SDM Lab Assignment 2 - Knowledge Graphs (KG)

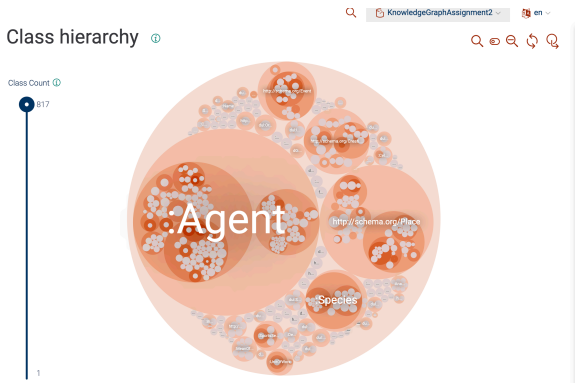
HanLing Hu, Lucia Victoria Fernandez Sanchez [1]

Knowledge graphs (KG) are pivotal for structuring interconnected data, enabling semantic reasoning and integration across heterogeneous sources, as emphasized in the foundational work on Web Data Management [1]. By leveraging formalisms like RDFS and OWL, KG support ontology inference, ensuring consistency and interoperability. Those keys to applications such as intelligent search and AI driven analytics.

A Exploring DBpedia

In this section, our primary objective is to familiarize ourselves with the GraphDB platform. As documented in official sources, GraphDB is a highly efficient, scalable, and robust graph database that supports RDF and SPARQL [2].

Figure A.1 illustrates the hierarchical structures within the platform. Notably, the database comprises over 817 classes, including the Agent class, which contains multiple subclasses, as depicted in A.2.



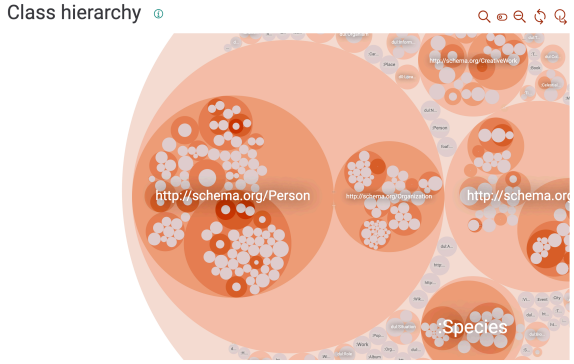


Figure A.2: Implementation of DBpedia hierarchies in GraphDB

B Ontology creation

1 TBOX definition

For the TBOX, we decided to create a whole new prefix

```
@prefix pap: <http://www.example.edu/papers/> .
...
pap:paper a owl:Class .
```

[1] GitHub Repository

```
pap:publication a owl:Class .
...
```

Thanks to this explicit prefix, we also make sure that all the properties, classes, and attributes are well-defined.

1.1 Classes and Sub-classes

As shown in the table B.1, here is an implementation decision related to the Classes:

We began by modeling the straightforward entities we were confident were classes, such as Paper and Keyword. However, we faced the first challenge with Author, which could also be a Reviewer. Considering that a person cannot be both in the same context,and following the examples from the lectures, we opted to create a parent class person with two subclasses: author and reviewer.

Regarding publication, we faced an issue since a Conference entity is quite similar to Workshop, therefore we decided to create a Class called event that stores the previous definitions as SubClasses, by doing so, we are making sure of the appropriate implication of using a Knowledge Graph. Nevertheless, we modeled journal as an independent class.

1.2 Inference

The class hierarchy implementation in GraphDB introduces two complex classes: *written_by* and *reviewed_by* (see the orange rows in the table D.2). Following the semantic modeling methodology given in the course [3]. This approach is optimal for handling attributed relationships, where complex properties are elevated to class entities within the ontology.

By modeling these relationships as distinct classes rather than simple properties, we achieve several practical benefits. First, it provides clearer semantic distinction between authorship (written_by) and review (reviewed_by). Finally, we have more awareness of the attributes such as position, score, etc. And avoid a mistake to relate the node to others that have nothing to do with the original context. in the Figure B.2 you can see the final schema implementation [1] .

Therefore, the implementation balances conceptual design with practical query efficiency. For a better and deeper analysis, see Query 2: Usefulness of a complex class.

