UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Efficient and Extensive Construction of Knowledge Graphs from Diverse Data with Declarative Mappings

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Julián Arenas Guerrero
MSc in Artificial Intelligence

Madrid, 2025


UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Doctoral Degree in Artificial Intelligence

Efficient and Extensive Construction of Knowledge Graphs from Diverse Data with Declarative Mappings

D O C T O R A L T H E S I S

Submitted for the degree of Doctor by:

Julián Arenas Guerrero
MSc in Artificial Intelligence

Under the supervision of:
Dr. Óscar Corcho
Dr. María S. Pérez

Madrid, 2025

Title: Efficient and Extensive Construction of Knowledge Graphs from Diverse Data with Declarative Mappings

Author: Julián Arenas Guerrero

Doctoral Programme: Artificial Intelligence

Thesis Supervision:

Dr. Óscar Corcho, full professor, Universidad Politécnica de Madrid

Dr. María S. Pérez, full professor, Universidad Politécnica de Madrid

External Reviewers:

Thesis Defense Committee:

Thesis Defense Date:

This investigation has been partially supported by the Euratom Research and Training Programme 2019–2020 (grant agreement No. 900018).

A mi familia.

Acknowledgement

La investigación que se presenta en esta tesis ha sido fruto de 4 años de trabajo en el grupo "Ontology Engineering Group" de la Universidad Politécnica de Madrid, y ha sido posible gracias a la financiación pública de la ciencia y al trabajo indirecto de muchas personas.

Agradezco a mi director y directora, Oscar y María, su apoyo durante este tiempo. Quiero expresar mi agradecimiento a Olaf, Manolis y Sebastián por acogerme en mis estancias en Linköping, Atenas y Santiago.

El motor de este trabajo ha sido el cariño, apoyo y cuidados de mi gente. Gracias a mi madre, mi padre, mi hermana y mi hermano; gracias a mis niñas: Carol, Karen, Lina y Carmen.

La ciencia es revolucionaria y contribuye a la liberación de la humanidad.
G. Politzer

Abstract

Heterogeneous data is typically integrated into a knowledge graph using declarative mappings. There are two ways of constructing the graph with the mappings: materialization and virtualization. Historically, virtualization has focused on relational databases due to their predominance and because of the possibility of pushing down costly computations in SQL. However, the current prevalence of diverse data has rendered existing virtualization techniques insufficient. This diversity is not a problem in materialization, but it must be optimized to create large graph dumps.

The first research problem addressed in this thesis is the optimization of materialization. We propose mapping partitioning, which divides rules into groups with non-overlapping outputs. Two strategies are introduced to optimize materialization: parallel processing of the mapping groups reduces execution times, and sequential processing reduces the consumption of memory. We devise algorithms for partitioning a mapping and validate their effectiveness on several benchmarks and use cases. We empirically compare materialization with mapping partitioning against a baseline without partitioning and state-of-the-art systems. The results demonstrate the positive impact of partitioning on materialization with a large reduction in execution times and memory usage.

Building on optimized materialization, we address our second research problem: generic virtualization over multiple and diverse sources. We propose shifting from the established query translation approach to a data translation approach. The idea is to converge on the fly the relevant subset of data for a SPARQL query into an intermediate representation. Coalescing heterogeneous data into this representation is easier than translating queries into the potentially diverse query languages of the local sources. We present the `ITT` architecture, based on star-shaped mapping candidate selection to find the relevant set of rules, rapid materialization of intermediate results into a triple table, and unfolding SPARQL queries to SQL over the triple table. Experiments with mixed data sources demonstrate the generic nature of `ITT`. Compared to query translation with Ontop, the experimental results showed that `ITT` outperformed query translation in many cases. While `ITT` exhibited a more intensive use of memory, it did not experience timeouts, unlike Ontop.

The thesis also addresses other challenges in KG construction arising from the emergence of the RDF-star graph model and the need for flexible data transformations. For RDF-star, we devise the RML-star mapping language, a baseline materialization algorithm, and a method for the automatic extraction of RML-star from property graphs to mitigate labor-intensive mapping development. Regarding flexible data transformations, we extend RML views to tabular files to leverage SQL, overcoming RML's limitations with complex joins and composite data values. We also propose unfolding data transformations defined in RML-FNML to RML views, enabling query translation-based virtualization systems to process RML-FNML with prior unfolding of the mappings. Finally, we implemented Python user-defined functions in RML to allow more flexible data transformations using a high-level programming language; their positive impact on knowledge graph construction has been demonstrated at an industry scale in the chemical sector.

Resumen

La integración de datos heterogéneos en un grafo de conocimiento se realiza comúnmente mediante reglas de mapeo declarativas. Existen dos formas de construir el grafo con estas reglas: materialización y virtualización. Históricamente, la virtualización se ha centrado en las bases de datos relacionales debido a su predominancia y a la posibilidad de delegar tareas computacionalmente intensivas mediante SQL. No obstante, las técnicas de virtualización existentes son insuficientes para manejar la diversidad de los datos en el contexto actual. Esta diversidad no supone un problema para la materialización, pero este proceso debe optimizarse para generar grafos de gran tamaño de manera eficiente.

El primer problema de investigación abordado en esta tesis es la optimización de la materialización. Para ello, proponemos el particionamiento de las reglas de mapeo, dividiéndolas en grupos que generan subgrafos sin solapamiento. Presentamos dos estrategias para optimizar la materialización: la paralelización de los grupos de mapeo, que reduce los tiempos de ejecución, y el procesamiento secuencial, que disminuye el consumo de memoria. Proponemos algoritmos para particionar las reglas y validamos su efectividad en varios benchmarks y casos de uso. Comparamos empíricamente la materialización con particionamiento frente a una referencia sin particionamiento y otros sistemas. Los resultados demuestran que nuestra propuesta disminuye de forma significativa los tiempos de ejecución y el uso de memoria.

Basándonos en lo anterior, abordamos un segundo problema de investigación: la virtualización genérica sobre múltiples fuentes de datos diversas. Para ello, proponemos cambiar el enfoque clásico de traducción de consultas a uno de traducción de datos. La idea es fusionar al vuelo el subconjunto relevante de datos para una consulta SPARQL en una representación intermedia. Traducir datos heterogéneos a esta representación es más sencillo que traducir consultas a los lenguajes potencialmente diversos de las fuentes locales. De esta manera, introducimos la arquitectura `ITT`, que se basa en la selección de reglas de mapeo con forma de estrella, la materialización eficiente de resultados intermedios en una tabla de tripletas empleando particionamiento de las reglas, y la traducción de consultas SPARQL a SQL sobre la tabla de tripletas. Los experimentos con fuentes de datos mixtas demuestran el carácter genérico de `ITT`. En comparación con la traducción de consultas con Ontop, los experimentos arrojan un mejor rendimiento de `ITT` en muchos casos. Aunque `ITT` hace un uso intensivo de la memoria, a diferencia de Ontop no supera los límites de tiempo establecidos en ningún caso.

La tesis también aborda otros retos en la construcción de grafos de conocimiento. El primero es la generación de RDF-star, para lo que proponemos el lenguaje de mapeo RML-star, un algoritmo de materialización y un método para la extracción automática de RML-star a partir de un grafo de propiedades, con el fin de mitigar el desarrollo laborioso de los mapeos. En cuanto a desafíos en la transformación de los datos, extendemos las vistas de RML para procesar ficheros tabulares, utilizando SQL para superar limitaciones de RML respecto a *joins* complejos. Además, proponemos la traducción de las transformaciones de datos definidas en RML-FNML a vistas de RML, permitiendo que los sistemas de virtualización basados en la traducción de consultas soporten RML-FNML mediante un preprocesamiento previo. Finalmente, implementamos funciones definidas por el usuario en Python para permitir transformaciones de datos más flexibles utilizando un lenguaje de programación de alto nivel.

Abbreviations and acronyms

DBMS	Database Management System
GTFS	General Transit Feed Specification
IQ	Intermediate Query
ITT	Intermediate Triple Table
KG	Knowledge Graph
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
OM	Object Map
OOM	Out Of Memory
PG	Property Graph
PM	Predicate Map
POM	Predicate-Object Map
R2RML	RDB 2 RDF Mapping Language
RDB	Relational DataBase
RDBMS	Relational DataBase Management System
RDF	Resource Description Framework
RML	RDF Mapping Language
SF	Scaling Factor
SM	Subject Map
SQL	Structured Query Language
TM	Triples Map
TO	Time Out
VKG	Virtual Knowledge Graph
W3C	World Wide Web Consortium

XML eXtensible Markup Language

Table of Contents

Chapter 1

Introduction

Data-driven organizations typically work with a large number of diverse data sources. This data is often sparse and isolated across divisions and departments, each maintaining its own databases to manage information, resulting in separated data silos. These silos hinder the unified utilization of data, a challenge that becomes even more difficult when data from different organizations needs to be integrated. The combined exploitation and analysis of these heterogeneous and potentially large volumes of data are critical in use cases that require the collection of value from the data in whole [102].

However, the unified management and access of data silos is complex. The problem of combining data in separated silos to create a single, comprehensive, interconnected, unified view of the data for its effective consumption is known as data integration [96]. Data integration comes with challenges. First, there is a need for a flexible data model for the unified view to which diverse data can be smoothly coalesced. Another problem is mapping the diverse data sources to the target view following the aforementioned data model. This process must be systematic, maintainable, and reproducible. Finally, data integration must scale to the large amounts of data prevalent today, for which efficient techniques are required.

For the representation of integrated data, knowledge graphs [77] (KGs) have been widely adopted in recent years. Although they have existed for some time, their wide acceptance today was particularly motivated by the Google KG launch [120] in 2012, their subsequent adoption by other large companies [109] and the emergence of open KGs such as DBpedia [95], YAGO [110] and Wikidata [132]. The suitability of graph data models for data integration comes from its flexibility, as they are schema-on-read, meaning they do not enforce a strict schema for the unified view. This flexibility allows for adaptation to evolving data and easy integration of additional sources as needed. Additionally, graph query languages enable rich exploitation of the integrated data.

Although diverse data can be incorporated into a KG in an ad hoc programmatic manner, it is not maintainable and a systematic approach is required. Declarative mappings are the most accepted approach for this. To illustrate the benefit of a declarative approach, consider relational databases. It is possible to query relational data using programming languages such as Java or Python. However, this approach is neither maintainable nor scalable. Instead,

declarative querying with the SQL language is preferred. Relational databases can reliably process complex SQL queries without repeatedly implementing execution procedures, and providing users with rich functionality. For data integration, a similar approach is desirable: mapping data from heterogeneous sources to a target view should be done in a high-level and declarative manner. Ideally, the mappings should be defined in a standard language, similar to SQL. In the case of mappings, the standard comes from the World Wide Web Consortium (W3C) and is called R2RML [44]. The W3C also provides many other standards for KG management, such as RDF for the graph data model [43], SPARQL for querying [65], among others (see Chapter 2).

Furthermore, another advantage of a declarative approach is that systems can transparently implement optimizations. In the case of SQL, efficient querying has been extensively studied over the last decades, and modern relational database management systems are highly optimized. As mentioned above, optimizations in data management, for querying, but also for integrating data, are critical given the large volumes of data available today.

Although in recent years there have been advances in data integration, further work is needed to achieve the full potential of integrating data as KGs with declarative mappings. This thesis is devoted to advancing this area, addressing current limitations of declarative mapping languages and systems, and proposing optimizations for KG construction at scale.

1.1 Motivation

1.1.1 Materialization vs. Virtualization

There are two approaches and associated systems to access a KG with declarative mappings: materialization and virtualization. Materialization systems [16] retrieve data from diverse sources, apply the schema and data transformations defined in the declarative mappings, and create a graph dump. The dump can then be exploited by loading it into a triplestore. This is also known as the extract, transform, and load (ETL) approach. In the virtualization approach [135], the user provides the mappings and a graph query, and systems obtain the query result on the fly, maintaining the data in its original form (e.g., a relational or document database).

Materialization can be challenging and costly due to the transformation of potentially large data sources. Although some optimizations have been proposed (see Section 3.1), they are sometimes not sufficient to handle large volumes of data. At the beginning of this work, we carried out an experimental evaluation of state-of-the-art systems and showed that they timeout or run out of memory when creating a KG comprising ~400 million triples. Considering that knowledge bases like YAGO 4 [110] have ~2 billion triples, materialization systems need to scale to create larger KGs.

The expensive transformation of data is avoided with virtualization. However, a limitation of virtualization systems is that they are mainly focused on relational databases [135]. This comes from the prevalence of the relational data model years ago. The focus of virtualization in relational databases is evident, for instance, in the R2RML standard, which targeted the

relational data model and omitted others. Virtual knowledge graph systems access relational databases by translating SPARQL queries to SQL, exploiting mature SQL infrastructure, and pushing down computations to relational database management systems. However, today data is stored in a variety of data formats beyond the relational model. Moreover, the current query translation approach is limited to a single relational database, it is not possible to directly access multiple databases [78]. Although some work addressed document stores, the techniques still present the limitation of being data model-specific. Therefore, virtualizing over multiple and diverse sources is not possible. Generic virtualization over multiple and diverse data sources is therefore needed.

1.1.2 Data Transformations

As previously mentioned, R2RML is the standard mapping language for data integration. However, it has certain limitations. The most significant one is that it only supports access to relational databases, while data is often heterogeneous and stored in NoSQL databases or files. This limitation has been addressed by several extensions, with RML [81] being the most notable and widely used today.

R2RML defines schema transformations between a relational database and a target ontology, with data transformations and joins specified using SQL. However, due to the generic nature of RML for integrating diverse data, delegating data transformations to the underlying sources is not always feasible. When possible, it often involves multiple query languages tailored to different data models and formats. To address this, RML-FNML [46] was incorporated in the RML ecosystem to define data transformations in a source-agnostic manner. However, its execution by virtualization systems based on query translation remains unstudied. In addition, while data transformations should be defined declaratively when possible, sometimes more flexibility and expressiveness is needed. This can be achieved with user-defined functions in a high-level programming language, but implementations are missing.

While RML-FNML offers generic data transformations, it does not address RML's limitations concerning joins. In RML, joins for relational data are defined using SQL, but defining multiple or theta joins for other models and formats is not possible. However, SQL could also be utilized with tabular sources such as CSV to overcome this limitation.

1.1.3 RDF-star Generation

Sometimes, it is convenient to annotate statements in a KG, for instance, with the date in which they were created. This is known as reification, and it is intrincate in the RDF data model. To address this, RDF-star was proposed, treating statement-level annotations as first-class citizens [67]. While the SPARQL query language has been extended to support RDF-star, the RML mapping language has not. To exploit RDF-star graphs, they must first be created; therefore, extending RML to support RDF-star is necessary.

1.2 Research Problems

Given the insufficient performance of current materialization approaches that we previously identified [16], more efficient techniques are necessary. Therefore, the first research problem for this thesis is to optimize KG materialization exploiting information in declarative mappings. Specifically, we aim to partition declarative mappings into groups of rules that have non-overlapping outputs, i.e., generating disjoint RDF subgraphs. This allows us to execute groups of mapping rules in parallel, speeding up materialization. Alternatively, groups of rules can be processed sequentially to reduce memory usage, albeit with longer materialization times. We will investigate methods for partitioning mapping rules, their practical feasibility, and their impact on optimizing materialization.

Building on this optimized partition-based materialization, we aim to address the second research problem of the thesis, as identified by Hoseini et al. [78]: the generalization of virtualization to multiple, diverse data sources. The main research question is whether it is possible to achieve general virtual knowledge graphs with sufficient performance. Our hypothesis is that this can be achieved by shifting from a query shipping to a data shipping approach. This is because translating data in various models and formats is generally simpler than translating queries, which may not always be feasible. We propose a solution based on fast materialization with mapping partitioning to coalesce the relevant subset of data for an SPARQL query into a schema-oblivious, intermediate graph representation, which can then be queried uniformly. We will investigate the feasibility of this approach and whether it exhibits comparable performance to query translation approaches.

Additionally, we aim to tackle other technical problems related to the declarative generation of RDF-star graphs and data transformations in mappings. The primary challenge with RDF-star lies in its recursive nature. We plan to address this by enabling the nesting of RML mapping rules. This solution enables the creation of RDF-star from property graphs, motivated by their inherent interoperability [66]. However, this raises the problem of labor-intensive and error-prone creation of mappings for existing property graphs. For relational databases this has been mitigated by automatically extracting default mappings. Similarly, we aim to alleviate this issue for property graphs by bootstrapping default mappings.

