School of Informatics Aristotle University of Thessaloniki

M.Sc. Artificial Intelligence

Knowledge Graphs & Ontology Engineering

Lelekas Pantelis plele@csd.auth.gr

Grosomanidis Odysseas ogros@csd.auth.gr

GitHub

Table of Contents

1 Introduction

The primary goal of this project is to develop a complete end-to-end solution using semantic web technologies such as RML (RDF Mapping Language), RDF4J, and GraphDB. The process involves selecting and preprocessing real-world datasets, designing and applying ontologies, mapping the data into RDF format, and storing it in a triplestore to support complex SPARQL queries.

Two datasets were used for this purpose: the IMDb Top 1000 Movies dataset and the Goodreads Books dataset. To represent their semantics, the CAMO (Context-Aware Movie Ontology) was applied for movies, while a custom ontology was created for books, building on established vocabularies like FOAF and BIBO. The datasets were converted to RDF using RML rules and then RDF4J was used to add extra meaning by connecting books to their movie adaptations.

The resulting RDF graph, which combines both datasets along with their ontologies and enriched relationships, was uploaded to a GraphDB repository. A set of SPARQL queries was then executed to test the model and extract valuable insights, highlighting the strength of linked data in integrating and querying complex information sources.

This report provides a detailed explanation of each step in the pipeline, outlining the methods, tools, and reasoning behind the transition from raw data to structured, queryable knowledge.

2 Knowledge Models and Datasets

2.1 Ontology Description

This subsection describes the ontologies chosen for the project implementation: one for movies and another for books.

2.1.1 Movies Ontology

The CAMO (Context-aware Movie Ontology) [1] is a semantic framework which is designed to model movies. It was built up using contextual features extracted from Linked Open Data (LOD) and movie databases (Rotten Tomatoes and IMDB).

The CAMO ontology is centered around the Movie class, which serves as the root of the ontology and connects to all major subclasses. This class encapsulates core movie metadata, including the title, abstract, rating, source material (based-on), and release dates (theatrical and DVD). The Movie class consists of 5 main subclasses [Table 1].

Table 1. Main subclasses of Movie class.

Subclass	Description
CastandCrew	Represents individuals involved in movie production, including actors, actresses and directors.
Certification	Describes content ratings assigned to movies based on content suitability.
Genre	Categorizes movies into thematic groups such as action, drama, or comedy, and supports subgenres.
Reviews	Captures user and critic review statistics and sentiment-related metadata.
Awards	Represents awards received by movies like Oscars, Golden Globes and BAFTAs.

Each of these main subclasses is further specialized into more detailed subclasses, allowing for a richer representation of movie-related information. The Table 2 outlines these subclass hierarchies.

Table 2. Subclasses Derived from Main Movie Subclasses.

Main Subclass	Subclass(es)
CastandCrew	- Actor - Actress - Director
Certification	- General(G) - NC-17(AdultsOnly) - NotRated(NR) - ParentalGuidance(PG) - ParentalGuidance-13(PG-13) - Restricted(R)
Genre	- GenreCategory1 - GenreCategory2 - GenreCategory3
Reviews	- NumberofUserReviews - NumberofCriticReviews
Awards	- Oscars - GoldenGlobe - Bafta

The diagram in Figure 1 illustrates the subclass hierarchy of the Movie class, showcasing its main subclasses such as Genre, CastandCrew, Certification, Reviews, and Awards, along with their respective subcategories.

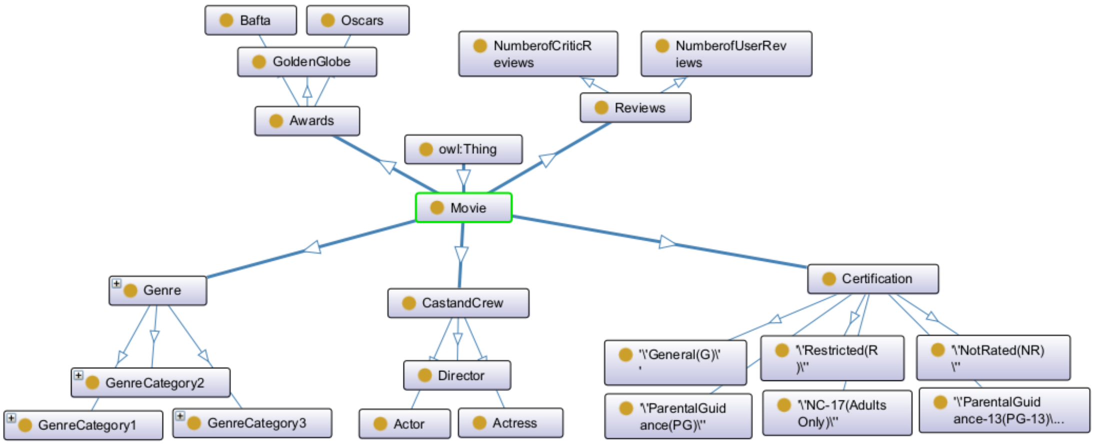


Figure 1. Subclass Structure of the Movie Class.

The CAMO ontology includes datatype properties [Table 3] that describe basic details about a movie like title, rating, release dates etc. These properties link movies to simple values like text, numbers or dates.

Table 3. Datatype properties of CAMO ontology.

Property	Domain	Range
about	Movie	string
abstract	Movie	string
awardsSentimentValue	Movie	double
basedOn	Movie	string
castPerformancesSentimentValue	Movie	double
cinematographySentimentValue	Movie	double
directionSentimentValue	Movie	double
dvdReleaseDate	Movie	dateTime
musicSentimentValue	Movie	double
rating	Movie	double
scenesSentimentValue	Movie	double
storySentimentValue	Movie	double
theatreReleaseDate	Movie	dateTime
title	Movie	string
totalUserReviews	Movie	integer
visualEffectsSentimentValue	Movie	double

Object properties in the CAMO ontology [Table 4] are used to connect movies to other things like the director, the genre or the awards they've won. Instead of linking to plain values, they connect one part of the ontology to another.

Table 4. Object type properties of CAMO ontology.

Property	Domain	Range	Inverse Of
aboutAwards	Movie		
directedBy	Movie	Director	
directedMovie	Director	Movie	directedBy
hasAwardsSentiments	Movie		
hasCast	Movie	Actor, Actress	
hasCastPerformancesSentiments	Movie		
hasCertification	Movie	General(G), NC-17(AdultsOnly), NotRated(NR), PG, PG-13, R	
hasCinematographySentiments	Movie		
hasDirectionSentiments	Movie		
hasGenre	Movie	GenreCategory1, GenreCategory2, GenreCategory3	
hasMusicSentiments	Movie		
hasScenesSentiments	Movie		
hasStorySentiments	Movie		
hasSubGenre	Genre	SubGenre1, SubGenre2, SubGenre3	
hasVisualEffectsSentiments	Movie		
hasWonAwardsBafta	Movie	Bafta	
hasWonAwardsGoldenGlobe	Movie	GoldenGlobe	
hasWonOscarsActor	Movie	Oscars	
hasWonOscarsActress	Movie	Oscars	
hasWonOscarsAnimatedMovie	Movie	Oscars	
hasWonOscarsBestPicture	Movie	Oscars	
hasWonOscarsCinematography	Movie	Oscars	
hasWonOscarsDirector	Movie	Oscars	
hasWonOscarsVisualEffects	Movie	Oscars	
reviewedByCritics	Movie	NumberofCriticReviews	
reviewedByUser	Movie	NumberofUserReviews	

2.1.2 Book Ontology

The book ontology is designed to represent relationships between books authors, book ratings and publishers. The ontology reuses classes from the FOAF [2] and BIBO [3] ontologies as external vocabularies to describe people and bibliographic entities. The BIBO (Bibliographic Ontology) is an RDF-based ontology designed to describe bibliographic entities such as books, articles, and documents on the Semantic Web. The FOAF (Friend of a Friend) Ontology is an RDF vocabulary used to describe people, their relationships, and activities on the web.

The main classes of book ontology are represented in Table 5

Table 5. Main classes of the book ontology.