In addition, we managed to create four sub-Properties. The first related to the property **published_in** which expands the concept of **volume** and **edition** (both subClasses of publication), in order to have an inference and review the proper domains.

The second ones defined as **belongs_to_conference** and **belongs_to_workshop** are based on the Property **belongs_to**. This implementation aims to make a difference between a workshop and a conference. In the table B.2, we can observe the subproperties and in color Orange, we see the properties.

2 ABOX definition

For the current assignment, we started by using the schema logic from assignment 1, however, we made sure to modify two main things:

1. **Feedback**: Before going further in the implementation for this whole new task, we made

[1] Better Quality Schema Version.

Class	SubClassOf	Description
paper	-	Related to the publication format, here we get everything related to the basic paper information.
publication	volume	Useful for the paper and journal instances.
	edition	Useful to identify and connect a paper with conference or workshop.
event	workshop	Related to all the papers published in a workshop. The Class is an event since conference and workshop are modeled similar way.
	conference	To identify the event type, check the property *e_type*.
person	author	By sharing class with reviewer and while there is not significant difference between the two, it was important to make a subclass only for authors and reviewers.
	reviewer	Useful to fully visualize the constraint that an author and a reviewer should not be the same given one paper.
keyword	-	Related to keywords.
journal	-	Since conference and journal share the event class, here we decided to create journal as a class.
written_by	-	Practical to have attributes in the implementation.
reviewed_by	-	Complex class, similar to the previous one, convenient for having Metadata.

Table B.1: Class Hierarchy with Subclass Relationships

Domain	Predicate	Range
edition	**belongs_to**	**event**
edition	belongs_to_conference	conference
	belongs_to_workshop	workshop
paper	**published_in**	**publication**
paper	published_in_volume	volume
	published_in_edition	edition

Table B.2: Triplets using subclasses and subproperties

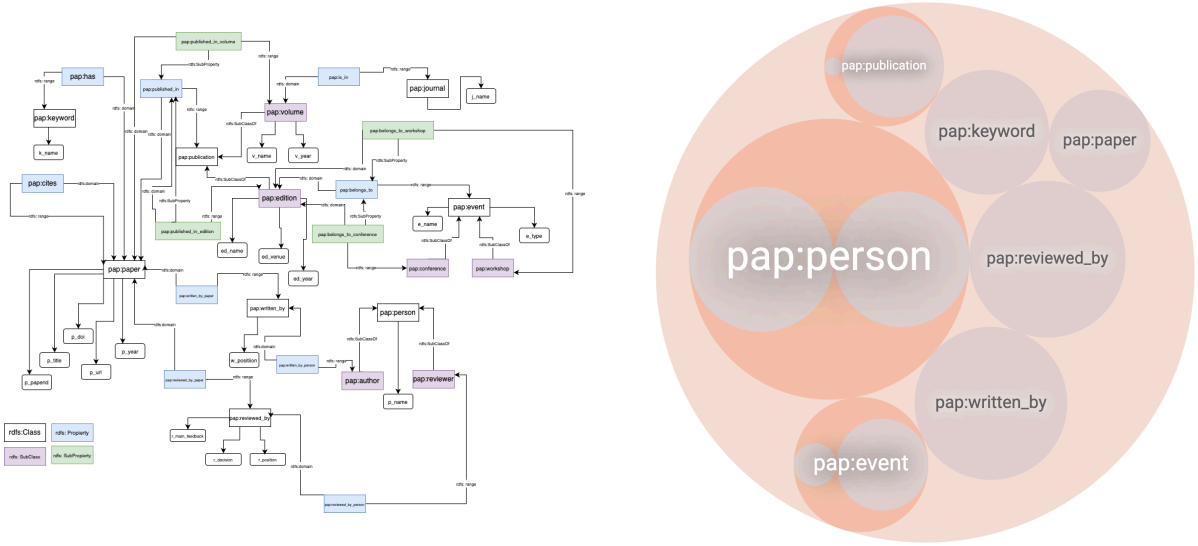


Figure B.2: Schema and implementation of Class hierarchies in GraphDB

sure to change all the differences that the professor Anna gave us during the Assignment 1 grade.These refinements, documented in the Appendix's *Feedback Approach section*, ensured a stronger foundation for the current delivery.