Regarding data transformation in declarative mappings, we aim to address various technical problems. The first is the execution of RML-FNML by virtualization systems. We will see that this is inherently supported in data translation-based virtualization, as proposed to address the second research problem. For query translation-based virtualization, we propose extending the idea of pushing down computations in mappings to RML-FNML as well. The goal is to unfold RML-FNML into RML views that can be executed by virtualization systems. The second problem we address is the limited support for joins and composite data values in RML for tabular files. We observe that these limitations are resolved for relational databases through the expressiveness of SQL. Therefore, we devise the extension of RML views to tabular files to overcome these limitations in a similar manner. Finally, we aim to address scenarios where built-in functions are insufficient and more expressivity is required through user-defined functions. We tackle this issue by incorporating Python user-defined functions into RML, leveraging Python's usability and data processing libraries.

1.3 Contributions

This work addresses the first research problem in Section 1.2 with the following specific contributions:

- The formalization of an RML mapping partition.
- Algorithms to compute a mapping partition from a set of RML documents.
- Methods for optimizing the materialization of RDF graphs with a mapping partition.
- Experimental results demonstrating the positive impact of mapping partitioning on the efficiency of RDF materialization.

To tackle the second research problem in Section 1.2 we contribute the following:

- The intermediate triple table architecture based on data translation for virtualizing over multiple and diverse sources.
- The formalization of star-shaped mapping candidate selection.
- Experimental results evidencing the generality of intermediate triple table for virtualizing diverse sources, and its comparable efficiency with respect to the query translation-based approach.

The technical problems related to the declarative generation of RDF-star graphs are addressed with the following concrete contributions:

- The RML-star mapping language to declare the generation of RDF-star graphs from diverse data.
- A baseline materialization algorithm for RML-star.
- A method to automatically extract RML-star mappings from a property graph.

For the technical problems regarding data transformation in mappings, we contribute:

- An extension of RML views to tabular files.
- The implementation of an RML-FNML module for Python user-defined functions.
- The implementation of a research prototype for unfolding RML-FNML into RML views.

Additionally, we contribute the following technological and transversal advancements:

- Morph-KGC: An effient and feature-rich materialization system that implements various proposal of the thesis, specifically: mapping partitioning, RML-star, RML tabular views, and Python user-defined functions.
- LUBM4OBDA: An extension of the LUBM [61] benchmark for evaluating virtual knowledge graph systems, which also encompasses reification approaches including RDF-star.
- The implementation of the intermediate triple table architecture and the bootstrapping of RML-star mappings from a property graph.

1.4 Thesis Outline

This thesis starts presenting in **Chapter 2** the relevant concepts and languages. It first introduces data integration and then describes relevant W3C standards, specifically the RDF data model, the SPARQL query language, and the R2RML and RML mapping languages.

In **Chapter 3** we focus on optimization of materialization. We present our mapping partitioning approach, provide algorithms for obtaining a partition from a set of mapping rules, and study the execution strategies of a mapping partition to reduce materialization times and memory usage. We also introduce here redundant self-join elimination in mappings. We conduct extensive experiments that show the effectiveness of the optimizations compared to the state of the art.

Chapter 4 is devoted to virtualization for which we introduce the intermediate triple table architecture. It presents the major advantage of being independent of the underlying sources, i.e., it works over any data model and format. The architecture is based on data translation with reduced computations. We conduct experiments with multiple and heterogeneous data sources that evidence its source independence and its efficiency in terms of query execution time. The intermediate triple table even evaluates complex graph queries over a single relational database faster than previous approaches, but with the tradeoff of a more intensive use of memory.

In **Chapter 5** we address the generation of RDF-star graphs from diverse data. First, we extend the RML mapping language with a new module, namely RML-star, to declare the generation statement-level annotations. Next, we introduce a procedure to process RML-star mappings and materialize RDF-star graphs. In **Chapter 6** we study the automatic extraction of RML-star mappings. This is relevant since mapping development is a time-consuming task that automation can potentially accelerate. Specifically, a basic draft with the mappings can be automatically extracted, which users can then edit according to a domain vocabulary.

In **Chapter 7** we present some proposals involving the definition and execution of data transformations in declarative mappings. First, we address data transformation in tabular data sources by declaring computations directly in SQL. This simple solution solves current limitations of RML for complex joins and composite data values on tabular data. We also bring Python user-defined functions to RML. This contribution empowers the creation of KGs with the flexibility of user-defined functions required in many data integration use cases with a widely adopted programming language for data processing. Finally, we present an approach for the execution of RML-FNML in virtualization systems. The approach is based on the translation of mappings without the need to modify current virtualization systems.

Chapter 8 summarizes the main conclusions of the thesis and outlines future research lines.

Publications

The materialization optimizations presented in Chapter 3 are based on the publication:

- Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M. S., & Corcho, O. “Morph-KGC:

Scalable knowledge graph materialization with mapping partitions". In: *Semantic Web* 15.1 (2024), pp. 1–20. IOS Press. doi: 10.3233/SW-223135.

The virtual KG architecture presented in Chapter 4 is based on the following publication:

- Arenas-Guerrero, J., Corcho, O., & Pérez, M. S. "Intermediate triple table: A general architecture for virtual knowledge graphs". In: *Knowledge-Based Systems* 314 (2025). Elsevier. doi: 10.1016/j.knosys.2025.113179.

RML-star, its integration in the RML ecosystem and the materialization procedure and implementation described in Chapter 5 have been disseminated in the following publications:

- Arenas-Guerrero, J., Iglesias-Molina, A., Chaves-Fraga, D., Garijo, D., Corcho, O., & Dimou, A. "Declarative Generation of RDF-star graphs from heterogeneous data". In: *Semantic Web* 16.2 (2025). IOS Press. doi: 10.3233/SW-243602.
- Delva, T., Arenas-Guerrero, J., Iglesias-Molina, A., Corcho, O., Chaves-Fraga, D., & Dimou, A. "RML-star: A Declarative Mapping Language for RDF-star Generation". In: *International Semantic Web Conference, P&D*. Vol. 2980. CEUR Workshop Proceedings, 2021. url: http://ceur-ws.org/Vol-2980/paper374.pdf.
- Iglesias-Molina, A., Van Assche, D., Arenas-Guerrero, J., De Meester, B., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D., & Dimou, A. "The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF". In: *Proc. of the 22nd International Semantic Web Conference*. Springer Nature Switzerland, 2023, pp. 152–175. doi: 10.1007/978-3-031-47243-5_9.

The automatic extraction of RML-star mappings presented in Chapter 6 is based on the following publication:

- Arenas-Guerrero, J., & Espinoza-Arias, P. "Automatic Extraction of RML-star Mappings from Property Graphs". In: *Proc. of the 26th International Conference on Information Integration and Web Intelligence*. Springer Nature Switzerland, 2024, pp. 298-303. doi: 10.1007/978-3-031-78090-5_25.

The techniques for processing data transformations in declarative mappings presented in Chapter 7 were published in:

- Arenas-Guerrero, J., Alobaid, A., Navas-Loro, M., Pérez, M. S., & Corcho, O. "Boosting Knowledge Graph Generation from Tabular Data with RML Views". In: *Proc. of the 20th Extended Semantic Web Conference*. Springer Nature Switzerland, 2023, pp. 484–501. doi: 10.1007/978-3-031-33455-9_29.
- Arenas-Guerrero, J., Espinoza-Arias, P., Bernabé-Diaz, J. A., Deshmukh, P., Sánchez-Fernández, J. L., & Corcho, O. "An RML-FNML module for Python user-defined functions in Morph-KGC". In: *SoftwareX* 26 (2024), p. 101709. Elsevier. doi: 10.1016/j.softx.2024.101709.
- Arenas-Guerrero, J. "Handling Data Transformations in Virtual Knowledge Graphs with RML View Unfolding". In: *Proc. of the 24th International Conference on Web*

Engineering. Springer Nature Switzerland, 2024, pp. 424–427. doi: 10.1007/978-3-031-62362-2_38.

The LUBM4OBDA benchmark employed for experimentation in Chapters 4, 5 and 7 was published in:

- Arenas-Guerrero, J., Pérez, M. S., & Corcho, O. "LUBM4OBDA: Benchmarking OBDA Systems with Inference and Meta Knowledge". In: *Journal of Web Engineering* 22.8 (2024), pp. 1163–1186. River Publishers. doi: 10.13052/jwe1540-9589.2284.

Open Science Materials

This thesis produced artifacts that are openly available and that can be reused to support further research:

- The source code of Morph-KGC is openly available in GitHub[1] and archived at Zenodo [12] under the Apache 2.0 License. These repositories include the modules for RML-star, RML-FNML, RML tabular views and Python user-defined function. Morph-KGC is registered on the Intellectual Property Registry of the Community of Madrid under the reference *M-003951/2021*.
- The source code for the intermediate triple table architecture devised in Chapter 4 is openly available in GitHub[2] and archived at Zenodo [11] under the Apache 2.0 License.
- The implementation of the RML-star mapping bootstrapper presented in Chapter 6 is openly available in GitHub[3] under the Apache 2.0 License.
- The implementation of the translator from RML-FNML to RML views outlined in Chapter 7 is openly available in GitHub[4] under the Apache 2.0 License.
- The resources of the LUBM4OBDA benchmark are available in GitHub[5] and archived at Zenodo [10] under the MIT License.

[1] https://github.com/morph-kgc/morph-kgc/
[2] https://github.com/arenas-guerrero-julian/ITT
[3] https://github.com/arenas-guerrero-julian/pg2rml-star
[4] https://github.com/arenas-guerrero-julian/fnml-translator
[5] https://github.com/oeg-upm/lubm4obda

Chapter 2

Preliminaries

In this chapter, we first outline data integration and its components. Then, we briefly describe the standard languages of these components. Finally, we present some assumptions, notation, and conventions that will be used in the thesis.

2.1 Data Integration

Data is usually distributed across sparse, isolated sources, resulting in separated silos. This hinders the combined exploitation and analysis of diverse data, which is critical in use cases that require the collection of value from the data in whole [102]. Data integration is the problem of collecting and combining a set of sources and providing users with a common view of the data [96].

A data integration system is a triple $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{O}$ is the ontology or *global schema*, $\mathcal{S}$ is the set of data sources or *local schemas*, and $\mathcal{M}$ are the mappings that describe the relationships between $\mathcal{O}$ and $\mathcal{S}$ [96]. The local schemas describe the underlying sources containing the data, the global schema provides the abstract, integrated, and common view of the local sources, and the mapping contains the assertions that connect the elements in the local and global schemas.

Mappings are expressed as declarative rules that define the correspondences between $\mathcal{O}$ and $\mathcal{S}$. There are two fundamental approaches for mappings: *global-as-view* and *local-as-view*. The work presented in this thesis focuses on global-as-view mappings, which associate for each element in $\mathcal{O}$ a query over $\mathcal{S}$. Global-as-view mappings consist of a set of assertions of the form $g \rightsquigarrow q_{\mathcal{S}}$, where $q_{\mathcal{S}}$ is a query over the local schemas and g is a function defining how to construct the elements in $\mathcal{O}$ from the result set of $q_{\mathcal{S}}$. To illustrate this, imagine that $\mathcal{S}$ is given by a relational database (RDB); then each rule in $\mathcal{M}$ defines an SQL query over the RDB, which populates the entities and properties in $\mathcal{O}$.

The W3C provides standard languages for each of the components in the data integration system, namely RDF is the graph data model for the integrated data, OWL and RDFS are used to encode ontologies that define the target view, SPARQL is used for querying, and R2RML for the mappings. In the following sections, we briefly describe the most relevant

components for our work.

2.2 Data Representation and Query Languages (RDF and SPARQL)

The Resource Description Framework [43] (RDF) is the standard graph data model used in the Semantic Web [25] and stands out for its flexibility. RDF consists of terms, triples, and graphs, which are introduced below.

The set of *RDF terms* is given by the union of the mutually disjoint sets of $\mathcal{I}$, $\mathcal{B}$ and $\mathcal{L}$. $\mathcal{I}$ are IRIs (e.g., `https://www.wikidata.org/wiki/Q8717` to identify the city of Seville), $\mathcal{B}$ are "existential variables" (e.g., a blank node may identify the country in which Seville is located in without naming it), $\mathcal{L}$ are primitive values that can be typed according to XML Schema (e.g., `"5"^^xsd:integer`) or that can be language-tagged in the case of strings (e.g., `"Sevilla"@es`).

An *RDF triple* is a tuple $(s, p, o) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, where s is the subject, p is the predicate, and o is the object ($s \xrightarrow{p} o$). The triple :Q8717 $\xrightarrow{\text{:Property:P17}}$:Q29 denotes that Seville is located in Spain (using Wikidata IRIs). An *RDF graph* is a set of RDF triples. Finally, an *RDF dataset* is a set of RDF graphs. An RDF dataset can be visualized as a set of *RDF quads* $(s, p, o, g) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}) \times \mathcal{I}$, where g is the named graph in which the triple is contained. N-Triples [23] and Turtle [24] are the most notable serializations of RDF graphs. The former simply encodes each triple in a line, while the latter syntax is more compact.

SPARQL [65] is the standard language for querying RDF. The core concept of SPARQL is the *triple pattern*. An RDF triple pattern is a tuple $(S, P, O) \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{V}) \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V})$, where $\mathcal{V}$ is the set of variables. SPARQL queries typically contain sets of triple patterns known as *basic graph patterns*. Triple and basic graph patterns are used to *match* subgraphs of RDF graphs. A triple (s, p, o) in an RDF graph matches a triple pattern (S, P, O) if:

- $S \in \mathcal{I} \cup \mathcal{B}$ then $S = s$
- $P \in \mathcal{I}$ then $P = p$
- $O \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ then $O = o$

2.3 Declarative Mapping Languages (R2RML and RML)

The RDB to RDF Mapping Language [44] (R2RML) is the W3C Recommendation mapping language that links relational databases to the RDF data model. The RDF Mapping Language [51] (RML) is a well-known extension of R2RML that supports input data formats beyond RDBs (e.g., CSV, JSON, or XML). It must be noted that during the course of this thesis, RML was redesigned and modularized by the W3C Knowledge Graph Construction Community Group[1] to incorporate new features [81].[2] The RML-Core module in the new

[1] `https://www.w3.org/community/kg-construct/`

[2] Indeed, Chapter 5 presents an extension of RML that is a module within the new RML redesign.

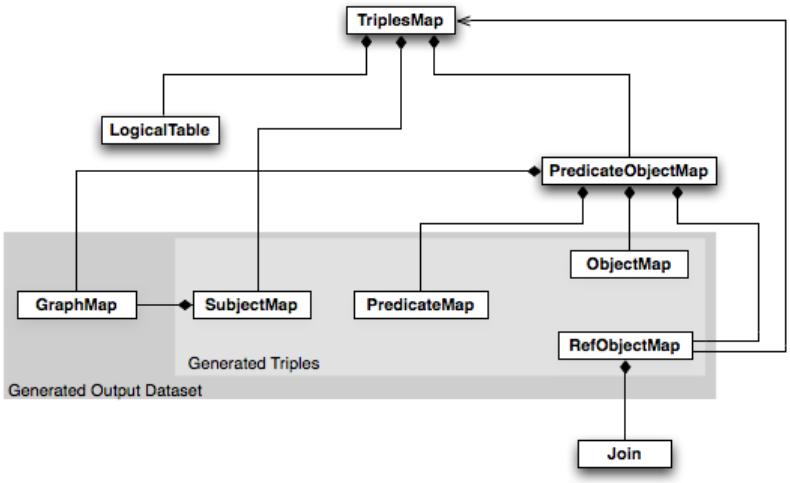


Figure 2.1: An overview of a triples map in R2RML, based on [44]. In RML, a triples map presents the same structure, but it replaces `LogicalTable` by the more generic `LogicalSource`.

design roughly comprises the features of R2RML and the original RML proposal. In this thesis, when we use RML we typically refer to the RML-Core module, but it may be used in a more general way to refer to all modules. As RML is a superset of R2RML, in this section, we present the main notions of these mapping languages focusing solely on RML.