Class IRI	Label	Subclass of
ex:Book	Book	bibo:Book
ex:Author	Author	foaf:Person
ex:Publisher	Publisher	foaf:Agent
ex:Rating	Rating	bibo:Review

The diagram in Figure 2 displays the high-level class hierarchy, encompassing major classes such as foaf:Person, Review, Book, and Movie, with further subclass relationships including Author and Rating.

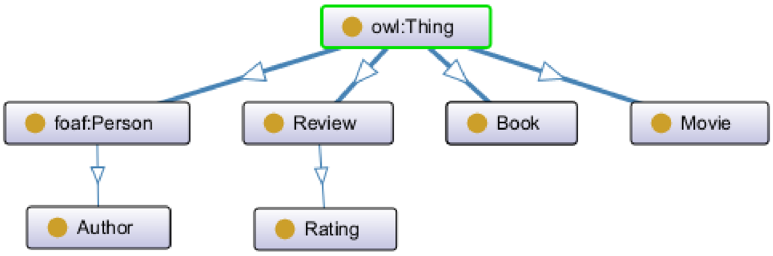


Figure 2. Subclass Structure of the Book ontology.

The Table 6 and

Table 7 summarizes the key components of the ontology, including the defined classes, object properties, and datatype properties.

Table 6. Object type properties of Book ontology.

Property IRI	**Domain**	**Range**
adaptationOf	camo:Movie	ex:Book
adaptedInto	ex:Book	camo:Movie
hasAuthor	ex:Book	ex:Author
hasRating	ex:Book	ex:Rating
hasPublisher	ex:Book	ex:Publisher

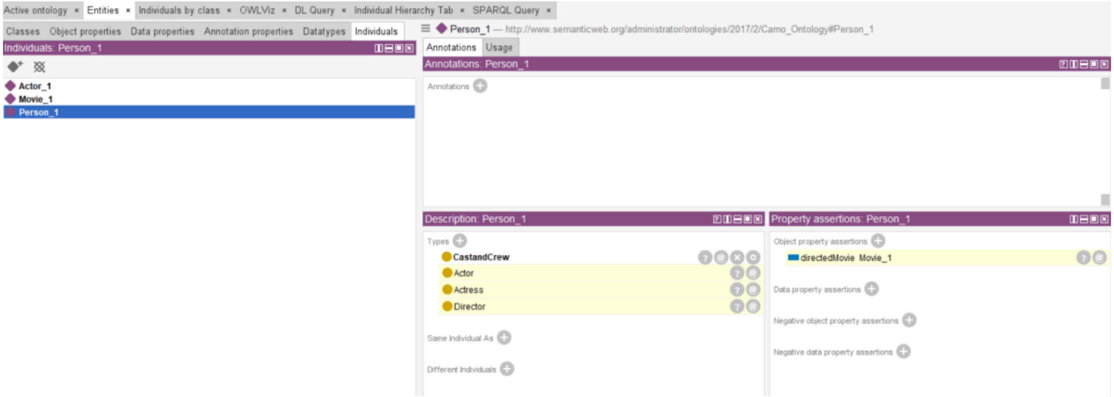
Table 7. Data type properties of Book ontology.

Property	**Domain**	**Range**
publishedYear	Book	xsd:gYear
ratingValue	Rating	xsd:decimal
publisherName	Publisher	xsd:string
publisherLocation	Publisher	xsd:string

2.1.3 Reasoner

At this point, we applied reasoning to the Movie ontology in order to examine its behavior and verify the correct functioning of the logical rules and relationships we have defined. The purpose of using a reasoner is to identify implicit inferences, such as the automatic classification of individuals into appropriate classes based on their properties. For this reason, the Protégé ontology editor was used in combination with the HermiT 1.4.3.456 reasoner.

We define Movie_1 as an individual belonging to the class Movie, and Person_1 as an individual of the class CastandCrew. We then assign two object properties: Movie_1 is linked to Person_1 through the hasCast property, and also through the directedBy property. Since the property hasCast has both Actor and Actress as its range, and the property directedBy has Director as its range, we expect that after running the reasoner, Person_1 will be inferred to belong to the classes Director, Actor, and Actress.



Person_1 — http://www.semanticweb.org/administrator/ontologies/2017/2/Camo_Ontology#Person_1

Individuals: Person_1

- Actor_1
- Movie_1
- Person_1

Annotations: Person_1

Description: Person_1

Types

- CastandCrew
- Actor
- Actress
- Director

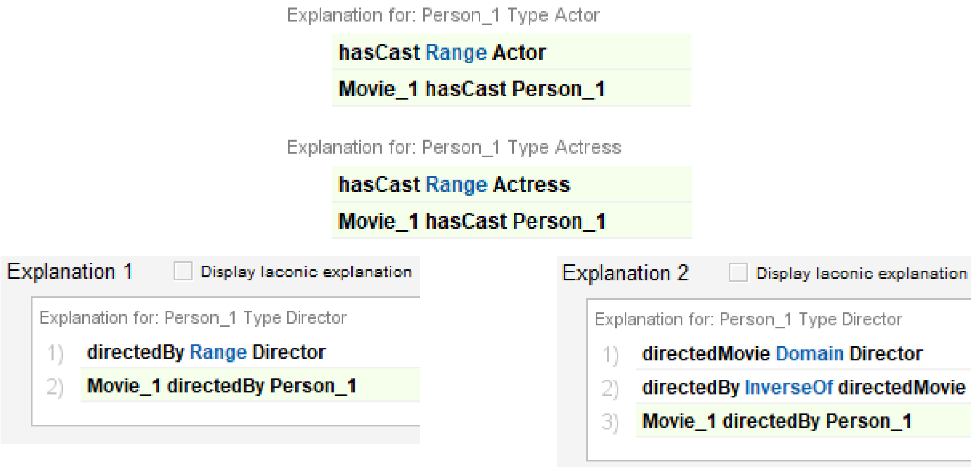
Property assertions: Person_1

Object property assertions

- directedMovie Movie_1

Figure 3. According to the reasoner, Person_1 is inferred to be an instance of Actor, Actress, and Director classes.

Indeed, after applying the reasoner, the individual Person_1 was classified under the classes Actor, Actress, and Director, as was expected. The explanation provided by the reasoner for this classification is shown in the Figure 4.



Explanation for: Person_1 Type Actor

hasCast Range Actor
Movie_1 hasCast Person_1

Explanation for: Person_1 Type Actress

hasCast Range Actress
Movie_1 hasCast Person_1

Explanation 1 □ Display laconic explanation

Explanation for: Person_1 Type Director

1) directedBy Range Director
2) Movie_1 directedBy Person_1

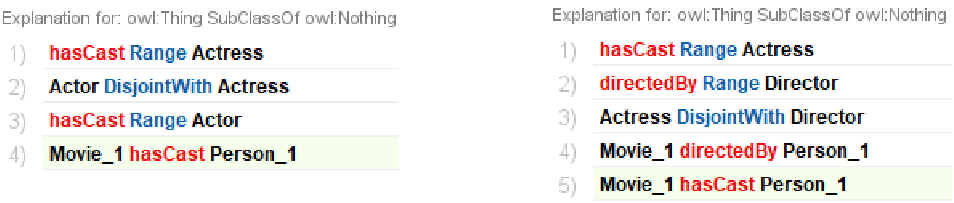
Explanation 2 □ Display laconic explanation

Explanation for: Person_1 Type Director

1) directedMovie Domain Director
2) directedBy InverseOf directedMovie
3) Movie_1 directedBy Person_1

Figure 4. Explanations of the reasoner's decision.

Now, if the classes Actor, Actress, and Director are made disjoint with each other and the reasoner is run again, an inconsistency will occur in our ontology. The reasoner returns 5 different explanations for this inconsistency. Those explanations are shown in the Figure 5.