2. **Implementation considerations**: During the implementation of the adaptaded schema 1 version in GraphDB, we encountered a new issue: **it was not possible to create two instances with the same property name under different classes**. When attempted, the implicit property incorrectly assigned the implementation to both classes. To resolve this issue, we modified the schema by renaming all duplicate properties. The updated version of the schema is available in the Appendix's figure D.3.

2.1 Processing Data

The main idea of the file *request.py* is bringing the Data from the dblp computer science bibliography repository. We have other Python file called *synthetic_data.py* that helps to bring the Keywords related to Computer Science fields and mentioned in the Assignment 1, the event information and paper reviews. For the latter synthetic data, we decided to make sure two things:

1. All papers should have review's score greater than 5 (range: 6-10)
2. If the paper has score 6, the paper has **partially accepted** as decision, greater value score has **accepted**
3. Each paper has the same main review, same score but different position, therefore each

reviewer has an instance of reviewed_by with the difference of position. See image B.3 for an extract an example.

	paperTitle	reviewerName	reviewerPosition	reviewScore	reviewFeedback
1	"Computing in Data Science or Data in Computer Science? Exploring the Relationship between Data Science and Computer Science in K-12 Education."	"Mathieu Roche"	"1"^^xsd:int	"8"^^xsd:int	"Excellent practical applications"
2	"Computing in Data Science or Data in Computer Science? Exploring the Relationship between Data Science and Computer Science in K-12 Education."	"Gholamreza B. Khosrovshahi"	"2"^^xsd:int	"8"^^xsd:int	"Excellent practical applications"
3	"Computing in Data Science or Data in Computer Science? Exploring the Relationship between Data Science and Computer Science in K-12 Education."	"Mohammad Amin Shokrollahi"	"3"^^xsd:int	"8"^^xsd:int	"Excellent practical applications"

Figure B.3: Extract of one paper having the same score and main feedback with different position per reviewer.

In the final section of our implementation, we include two key functions to standardize publication types. These functions ensure that regardless of the paper types retrieved from the API, our final JSON output consistently contains only the three designated categories: Conference, Workshop, and Journal.

Regarding institutional affiliations (including university and company details), while these elements are not utilized in our current implementation, we deliberately maintained them in the codebase. This decision reflects our approach to preserve the integrity of the original data structure of the previous deliverable, avoiding unnecessary modifications to established components.

2.2 Constraints

Continuing the previously established quality constraint (all papers having a minimum score of 6), we introduced additional validation rules during the data request process to ensure consistency and reliability.

First, all papers are required to include the keyword **Computer Science** as a common identifier. This implicit filtering guarantees thematic coherence across the dataset while preventing potential conflicts with broader categories such as Big Data Management.

To maintain uniformity, each paper is assigned exactly three synthetic keywords, generated through a controlled randomization process. Additionally, our implementation automatically assigns one review (as stated before) and one institutional affiliation per paper, both selected at random from predefined arrays of valid entries.

Furthermore, we enforce strict separation between authors and reviewers. Specifically, the system validates that none of the three assigned authors overlap with the three randomly selected reviewers.

Finally, we made sure during our ABOX implementation that the triplets are perfectly fine and the are instanced in their corresponding sub-property and/or their sub-class. For example, person does not have any instance by itself, however the inference in the system makes sure

that an author or reviewer, it is an implicit instance of person. See figure B.4, we have the author "Ali Shokoufandeh" who also is a person, this is one of the biggest assets regarding the Knowledge graphs.

	authorName	class	classType
1	"Ali Shokoufandeh"	pap:author	"Direct instance"
2	"Ali Shokoufandeh"	pap:person	"Inferred superclass"

Figure B.4: Extract of one author named "Ali Shokoufandeh".

3 Querying the ontology

3.1 First query: Pattern Matching

This SPARQL query analyzes relationships between papers by examining their citation papers and shared research topic (keywords). Since in the pre processing part is restricted to never be auto-cited, adding a filter in the query is useless.