An RML mapping is represented as an RDF graph, and it is generally written in Turtle syntax. An *RML mapping document* encodes an RML mapping, and it consists of one or more *triples maps*. Figure 2.1 depicts the structure of a triples map in R2RML; in RML the structure is similar but a table is generalized to any source. A triples map has one *logical source* and contains the rules for generating the RDF triples. A triples map consists of one *subject map* and zero or more *predicate-object maps*. Each predicate-object map has, in turn, one or more *predicate* and *object maps*. Subject, predicate, and object maps are functions called *term maps* specifying how to generate the RDF terms in the homonymous positions of the triples. Figure 2.2 shows the structure of a term map in R2RML; in RML it presents a similar structure. Term maps can be constant-valued (always generate the same value), reference-valued (the values are obtained directly from the logical source, e.g., a column in a table of an RDB), or template-valued (which generate RDF terms with some parts given by constants and others given by references). A template-valued term map comprises a *string template*, that defines how RDF terms are generated from one or more references (enclosed in curly braces) over the logical source. *Constant shortcut properties* are a compact method of expressing constant-valued term maps. A *referencing object map* allows to generate triples in which the object map is given by the subject map of another triples map, known as the *parent triples map*. A *join condition* is used when the logical sources of both triples maps are different. The evaluation of a triples map[3] produces an RDF graph whose triples consist of the generated RDF terms that result from applying its subject, predicate, and object maps to the input logical source.

[3] https://www.w3.org/TR/r2rml/#generated-triples

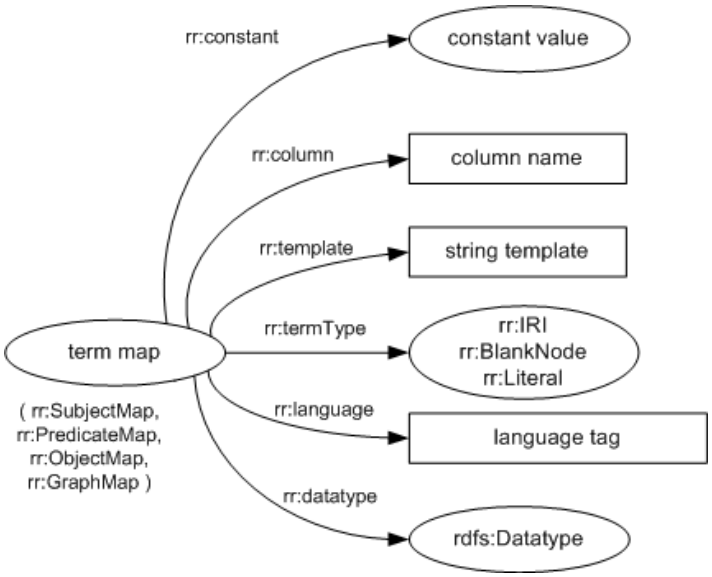


Figure 2.2: Structure of a term map in R2RML, based on [44]. RML presents a similar structure, but it replaces `rr:column` by the source-independent `rml:reference`. Term maps in RML also have properties to create dynamic language tags and datatypes (i.e., they are generated from input data and not as constant values).

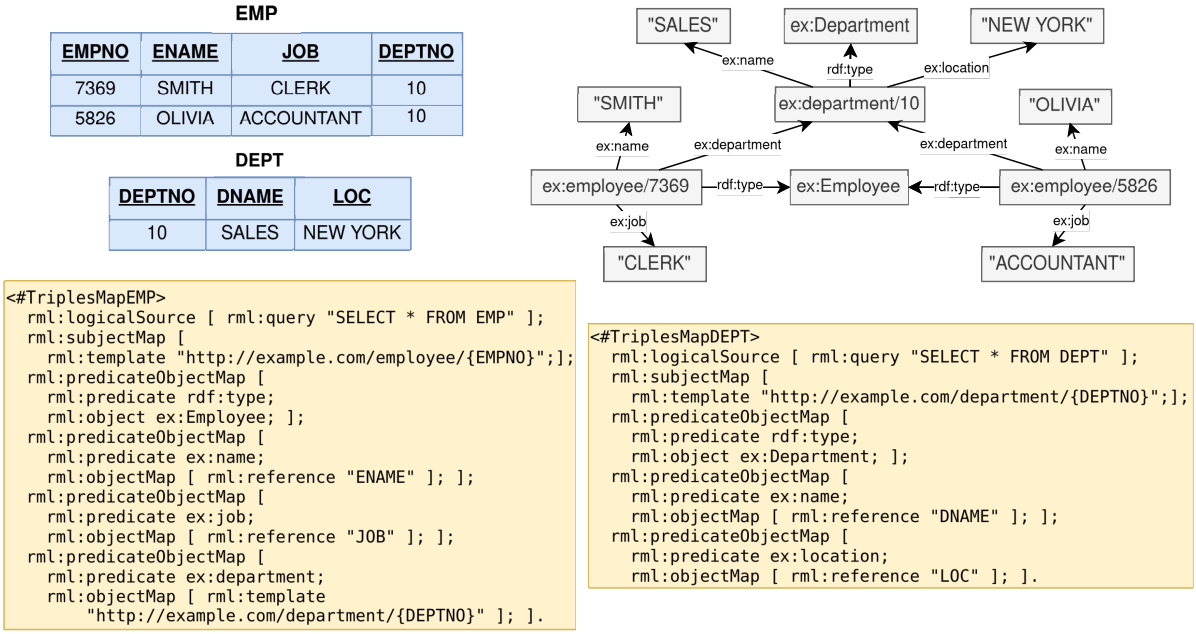


Figure 2.3: RML example based on [44] accessing an RDB with two tables (top left), the RML mapping (bottom), and the visual representation of the KG (top right).

Figure 2.3 illustrates RML. In the example, a relational database with two tables is mapped to RDF, for which the visual representation of the output graph is shown. The mapping consists of two triples maps, each with multiple predicate-object maps.

2.4 Notation and Conventions

In our work, we rely on the normalization of mappings as defined in [114]. A normalized mapping document does not contain shortcuts (which are expanded), uses predicate-object maps to type resources (instead of using `rml:class`), and all its triples maps contain a single predicate-object map with one predicate map and one object map. This is not restrictive, as any R2RML or RML document can be normalized. We refer to a normalized triples map as a *mapping rule*. In this document, mappings omit RDF prefix declarations (e.g., `rml:` and `xsd:`) for the sake of conciseness.

We target KG materialization where the resulting RDF graph does not contain duplicated triples, as assumed by most engines [31, 48, 79]. Given that an RDF graph is a set of triples [43], the presence of duplicated triples in the serialization of the RDF graph does not affect the final result. We add this restriction because it has an impact on memory and time consumption, as well as on the size of the resulting files.

Because RML encompasses R2RML, we use the former in this thesis. The presented solutions may be easily extended to other R2RML-based mapping languages. We refer to RDF triples and quads indistinctly. *TM*, *SM*, *POM* and *OM* denote triples, subject, predicate object and object maps, respectively.

Let $\mathcal{T}$ be a term map, $\mathfrak{T}$ the set of all possible term maps, and $\mathbb{P}$ the set of positions that the RDF terms generated by the term map $\mathcal{T}$ can occupy in a quad, i.e., $\{subject, predicate, object, graph\}$. We define *position* as a function mapping $\mathfrak{T}$ to $\mathbb{P}$. Let $\mathbb{T}$ be the set of possible types of term maps, i.e., $\{IRI, Literal, BlankNode\}$, then *type* is a function mapping $\mathfrak{T}$ to $\mathbb{T}$. Let $\mathbb{V}$ be the set of possible values that a term map can have, i.e., $\{constant, reference, template\}$, then *value* is a function that maps $\mathfrak{T}$ to $\mathbb{V}$. Let $\mathbb{C}$ be the set of possible constant values that a constant-valued term map can take, then we define *const* as a function mapping $\mathfrak{T}'$ to $\mathbb{C}$ where $\mathfrak{T}' = \{\mathcal{T} \in \mathfrak{T} \mid value(\mathcal{T}) = constant\}$. Let $\mathbb{I}$ be the set of all possible values of the specified language tags or datatypes (as defined in [44]), then we define *literaltype* as a function mapping $\mathfrak{T}''$ to $\mathbb{I}$ where $\mathfrak{T}'' = \{\mathcal{T} \in \mathfrak{T} \mid type(\mathcal{T}) = Literal\}$.

Chapter 3

Efficient Knowledge Graph Materialization with Mapping Partitioning

In this chapter, we address the problem of scalability in KG materialization from diverse data sources using declarative mapping rules. Previously to the start of the work presented here, we analyzed the performance of existing materialization systems in terms of execution time and memory usage [16]. Our work starts from the observation that most of the systems in the state of the art were not capable of processing large volumes of data, producing timeouts and out-of-memory errors.

The main proposal that we introduce in this chapter is the concept of *mapping partition*. A mapping partition groups rules in the input mapping documents that generate disjoint sets of RDF triples. The partition can then be used to optimize the materialization process while ensuring that the output KG contains no duplicate triples. This guarantee about duplicates presents the advantages of preventing the creation of unnecessary large files and processing less amounts of data, which minimizes memory usage and execution times.

A mapping partition can be obtained by exploiting information within the mapping rules. We describe here how this is achieved and devise algorithms that create a mapping partition from a set of mapping rules. We also show the effectiveness of the proposed algorithms by applying them in benchmarks and real-world use cases, demonstrating that, in practice, it is possible to obtain partitions with a large number of mapping groups. We then study two strategies to optimize the materialization process with a partition: (i) parallel execution, which minimizes materialization times with the concurrent processing of mapping rules, and (ii) sequential execution, which minimizes memory usage by processing subsets of data independently.

3.1 Related Work

For the specific case of RDF materialization from RDBs, Ontop[1] [31, 137] leverages the fact that predicate maps are generally constant-valued. It generates one SPARQL query for each predicate with unbounded subject and object. These queries are then translated to SQL and optimized by applying a set of structural optimizations (e.g., subqueries elimination) and semantic query optimizations (e.g., redundant self-joins removal). It also avoids retrieving large query result sets at once with chunking.

The work presented in [85] exploits knowledge encoded in the mapping documents to project the attributes appearing in a triples map, reducing the size of the data sources that need to be processed. Similarly, to diminish the impact of duplicates in the evaluation of join conditions, it also pushes down projections into joins. SDM-RDFizer [79] proposes physical data structures to store the KG in memory in a way that allows to efficiently remove duplicates and avoid unnecessary operations. In particular, it uses one hash table for each predicate, called *Predicate Tuple Tables*, where the hash key combines the subject and the object of the triple, and the value is the triple itself. It also proposes to speed up joins by creating the *Predicate Join Tuple Table*, a hash table using the values matching the join condition as the hash key, and being the values of the hash the set of the generated values by the parent triples map. These hash tables are checked every time a new triple is to be generated; in the case that the triple already exists, it is discarded; otherwise it is added to the KG, and the corresponding hash table is updated. Moreover, SDM-RDFizer considers data compression techniques that reduce the memory usage of the data structures that store intermediate results during materialization.

Recently, triples map planning has been proposed in [80]. Here, a bushy tree plan is created specifying an optimized execution order for executing mapping assertions. These bushy tree plans are obtained heuristically by relying on a greedy algorithm. Operating system commands are used with these execution plans that allow to efficiently execute different KG materialization engines.

Karma [88] tackles the scalability of KG construction from large data sources using batch processing. Instead of loading all data into memory, when operating in batch mode, the engine continuously loads fractions of the data, transforming them into the nested relational model [97] and then materializing the corresponding triples. In this manner, memory usage is reduced, as it is not required to maintain the entire KG in memory.

Parallelization has also been proposed to speed up the materialization process. The work presented in [63] divides this process into three tasks following the *producer-consumer* paradigm: ingestion of data from the sources, mapping to RDF, and combination of the RDF. Parallelization is done up to the data record level. Nevertheless, the proposed approach does not tackle duplicate elimination. Other works also parallelize at the triples map level [117, 79].

[1] Ontop is primarily a virtualization system, but it also has a materialization mode.

3.2 Self-Join Elimination in Mappings

Most virtualization systems in the state of the art (e.g., [111, 31]) remove redundant self-joins in the SQL queries that they generate, with the objective of making query evaluation more efficient. However, most materialization engines do not address self-joins that can occur in a mapping, which in most cases are locally executed.

Definition 1 (*Redundant self-join* in an RML document)**.** A redundant self-join in an RML document appears when a referencing object map joins two triples maps with the same logical source and it has join conditions with the same unique references.

A redundant self-join in an RML document can be removed by replacing the referencing object map with an object map given by the subject map of the parent triples map, producing the same set of RDF triples.

Example 1. Consider the RML mapping rules with a self-join taken from GTFS-Madrid-Bench [37]:

```
<#shapesTM>
    rml:logicalSource [ rml:tableName "SHAPES" ];
    rml:subjectMap [
        rml:template "http://transport.linkeddata.es/madrid/metro/shape/{shape_id}"
    ];
    rml:predicateObjectMap [
        rml:predicate gtfs:shapePoint;
        # referencing object map (self-join)
        rml:objectMap [
            rml:parentTriplesMap <#shapePoints> ;
            rml:joinCondition [
                rml:child "shape_id";
                rml:parent "shape_id";
            ];
        ];
        # object map (no join)
        rml:objectMap [
            rml:template "http://transport.linkeddata.es/madrid/metro/shape_point/
                          {shape_id}-{shape_pt_sequence}"
        ]
    ].

<#shapePointsTM>
    rml:logicalSource [ rml:tableName "SHAPES" ];
    rml:subjectMap [
        rml:template "http://transport.linkeddata.es/madrid/metro/shape_point/
                      {shape_id}-{shape_pt_sequence}"
    ].
```

Both triples maps use the same database table, and the referencing object map uses the same unique column to join both triples maps. This can be transformed into an object map without a join condition (the second object map in the triples map `#shapesTM`).

As defined in the R2RML Recommendation, the *effective SQL query*[2] of the referencing object map in the triples map `#shapesTM` of *Example 1* is:

```
1    SELECT * FROM
2        ( SELECT * FROM SHAPES ) AS child,
3        ( SELECT * FROM SHAPES ) AS parent
4    WHERE child.shape_id=parent.shape_id
```

Removing this kind of SQL join is widely studied in the literature, known as semantic query optimization [34, 119]. Under the assumption that the join references in a mapping are unique, self-joins can be eliminated in mapping documents for any data format.

The impact of redundant self-joins was observed in our previous analysis of materialization systems [16]. We propose to remove redundant self-joins within the mapping documents to improve the performance of KG construction engines without the need to modify their current materialization procedures. In this way, an RML document without redundant self-joins does not contain referencing object maps involving two triples maps with the same logical source and with the same unique references in the join conditions. This redundant self-join elimination approach is independent of the underlying data format, as opposed to the previous techniques (e.g., [31, 111] address RDBs only).

Definition 2 (*Canonical* RML document w.r.t. joins)**.** A canonical RML document is a normalized RML document without redundant self-joins.

An RML document and its canonicalization are equivalent; this implies that any transformation of a mapping document comprised in *Definition 2* (i.e., normalization and redundant self-join elimination) also generates an equivalent mapping document. Algorithm 1 obtains the canonicalization of any mapping document. First, it normalizes the document (see [114]). Next, it discards referencing object maps with different logical sources in the triples map and the parent triples map (*lines 4-6*). After that, the algorithm checks that the fields in all join conditions match (*lines 7-11*). When that happens, the self-join can be removed and the object map is replaced by the subject map of the parent triples map (*lines 12-13*).

3.3 Mapping Partitions

Given a set of mapping rules, we aim at identifying those that produce disjoint sets of triples, i.e., the mapping rules will be grouped so that those in different groups generate sets of RDF triples that do not overlap. In the following, when we refer to the sets of generated triples, we consider them to be composed of all the triples that a mapping rule, group of mappings rules, or mapping document generate given a data source.

Definition 3 (*Mapping Partition* of an RML document)**.** Let $\mathcal{M}$ be a normalized RML document with a set of mapping rules m_1, m_2, ..., m_n that generates the triple set T. Then, a mapping partition $\mathcal{P}$ of $\mathcal{M}$ is a set of subsets of $\mathcal{M}$, designated as mapping groups $\mathcal{G}_1$, $\mathcal{G}_2$, ..., $\mathcal{G}_k$, that generate the triple sets t_1, t_2, ..., t_k, satisfying the following conditions:

[2] https://www.w3.org/TR/r2rml/#dfn-effective-sql-query

Algorithm 1: *Canonicalization* of an RML document, $\mathcal{M}$.