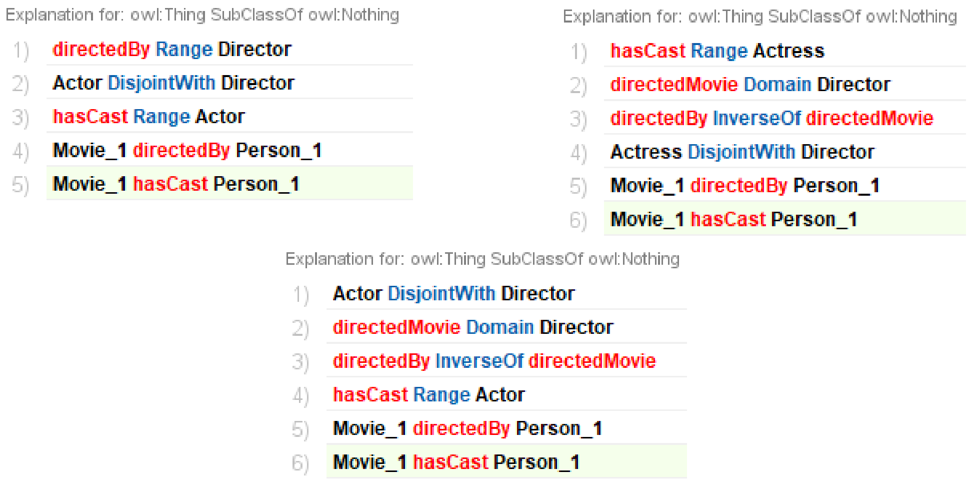


Explanation for: owl:Thing SubClassOf owl:Nothing

1) hasCast Range Actress
2) Actor DisjointWith Actress
3) hasCast Range Actor
4) Movie_1 hasCast Person_1

Explanation for: owl:Thing SubClassOf owl:Nothing

1) hasCast Range Actress
2) directedBy Range Director
3) Actress DisjointWith Director
4) Movie_1 directedBy Person_1
5) Movie_1 hasCast Person_1



Explanation for: owl:Thing SubClassOf owl:Nothing

1) directedBy Range Director
2) Actor DisjointWith Director
3) hasCast Range Actor
4) Movie_1 directedBy Person_1
5) Movie_1 hasCast Person_1

Explanation for: owl:Thing SubClassOf owl:Nothing

1) hasCast Range Actress
2) directedMovie Domain Director
3) directedBy InverseOf directedMovie
4) Actress DisjointWith Director
5) Movie_1 directedBy Person_1
6) Movie_1 hasCast Person_1

Explanation for: owl:Thing SubClassOf owl:Nothing

1) Actor DisjointWith Director
2) directedMovie Domain Director
3) directedBy InverseOf directedMovie
4) hasCast Range Actor
5) Movie_1 directedBy Person_1
6) Movie_1 hasCast Person_1

Figure 5. Explanation of the inconsistency after making the classes Actor, Actress, and Director disjoint.

2.2 Dataset Description

2.2.1 Movies Dataset

For the movie dataset, the [4] was selected. The IMDb Top 1000 Movies Dataset includes information on 1,000 of the highest-rated movies on IMDb. This dataset is useful for exploring patterns in movie quality, genre popularity, box office success, and the impact of key contributors like directors and actors.

Each record in the dataset represents a single movie and includes the features listed in the Table 8.

Table 8. Features of IMDb Top 1000 Movies Dataset.

Column Name	Description
Poster_Link	URL to the movie poster image
Series_Title	Title of the movie
Released_Year	Year the movie was released
Certificate	Movie rating certificate (e.g., PG-13, R)
Runtime	Duration of the movie
Genre	Genre(s) of the movie (comma-separated)
IMDB_Rating	IMDb rating (e.g., 8.3)
Overview	Short description of the movie
Meta_score	Metacritic score
Director	Name of the director
Star1	First main actor/actress
Star2	Second main actor/actress
Star3	Third main actor/actress

Column Name	Description
Star4	Fourth main actor/actress
No_of_Votes	Number of user votes on IMDb
Gross	Gross earnings in USD (where available)

2.2.2 Book Dataset

The Goodreads Books Dataset [5] was selected as book dataset for this project. It includes information on over 23,000 books listed on Goodreads. This dataset is great for exploring trends in book ratings, genres, author popularity, and more. Each entry represents a single book and contains details like title, author, average rating, number of pages, publication year and more, as shown in Table 9.

Table 9. Features of Goodreads-books Dataset.

Column Name	Description
bookID	Unique identifier for each book
title	Title of the book
authors	Author(s) of the book
average_rating	Average rating of the book on Goodreads
isbn	International Standard Book Number (ISBN)
isbn13	13-digit ISBN number
language_code	Language code of the book (e.g., 'eng' for English)
num_pages	Number of pages in the book
ratings_count	Total number of ratings the book has received
text_reviews_count	Number of text reviews for the book
publication_date	Publication date of the book
publisher	Publisher of the book

3 Implementation

3.1 Implementation Architecture

As previously mentioned, the main objective of this study is the transformation of heterogeneous data into RDF format, their storage in a GraphDB database, and their utilization through SPARQL queries. The individual steps followed to achieve this goal are presented in the flowchart in Figure 6, which briefly illustrates the overall architecture, and the logical sequence of the technologies and tools used.

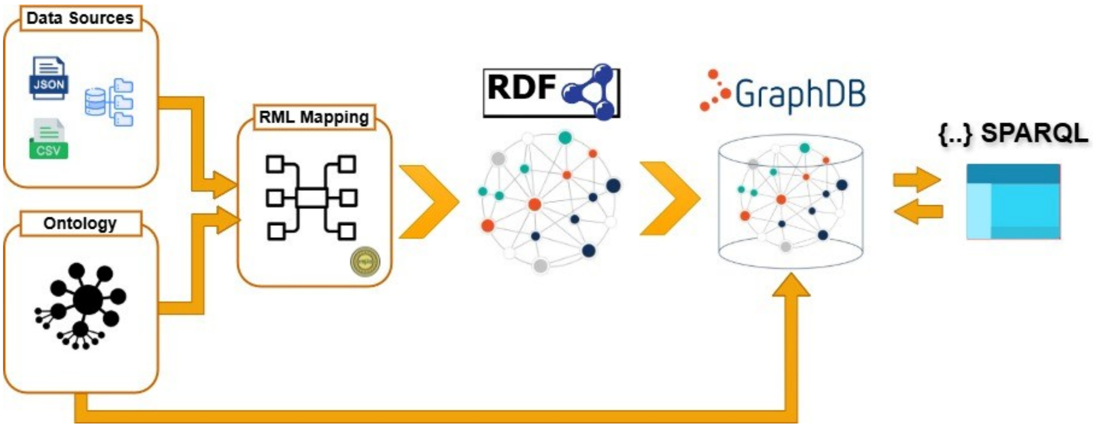


Figure 6. End-to-End Semantic Data Integration and Querying Pipeline

The first stage involves the selection or design of an ontology in OWL (Web Ontology Language), which defines the main classes, properties, as well as the constraints or rules governing the relationships. Along with ontology, a dataset is also provided as input to the system, which refers to the ontology or a part of it. The structure and content of the dataset must allow for its semantic modeling and subsequent transformation into RDF.

This is followed by the definition of rules for transforming the data into RDF using the RML (RDF Mapping Language). The RML rules describe how each field of data sources is mapped to RDF triples (subject-predicate-object), based on the structure of the ontology.

The RDF graph is imported into a Graph database named GraphDB by creating a repository named MiniProject. Along with the data, the ontology is also loaded, so that queries can be executed on a complete semantic model. The storage is performed either through the GraphDB web interface or programmatically using the RDF4J framework.

The final phase involves submitting SPARQL queries to GraphDB with the aim of extracting information and evaluating the RDF model. The queries designed must utilize the concepts of the ontology and highlight the importance of data interlinking.

3.2 Data Preprocessing

3.2.1 Movies Data Preprocessing

In order to align the movie data more closely with the ontology, a preprocessing step was performed. First, a mapping dictionary was created to match the certification labels in the movie dataset with those defined in the ontology. This transformation was applied using the map() function, while any missing (NaN) or unmapped values were preserved using the combine_first() function.