First, it manages to connect the paper and identifies if there is a "relationship" with other paper (cites). Then determines how closely related these papers are based on their subject matter. The query focuses specifically on finding pairs of papers connected through citations that share common keywords, all of them related to Computer Science as implicitly stated in the Assignment instructions.

For each of these citation relationships, it examines the keywords assigned to both the citing paper and the cited paper. The system then identifies any keywords that appear in both papers, indicating shared research topics or subject areas. These common keywords are compiled into a list for each paper pair, showing precisely which topics connect the two works.

The output shows three key pieces of information: the title of the paper, the title of the paper it references, and a complete list of all topics they have in common. The query sorts these results to show the most strongly connected pairs first (those papers that share the greatest number of keywords).

This type of analysis is particularly valuable for researchers and academics because it reveals important patterns in how knowledge develops within a field. Scholars can better understand how research ideas evolve and spread through the academic literature.

```
PREFIX pap: <http://www.example.edu/papers/>

SELECT ?Paper ?Cited
       (GROUP_CONCAT(DISTINCT ?sharedKeyword; SEPARATOR=", ") AS ?Keywords)
WHERE {
  ?paper a pap:paper ;
           pap:p_title ?Paper ;
           pap:cites ?cited .

  ?cited a pap:paper ;
           pap:p_title ?Cited .
```

```
  ?paper pap:has_keyword ?keyword .
  ?keyword pap:k_name ?sharedKeyword .

  ?cited pap:has_keyword ?citedKeyword .
  ?citedKeyword pap:k_name ?sharedKeyword .

}
GROUP BY ?Paper ?Cited
ORDER BY DESC(COUNT(DISTINCT ?sharedKeyword))
LIMIT 100
```

As we can see in the image B.5, in row 1 we got a paper with its cited paper having in common the keywords **"Machine Learning"** and **"Big Data"**, while, in the 2nd row we have the same papers but the other way around (the cited paper became the main paper and viceversa), having in common the same keywords. Similar logic for rows 3 and 4. As in the 5, we see a result with only one similar keyword in common (**"Data Modeling"**), proving that the query order by number of keywords as well.

	Paper	Cited	Keywords
1	"Computing in Data Science or Data in Computer Science? Exploring the Relationship between Data Science and Computer Science in K-12 Education."	"SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Bratislava, Slovak Republic, January 20-23, 2025, Proceedings, Part I"	"Machine Learning, Big Data"
2	"SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Bratislava, Slovak Republic, January 20-23, 2025, Proceedings, Part I"	"Computing in Data Science or Data in Computer Science? Exploring the Relationship between Data Science and Computer Science in K-12 Education."	"Machine Learning, Big Data"
3	"Research in Computer Science - 17th African Conference on Research in Computer Science and Applied Mathematics, CARI 2024, Bejaïa, Algeria, November 24-26, 2024, Proceedings"	"SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Bratislava, Slovak Republic, January 20-23, 2025, Proceedings, Part II"	"Data Modeling, Virtualization"
4	"SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Bratislava, Slovak Republic, January 20-23, 2025, Proceedings, Part II"	"Research in Computer Science - 17th African Conference on Research in Computer Science and Applied Mathematics, CARI 2024, Bejaïa, Algeria, November 24-26, 2024, Proceedings"	"Data Modeling, Virtualization"
5	"Theoretical Aspects of Computer Science, Advanced Lectures (First Summer School on Theoretical Aspects of Computer Science, Tehran, Iran, July 2000)"	"Research in Computer Science - 17th African Conference on Research in Computer Science and Applied Mathematics, CARI 2024, Bejaïa, Algeria, November 24-26, 2024, Proceedings"	"Data Modeling"

Figure B.5: Extract of the output of the query one.

3.2 Second query: Usefulness of Complex Class

This query helps identify which reviewers are consistently giving the highest scores to research papers. It analyzes all review assignments in the system, calculates each average score across the papers they have evaluated, and shows how many papers they have reviewed (more than 1). The results are sorted to show reviewers with the highest average scores at the top, giving you a clear picture of who tends to be most favorable in their evaluations.