Result: Canonical $\mathcal{M}$

```
M = normalize(M) // see [114]
for TM ∈ M do
    for OM ∈ TM do
        if isRefOM(OM) then
            parentTM = OM.parentTM
            if TM.source == parentTM.source then
                removeJoin = True
                for joinCond ∈ OM do
                    if joinCond.child ≠ joinCond.parent then
                        removeJoin = false
                end
                if removeJoin then
                    OM = parentTM.SM
    end
end
```

- $\bigcap_{i=1}^{k} t_i = \emptyset$, i.e., the triple sets generated by each mapping group are disjoint.
- $\bigcup_{i=1}^{k} t_i = T$, i.e., the union of the triple sets generated by all the mapping groups is equivalent to T.

Multiple mapping partitions can exist for $\mathcal{M}$. The most trivial mapping partition is the one with only one mapping group (i.e., the singleton set), and we denote it with $\mathcal{P}_\emptyset$. Mind that this definition of mapping partition does not entail that a mapping group can be considered as a new mapping document. A mapping rule in a mapping group can still have a join condition involving a rule from a different group of mappings.

Example 2. Consider the mapping rules (based on [44]):

```
<#TM1>
    rml:logicalSource [ rml:tableName "DEPT" ];
    rml:subjectMap [
        rml:template "http://data.example.com/department/{DEPTNO}";
    ];
    rml:predicateObjectMap [
        rml:predicate <http://data.example.com/name>;
        rml:objectMap [ rml:reference "DNAME" ];
    ].

<#TM2>
    rml:logicalSource [ rml:tableName "EMP" ];
    rml:subjectMap [
        rml:template "http://data.example.com/employee/{EMPNO}";
    ];
    rml:predicateObjectMap [
        rml:predicate <http://data.example.com/department>;
        rml:objectMap [
```

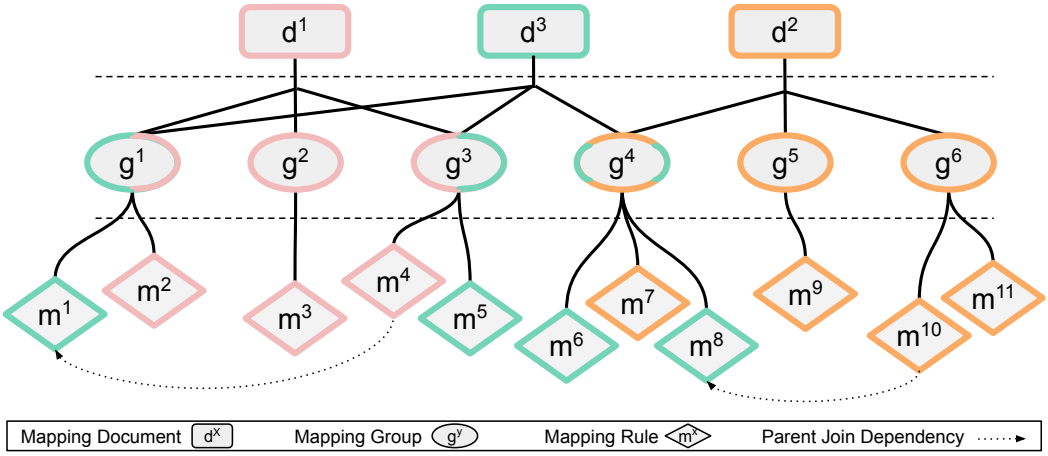


Figure 3.1: Mapping partition of three mapping documents with eleven normalized mapping rules in total. The mapping partition is composed of six mapping groups that have between one and three mapping rules. In addition, there are two join dependencies among different groups of mappings.

```
19              rml:parentTriplesMap <#TM1> ;
20              rml:joinCondition [
21                  rml:child "DEPTNO";
22                  rml:parent "DEPTNO";
23              ];
24          ];
25      ].
```

Both mapping rules can be assigned to different mapping groups as they do not generate common triples, given that the predicates are constants and that they are different. Nonetheless, the mapping rule in `#TM2` is dependent on `#TM1`, since the object map of the former is given by the subject map of the latter, and this results in a join dependency between those mapping groups. This prevents both mapping groups from becoming independent mapping documents.

Figure 3.1 depicts an example of mapping partitioning that involves three mapping documents with eleven normalized mapping rules in total. Six mapping groups are formed, which have between one and three mapping rules. As it can be seen, there are join dependencies between different groups of mappings; nonetheless, they are still disjoint in terms of the set of triples that they generate.

We now delve into the rationale for obtaining partitions of a mapping document $\mathcal{M}$. This is done incrementally by first examining the disjointness of term maps, next of mapping rules, and finally of mapping groups.

Definition 4 (*Prefix* of a template)**.** We define the prefix of a template as the constant (or immutable) part of its string template preceding the first reference in it. If a template starts with a reference, then its prefix is empty ($\emptyset$). Note that this is different to the notion of prefix declaration in RDF documents.

Example 3. Consider the following string templates along with their prefixes:

```
Template:     http://data.example.com/employee={EMPNO}/department={DEPTNO}
Prefix:       http://data.example.com/employee=

Template:     http://data.example.com/roles/{ROLE}
Prefix:       http://data.example.com/roles/

Template:     {EMPNO}-{DEPTNO}
Prefix:       ∅
```

The prefixes are obtained by eliminating the first reference and what follows. The beginning of the last string template is data source-dependent and therefore its prefix is $\emptyset$.

Definition 5 (*Invariant* of an RML term map)**.** The invariant $\mathcal{I}$ of a term map $\mathcal{T}$ is the longest common initial part of all the RDF terms that can be generated by $\mathcal{T}$. $\mathcal{I}$ is an intrinsic property of $\mathcal{T}$ and remains immutable regardless of data coming from the input sources. $\mathcal{I}$ depends on $\mathbb{V}$, and it is obtained as follows:

- If $value(\mathcal{T}) = constant$, then $\mathcal{I} = const(\mathcal{T})$.
- If $value(\mathcal{T}) = template$, then $\mathcal{I} = prefix(\mathcal{T})$.
- If $value(\mathcal{T}) = reference$, then $\mathcal{I} = \emptyset$.

Notation 1 (*Invariants*)**.** For simplicity and clarity, we define the following notation for invariants:

- $\mathcal{I}_\emptyset$ denotes the empty invariant.
- $\mathcal{I}_1 < \mathcal{I}_2$, denotes that the length of $\mathcal{I}_1$ is shorter than $\mathcal{I}_2$, with the length given by the number of characters of the invariants.
- $\mathcal{I}_1 \subset \mathcal{I}_2$, denotes that $\mathcal{I}_1$ matches the beginning of $\mathcal{I}_2$. This entails $\mathcal{I}_1 < \mathcal{I}_2$.

Example 4. Consider the three templates in *Example 3* and their invariants $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ respectively (given by their prefixes). Then, the following applies:

- $\mathcal{I}_3 = \mathcal{I}_\emptyset$.
- $\mathcal{I}_3 < \mathcal{I}_2 < \mathcal{I}_1$.
- $\mathcal{I}_3 \subset \mathcal{I}_1$, $\mathcal{I}_3 \subset \mathcal{I}_2$, $\mathcal{I}_2 \not\subset \mathcal{I}_1$ and $\mathcal{I}_1 \not\subset \mathcal{I}_2$.

Definition 6 (*Disjoint Term Maps*)**.** Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two term maps and $\mathcal{I}_1$, $\mathcal{I}_2$ their respective invariants. $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint iff the sets of RDF terms that they can generate are in turn disjoint, regardless of the input data. The disjoint property for $\mathcal{T}_1$ and $\mathcal{T}_2$ applies iff at least one of the following conditions hold:

1. $type(\mathcal{T}_1) \neq type(\mathcal{T}_2)$.
2. $\mathcal{I}_1 \neq \mathcal{I}_2$, $\mathcal{I}_1 \not\subset \mathcal{I}_2$ and $\mathcal{I}_2 \not\subset \mathcal{I}_1$.
3. $\mathcal{I}_1 < \mathcal{I}_2$, $type(\mathcal{T}_1) = constant$, or *vice versa*.
4. $type(\mathcal{T}_1) = type(\mathcal{T}_2) = Literal$, and $literaltype(\mathcal{T}_1) \neq literaltype(\mathcal{T}_2)$.

Disjointness of term maps depends on: $\mathbb{T}$, invariants, and $\mathbb{I}$. Term maps with different $\mathbb{T}$ enforce the generation of distinct (disjoint) RDF terms (*first condition*). We now focus on

term maps with similar $\mathbb{T}$. For term maps with distinct invariants, the generated triple sets are disjoint if none of the invariants matches the beginning of the other (*second condition*). The latter is necessary to not make any assumptions on data coming from the sources. Note that in this case $\mathcal{I}_\emptyset$ prevents term map disjointness to apply. When the value of the term map with the shortest invariant is constant, the *second condition* can be relaxed, and it is only necessary that the invariant of this term is shorter than the other invariant (*third condition*). This is because the term map with the shortest invariant is not dependent on data and the RDF terms will always be shorter (and therefore distinct) than those generated by the other term map. For the specific case that two term maps generate literals, they are disjoint if they have distinct $\mathbb{I}$ (*fourth condition*). This is because RDF literals with different datatypes or language tags are different. This is also true for the empty literal type, e.g., a typed literal will always be different from a non-typed literal. Abusing notation, given two term maps $\mathcal{T}_1$ and $\mathcal{T}_2$, we use $\mathcal{T}_1 \bigcap \mathcal{T}_2 = \emptyset$ to denote that they are disjoint.

Definition 7 (*Disjoint Mapping Rules*)**.** Two mapping rules m_1, m_2 that generate triple sets t_1, t_2 respectively, are disjoint iff they do not generate common triples, i.e., $t_1 \bigcap t_2 = \emptyset$, regardless of the input data sources. Disjointness of m_1 and m_2 can be determined as follows:

$$\exists \mathcal{T}_1 \in m_1, \exists \mathcal{T}_2 \in m_2 \mid \mathcal{T}_1 \bigcap \mathcal{T}_2 = \emptyset, position(\mathcal{T}_1) = position(\mathcal{T}_2)$$

For two mapping rules to be disjoint, it is required that at least two position-wise term maps are disjoint. This is true because once two triples have a different subject, predicate, object, or graph, then the triples are immediately distinct. Abusing notation, given two mapping rules m_1 and m_2, we use $m_1 \bigcap m_2 = \emptyset$ to denote that they are disjoint.

Definition 8 (*Disjoint Mapping Groups* of an RML document)**.** Two mapping groups $\mathcal{G}_1$ and $\mathcal{G}_2$ of $\mathcal{M}$ are disjoint iff they generate disjoint sets of triples. This property holds when all the mapping rules in $\mathcal{G}_1$ are disjoint of all the mapping rules in $\mathcal{G}_2$. As a consequence, a mapping rule cannot belong simultaneously to disjoint mapping groups. Formally:

$$\forall m_1 \in \mathcal{G}_1, \forall m_2 \in \mathcal{G}_2 \mid m_1 \bigcap m_2 = \emptyset$$

Definition 9 (*Maximal Mapping Partition* of an RML document)**.** The maximal mapping partition of $\mathcal{M}$ (denoted with $\mathcal{P}_{max}$) is the one with the largest number of mapping groups.

Given a mapping document, its maximal mapping partition is not necessarily unique, i.e., there may be several maximal mapping partitions for the original mapping document. When all mapping rules in a mapping document are disjoint, each mapping group in $\mathcal{P}_{max}$ is a singleton set.

3.4 Materialization Strategies

KG construction can leverage mapping partitions to reduce execution time and memory consumption. Before this, a partition of the mapping needs to be performed. We propose Algorithm 2, which generates partial mapping partitions by $\mathbb{P}$, and aggregates them for further partitioning. The purpose of this algorithm is to find a good partition (i.e., with a high

number of mapping groups) while keeping it simple and with a low computational cost.

Algorithm 2: *Partial-Aggregations Partitioning* of an RML document, $\mathcal{M}$.

Result: $\mathcal{P}$ of $\mathcal{M}$

```
1  M = normalize(M)
2  // Repeat for subj., pred., obj., and graph
3  for p ∈ ℙ do
4      // Lexicographic sort
5      M = sortByTermTypeAndLitTypeAndInv(M, p)
6      curTermType = ∅
7      curLitType = ∅
8      curGroup = 0
9      // Iterate over triples maps in M with a certain ℙ
10     for T ∈ M[p] do
11         I = invariant(T)
12         if type(T) ≠ curTermType then
13             curTermType = type(T)
14             curInv = ∅
15             curGroup++
16         else if type(T) = Literal ∧ literaltype(T) ≠ curLitType then
17             curLitType = literaltype(T)
18             curInv = ∅
19             curGroup++
20         else
21             if allTMsConstants(M, p) ∧ curInv < I then
22                 curGroup++
23             else if curInv ⊄ I then
24                 curGroup++
25             curInv = I
26         T.group = curGroup
27     end
28 end
29 P = aggregatePartialPartitions(M)
```

The outer for-loop (*line 3*) of Algorithm 2 iterates four times to retrieve partial mapping partitions by subject, predicate, object and graph. The normalized mappings are lexicographically sorted by $\mathbb{T}$, $\mathbb{I}$ and invariants (*line 5*) in the position being processed. Term maps for each $\mathbb{P}$ are then iterated (*line 10*). The *first condition* in *Definition 6* is fulfilled with *lines 12-15*, that create a new $\mathcal{G}$ when a $\mathcal{T}$ with a different $\mathbb{T}$ is reached. The *fourth condition* in *Definition 6* is satisfied with *lines 16-19*, which create a new a new $\mathcal{G}$ if $\mathbb{I}$ differs from that of the previous $\mathcal{T}$. If it was not possible to generate a new $\mathcal{G}$ with the former, we proceed to partition by invariant. When all the term maps in a specific $\mathbb{P}$ have constant values (*line 21*), the *third condition* in *Definition 6* can be applied. Otherwise, it is checked if *condition 2*, which is more restrictive, is fulfilled (*line 23*). The final $\mathcal{P}$ is the result of aggregating the partial mapping partitions by $\mathbb{P}$ (*line 29*), e.g., a mapping rule with partial mapping partitions *subject(4)*,

Algorithm 3: *Maximal Partitioning* of an RML document, $\mathcal{M}$.

Result: $\mathcal{P}_{max}$ of $\mathcal{M}$

```
1  M = normalize(M)
2  P_max = ∅
3  for order ∈ permutations(ℙ) do
4      for p ∈ order do
5          P = aggregatePartialPartitions(M)
6          for G ∈ P do
7              Apply lines 4-27 of Algorithm 2 to mapping rules in G
8          end
9      end
10     P = aggregatePartialPartitions(M)
11     if size(P) > size(P_max) then
12         P_max = P
13     M = resetPartition(M)
14 end
```

predicate(23), *object(11)* and *graph(1)* would be assigned the final partition *4-23-11-1*.

The time complexity of Algorithm 2 is $\mathcal{O}(n\ log\ n)$, where n is the number of mapping rules. This is because the outer for-loop can be removed by repeating its inner code four times (once per each $\mathbb{P}$). The complexity is then determined by the lexicographic sorting of the mapping rules.

We also propose Algorithm 3 that generates the maximal mapping partition of an RML document, i.e., it solves the problem of finding $\mathcal{P}_{max}$ of a mapping document. The assumption here is that exploring every possible mapping partition of an RML document guarantees obtaining the maximal one. To do so, it considers all orderings of $\mathbb{P}$ (*line 3*) and iterates over them (*line 4*). Partitioning is performed independently by $\mathcal{G}$ (*lines 6-7*), thus the aggregation of the partial partitions is done before (*line 5*) to generate these groups. Once the full partition has been created for an order of $\mathbb{P}$ (*line 10*), it is checked whether it has more groups than any other previously created (*lines 11-12*). Finally, the partial mapping partitions are reset (*line 13*) to prepare them for the next order processing.