Table 10. Mapping between movie dataset certification labels and ontology-defined labels

Original Certification	Mapped Certification
A	ParentalGuidance(PG)
UA	ParentalGuidance-13(PG-13)
U	General(G)
PG-13	ParentalGuidance-13(PG-13)
R	Restricted(R)
PG	ParentalGuidance(PG)
G	General(G)
Passed	General(G)
TV-14	ParentalGuidance-13(PG-13)
16	Restricted(R)
TV-MA	NC-17(AdultsOnly)
Unrated	NotRated(NR)
GP	ParentalGuidance(PG)
Approved	General(G)
TV-PG	ParentalGuidance(PG)
U/A	ParentalGuidance-13(PG-13)

Afterward, the Genre column, which contains a comma-separated list of genres, was split into up to three separate columns (Genre1, Genre2, and Genre3) in order to facilitate the RML mapping procedure. After this transformation, the original Genre column was removed from the dataset.

3.2.2 Book Data Preprocessing

To prepare the books.csv dataset for RDF transformation, a preprocessing script was implemented. The script performs the following key tasks: Standardizing Publication Dates, Adding Unique Identifiers and Saving the Cleaned Dataset.

To standardize date formats in the books.csv dataset, a Python script was implemented using the pandas and datetime libraries. The objective was to convert all values in the publication_date column to ISO 8601 format (yyyy-mm-dd), ensuring consistency and compatibility for RDF conversion and SPARQL querying.

3.3 RML Mapping

3.3.1 Movies RML Mapping

To transform the dataset imdb_top_1000.csv into RDF according to the CAMO ontology, we implemented a set of RML mapping rules that define how CSV data is converted into RDF entities and relationships. These rules cover the core concepts of ontology, such as Movie, Person and Genre, as well as their associated properties.

MovieMapping

The main RML rule represents each movie as an instance of the Movie class. A unique URI is generated for each movie using the Series_Title field (rr:template). Additionally, several literal properties are assigned, including:

Released_Year → hasReleaseYear

Runtime → hasRuntime

IMDB_Rating → hasIMDBRating

Meta_score → hasMetaScore

Overview → hasOverview

Certificate → hasCertificate

Gross → hasGross

DirectorMapping

For each record, a Person entity is created to represent the movie's director, using the Director field. A URI is dynamically generated, and the movie is linked to this entity via the hasDirector property.

StarMapping

Since the dataset contains four columns for actors (Star1, Star2, Star3, Star4), we defined a separate mapping rule for each one. Each actor is represented as an instance of the Person class, and a URI is created based on their name. The movie is then linked to each actor using the hasActor property.

Star1 → has Actor (Person1)

Star2 → hasActor (Person2)

Star3→ hasActor (Person3)

Star4 → hasActor (Person4)

This approach ensures that all actors are correctly represented, even when some fields are missing or null.

It should be noted that all performers (male or female) were classified under the "Actor" class, as the available dataset did not provide sufficient information to determine whether they were actors or actresses.

GenreMapping

Each genre is modeled as an instance of the Genre class and linked to the corresponding movie using the hasGenre property. URIs are generated based on genre values (e.g., Drama, Action, Crime, etc.).

Genre1 → hasGenre (Genre1)

Genre2 → hasGenre (Genre2)

Genre3 → hasGenre (Genre3)

All mappings make use of rr:subjectMap in combination with rr:template to generate unique URIs for each entity based on the dataset fields. Properties are defined using rr:predicateObjectMap, which associates predicates with their corresponding object mappings. Literal values are appropriately typed using rr:datatype, with data types such as xsd:string, xsd:integer, and xsd:float to ensure correct semantic representation. Furthermore, in cases where fields are empty or contain missing values (e.g., NaN), the RML processor automatically skips the generation of those triples, resulting in clean and semantically valid RDF output.

3.3.2 Books RML Mapping

To transform the books dataset [] into RDF according to the custom Books ontology, we implemented a set of RML mapping rules that defines how the data of the dataset csv is converted into RDF triples. These mappings align with the concepts defined in the custom created Books ontology, such as Book, Author, etc., while also establishing properties to represent relationships between them.

BooksMap

The most fundamental RML rule maps each book as an instance of the ex:Book class. A unique URI is generated for each book using the bookID field of the given dataset. The bookID field was created in a post-processing step in the book's dataset because as a unique identifier it is safer to be used compared to other fields like titles that have the danger of having duplicate entries.

title → dc:title

author → ex:hasAuthor

publication_date → ex:publishedYear (xsd:date)

average_rating → ex:ratingValue (xsd:decimal)

The book title is extracted directly from the title filed and mapped as a literal string using the Dublin Core dc:title property.

Afterwards, each book is associated with an author, using the author's field of the given dataset. An object property ex:hasAuthor is used to link the book to a URI that identifies the author of the book uniquely. The author URIs which is created, uses the author's name and has the type of ex:Author.

Also, the book's publication date is extracted from the dataset, and mapped as a literal with datatype xsd:date. It is assigned to the ontology property ex:publishedYear.

Finally, each book's rating is directly mapped from the corresponding field of the dataset. A xsd:decimal literal value is generated using the ex:ratingValue property. This approach simplifies the representation of the book's ontology by eliminating the need for another class, for the Rating of each book.

3.3.3 RML Mapping execution

For the automated execution of the RML mappings described in the previous sections, a JAVA application was created. Three major steps implemented in this application, the execution of the RML mappings with the generation of the RDF output, afterwards the usage of the RDF4J framework to match the titles of the books and movies dataset to combine them using the adaptedInto and adaptationOf relationships, and finally the integration of the final RDF files and ontologies to GraphDB.

The first step in the pipelines, as already mentioned, involves the conversion of the given datasets into RDF. Firstly, the two mapping files are passed in the executeMapping() function, which is responsible for parsing the RML mapping file, executing the transformation, and returning the resulting RDF model. For this purpose, the QuadStoreFactory used to read the mapping stream, and the RecordsFactory to prepare the data records.

```
QuadStore rmlStore = QuadStoreFactory.read(mappingStream);

RecordsFactory factory = new RecordsFactory(mappingFile.getParent());
```

Once the mapping for both books and movies are executed, the RDF models produced are merged into one, using the RDF4J .addAll() function. The resulting RDF graph is serialized and written into an output file as an intermediate output which will be the input for the second step of post-processing.

```
bookModel.addAll(movieModel);

Writer output = new FileWriter("./src/main/resources/outputFile.ttl");
Rio.write(bookModel, output, RDFFormat.TURTLE);
```

3.3.4 Rdf4j postprocessing

After the RDF generation described in the previous section, the RDF4J framework was used to perform a semantic enrichment of the generated model. This post-processing step is responsible for establishing a connection between books and their movie adaptations, enhancing the knowledge graph with interlinked data that would be otherwise impossible to express in static RML rules.

The primary goal of this step is to match book titles and movie titles. Thus, the model generated is filtered to collect into the two hash maps of the books and movie titles. These hash maps enable efficient lookups during the matching process.

```
model.filter(null, dcTitle, null).forEach(stmt -> {
  String title = stmt.getObject().stringValue().toLowerCase().trim();
  IRI subject = (IRI) stmt.getSubject();
  books.put(title, subject);
});
```

Using the collected data the post-processing step compares the cleaned and lowercased titles of books and movies and checks whether one title starts with the other. If a match is found, two RDF triples are added, one expressing a book was adaptedInto a movie, and the other denoting the inverse relationship of movie being an adaptationOf a book.

```
if (movieTitle.startsWith(bookTitle) || bookTitle.startsWith(movieTitle)) {
  model.add(bookIRI, vf.createIRI("http://example.org/mini#adaptedInto"), movieIRI);
  model.add(movieIRI, vf.createIRI("http://example.org/mini#adaptationOf"), bookIRI);
  System.out.println("Linking: Book \"" + bookTitle + "\" -> Movie \"" + movieTitle + "\"");
  break;
}
```

The enhanced RDF model is then serialized again and saved into the final output file which also includes the new inferred relationships. This final RDF model is the one that will be integrated in GraphDB in the final step.

3.4 GraphDB Integration

GraphDB is a Semantic Graph Database (also called RDF triplestores), compliant with W3C Standards. A graph database is a type of database designed to represent and store data in terms of nodes, edges and properties. The connections between the nodes indicate their relationships. This makes it easier to access data and often allows it to be retrieved in a single query.

So, the last step in the pipeline is to publish the RDF model into the GraphDB, which acts as the backend for storing, querying and exploring the RDF data. This step is responsible for ensuring that the ontologies and the final output of the enriched RDF instances is integrated into GraphDB and available for queries through SPARQL.