Additionally, in this implementation we are using the complex class **reviewed_by** in order to have a easier and robust way to query between the paper and the reviewer.

```
PREFIX pap: <http://www.example.edu/papers/>

SELECT ?reviewerName
       (AVG(?score) AS ?averageScore)
       (COUNT(DISTINCT ?paper) AS ?NumberOfPapers)
WHERE {
  ?review a pap:reviewed_by ;
          pap:reviewed_by_paper ?paper ;
          pap:reviewed_by_person ?reviewer ;
          pap:r_score ?score .

  ?reviewer pap:per_name ?reviewerName .
}
GROUP BY ?reviewerName
HAVING (COUNT(DISTINCT ?paper) > 1)
ORDER BY DESC(?averageScore)
LIMIT 10
```

As we can see in the output shown in the image B.6, it is useful for conference organizers, this information is valuable for understanding reviewer behavior and maintaining quality control. Also, it reveals which reviewers might be consistently tolerant (high average scores) or particularly strict (low scores).

	reviewerName	averageScore	NumberOfPapers
1	"Zarek Drozda"	"8"^^xsd:decimal	"2"^^xsd:integer
2	"Mathieu Roche"	"8"^^xsd:decimal	"2"^^xsd:integer
3	"Kamel Barkaoui"	"7.6666666666666666666666667"^^xsd:decimal	"3"^^xsd:integer
4	"Kathi Fisler"	"7.5"^^xsd:decimal	"2"^^xsd:integer

Figure B.6: Extract of the output of query 2.

Finally, in the ABOX we defined a function that shows the statistics per each ingestion to the system, as we can see in the image B.7, we can see that there was not any instance related to Journal (therefore neither a volume created). Also we can recall that the json file had 5 papers with 12 authors, who 10 of them are reviewers as well.

This simple but useful function coded will help us to have a brief but straightforward details of the main unstructured data. Thus, we can make it easily scalable given more data, or in this context, giving more paper data.

C Conclusion

1 Property or Knowledge Graphs

During the implementation of the assignment 1 schema in GraphDB, we encountered an issue where it was not possible to create two instances with the same property name under different classes.

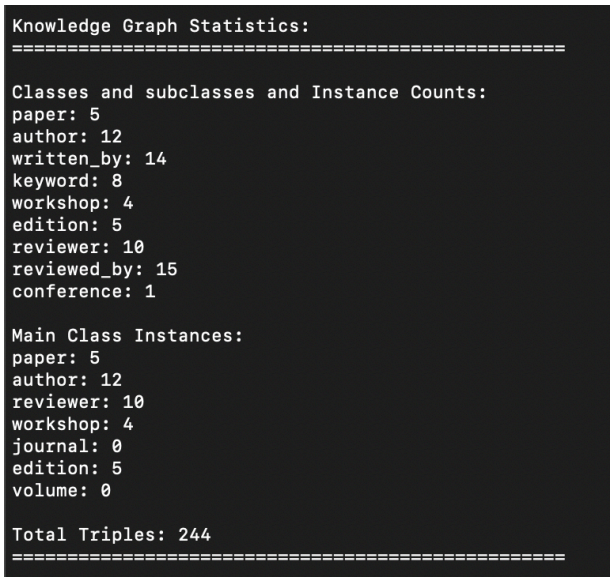


Figure B.7: Extract of statistics.

When attempted, the implicit implementation incorrectly assigned the property to both classes. To resolve this issue, we modified the schema by renaming all duplicate properties. The updated version of the schema is available in the repository.

For example, if we have an attribute called **name** for the Concept **e__name**, the first letter(s) before the _ is making a reference to the actual concept referred to.

Additionally, we needed to implement a proper normalization of words, by doing so, we made sure that the author names (and therefore the reviewers as well), are normalized. Otherwise the system would create two instances of the same person. For example, the person **Rastislav Kr̈0̈0c3̈0̈0a1lovic** should be instead **Rastislav Krlovic**.

In further details, we needed to create and use the concepts of Sub-Class and Sub-propierties since in our logic there are some instances that should be author (specific), instead of person (who could be author or reviewer). It was important during our implementation to make sure two things: the instances are well defined depending on the paper context and second, the inference was well done. This is the main difference between properties graphs and knowledge graphs, since the latter brings more metadata that can be inferred by the instances instead of explicitly handle them.