The time complexity of Algorithm 3 is upper bounded by $\mathcal{O}(n\ log\ n)$, where n is the number of mapping rules. The worst case for *lines 6-7* happens when the mapping partition has only one mapping group and they all need to be sorted (*line 5* of Algorithm 2). It must be noted that *lines 3-4* do not affect the time complexity since they are fixed by $\mathbb{P}$. Although the time complexity of Algorithms 2 and 3 is similar, the latter needs to perform more operations, since it considers every permutation of $\mathbb{P}$.

The construction of a KG based on a mapping partition can be done in two different ways. The first (Figure 3.2) processes each mapping group sequentially. Hence, only the triples of a single mapping group are kept in memory simultaneously to remove duplicated triples. Memory usage is bounded by the largest group of mappings (in terms of the number of triples that it generates). The second (Figure 3.3) processes each group of mappings in parallel. As

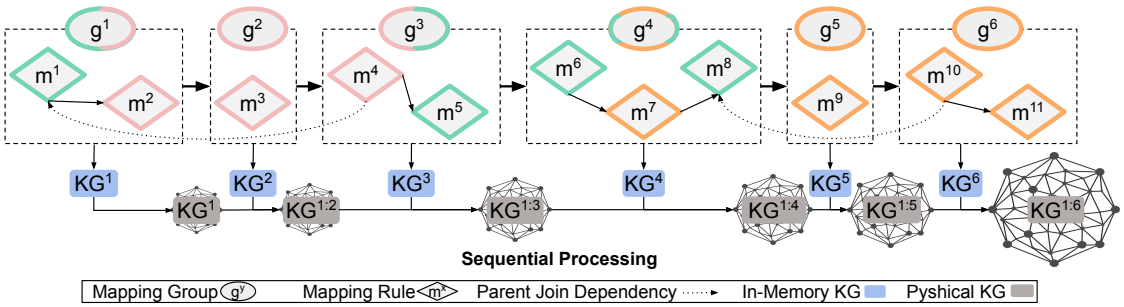


Figure 3.2: Sequential KG construction

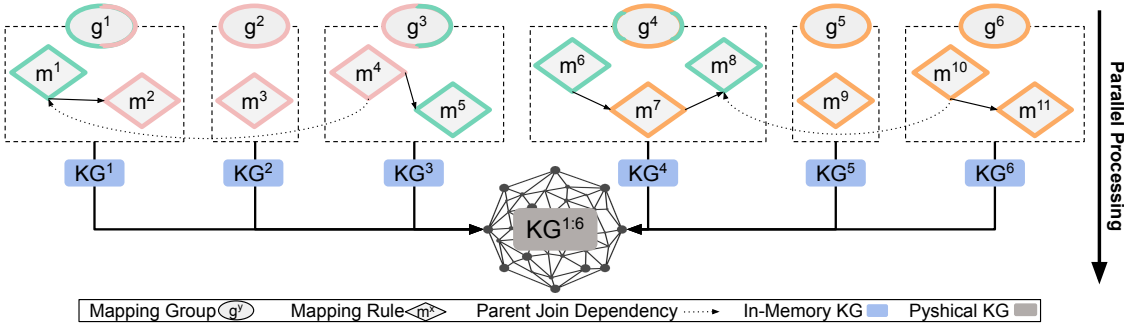


Figure 3.3: Parallel KG construction

Figure 3.4: Example of sequential and parallel processing of a mapping partition for constructing a KG. While the former creates the KG by processing one mapping group at a time, reducing memory consumption, the latter generates triples for several mapping groups simultaneously, reducing execution time.

a consequence, the execution time is reduced at the cost of increasing the maximum memory required, since multiple triple sets of different groups of mappings are maintained in memory at the same time.

3.5 Significance and Applicability of Mapping Partitions

The performance of partition-based KG construction strongly depends on the ability to partition mapping documents. If the conditions required to generate mapping partitions are not generally met, then mapping partitioning would not be feasible in practice (for instance, $\mathcal{I}_{\emptyset}$ prevents partitioning in the general case). In addition, the ability to generate a high number of mapping groups affects the improvement in the performance. In general, a higher number of mapping groups entails a higher parallelization capacity (bounded by the number of CPU cores), and a lower number of mapping rules in each of the groups, and therefore less memory consumption in the case of sequential processing.

We have compiled information on mapping partitioning for several well-known benchmarks (namely, NPD [90], BSBM [26], GTFS-Madrid-Bench [37] and LSLOD [71]), the DevOps ICT

KG [41], and other real use cases from the Knowledge Graph Construction W3C Community Group[3] in Table 3.1. We have included whether all the predicate maps are constant-valued, so that the *third condition* in *Definition 6* applies. We select predicate maps for this purpose because in real settings constant-valued term maps usually appear in this position (and in graph maps, but they are not used in the selected cases). The number of mapping groups and the maximum number of mapping rules in a group have been obtained using Algorithms 2 and 3. In all cases, it has been possible to obtain a mapping partition beyond $\mathcal{P}_{\emptyset}$. In most of the cases, the partitioning conditions are very advantageous, and low $\frac{\#\ mapping\ rules}{\#\ groups}$ ratios as well as small groups of mappings (with few mapping rules) are obtained. It can also be observed that both algorithms obtain a similar number of mapping groups in many cases. The most significant difference is found in the case of Data Hub - Ontopic, for which Algorithm 3 obtains a partition with a number of groups more than three times higher and drastically reduces the number of mapping rules in the largest group.

Table 3.1: Mapping partitioning in benchmarks and real use cases.

Benchmark or Real Use Case	all pred. constants	# mapping rules	partial-aggregations (Alg. 2)		maximal (Alg. 3)	
			# $\mathcal{G}$	max # rules in $\mathcal{G}$	# $\mathcal{G}$	max # rules in $\mathcal{G}$
GTFS-Madrid-Bench	yes	86	83	2	84	2
LSLOD - Bio2RDF	yes	182	117	37	117	37
LSLOD - Linkedct	yes	143	126	14	126	14
LSLOD - TCGA	yes	2450	388	380	409	55
LSLOD - Dailymed	yes	261	212	18	212	18
NPD	yes	1177	477	116	745	14
BSBM	yes	75	57	4	62	3
Open Cities - UPM	yes	122	99	6	99	6
Btw Our Worlds - IDLab	yes	62	47	4	47	4
SDM-Genomics - TIB	yes	169	105	8	105	8
Drugs4Covid - UPM	yes	75	27	29	38	19
Data Hub - Ontopic	yes	270	69	83	232	6
DevOps ICT KG	yes	326	299	3	299	3

3.6 Experiments

In this section, we experimentally evaluate our proposal. We aim at studying (i) the impact of mapping partitioning in the execution time and the memory consumption during the materialization of KGs, (ii) the effect that the number of groups in a mapping partition has in the materialization process, and (iii) comparing mapping partitioning and our implementation,Morph-KGC, with state-of-the-art systems. In the following, we describe the experimental setup.

3.6.1 Experimental setting

Implementation We developed a materialization system from scratch, Morph-KGC, and implemented self-join elimination and mapping partitioning on it. The system is written in Python and built on top of the Pandas library [103]. It executes the mappings creating a *triple table* [3] as a DataFrame, which is then serialized into N-Triples.

[3] https://github.com/kg-construct/use-cases

Benchmarks We evaluate our proposal on three different testbeds. First, we use GTFS-Madrid-Bench [37], a benchmark in the transport domain, for testing the performance and scalability of our proposal over different tabular data formats and sizes. After that, we use the SDM-Genomic-Datasets [79] from the biomedical domain to evaluate our proposal over different mapping configurations. Finally, we use the Norwegian Petroleum Directorate (NPD) benchmark [90], from the energy domain, to compare different configurations of Morph-KGC. We used MySQL v8.0 as relational database management system (RDBMS). The NPD benchmark provides the mappings in R2RML, the SDM-Genomic-Datasets in RML, and GTFS-Madrid-Bench provides the mappings in both languages.

Engines We use Morph-KGC v1.1.0[4] and consider five configurations of it: i) Morph-KGC as the baseline (without mapping partitioning); ii) Morph-KGC^{p}, which uses partial-aggregations for mapping partitioning and sequential processing; iii) Morph-KGC$^{p}_{+}$, which uses partial-aggregations for mapping partitioning and parallel processing, iv) Morph-KGC^{m} which uses maximal partitioning and sequential processing; and v) Morph-KGC$^{m}_{+}$ which uses maximal partitioning and parallel processing. We also compare our proposal against state-of-the-art KG materialization engines. Based on the results of our previous analysis [16], we select two R2RML engines, Ontop v4.1.0 and R2RML-F v1.2.3, and two RML interpreters, SDM-RDFizer v4.1.1 and Chimera v2.1. In our experimentation, we consider RDBs and CSV files, SDM-RDFizer and R2RML-F can process both of them, Ontop only processes the former and Chimera only the latter. It is important to mention that both selected RML processors parallelize the execution of the mappings.

Metrics *Execution time*: Elapsed time spent by an engine to complete the construction of a KG; it is measured as the absolute wall-clock system time as reported by the *time* command of the Linux operating system. *Memory consumption*: The memory used by an engine to construct the KG measured in time slots of 0.1 seconds. In addition, we have verified that the generated RDF are the same for all engines in terms of the number of triples and its correctness. All experiments were executed three times and the average execution time and memory consumption are reported. A timeout of 24 hours is used. The experiments are run on a CPU Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, 20 cores, 128 GB RAM, and a SSD SAS Read-Intensive 12 GB/s.

3.6.2 GTFS-Madrid-Bench results

We consider two distributions of the GTFS-Madrid-Bench based on the data format: GTFS^{csv} and GTFS^{rdb}. We also generated different data sizes of these distributions considering the scaling factors (SFs): 1, 10, 100, and 1.000. As reported in Table 3.1, the partial-aggregations and maximal partitioning algorithms return very similar mapping partitions (differing only in one mapping group). Thus, in this experiment, we only take into account partial-aggregations for mapping partitioning, avoiding the extra computational cost of maximal partitioning. Although the performance of Morph-KGC and Ontop is not impacted by self-joins because they remove them, the rest of the considered engines are extraordinarily affected by them.

[4] https://github.com/morph-kgc/morph-kgc/releases/tag/1.1.0

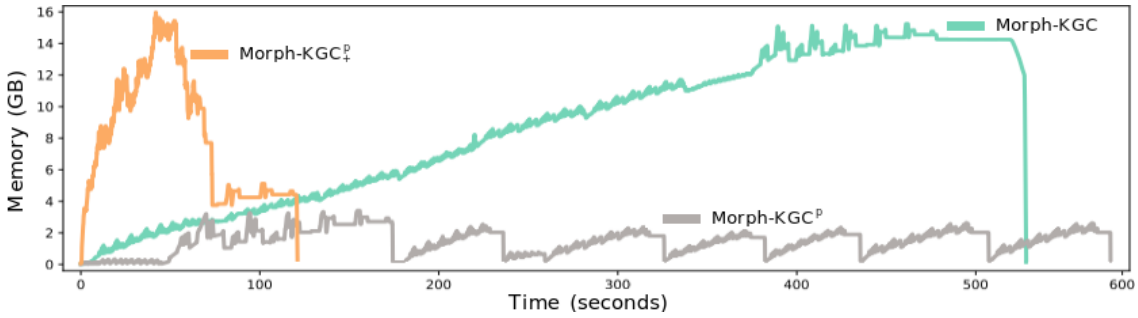


Figure 3.5: Memory over time $GTFS^{rdb}_{100}$

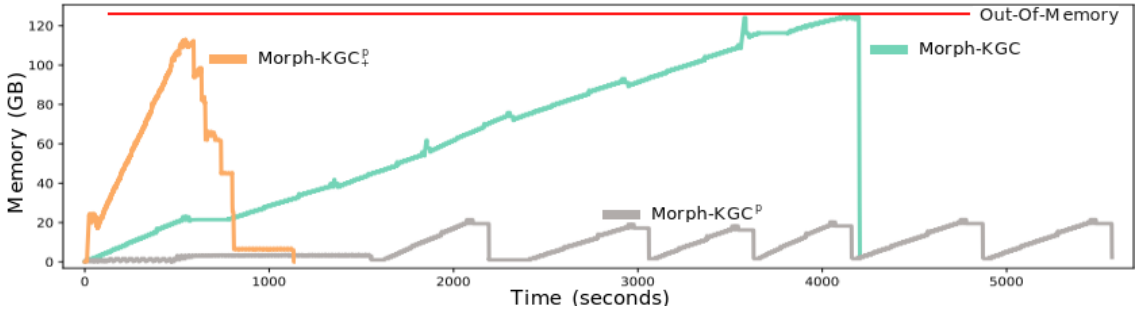


Figure 3.6: Memory over time $GTFS^{rdb}_{1.000}$

Figure 3.7: Morph-KGC over GTFS-Madrid-Bench. Memory over time in the materialization of $GTFS^{rdb}$ for three different configurations of the Morph-KGC engine: without mapping partitions (Morph-KGC), with mapping partitions and sequential processing (Morph-KGC^{p}), and with mapping partitions and parallel processing (Morph-KGC$^{p}_{+}$).

For this reason, we have manually removed self-joins from the original mappings. Mind that Ontop is only able to process GTFS^{rdb}, and Chimera GTFS^{csv}, as they do not support CSV and RDBs, respectively.

The impact of mapping partitions on the materialization of large input data sources can be observed in Figure 3.7. Regarding memory consumption, we can observe that the baseline, Morph-KGC, follows a growing trend over time. The reason is that it keeps the entire KG in memory to avoid the generation of duplicate triples. Indeed, Figure 3.6 shows that this approach produces an *out-of-memory* issue due to the size of the final KG. In the case of Morph-KGC^{p}, it is observed how memory is freed every time a group of mappings is materialized. In this configuration, the maximum memory peak is given by the largest group of mapping rules (in terms of the total number of triples generated), and it is significantly lower than the other two configurations. However, this comes at the cost of a small overhead in the execution time w.r.t. the baseline. Morph-KGC$^{p}_{+}$ demonstrates a great improvement w.r.t. the baseline regarding execution time, although the maximum peak of memory used is similar due to Morph-KGC$^{p}_{+}$ maintaining multiple groups of mappings in memory at the same time, as they are being processed concurrently. Note that mapping partition-based KG materialization is bounded by the parallelization capacity of the processor and by the mapping partition itself (e.g., the number of mapping groups or the differences in size among them).

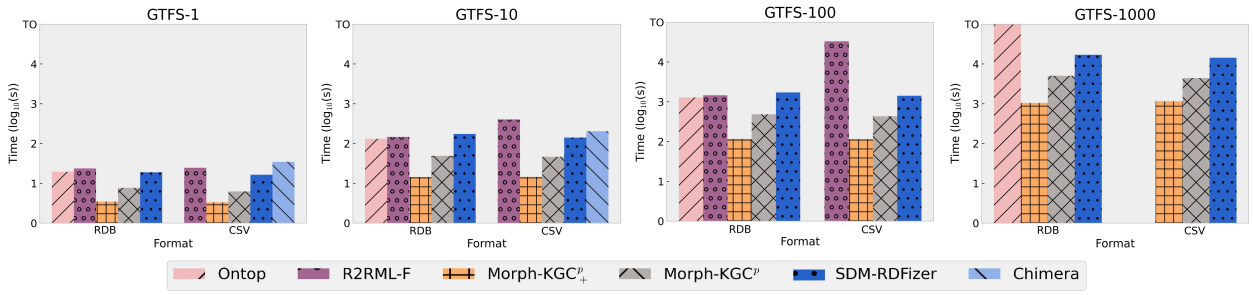


Figure 3.8: Total execution time in GTFS-Madrid-Bench. The absence of the bar indicates an *out-of-memory* issue. The bars reaching the top means a *timeout* issue.

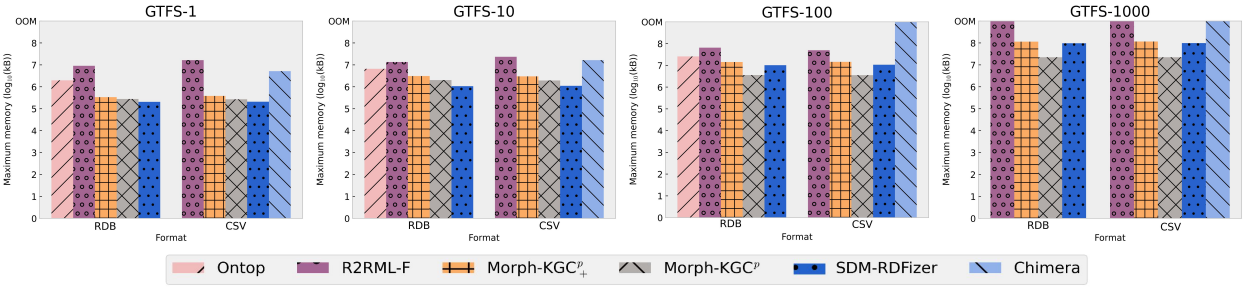


Figure 3.9: Memory consumption peak in GTFS-Madrid-Bench. The absence of the bar indicates a *timeout* issue. The bars reaching the top means an *out-of-memory* issue.