This integration is implemented using RDF4J framework, which allows the application to connect to a remote GraphDB repository via its HTTP API. The repository is initialized using the GraphDB server URL and the target repository name in this case was MiniProject.

```
HTTPRepository repo = new HTTPRepository(SERVER_URL + "/repositories/" + REPOSITORY_ID);
repo.initialize();
try (RepositoryConnection conn = repo.getConnection()) {
  conn.clear();
  conn.begin();
```

Before uploading data, the repository is cleared to ensure safe publish and no conflicting data exist from previous runs. Then, both the custom book ontology and the CAMO movie ontology are loaded into the repository using the RDF/XML format. These ontologies provide the schema definitions that underpin the RDF data, defining the classes, properties, and relationships used in the model. Also, the enriched RDF graph is uploaded to the repository, which includes all books, movies and the semantic links connecting them.

```
conn.add(new FileInputStream("./src/main/resources/books/bookOntology.owl"),
    "urn:base",
    RDFFormat.RDFXML);
conn.add(new FileInputStream("./src/main/resources/movies/CAMO_Ontology.owl"),
    "urn:base",
    RDFFormat.RDFXML);
conn.add(new FileInputStream("./src/main/resources/outputFileFinal.ttl"),
    "urn:base",
    RDFFormat.TURTLE);
```

After this publishing step to GraphDB, the data are available in the repository and can be queried using SPARQL to uncover hidden relationships and other useful insights like books authored by a specific writer that were later adapted into a high-rated movies, which was one of the core goals of this project.

3.5 SPARQL Queries

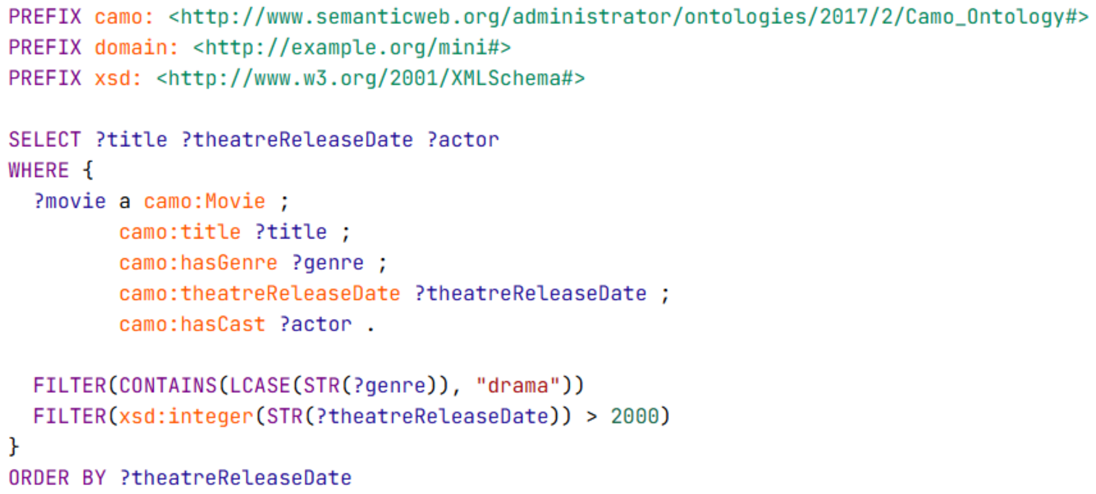
The final step of this project is to execute SPARQL queries that reflect real-world scenarios. These include queries related to movies, queries related to books and some that combine information from both ontologies.

3.5.1 SPARQL Queries on Movie Ontology

To begin with Movie ontology, the SPARQL query in Figure 7 retrieves information about movies that belong to the drama genre and were released after the year 2000. Specifically, it returns the movie's Title (?title), Release year (?theatreReleaseDate) and Actors (?actor).

Specifically, the query:

1. Filters only those genres that contain the word "drama" (case-insensitive).
2. Convert the release date to an integer to filter for dates after 2000.
3. Orders the results by release date (ORDER BY ?theatreReleaseDate).



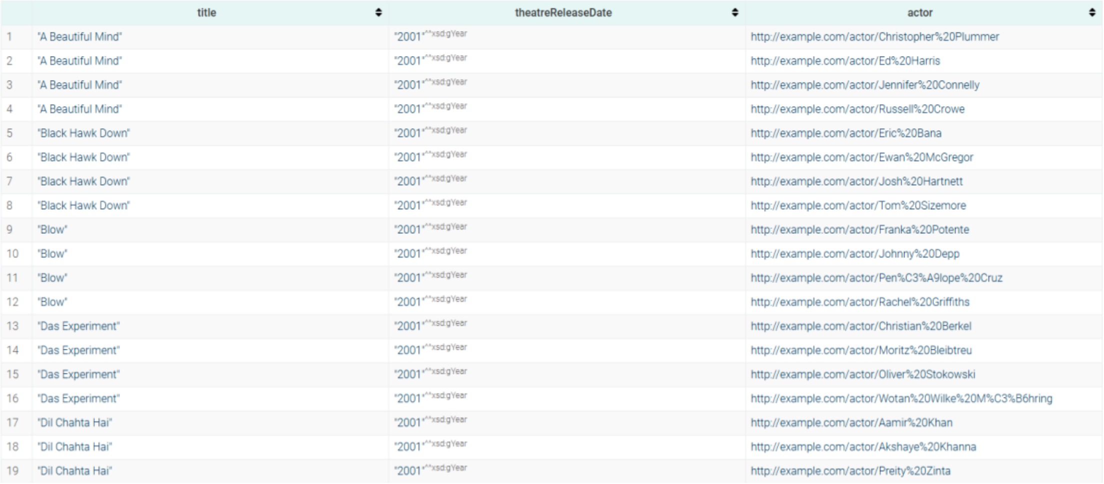
```
PREFIX camo: <http://www.semanticweb.org/administrator/ontologies/2017/2/Camo_Ontology#>
PREFIX domain: <http://example.org/mini#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?title ?theatreReleaseDate ?actor
WHERE {
  ?movie a camo:Movie ;
         camo:title ?title ;
         camo:hasGenre ?genre ;
         camo:theatreReleaseDate ?theatreReleaseDate ;
         camo:hasCast ?actor .

  FILTER(CONTAINS(LCASE(STR(?genre)), "drama"))
  FILTER(xsd:integer(STR(?theatreReleaseDate)) > 2000)
}
ORDER BY ?theatreReleaseDate
```

Figure 7. SPARQL query retrieves information about movies that belong to the drama genre and were released after the year 2000.

The results of this query are shown in Figure 8.