Overall, it was crucial for us to have good fundamentals of the concept and a better understanding of the implementation because each of the ideas needed to be carefully coded.

D Appendix

1 Feedback approach

In this section, our aim is to explain in simple words the changes that we needed to improve.

First, for coding purposes, we changed any capital letter to lower case for every single element in our schema. This change was due only for a better handling in Python.

Additionally, we decided to switch the **paper** and **:published__in** relationship. See image D.1 to see the graphic changes:

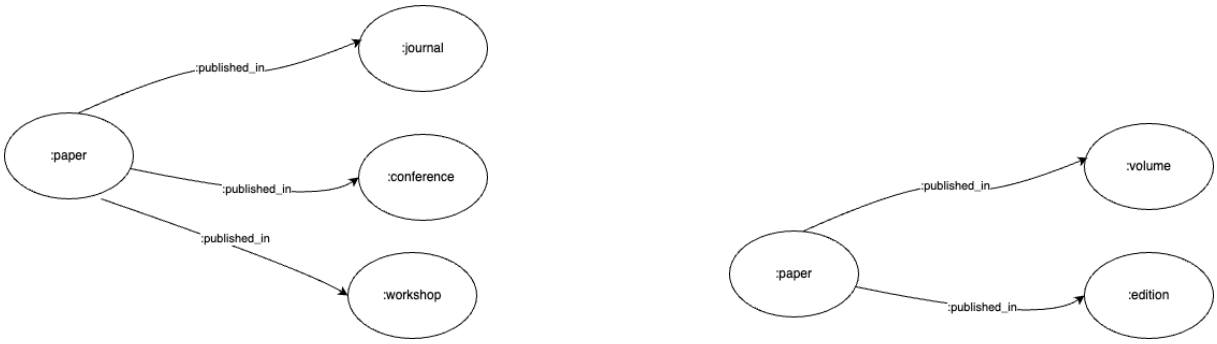


Figure D.1: *published_in* changes.

Also, we deleted the **main_author** attribute in the paper concept.Finally, we store reviews information into the relationship *reviewed_by*.

Here is the final edited schema; this Property graph will be adapted to be a Knowledge Graph.