Figure 3.10: Total execution time and memory consumption peak in GTFS-Madrid-Bench. Materialization time in seconds and memory consumption time in kB (logarithmic scale) of the tabular datasets from GTFS-Madrid-Bench with data scaling factors 1, 10, 100 and 1.000.

Figure 3.8 shows the total execution time of Morph-KGC compared to the rest of the selected engines. Morph-KGC$^{p}_{+}$ clearly outperforms the rest of the engines for all data formats and data SFs. The optimizations implemented by SDM-RDFizer make them the only competitor capable of scaling to GTFS$_{1.000}$. Figure 3.9 depicts the maximum peak of memory used by each engine. It is observed that compression in SDM-RDFizer achieves the lowest memory usage for GTFS$_1$ and GTFS$_{10}$. In the case of larger GTFS distributions, Morph-KGC^{p} outperforms SDM-RDFizer given that it reduces the maximum memory peak used to that of the largest mapping group.

3.6.3 SDM-Genomic-Datasets results

The SDM-Genomic-Datasets [79] provide a set of configurations taking into account different parameters that are relevant for constructing KGs such as the type of mappings, the number of duplicates, and data size. More in detail, regarding the latter, four different datasets are provided with different number of rows: 10K, 100K, 1M, and 10M. Although simpler configurations are also provided in terms of the number of duplicates, we select the most complex one, i.e., 75% of duplicates with each duplicated value repeated 20 times. In addition,

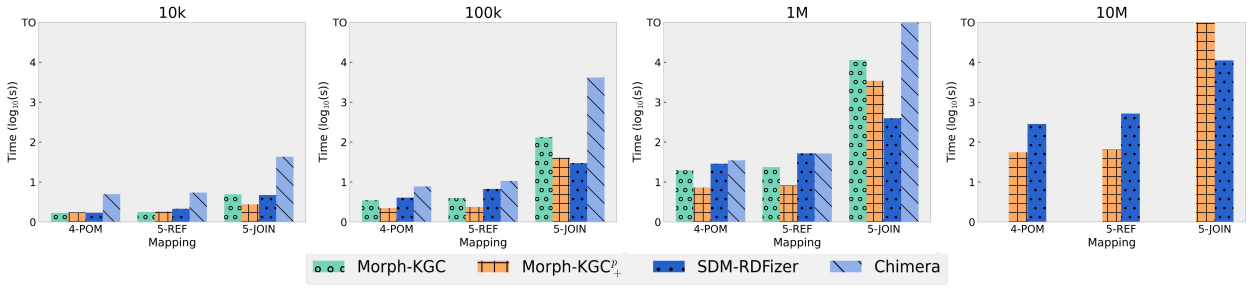


Figure 3.11: Total execution time in SDM-Genomic-Datasets. The absence of the bar indicates an *out-of-memory* issue. The bars reaching the top means a *timeout* issue.

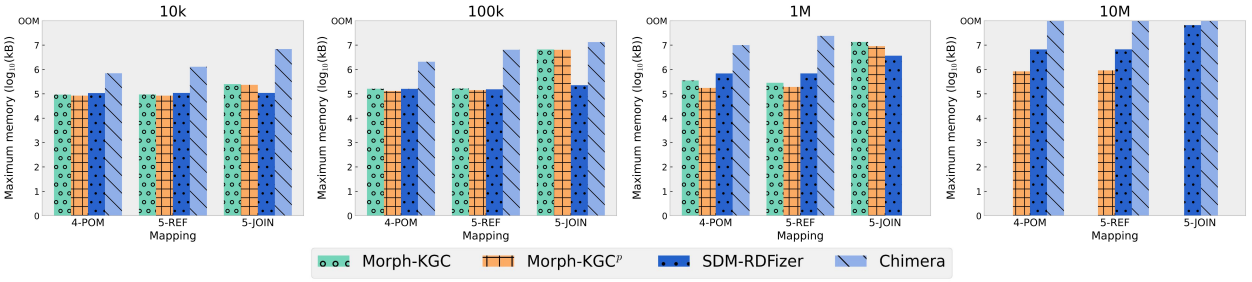


Figure 3.12: Memory consumption peak in SDM-Genomic-Datasets. The absence of the bar indicates a *timeout* issue. The bars reaching the top means an *out-of-memory* issue.

Figure 3.13: Total execution time and memory consumption peak in SDM-Genomic-Datasets. Materialization time in seconds and memory consumption time in kB (logarithmic scale) of the SDM-Genomic-Datasets with data scale factors 10K, 100K, 1M and 10M rows.

three mapping files with different types of predicate-object maps are considered: simple object map (POM), referencing object map with self-reference (REF), and referencing object map (JOIN). A number is used together with the name of each mapping type to specify the number of rules (e.g., 4-POM indicates 4 object maps). The testbeds provide the mappings in RML and data is in the form of CSV files. For this reason, we do not consider the R2RML processors in this experiment. In the same manner as for GTFS-Madrid-Bench, we only report the time and memory consumption for the case of partial-aggregations partitioning as the maximal partitioning algorithm generates the same mapping partition.

The total execution times for SDM-Genomic-Datasets are reported in Figure 3.11. As Morph-KGC$^{p}_{+}$ and Morph-KGC perform a self-join elimination over the REF mappings, they obtain similar results as in the POM ones, which is not the case of SDM-RDFizer and Chimera. While Morph-KGC$^{p}_{+}$ obtains the best results for the former configurations, SDM-RDFizer clearly outperforms it for JOIN mappings. The main reason is that SDM-RDFizer implements the Predicate Join Tuple Table as a specific physical data structure for improving the join conditions during the construction of the KG, and Morph-KGC does not implement any join optimization beyond redundant self-join elimination. Figure 3.12 shows that Morph-KGC^{p} outperforms its baseline and state-of-the-art engines regarding memory consumption for POM

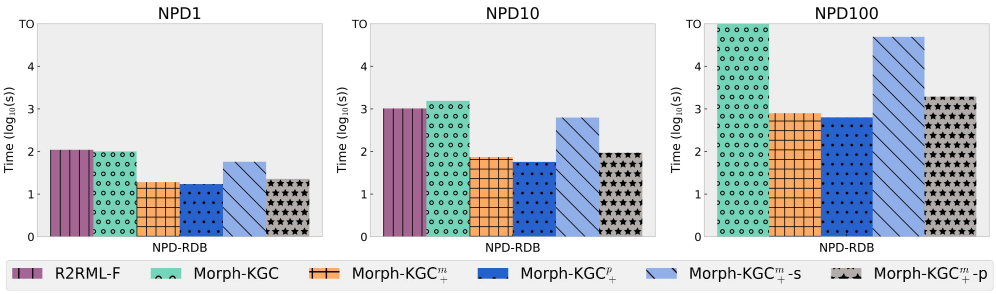


Figure 3.14: Total execution time in NPD Benchmark. The absence of the bar indicates an *out-of-memory* issue. The bars reaching the top means a *timeout* issue.

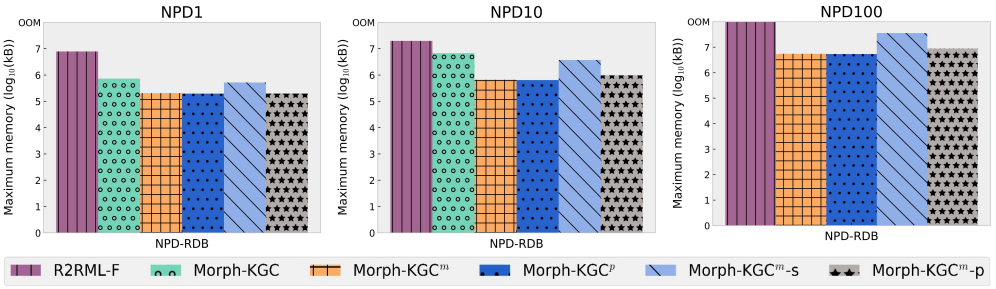


Figure 3.15: Memory consumption peak in NPD Benchmark. The absence of the bar indicates a *timeout* issue. The bars reaching the top means an *out-of-memory* issue.

Figure 3.16: Total execution time and memory consumption peak in NPD. Materialization time in seconds and memory consumption time in kB (logarithmic scale) of the NPD benchmark with data scaling factors 1, 10, 100.

and REF mappings. In the case of JOIN mappings, SDM-RDFizer obtains the best results given its data compression techniques.

3.6.4 NPD Benchmark results

The NPD benchmark [90] is a comprehensive testbed for virtual KG engines. It provides a set of SPARQL queries, a scalable instance of an RDB from the energy domain, and the corresponding mapping rules in R2RML. Although it has not been previously used for testing the performance of materialization engines, we notice that it is the only benchmark among those considered in Table 3.1 in which the difference in the number of mapping groups obtained by the partial-aggregations and the maximal partitioning algorithms is significant. This will allow us to compare the impact of maximal partitioning when it achieves a mapping partition with more groups than partial-aggregations. To further evaluate mapping partitions with different numbers of groups, we also include the configurations Morph-KGC$^{m}_{+}$-s and Morph-KGC$^{m}_{+}$-p which means that only subject and predicate, respectively, are taken into account to perform mapping partitioning (instead of every $\mathbb{P}$). We use the data generator of the benchmark [91] to obtain three different distributions with data SFs: 1, 10 and 100. Apart from our proposal, the only engine capable of parsing the R2RML mappings and generating the correct KG is R2RML-F.

The results obtained are shown in Figure 3.16. We observe that the best performance regarding execution time is obtained by Morph-KGC$^{p}_{+}$. Surprisingly, the partition generated by Morph-KGC$^{p}_{+}$ reports better results than the Morph-KGC$^{m}_{+}$ one, showing that a higher number of mapping groups does not always entail a better execution time in the materialization of the KG. A possible reason for this could be that parallel processing is bounded by the number of cores of the machine (20 in our case), and increasing the number of mapping groups (477 for Morph-KGC^{p} while there are 745 for Morph-KGC^{m}) does not result in a higher parallelization rate. Moreover, a higher number of mapping groups introduces an overhead as we saw previously in Figure 3.7, and obtaining the maximal partition is computationally more expensive. However, we observe that the materialization time of Morph-KGC$^{m}_{+}$ is very close to Morph-KGC$^{p}_{+}$, and that these two perform significantly better than Morph-KGC^{m}-s and Morph-KGC^{m}-p, with a lower number of mapping groups (17 and 327 respectively). This indicates that, in general, it is desirable to have a high number of mapping groups to increase the parallelization rate. Regarding memory consumption, we see that Morph-KGC^{p} and Morph-KGC^{m} obtain similar results. Note that in sequential processing, the peak in the amount of memory used is determined by the largest mapping group. If maximal partitioning is not able to further partition that specific mapping group, then a reduction in the peak of memory consumption is not expected.

3.7 Discussion

The experiments with SDM-Genomic-Datasets show that redundant self-join elimination is an effective technique and that it reduces the execution time of materialization. This behavior was also observed in our previous analysis [16] for GTFS-Madrid-Bench. Since our technique applies directly to mappings, it is engine-independent and can also be applied to any data format, as opposed to other proposals such as [31], that only work for RDBs. Any engine can benefit from this technique by preprocessing the mapping rules using Algorithm 1. However, this only applies to redundant self-joins, and the experiments with SDM-Genomics-Datasets also show that Morph-KGC is outperformed by SDM-RDFizer in complex joins, given that it implements the Predicate Join Tuple Table.

In general, our experiments show that mapping partitioning is an effective technique for KG materialization, as it avoids the generation of duplicate triples. Concurrent processing of mapping partitions has achieved the best execution times in many of the experimental configurations, except for those involving complex joins as previously mentioned. We have seen that in scenarios with several types of mappings (POM+REF+JOIN) (e.g., GTFS-Madrid-Bench or NPD), Morph-KGC$^{p}_{+}$ obtains better results than the rest of the engines. Moreover, sequential processing has obtained the lowest memory peak used in most of the experiment configurations, reducing the amount of memory used to that of the largest mapping group. It has also been observed that sequential processing adds a small overhead in the execution time. Overall, parallel processing is suitable for those cases in which KG materialization needs to be rapidly executed, and sequential processing for those cases in which it is necessary to keep memory usage low. Mapping partitioning can be implemented by other engines, in particular, those that still report performance issues such as Chimera, could benefit from this technique.

We have seen in the experiment with NPD benchmark that the number of mapping groups in a partition has an impact on the performance of materialization. Regarding the execution time, a higher number of mapping groups entail a faster execution in parallel processing. However, once the maximum parallelization capacity of the machine is reached, a higher number of mapping groups does not reduce the execution time and can even slightly increase it. Respecting memory consumption, sequential processing bounds the maximum peak of memory to that of the largest mapping group. A higher number of groups only reduces this peak in the case that the largest mapping group has been further partitioned.

3.8 Summary

In this chapter, we addressed the problem of efficient materialization of KGs from diverse data with declarative mappings. We started our work from the observation that existing systems result in timeout and out-of-memory issues when processing large volumes of data.

We presented the mapping partitioning optimization, which arranges rules into groups that generate disjoint sets of RDF triples. We propose algorithms to generate a partition from a set of mapping rules and applied them to several real-world use cases and benchmarks in the literature to show that they obtain partitions with a high number of mapping groups. We devised strategies for the efficient materialization of a partition based on parallel and sequential execution, which reduce materialization times and memory consumption, respectively. We evidenced the effectiveness of the proposal w.r.t. to a baseline without partitioning and showed that it outperforms state-of-the-art systems.

We also introduced an algorithm to remove redundant self-join from mappings. This optimization prevents the impact of unnecessary joins and can be applied by any materialization system as a preprocessing step.

It is also important to note that we developed a materialization system from scratch, Morph-KGC, to implement the aforementioned optimizations. The system is used as the basis to implement other techniques introduced in this thesis: `ITT` (Chapter 4), RML-star (Chapter 5), RML views for tabular data, and Python user-defined functions (Chapter 6). In the next chapter, we build on mapping partitioning to create virtual KGs based on the data translation paradigm.

Chapter 4

Intermediate Triple Table: A General Architecture for Virtual Knowledge Graphs

Current virtual knowledge graph (VKG) systems are based on query translation [137, 111, 106, 32, 119]. This technique has been extensively studied for scenarios in which the underlying data source is an RDB, by translating SPARQL queries into SQL. However, it is not so mature for NoSQL stores [78]. Moreover, VKG systems are limited to a single RDB; simultaneous access of multiple of them is not directly supported [78, 135].

In this chapter, we present *intermediate triple table* (`ITT`), a general architecture for VKGs. Our proposal is based on the data shipping paradigm [89] for data processing and addresses heterogeneity by adopting an *schema-oblivious* [18] graph representation, namely a triple table, that intervenes between the local data sources and the graph queries. A graph query is evaluated by populating the triple table with the subset of data relevant to the query. To find the relevant subset of the local sources, we rely on *star-shaped* query processing [129] and extend this technique to *mapping candidate selection* [126]. Moreover, data is rapidly transformed by parallelizing with mapping partitioning as presented in Chapter 3. This technique additionally guarantees that the intermediate triple table is duplicate-free, given the set semantics of RDF, reducing memory consumption. The queries are then homogeneously evaluated by translating SPARQL into SQL over the triple table [107]. A key point of `ITT` compared to query translation-based VKG systems is its *generality* in the sense that it is independent of the data model and formats of the underlying sources.

4.1 Related Work

4.1.1 Query translation

Query shipping or translation, and data shipping or translation are paradigms for data processing [89, 64]. Query shipping refers to queries being moved to the environment where the data is located. Query shipping applied to VKGs entails translating SPARQL queries

into the native query language of the underlying data sources (e.g., SQL, Cypher, MongoDB Query Language), which are then shipped and evaluated in the local data source system. Data shipping refers to data being transferred to the environment where the computation occurs. Data shipping applied to VKGs involves moving raw data from the underlying data sources to the VKG system and processing it to construct the final query result set.