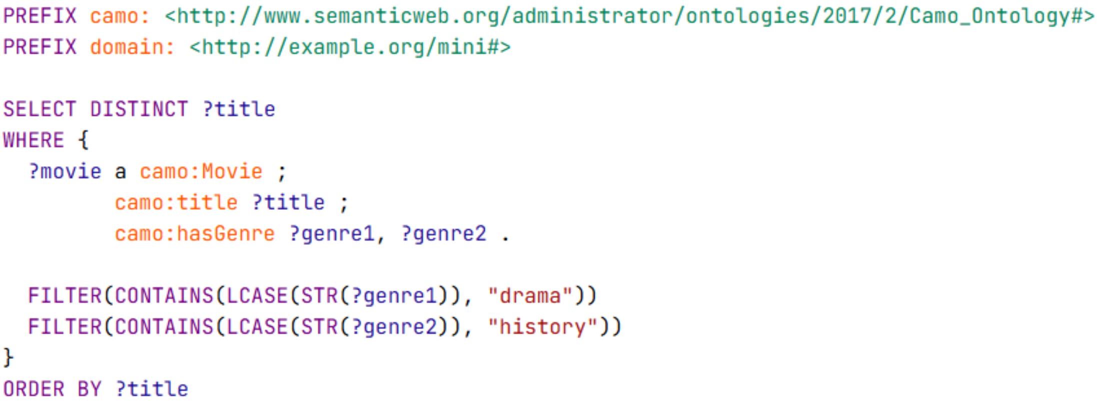
	title	theatreReleaseDate	actor
1	"A Beautiful Mind"	"2001"^^xsd:gYear	http://example.com/actor/Christopher%20Plummer
2	"A Beautiful Mind"	"2001"^^xsd:gYear	http://example.com/actor/Ed%20Harris
3	"A Beautiful Mind"	"2001"^^xsd:gYear	http://example.com/actor/Jennifer%20Connelly
4	"A Beautiful Mind"	"2001"^^xsd:gYear	http://example.com/actor/Russell%20Crowe
5	"Black Hawk Down"	"2001"^^xsd:gYear	http://example.com/actor/Eric%20Bana
6	"Black Hawk Down"	"2001"^^xsd:gYear	http://example.com/actor/Ewan%20McGregor
7	"Black Hawk Down"	"2001"^^xsd:gYear	http://example.com/actor/Josh%20Hartnett
8	"Black Hawk Down"	"2001"^^xsd:gYear	http://example.com/actor/Tom%20Sizemore
9	"Blow"	"2001"^^xsd:gYear	http://example.com/actor/Franka%20Potente
10	"Blow"	"2001"^^xsd:gYear	http://example.com/actor/Johnny%20Depp
11	"Blow"	"2001"^^xsd:gYear	http://example.com/actor/Pen%C3%A9lope%20Cruz
12	"Blow"	"2001"^^xsd:gYear	http://example.com/actor/Rachel%20Griffiths
13	"Das Experiment"	"2001"^^xsd:gYear	http://example.com/actor/Christian%20Berkel
14	"Das Experiment"	"2001"^^xsd:gYear	http://example.com/actor/Moritz%20Bleibtreu
15	"Das Experiment"	"2001"^^xsd:gYear	http://example.com/actor/Oliver%20Stokowski
16	"Das Experiment"	"2001"^^xsd:gYear	http://example.com/actor/Wotan%20Wilke%20M%C3%B6hring
17	"Dil Chahta Hai"	"2001"^^xsd:gYear	http://example.com/actor/Aamir%20Khan
18	"Dil Chahta Hai"	"2001"^^xsd:gYear	http://example.com/actor/Akshaye%20Khanna
19	"Dil Chahta Hai"	"2001"^^xsd:gYear	http://example.com/actor/Preity%20Zinta

Figure 8. Results of a SPARQL query that lists all drama movies released after the year 2000, along with their titles, release years and actors.

The next SPARQL query [Figure 9]retrieves the titles of movies that belong to both the "drama" and "history" genres.

1. It searches for resources of type camo:Movie.
2. It filters the results to include only those where:
    - One camo:hasGenre value contains "drama", and
    - Another contains "history" (case-insensitive checks using LCASE).
3. The results are distinct (no duplicates) and sorted alphabetically by title.

The query assumes that each movie can have multiple camo:hasGenre properties, which it binds to ?genre1 and ?genre2.



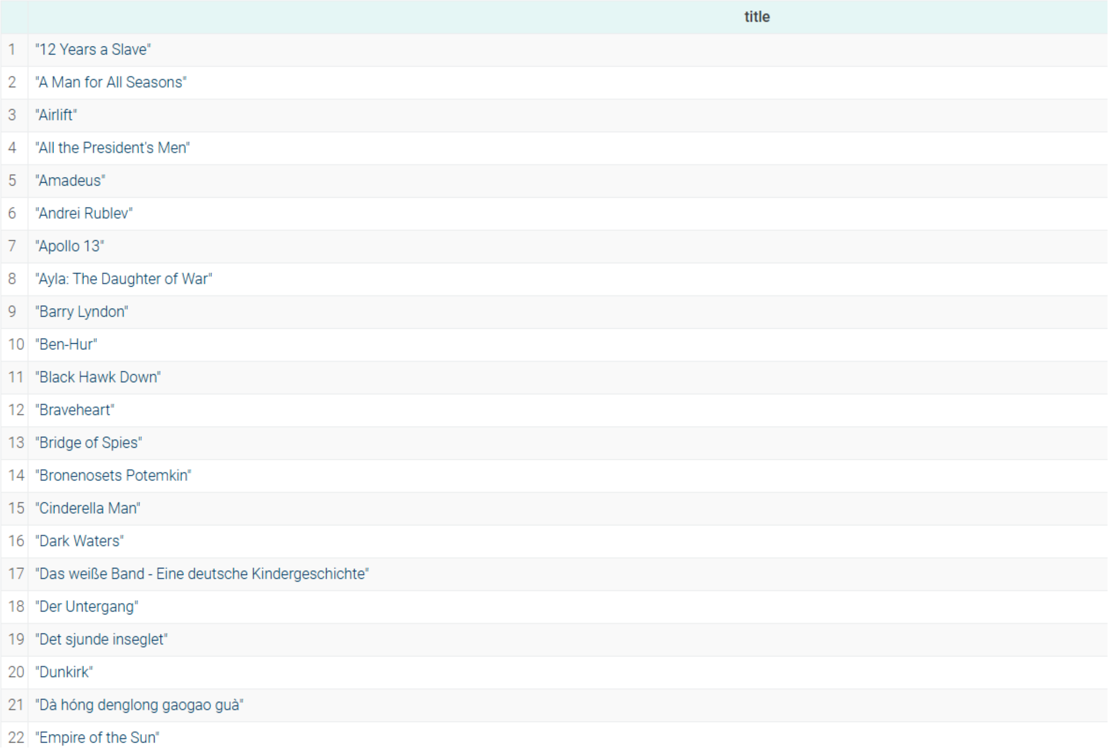
```
PREFIX camo: <http://www.semanticweb.org/administrator/ontologies/2017/2/Camo_Ontology#>
PREFIX domain: <http://example.org/mini#>

SELECT DISTINCT ?title
WHERE {
  ?movie a camo:Movie ;
         camo:title ?title ;
         camo:hasGenre ?genre1, ?genre2 .

  FILTER(CONTAINS(LCASE(STR(?genre1)), "drama"))
  FILTER(CONTAINS(LCASE(STR(?genre2)), "history"))
}
ORDER BY ?title
```

Figure 9. SPARQL query to retrieve movies classified under both "drama" and "history" genres.

Figure 10 shows the movies that belong to both the 'drama' and 'history' genres.



	title
1	"12 Years a Slave"
2	"A Man for All Seasons"
3	"Airlift"
4	"All the President's Men"
5	"Amadeus"
6	"Andrei Rublev"
7	"Apollo 13"
8	"Ayla: The Daughter of War"
9	"Barry Lyndon"
10	"Ben-Hur"
11	"Black Hawk Down"
12	"Braveheart"
13	"Bridge of Spies"
14	"Bronenosets Potemkin"
15	"Cinderella Man"
16	"Dark Waters"
17	"Das weiße Band - Eine deutsche Kindergeschichte"
18	"Der Untergang"
19	"Det sjunde inseglet"
20	"Dunkirk"
21	"Dà hóng denglong gaogao guà"
22	"Empire of the Sun"

Figure 10. Results of a SPARQL query retrieving titles of movies that belong to both the "drama" and "history" genres.

3.5.2 SPARQL Queries on Book Ontology

The SPARQL query in Figure 11 is designed to retrieve a list of books authored by J.R.R. Tolkien, by containing the name "Tolkien" (not case-sensitive) that have received high average ratings, specifically greater than 4.4.

The query selects the following data fields for each matching book, the URI of the book resource (?book), the title of the book (?bookTitle), the average rating (?bookRating), the publication year (?publishedYear) and the author resource (?author).

The FILTER clause applies to two conditions:

1. The author data field must contain the string “tolkien” (case-insensitive).
2. The book's rating must be greater than 4.4.

This query enables the filtering of Tolkien’s most well-known books.