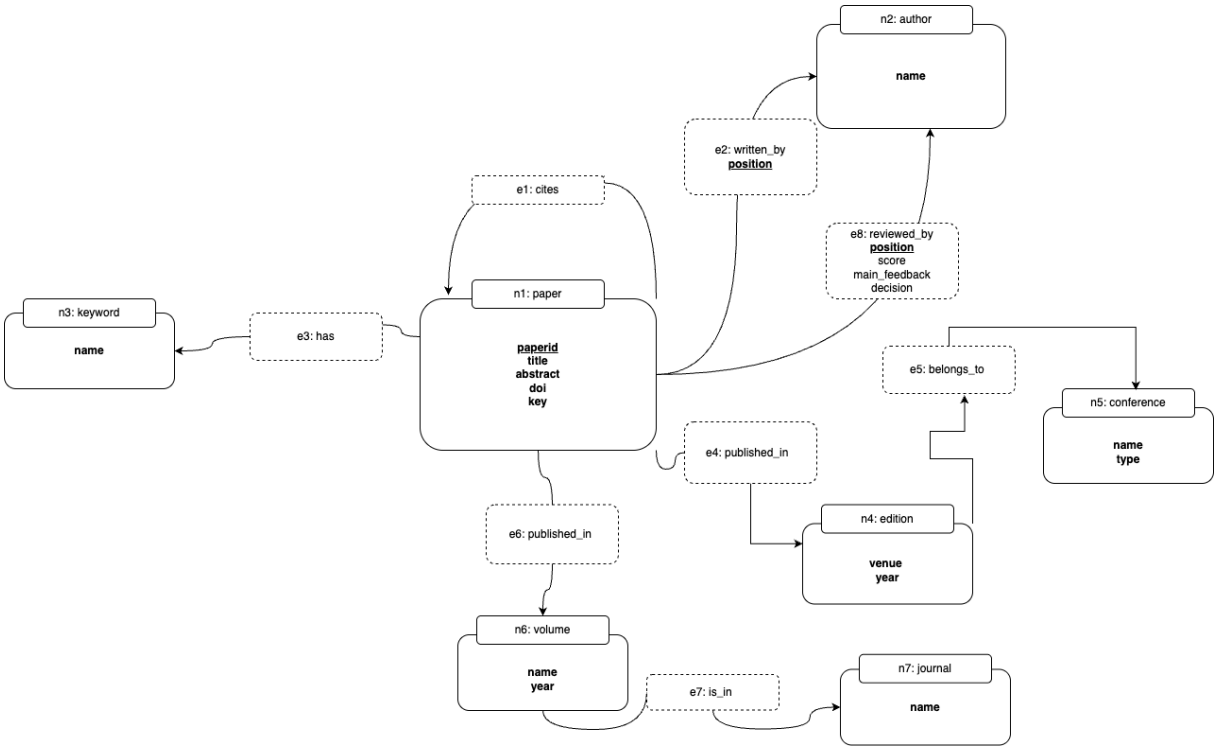


Figure D.2: Schema from Assignment 01 with all the changes from the Feedback.

2 Constraints approach

In this section, we outline the different strategies explored for imposing constraints in our knowledge graph, referencing insights from *Web Data Management* [1].

Initially, we attempted to encode constraint restrictions using OWL semantics in GraphDB. However, due to the RDFS-based configuration specified in the assignment (see Figure D.4), we found that OWL constraints could not be fully enforced.

We noticed that if we tried to code a constraint restriction using the OWL semantics on GraphDB, it is not possible to be fully followed. This is due to the RDFS configuration mentioned in the assignment specifications.

Below is an example of an OWL restriction limiting a paper to a maximum of three authors:

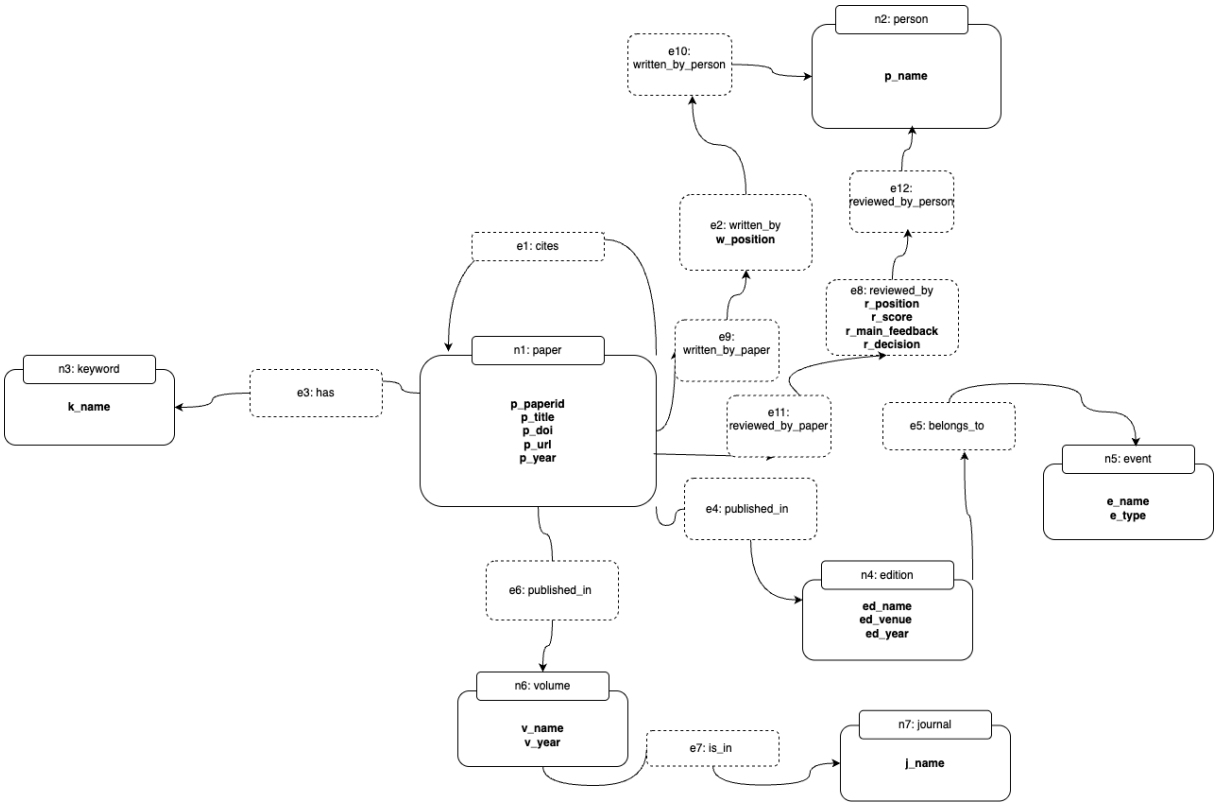


Figure D.3: Adapted version from schema 02 version.

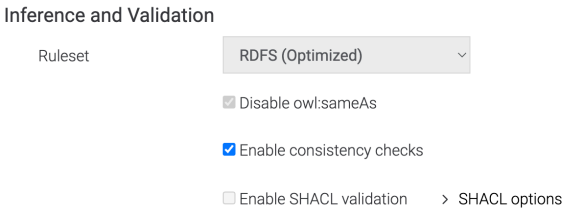


Figure D.4: Ruleset RDFS (Optimized) configuration.

```
pap:paper rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty pap:written_by ;
    owl:maxCardinality "3"^^xsd:nonNegativeInteger ] .
```

Despite proper syntax, GraphDB's RDFS ruleset prevented strict adherence to this constraint.

We then investigated **SHACL**, a more robust framework for defining constraints. However, our GraphDB ruleset configuration did not support SHACL validation (see Figure D.5).

The equivalent extract SHACL implementation for the cardinality restriction is:

```
@prefix pap: <http://www.example.edu/papers/>
@prefix sh: <http://www.w3.org/ns/shacl#> .

pap:paper a sh:NodeShape ;
    sh:targetClass pap:paper ;
    sh:property [
```

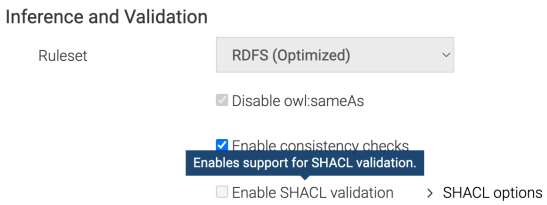


Figure D.5: Enable SHACL validation disabled option.

```
sh:path pap:written_by_paper ;
sh:maxCount 3 ;
....
```

Given these limitations, we finally implemented constraint checks during **pre-processing**, where we merged API data and synthetic data. This approach ensured compliance before data ingestion.

Class	**Subclass**	**Properties**
paper	-	p_paperid, p_title, p_doi, p_url, p_year
publication	volume	v_name, v_year
publication	edition	ed_venue, ed_year, ed_name
event	workshop	e_name, e_type
event	conference	e_name, e_type
person	author	per_name
person	reviewer	per_name
keyword	-	k_name
journal	-	j_name
written_by	-	w_position
reviewed_by	-	r_position, r_score, r_main_feedback, r_decision

Table D.1: Complete Class Hierarchy with Properties.

Bibliography

[1] Serge Abiteboul et al. *Web Data Management*. Cambridge University Press, 2011.

[2] Ontotext. *GraphDB Documentation*. [Online; accessed 07-Jun-2025]. June 2024. URL: `https://graphdb.ontotext.com/documentation/11.0/index.html`.

[3] Óscar Romero and Anna Queralt. *Semantic Data Management*. 2023.

Domain	Predicate	Range	Description
paper	cites	paper	A paper citing another paper
	has_keyword	keyword	A paper associated with a keyword
	published_in	publication	A paper published in a publication
edition	belongs_to	event	An edition belonging to an event
volume	is_in	journal	A volume contained in a journal
reviewed_by	reviewed_by_paper	paper	A review associated with a paper
	reviewed_by_person	reviewer	A review conducted by a reviewer
written_by	written_by_paper	paper	An authorship relation to a paper
	written_by_person	author	An authorship relation to an author

Table D.2: Object Properties with Domains and Ranges