So far, query translation has been the state-of-the-art technique for VKGs [135]. Given a query $\mathcal{Q}$ over $\mathcal{O}$ that produces the result set $\mathcal{R}$, the query translation technique consists in producing a query $\mathcal{Q}'$ over $\mathcal{S}$ which also produces $\mathcal{R}$. Here, $\mathcal{M}$ is used to translate (or unfold) $\mathcal{Q}$ to $\mathcal{Q}'$.

Query translation has mainly focused on RDBs, i.e., on SPARQL-to-SQL query translation. Well-known systems are Ontop [31] and Morph-RDB [111], which are open source, and Mastro [32] and Ultrawrap [119], which are proprietary. Since queries are executed by RDBs, the performance of this approach depends on that of the underlying RDBMS. Research in this VKG paradigm focused on optimizing unfolded queries [114, 111, 119, 35, 136].

Intermediate query [137] (`IQ`) was proposed by Ontop to unify SPARQL and SQL. `IQ` is used as a mid-representation between the input SPARQL queries and the unfolded SQL queries. In the work presented in this chapter, we do not use an intermediate representation of queries but of data. We partially transform input data sources into a relational representation of RDF, `ITT`, and we translate SPARQL to SQL queries over it. It must be noted that query unfolding in `IQ` depends on the data model of the local schema, while query unfolding in `ITT` is independent of the local schemas. `IQ` is limited to SQL and relational data sources, but `ITT` is not, since any data is transformed into `ITT` before query evaluation.

Other works generalized VKGs beyond RDBs. Botoeva et al. [28] proposed a generalized VKG framework based on `IQ` in which relational wrappers are used for NoSQL databases. These wrappers provide flat relational views of data that allow to reuse existing VKG techniques for RDBs. The lack of support for some computations (e.g., joins) in many NoSQL databases is solved by creating multiple native subqueries that are evaluated by the local sources and whose results are combined in a postprocessing step to construct the final SPARQL result set. The work presented in [106] also virtualizes over NoSQL databases in two phases. They first translate the SPARQL query into a *pivot abstract query*, independent of the target database, and then unfold the abstract query into the query language of the underlying database.

4.1.2 Data translation

Sequeda et al. [118] analyzed KG virtualization and materialization trade-offs in the context of RDBs. They propose to store the output triples of some mappings as materialized SQL views [60] to speed up query execution. In their approach, triples generated by a mapping rule are persisted as a materialized *triple-view* in the database. This approach, like ours, relies on the construction of a triple table from the mappings. However, their approach entails additional storage (triple-views are persisted on disk) and view maintenance costs, while we create a partial `ITT` on the fly that is limited in scope to a specific SPARQL query. In addition, their solution is specific to RDBs (views are materialized in the same database that contain the local sources), while ours is data model agnostic.

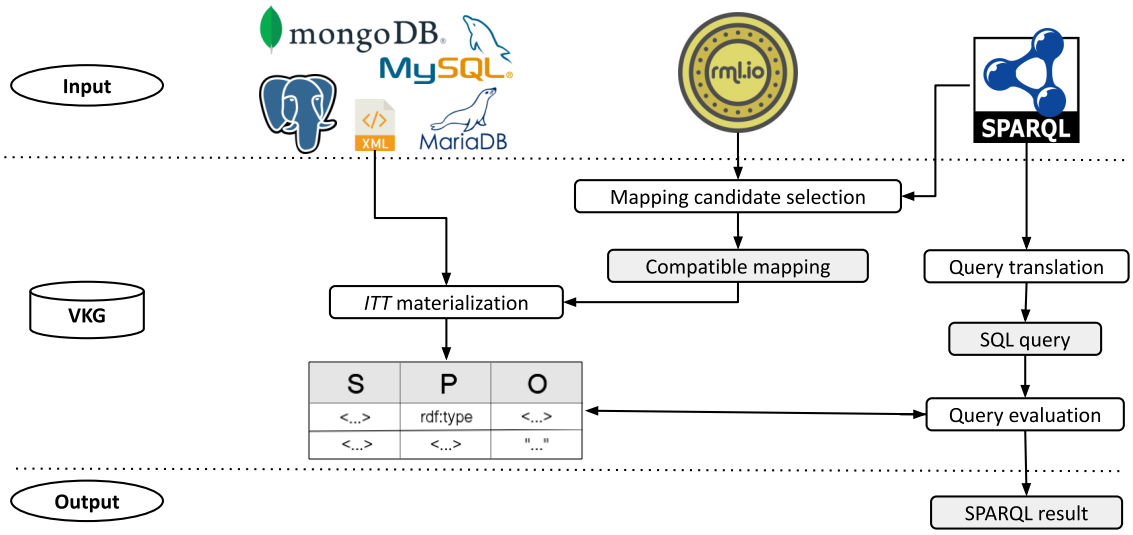


Figure 4.1: Overview of the ITT architecture.

Squerall [98], similar to our approach, is source independent and also relies on intermediate data structures that are populated on the fly, namely *ParSets*. However, there are significant differences between both solutions. On the one hand, ParSets are *schema-aware* [18] data structures: a *property table* [3] is constructed for each *star-shaped* subquery (defined as a group of triple patterns with a shared variable in the subject) in the input SPARQL query. This means that the organization of ParSets (e.g., the number of ParSets or the number of properties in the property tables) depends on the global schema, i.e., the ontology. On the other hand, ITT is a *schema-oblivious* [18] data structure, i.e., it is independent of the global schema. ITT also conforms to the output of the mappings, i.e., triples. This is a fundamental difference between the approaches: ITT can be directly built from the mappings, but creating and populating ParSets requires also mapping triples (resulting from mapping assertions) to property tables (ParSets). This involves heavy data computation that is performed at query execution time. Property tables are also problematic for multivalued and optional properties. Squerall is also a distributed solution based on technologies such as Apache Spark [17], while ours is single-node. Squerall does not support queries with unbounded predicates, i.e., with a variable in the predicate of a triple pattern. Kalaycı et al. [86] empirically showed that query translation with the Ontop system significantly outperforms Squerall in the general case.

Our work is closely related to the optimization of KG materialization. Constructing the ITT requires generating the RDF terms and arranging them into triples. N-Triples is the RDF serialization closest to ITT since it is based on a plain collection of triples; this concrete RDF syntax is used by Morph-KGC. The optimizations for KG materialization presented in Section 3.1 have a direct impact on our proposal since ITT construction is a heavy data processing task in our architecture.

4.2 Architecture Design

In this section, we introduce ITT. We first provide a general overview of the architecture and then describe each of its elements in detail.

For simplicity, we assume here *ground* RDF graphs, i.e., that do not contain blank nodes. This is not a limitation, since blank nodes can be converted to IRIs with skolemization [76]. In this regard, a *ground mapping* is one that does not contain term maps with blank node types. We also omit named graphs; nonetheless, our approach is easily extensible to RDF datasets by using an *intermediate quad table* (IQT).

4.2.1 ITT design

ITT follows the data shipping paradigm. It is inspired by intermediate representation of queries in the VKG query translation approach (see Section 4.1). Here, we propose to use an intermediate representation of data instead of queries. The data structure that we chose for graph data processing is a *triple table* [3], i.e., a single ternary relation $triple(s, p, o)$, and it is the fundamental element of the architecture. We refer to both, the architecture and the data structure, as ITT.

Figure 4.1 presents an overview of the ITT architecture. The system receives as input a set of local sources, which are diverse and can be in any data model and format (e.g., relational or document databases), a set of RML mappings, and the SPARQL query to be evaluated. The system performs the following tasks to produce the output result set for the query:

1. First, it finds the mapping rules that are relevant to evaluate the query with mapping candidate selection. As a result, a reduced set of mappings is obtained.
2. Next, the ITT is populated with data from the diverse local sources using the reduced mapping set.
3. Then, the SPARQL query is translated to an SQL query over the ITT.
4. Finally, the SQL query is evaluated over the ITT using an in-process analytical RDBMS. The results of this query is the final result set for the input SPARQL query.

The choice of a triple table as the intermediate data structure of the architecture is driven by the following observations: (i) it conforms with the output of the mappings (mappings emit triples that can be directly appended to the triple table), minimizing data transformation, and, as a result potentially yielding fast query evaluation; (ii) it is generic or schema-oblivious, combining graphs from different mapping rules is straightforward; (iii) state-of-the-art materialization systems build on this data structure such as maplib [20] and Morph-KGC; (iv) it is proven to be efficient for query evaluation [1, 107, 53]; and (v) its simplicity, graph data is stored as a single relation. In the following, each task is described in detail.

4.2.2 Mapping candidate selection

A naive strategy for ITT is to materialize the entire KG and create a giant triple table resulting from the execution of all the mapping rules. This entails translating all data on the

fly each time a query is evaluated over the VKG, which is inefficient, since unnecessary data is processed. Specifically, extra triples are created and the population of the `ITT` takes longer affecting the overall query execution time. Moreover, this naive approach also consumes additional memory, since the in-memory triple table grows in size due to the extra triples.

However, queries typically target a small fraction of the entire VKG; hence, most triples do not need to be accessed. An enhancement of the aforementioned strategy is to process only those entities and properties targeted by the SPARQL query to reduce the amount of processed data. In this regard, the relevant or *compatible* (denoted with $\top$) mapping rules w.r.t. a query are those rules that contribute to the evaluation of the query. Finding compatible mapping rules for a given SPARQL query is called *mapping candidate selection* [126] (or mapping-query binding).

Definition 1 (Binding of a mapping document w.r.t. a query)**.** Let q be a query, and let $\mathcal{M}$ be a set of mapping rules m_1, m_2, ..., m_n. The binding $\mathcal{M}_q$ of $\mathcal{M}$ w.r.t. q is a subset of $\mathcal{M}$, $\mathcal{M}_q \subset \mathcal{M}$, s.t. compatible m_i w.r.t. q are contained in $\mathcal{M}$.

$$\nexists m \in (\mathcal{M} \setminus \mathcal{M}q) \mid m \top q$$

The mapping rules that do not belong to the binding, i.e., outside $\mathcal{M}_q$, are not compatible w.r.t. the query. This definition is relaxed in the sense that incompatible mapping rules w.r.t. the query can be included in $\mathcal{M}_q$. We rely on star-shaped subqueries, widely used in SPARQL query evaluation [129], to find mapping candidates. To achieve this, we group mapping rules in a star-shaped fashion and then match them with star-shaped subqueries. The mapping rules are grouped by joining subject term maps with overlapping ranges, for which we rely on the notion of term map compatibility [126].

Definition 2 (Term map compatibility)**.** Two term maps $\mathcal{T}_1$ and $\mathcal{T}_2$ are compatible if their ranges overlap.

$$range(\mathcal{T}_1) \bigcap range(\mathcal{T}_2) \neq \emptyset$$

The compatibility of two term maps can be determined with term map disjointness as presented in Section 3.3. Specifically, this is done by checking the invariant of a term map.[1]

Definition 3 (Star-shaped mapping group)**.** Each mapping rule with term maps $\{\mathcal{T}_s\mathcal{T}_p\mathcal{T}_o\}$ s.t. $invariant(\mathcal{T}_s) = \mathcal{I}$ is a star-shaped mapping group w.r.t. $\mathcal{I}$, or $\mathcal{I}^* - \mathcal{MG}$.

The subject term maps in $\mathcal{I}^* - \mathcal{MG}$ are compatible. The normalization of a triples map forms a $\mathcal{I}^* - \mathcal{MG}$ given that all its mapping rules have the same subject term map. Other triples maps may also have a compatible subject invariant, which may result in a larger star-shaped mapping group. SPARQL queries and RML mapping documents are decomposed into star-shaped subqueries and mapping groups, which are then matched to find mapping candidates. Star-shaped decomposition of mappings include rules in one group only and two groups cannot be compatible, i.e, with compatible subject maps. Following this, given two

[1]Term map disjointness is also affected by term types, but here subject maps can only be IRIs, as enforced by ground mappings.

Algorithm 4: Star-shaped mapping candidate selection.

Input: Mapping document $\mathcal{M}$
Input: SPARQL query q
Result: Compatible mapping set $\mathcal{M}_q$ of $\mathcal{M}$

```
1  M_q = φ // initial empty set for mapping bindings
2  SPG∫ = retrieveStarPatterns(q)
3  SMG∫ = retrieveStarMappings(M)
4  for SPG ∈ SPG∫ do
5      for SMG ∈ SMG∫ do
6          if types(SPG) ⊄ types(SMG) then
7              continue
8          if ∀tp ∈ SPG, ∃m ∈ SMG | tp⊤m then
9              for m ∈ SMG do
10                 if ∃tp ∈ SPG | tp⊤m then
11                     M_q.add(m)
12             end
13     end
14 end
```

mapping groups $\mathcal{I}_1^* - \mathcal{MG}$ and $\mathcal{I}_2^* - \mathcal{MG}$, if $\mathcal{I}_1 \subset \mathcal{I}_2$ then both mapping groups fuse into a single star-shaped group given by $\mathcal{I}_1$. Binding of star-shaped mapping groups and subqueries is based on the compatibility of term maps in the former with the RDF terms in the triple patterns of the latter.

Definition 4 (Term map and RDF term compatibility)**.** A term map $\mathcal{T}$ and an RDF term r are compatible if $r \in range(\mathcal{T})$. If r is a variable, then it is always compatible with $\mathcal{T}$, otherwise r and $\mathcal{T}$ are compatible if they do not satisfy any of the following conditions:

- $type(r) \neq type(\mathcal{T})$.
- $type(r) = type(\mathcal{T}) = Literal$, and $literaltype(r) \neq literaltype(\mathcal{T})$.
- $\mathcal{T}$ is template- or constant-valued, and it cannot match r.

Definition 5 (Mapping rule and triple pattern compatibility)**.** A mapping rule m with term maps $\{\mathcal{T}_s\mathcal{T}_p\mathcal{T}_o\}$ and a triple pattern tp with RDF terms $\{r_s r_p r_o\}$ are compatible if their respective term maps and RDF terms are in turn position-wise compatible. Formally:

$$\forall \mathcal{T} \in m, \exists r \in tp \mid r \top \mathcal{T}, position(r) = position(\mathcal{T})$$

Finally, the binding of star-shaped subqueries and mappings is presented in Algorithm 4. The first condition checked is that the subquery and mapping types match. A typed subquery contains at least one triple pattern *{?s a :class}* and a typed mapping group contains at least one mapping rule with term maps *{$\mathcal{T}$ a :class}*, i.e., where predicate and object maps are constant-valued indicating that the resources generated by the subject map are an instance of a class. If there is a type in the subquery that is not contained in the mapping group, the generated triples cannot satisfy the subquery. Next, all triple patterns must be compatible

```
SELECT ?loc WHERE {
    ?emp rdf:type ex:Employee .
SQ1 ?emp ex:job "ACCOUNTANT" .
    ?emp ex:department ?dept .
    ?dept rdf:type ex:Department .
SQ2 ?dept ex:location ?loc .
}
```

Figure 4.2: Example SPARQL query over the VKG in Figure 2.3 that retrieves the location of departments with accountants. The query is composed of two star-shaped subqueries, one associated to employees (`?emp`) and another associated to departments (`?dept`).

with at least one rule in the mapping group. Only rules in the mapping group that are compatible with the triple patterns are selected as candidates, since incompatible ones yield irrelevant results for the subquery. In practice, the feasibility of star-shaped mapping candidate selection is based on the fact that predicates in triple patterns are bounded and that predicated maps are constant-valued, as shown in Section 3.5. Figure 4.2 shows an SPARQL query with two star-shaped subqueries that match both mapping groups in Figure 2.3 (given by `TriplesMapEMP` and `TriplesMapDEPT`).

4.2.3 Materialization of the intermediate triple table

Compatible mappings are used to translate the data into a queryable data structure, independent of the heterogeneity of the underlying data sources, the `ITT`. The translation of local sources to RDF is time-consuming since complex operations need to be computed over large amounts of data (percent encoding, string scaping, functions defined in mappings, etc.). This is minimized with mapping candidate selection, but it is fundamental for performance that the `ITT` is efficiently materialized. We adopt the mapping partitioning optimization, introduced in Chapter 3, which allows to:

- Minimize the size of the `ITT`. It finds dependencies between mapping rules in terms of the disjointness of the sets of the generated triples, which allows to generate a duplicate-free output. As a result, all triples in `ITT` are unique, its size is reduced, and memory usage is minimal.
- Minimize the time to materialize the `ITT`. Mapping rules dependencies are exploited to execute them in a parallel fashion, leveraging modern hardware, which typically consist of processors with many CPUs.