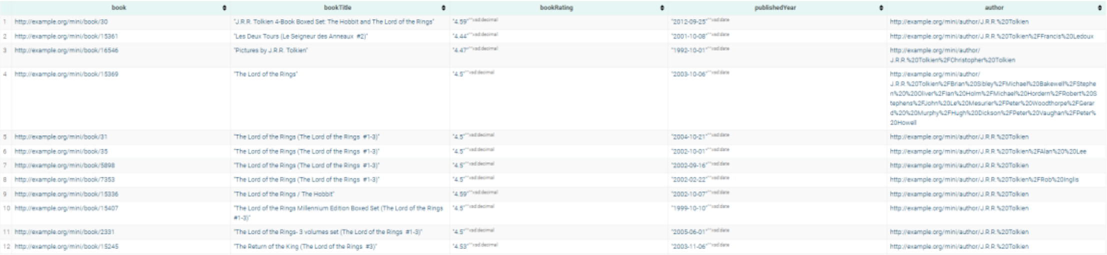
```
PREFIX ex: <http://example.org/mini#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?book ?bookTitle ?bookRating ?publishedYear ?author
WHERE {
  # Match the book, including its title, rating, and publication year
  ?book a ex:Book ;
        dc:title ?bookTitle ;
        ex:ratingValue ?bookRating ;
        ex:publishedYear ?publishedYear ;
        ex:hasAuthor ?author .

  # Filter to find only books authored by Tolkien
  FILTER(CONTAINS(LCASE(str(?author)), "tolkien") && ?bookRating > 4.4)  # Matching "Tolkien" case-insensitively
}
ORDER BY ?bookTitle
```

Figure 11. SPARQL query to retrieve highly rated books authored by Tolkien.

The results of the above query are shown in Figure 12.

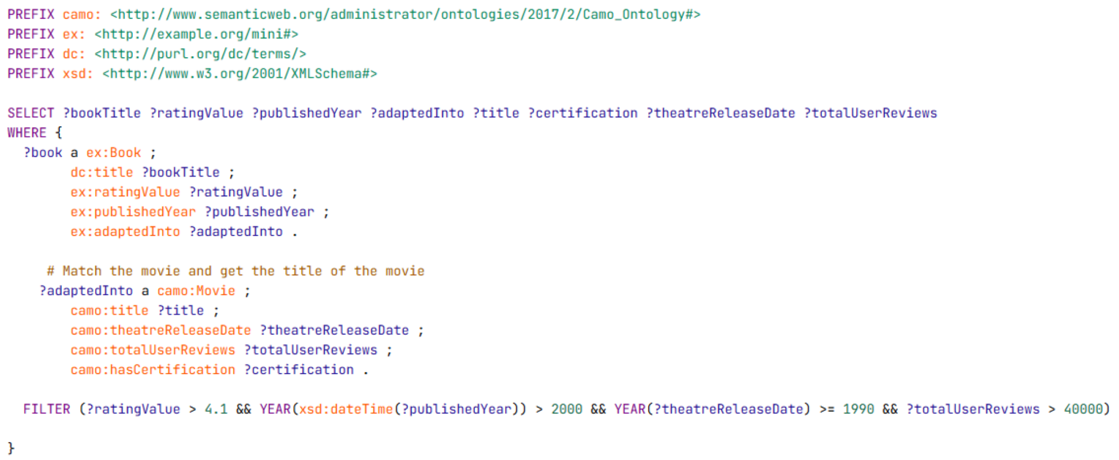


	book	bookTitle	bookRating	publishedYear	author
1	http://example.org/mini/book/30	"J.R.R. Tolkien 4-Book Boxed Set: The Hobbit and The Lord of the Rings"	"4.59"^^xsd:decimal	"2012-09-25"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien
2	http://example.org/mini/book/15361	"Les Deux Tours (Le Seigneur des Anneaux #2)"	"4.44"^^xsd:decimal	"2001-10-08"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien%2FFrancis%20Ledoux
3	http://example.org/mini/book/16546	"Pictures by J.R.R. Tolkien"	"4.47"^^xsd:decimal	"1992-10-01"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien%2FChristopher%20Tolkien
4	http://example.org/mini/book/15369	"The Lord of the Rings"	"4.5"^^xsd:decimal	"2003-10-06"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien%2FBrian%20Sibley%2FMichael%20Bakewell%2FStephen%20%20Oliver%2FIan%20Holm%2FMichael%20Hordern%2FRobert%20Stephens%2FJohn%20Le%20Mesurier%2FPeter%20Woodthorpe%2FGerard%20%20Murphy%2FHugh%20Dickson%2FPeter%20Vaughan%2FPeter%20Howell
5	http://example.org/mini/book/31	"The Lord of the Rings (The Lord of the Rings #1-3)"	"4.5"^^xsd:decimal	"2004-10-21"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien
6	http://example.org/mini/book/35	"The Lord of the Rings (The Lord of the Rings #1-3)"	"4.5"^^xsd:decimal	"2002-10-01"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien%2FAlan%20%20Lee
7	http://example.org/mini/book/5898	"The Lord of the Rings (The Lord of the Rings #1-3)"	"4.5"^^xsd:decimal	"2002-09-16"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien
8	http://example.org/mini/book/7353	"The Lord of the Rings (The Lord of the Rings #1-3)"	"4.5"^^xsd:decimal	"2002-02-22"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien%2FRob%20Inglis
9	http://example.org/mini/book/15336	"The Lord of the Rings / The Hobbit"	"4.59"^^xsd:decimal	"2002-10-07"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien
10	http://example.org/mini/book/15407	"The Lord of the Rings Millennium Edition Boxed Set (The Lord of the Rings #1-3)"	"4.5"^^xsd:decimal	"1999-10-10"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien
11	http://example.org/mini/book/2331	"The Lord of the Rings- 3 volumes set (The Lord of the Rings #1-3)"	"4.5"^^xsd:decimal	"2005-06-01"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien
12	http://example.org/mini/book/15245	"The Return of the King (The Lord of the Rings #3)"	"4.53"^^xsd:decimal	"2003-11-06"^^xsd:date	http://example.org/mini/author/J.R.R.%20Tolkien

Figure 12. Query results showing highly rated books authored by Tolkien.

3.5.3 Book-to-Movies Adaptations

At this point, some SPARQL queries were executed that link the two RDF graphs. The following query is designed to retrieve a list of highly rated books that have been adapted into popular movies, focusing specifically on modern publications and widely reviewed films.



```
PREFIX camo: <http://www.semanticweb.org/administrator/ontologies/2017/2/Camo_Ontology#>
PREFIX ex: <http://example.org/mini#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?bookTitle ?ratingValue ?publishedYear ?adaptedInto ?title ?certification ?theatreReleaseDate ?totalUserReviews
WHERE {
  ?book a ex:Book ;
        dc:title ?bookTitle ;
        ex:ratingValue ?ratingValue ;
        ex:publishedYear ?publishedYear ;
        ex:adaptedInto ?adaptedInto .

     # Match the movie and get the title of the movie
    ?adaptedInto a camo:Movie ;
        camo:title ?title ;
        camo:theatreReleaseDate ?theatreReleaseDate ;
        camo:totalUserReviews ?totalUserReviews ;
        camo:hasCertification ?certification .

  FILTER (?ratingValue > 4.1 && YEAR(xsd:dateTime(?publishedYear)) > 2000 && YEAR(?theatreReleaseDate) >= 1990 && ?totalUserReviews > 40000)

}
```

Figure 13. This SPARQL query retrieves books with a rating above 4.1, published after 2000, that have been adapted into movies released after 1990 with more than 40,000 user reviews.

This query is designed to retrieve a list of highly rated books that have been adapted into popular movies, focusing specifically on modern publications and widely reviewed films.

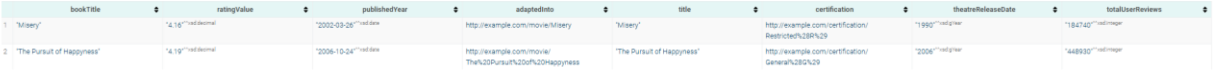
The query following data fields for each matching book and its corresponding movie such as, the title of the book (?bookTitle), the average rating of the book (?ratingValue), the year of publication (?publishedYear) etc.

The FILTER clause applies multiple conditions to ensure both the book, and its adaptation meet specific quality and popularity criteria:

1. The book must have a rating greater than 4.1
2. The book must have been published after the year 2000
3. The movie must have been released in or after 1990
4. The movie must have received more than 40,000 user reviews, indicating broad public engagement

This query enables the identification of modern, critically acclaimed books that were later adapted into successful and widely viewed films, giving useful insights into the latest adaptation trends.

