith"	"1968-09-25"^^xsd:date
23	:jeff_goldblum	"Jeff Goldblum"	"1953-10-22"^^xsd:date
24	http://semantics.id/ns/example#roland_emmerich	"Roland Emmerich"	
25	http://semantics.id/ns/example#dean_devlin	"Dean Devlin"	
26	:anne_dudley	"Anne Dudley"	
27	:gabriel_yared	"Gabriel Yared"	

Inference OFF

Unnamed | Unnamed 1 | Unnamed 2 | Unnamed 3 | Unnamed 4 | Unnamed 5 | Unnamed 6 | http://xmlns.com/foaf/0.1...

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://semantics.id/ns/example/film#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://semantics.id/ns/example#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person ?name ?birthdate
WHERE {
  ?person a foaf:Person.
  ?person :fullName ?name.
  OPTIONAL { ?person :dateOfBirth ?birthdate. }
}
```

Run

keyboard shortcuts

Table | Raw response | Pivot Table | Google Chart

Filter query results | Compact view | Hide row numbers

No results. Query took 0.1s, moments ago.

person	name	birthdate
No data available in table		

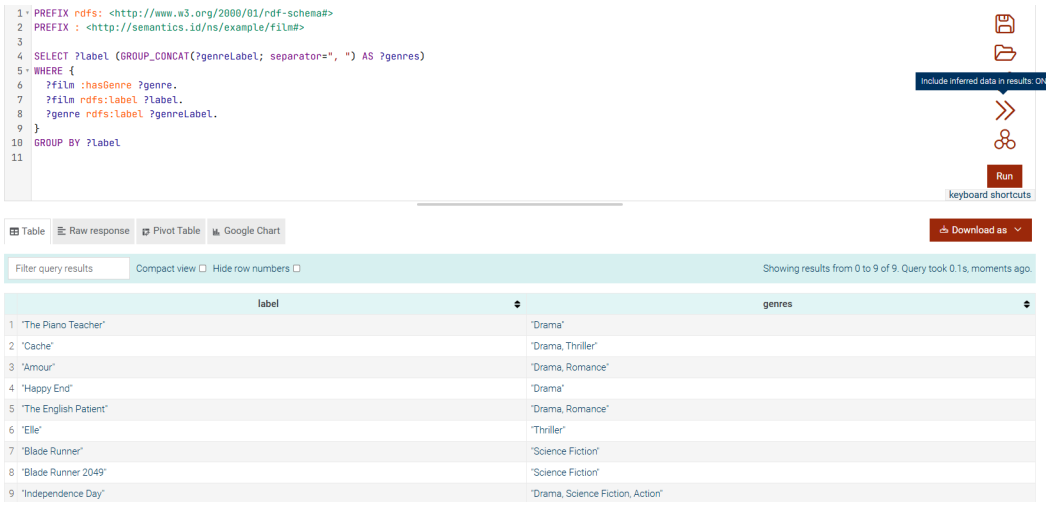
> Query Q16: Retrieve all movies and their respective genres

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://semantics.id/ns/example/film#>

SELECT ?label (GROUP_CONCAT(?genreLabel; separator=", ") AS ?genres)
WHERE {
  ?film :hasGenre ?genre.
  ?film rdfs:label ?label.
  ?genre rdfs:label ?genreLabel.
}
GROUP BY ?label
```

The :hasGenre predicate, one of the direct relationships established in the dataset, is used by the SPARQL query to obtain movies and the genres that go with them. Because the query depends only on relationships that are explicitly expressed in the RDF dataset, turning inference on or off has no effect. Results are only impacted by inference if the ontology defines indirect or derived links (such as parent-child genre relationships).

Inference ON



Inference OFF