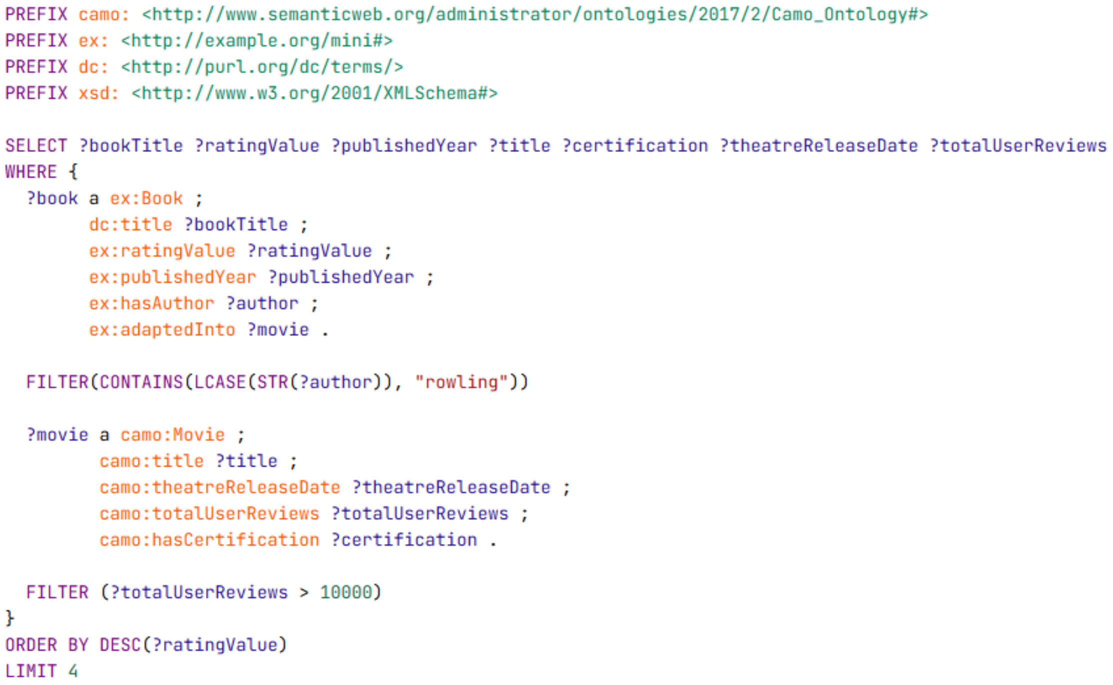
The results of the above query are shown in Figure 14. Results show two highly rated books, Misery and The Pursuit of Happyness, published after 2000 and adapted into movies released after 1990, each with over 40,000 user reviews.



	bookTitle	ratingValue	publishedYear	adaptedInto	title	certification	theatreReleaseDate	totalUserReviews
1	"Misery"	"4.16"^^xsd:decimal	"2002-03-26"^^xsd:date	http://example.com/movie/Misery	"Misery"	http://example.com/certification/Restricted%28R%29	"1990"^^xsd:gYear	"184740"^^xsd:integer
2	"The Pursuit of Happyness"	"4.19"^^xsd:decimal	"2006-10-24"^^xsd:date	http://example.com/movie/The%20Pursuit%20of%20Happyness	"The Pursuit of Happyness"	http://example.com/certification/General%28G%29	"2006"^^xsd:gYear	"448930"^^xsd:integer

Figure 14. Results show two highly rated books, Misery and The Pursuit of Happyness, published after 2000 and adapted into movies released after 1990, each with over 40,000 user reviews.

The next SPARQL query [Figure 15] is designed to extract the top 4 highly rated books written by J.K. Rowling that have been adapted into movies, prioritizing those with significant public reception based on user review counts. The query retrieves information for each book-movie pair such as, the title of the book (?bookTitle), book's average rating (?ratingValue), the publication year of the book (?publishedYear), etc.



```
PREFIX camo: <http://www.semanticweb.org/administrator/ontologies/2017/2/Camo_Ontology#>
PREFIX ex: <http://example.org/mini#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?bookTitle ?ratingValue ?publishedYear ?title ?certification ?theatreReleaseDate ?totalUserReviews
WHERE {
  ?book a ex:Book ;
        dc:title ?bookTitle ;
        ex:ratingValue ?ratingValue ;
        ex:publishedYear ?publishedYear ;
        ex:hasAuthor ?author ;
        ex:adaptedInto ?movie .

  FILTER(CONTAINS(LCASE(STR(?author)), "rowling"))

  ?movie a camo:Movie ;
         camo:title ?title ;
         camo:theatreReleaseDate ?theatreReleaseDate ;
         camo:totalUserReviews ?totalUserReviews ;
         camo:hasCertification ?certification .

  FILTER (?totalUserReviews > 10000)
}
ORDER BY DESC(?ratingValue)
LIMIT 4
```

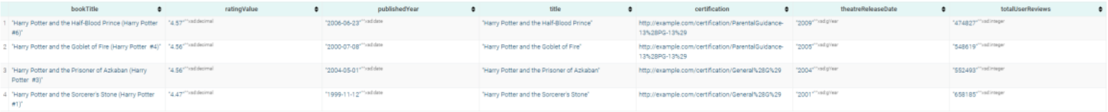
Figure 15. This query retrieves the top 4 highest-rated books authored by "Rowling" that have been adapted into movies with more than 10,000 user reviews, showing both book and movie details.

Two main filters are applied:

1. The FILTER(CONTAINS(LCASE(STR(?author)), "rowling")) ensures the author associated with the book includes the substring "rowling", allowing case-insensitive matches to identify all books authored by J.K. Rowling.
2. The second filter limits movies with more than 10,000 user reviews, highlighting only adaptations that resulted in notable users' attention.

The results are sorted in descending order by book rating to showcase the highest-rated works, and the output is limited to the top 4 results using LIMIT 4. This query supports analysis of the literary success and cinematic impact of J.K. Rowling's works, providing insights into which of her books have not only been highly rated by readers but also successfully transitioned into widely reviewed films.

The results of the above query are shown in Figure 16.



	bookTitle	ratingValue	publishedYear	title	certification	theatreReleaseDate	totalUserReviews
1	"Harry Potter and the Half-Blood Prince (Harry Potter #6)"	"4.57"^^xsd:decimal	"2006-06-23"^^xsd:date	"Harry Potter and the Half-Blood Prince"	http://example.com/certification/ParentalGuidance-13%28PG-13%29	"2009"^^xsd:gYear	"474827"^^xsd:integer
2	"Harry Potter and the Goblet of Fire (Harry Potter #4)"	"4.56"^^xsd:decimal	"2000-07-08"^^xsd:date	"Harry Potter and the Goblet of Fire"	http://example.com/certification/ParentalGuidance-13%28PG-13%29	"2005"^^xsd:gYear	"548619"^^xsd:integer
3	"Harry Potter and the Prisoner of Azkaban (Harry Potter #3)"	"4.56"^^xsd:decimal	"2004-05-01"^^xsd:date	"Harry Potter and the Prisoner of Azkaban"	http://example.com/certification/General%28G%29	"2004"^^xsd:gYear	"552493"^^xsd:integer
4	"Harry Potter and the Sorcerer's Stone (Harry Potter #1)"	"4.47"^^xsd:decimal	"1999-11-12"^^xsd:date	"Harry Potter and the Sorcerer's Stone"	http://example.com/certification/General%28G%29	"2001"^^xsd:gYear	"658185"^^xsd:integer

Figure 16. The query results display four top-rated Harry Potter books by "Rowling" that were adapted into movies with over 10,000 user reviews, showing their ratings, publication dates, and corresponding film details.

4 References

[1] V. Sejwal, «"CAMO," GitHub,». Available: https://github.com/vineet-sejwal/CAMO.

[2] D. B. a. L. Miller, «FOAF Vocabulary Specification,». Available: http://xmlns.com/foaf/spec/.

[3] B. D. a. F. Giasson, «The Bibliographic Ontology Specification,». Available: http://bibliontology.com/.

[4] H. Shankhdhar, «Top 1000 Movies by IMDB Rating,». Available: https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows.

[5] Soumik, «Goodreads-books,». Available: https://www.kaggle.com/datasets/jealousleopard/goodreadsbooks.