We adopt mapping partitioning as a baseline in our architecture, but other optimizations can also be applied to further optimize `ITT` materialization. We also maintain the `ITT` in memory to avoid reading and writing from disk. Theoretically, the `ITT` can also be spilled to disk in those cases in which it does not fit in memory. Figure 4.3 shows the materialized `ITT` for the SPARQL query in Figure 4.2.

s	p	o
ex:employee/7369	rdf:type	ex:Exployee
ex:employee/7369	ex:job	"CLERK"
ex:employee/7369	ex:department	ex:department/10
ex:employee/5826	rdf:type	ex:Exployee
ex:employee/5826	ex:job	"ACCOUNTANT"
ex:employee/5826	ex:department	ex:department/10
ex:department/10	rdf:type	ex:Department
ex:department/10	ex:location	"NEW YORK"

Figure 4.3: Materialized ITT for the SPARQL query in Figure 4.2 with the star-shaped mapping binding.

4.2.4 Querying the intermediate triple table

ITT is queried with SPARQL-to-SQL translation of queries, where the SQL queries scan the triple table. This translation is *generic*, since it does not depend on a certain database schema, i.e., it is schema-oblivious.

Querying triple tables has been widely studied in the literature. Well-known graph stores such as Virtuoso [53] and RDF-3X [107] demonstrate the efficiency of this data structure for query evaluation; specifically, Virtuoso is usually included in RDF graph store evaluations [5, 115] achieving state-of-the-art performance. SPARQL operators have their counterpart in the SQL algebra; namely, a JOIN in SPARQL is an INNER JOIN in SQL, OPTIONAL corresponds to LEFT JOIN, FILTER, with WHERE, operators UNION, LIMIT and OFFSET have homonym operators in SQL, and so on. Figure 4.4 shows the unfolded SQL query of the SPARQL query in Figure 4.2. Popular query optimization techniques for triple tables include vertical partitioning [1] and indexing [53].

We adopt the OLAP approach to evaluate unfolded SQL queries over the ITT. Unfolded queries are often complex with a high number of joins, and OLAP executes such queries fast. Moreover, OLAP optimizes for read operations, which fits with our architecture: once the ITT is materialized the nature of the system is *read-only*, which evaluates the unfolded query once.

4.2.5 Implementation

We implemented ITT on top of Morph-KGC. We adapted Morph-KGC to increase its materialization performance and also implemented missing components of the ITT architecture, as explained in this section.

Morph-KGC is built on top of the Pandas library, which is used to create a triple table. Triples are then serialized in the N-Triples syntax, but our architecture queries the triple table instead of serializing the graph. However, as pointed out by [20], the Polars[2] library provides similar functionality than Pandas with increased performance. For this reason, we replaced

[2] https://github.com/pola-rs/polars

```
SELECT
  loc
FROM
  ( ( ( ( ( (
              SELECT
                s AS dept, o AS loc
              FROM
                ITT
              WHERE
                p = 'ex:location'
            ) AS v950117
            INNER JOIN (
              SELECT
                s AS dept
              FROM
                ITT
              WHERE
                p = 'rdf:type' AND o = 'ex:Department'
            ) AS v208794 ON v950117.dept = v208794.dept
          ) AS v208794
          INNER JOIN (
            SELECT
              s AS emp, o AS dept
            FROM
              ITT
            WHERE
              p = 'ex:department'
          ) AS v320081 ON v208794.dept = v320081.dept
        ) AS v320081
        INNER JOIN (
          SELECT
            s AS emp
          FROM
            ITT
          WHERE
            p = 'ex:job' AND o = '"ACCOUNTANT"'
        ) AS v927160 ON v320081.emp = v927160.emp
      ) AS v927160
      INNER JOIN (
        SELECT
          s AS emp
        FROM
          ITT
        WHERE
          p = 'rdf:type' AND o = 'ex:Employee'
      ) AS v588866 ON v927160.emp = v588866.emp
    )
  )
```

Figure 4.4: SQL unfolding of the SPARQL query in Figure 4.2 over the `ITT` in Figure 4.3. The query is independent of the local data source (`EMP` and `DEPT` tables in Figure 2.3).

Pandas with Polars to transform data faster and obtain lower end-to-end query execution times. Furthermore, Morph-KGC uses SQLAlchemy [22] to access RDBs, but we replaced it with ConnectorX [133] which is faster and more memory efficient.

We used Lark[3] to parse SPARQL queries and we implemented star-shaped subquery and mapping group decomposition and binding. The materialization of the triple table with compatible mappings produces a Polars DataFrame. We implemented SPARQL-to-SQL query

[3] https://github.com/lark-parser/lark

translation and adopted the in-process analytical database DuckDB [113] for query evaluation which directly scans the ITT from the Polars DataFrame.

4.3 Experiments

In this section, we empirically analyze the ITT architecture. In our experiments, our aim is to (i) validate the generality of the ITT architecture, i.e., that it is independent of local sources, (ii) analyze its query execution performance, and (iii) compare it with the state-of-the-art VKG query translation approach.

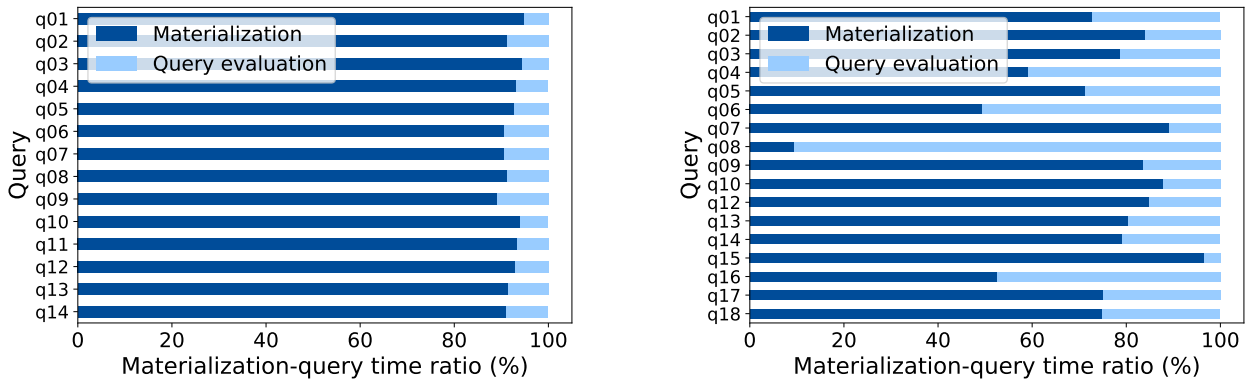
4.3.1 Experimental setting

Benchmarks We test VKG systems with two benchmarks. One is GTFS-Madrid benchmark [37], which was used for experimentation in the previous chapter. One of the advantages of this benchmark is that it provides multiple distributions of data in different formats (RDB, MongoDB, CSV, XML and JSON), which allows validating source independence of ITT. The data sources can be scaled in size, and in our experiments we consider data scaling factors 1, 10, 100 and 1.000. SF 1 amounts for ∼396 K triples and SF 1.000 amounts for ∼396 M triples. The query set consists of 18 complex queries with several triple patterns and SPARQL operators (UNION, OPTIONAL, ORDER BY, GROUP BY, etc.). We discarded query 17 because it contains the NOT EXISTS functional form, which none of the systems under test support. The other benchmark is LUBM4OBDA, which we created as a VKG extension of the popular benchmark for graph store evaluation LUBM [61]. This benchmark models the university domain (departments, professors, publications, etc.). The local data source of LUBM4OBDA is an RDB that can be scaled in size with the generator provided by the Linköping GraphQL Benchmark [39]. In our experiments we use data SFs 1, 10, 100 and 1.000 which range from ∼112 K triples to ∼149 M triples. The query set is composed of 14 queries that are simpler than those of the GTFS-Madrid benchmark, with a low number of triple patterns and without additional SPARQL clauses. Although the creation of this benchmark is not the central contribution of this chapter, we highlight here two points that distinguish it from existing VKG benchmarks: it allows to evaluate statement-level metadata approaches including RDF-star (this is related to the work presented in Chapter 5), and it allows to evaluate some inference capabilities that are not considered by previous benchmarks. We refer the interested reader in the LUBM4OBDA benchmark to [15].

Systems In addition to our ITT implementation (we use v1.0[4]), we select Ontop v5.1.0 as the representative system for the VKG query translation approach. Ontop is well-known to be the state-of-the-art VKG system. It is open source and has been applied in several use cases [86, 82, 134, 99]. We use MySQL v8.0.28 and PostgreSQL v14.2 as RDBMSs and MongoDB v7.0.0 as document store.

Environment The experiments were performed on an Intel® Core™ i7-1165G7 (2.80GHz) and a memory of 40 GB RAM DDR4 (3200 MHz). All the times reported are the average

[4] https://github.com/arenas-guerrero-julian/ITT/releases/tag/1.0



(a) LUBM4OBDA SF 100.

(b) GTFS-Madrid SF 100.

Figure 4.5: Ratio between the materialization time of the `ITT` and query time of the `ITT` for LUBM4OBDA SF 100 and GTFS-Madrid SF 100 with PostgreSQL.

time of 5 executions and we define a timeout of 1 hour.

4.3.2 Experimental results

Tables 4.1-4.4 show the end-to-end query execution times for LUBM4OBDA and Tables 4.5-4.8 for GTFS-Madrid for all data SFs and queries. The execution times of the VKG systems are grouped by the underlying database management system (DBMS). In the case of GTFS-Madrid, MongoDB is included, but only for `ITT`, since Ontop does not support NoSQL databases. For each query and underlying DBMS, the result of the VKG system that achieved the fastest execution time is highlighted in green and bold. Timeout and out-of-memory issues are highlighted in red.

Differences among benchmarks In general, `ITT` outperforms Ontop in GTFS-Madrid, and Ontop outperforms `ITT` in LUBM4OBDA. This is mainly due to the query workloads of both benchmarks. On the one hand, GTFS-Madrid consists of an analytical workload with complex queries that include many triple patterns, aggregates, and computationally intensive operations such as ORDER BY and GROUP BY. On the other hand, LUBM4OBDA resembles an operational workload, with simple queries that consist of a few triple patterns, some of them even of a single one. These workloads are known as OLAP and OLTP [58], respectively.

Workload characterization The results of `ITT` for both workloads are explained by the distribution of the time taken by each task in the architecture. The end-to-end query execution time in `ITT` is dominated by two tasks, (i) materialization of the `ITT`, and (ii) query evaluation over the `ITT`; other tasks, such as mapping candidate selection, are independent of data and do not significantly influence the execution time. The materialization of `ITT` involves the evaluation of many simple queries over the RDBMS, typically of the form *`SELECT col1, ..., colN FROM table`*. These queries correspond to the logical views defined in the compatible mappings. Query evaluation requires posing a single SQL query on the `ITT`. The complexity

Table 4.1: Results for LUBM4OBDA SF 1.

	MySQL		PostgreSQL	
	Ontop	ITT	Ontop	ITT
q01	0.36	**0.24**	**0.17**	0.35
q02	**0.23**	0.25	**0.2**	0.38
q03	478.64	**0.22**	**0.28**	0.32
q04	0.81	**0.3**	0.53	**0.45**
q05	**0.18**	0.19	**0.18**	0.3
q06	0.23	**0.17**	**0.19**	0.27
q07	**0.25**	0.28	**0.2**	0.38
q08	0.51	**0.26**	0.47	**0.38**
q09	0.32	**0.31**	22.05	**0.42**
q10	0.24	0.24	**0.16**	0.34
q11	**0.14**	0.18	**0.02**	0.29
q12	**0.18**	0.2	**0.17**	0.31
q13	**0.15**	0.19	**0.14**	0.3
q14	0.21	**0.19**	**0.21**	0.27

Table 4.2: Results for LUBM4OBDA SF 10.

	MySQL		PostgreSQL	
	Ontop	ITT	Ontop	ITT
q01	**1.55**	1.78	**0.46**	1.79
q02	**0.83**	1.21	**0.29**	1.33
q03	TO	**1.41**	**1.11**	1.48
q04	**1.84**	2.08	**0.86**	2.03
q05	**0.25**	1.11	**0.23**	1.21
q06	1.41	**0.95**	**0.8**	1.03
q07	**1.57**	1.97	**0.39**	2.0
q08	1.7	**1.4**	**0.78**	1.65
q09	3.21	**2.12**	**0.85**	2.25
q10	**1.64**	1.76	**0.69**	1.94
q11	**0.13**	0.93	**0.14**	1.02
q12	**0.18**	0.94	**0.2**	1.07
q13	**0.63**	0.98	**0.17**	1.14
q14	1.02	**0.94**	1.23	**1.03**

Table 4.3: Results for LUBM4OBDA SF 100.

	MySQL		PostgreSQL	
	Ontop	ITT	Ontop	ITT
q01	67.43	**20.98**	**4.09**	18.98
q02	**11.02**	13.38	**2.99**	12.56
q03	TO	**16.5**	16.9	**15.22**
q04	**14.17**	21.64	**3.69**	20.47
q05	**0.84**	12.26	**0.76**	11.62
q06	**12.38**	13.29	**5.56**	9.81
q07	38.42	**22.83**	**2.87**	20.62
q08	19.87	**15.84**	**6.91**	14.93
q09	52.6	**25.02**	**6.58**	22.03
q10	**17.05**	21.56	**6.34**	19.49
q11	**0.13**	10.43	**0.13**	9.32
q12	**0.17**	10.72	**0.19**	9.72
q13	**4.09**	10.97	**0.5**	10.13
q14	**9.2**	10.53	10.42	**9.57**

Table 4.4: Results for LUBM4OBDA SF 1000.

	MySQL		PostgreSQL	
	Ontop	ITT	Ontop	ITT
q01	709.91	**280.07**	**38.08**	195.37
q02	**129.06**	199.74	**29.05**	132.49
q03	TO	**230.52**	**91.81**	154.5
q04	**154.07**	OOM	**27.96**	OOM
q05	**1.52**	179.17	**5.7**	122.41
q06	**140.37**	150.37	**51.7**	105.36
q07	**400.74**	OOM	**80.39**	OOM
q08	**221.82**	224.88	**61.96**	158.73
q09	**674.01**	OOM	**219.9**	OOM
q10	**192.46**	288.54	**61.77**	197.62
q11	**0.15**	146.88	**0.14**	102.24
q12	**0.17**	149.8	**0.24**	101.15
q13	**53.69**	156.49	**28.29**	105.91
q14	**103.16**	161.46	108.02	**105.1**

Table 4.5: Results for GTFS-Madrid SF 1.

	MySQL		PostgreSQL		Mongo
	Ontop	ITT	Ontop	ITT	ITT
q01	2.1	**0.83**	1.95	**0.67**	**1.54**
q02	0.16	**0.11**	0.14	**0.12**	**0.13**
q03	0.32	**0.13**	0.3	**0.14**	**0.14**
q04	0.19	**0.14**	**0.17**	0.23	**0.15**
q05	0.21	**0.1**	0.2	**0.12**	**0.13**
q06	0.11	**0.07**	0.11	**0.08**	**0.08**
q07	25.39	**0.25**	0.5	**0.48**	**0.37**
q08	41.76	**0.31**	124.54	**0.39**	**0.36**
q09	0.77	**0.47**	3.5	**0.55**	**1.58**
q10	TO	**0.11**	0.35	**0.13**	**0.14**
q12	TO	**0.18**	0.4	**0.22**	**0.24**
q13	0.18	**0.09**	0.18	**0.11**	**0.12**
q14	TO	**0.15**	1.4	**0.21**	**0.22**
q15	0.25	**0.11**	0.25	**0.16**	**0.17**
q16	1.63	**0.15**	0.42	**0.27**	**0.19**
q17	1591.76	**0.13**	0.36	**0.17**	**0.18**
q18	0.22	**0.17**	0.23	**0.22**	**0.21**

Table 4.6: Results for GTFS-Madrid SF 10.

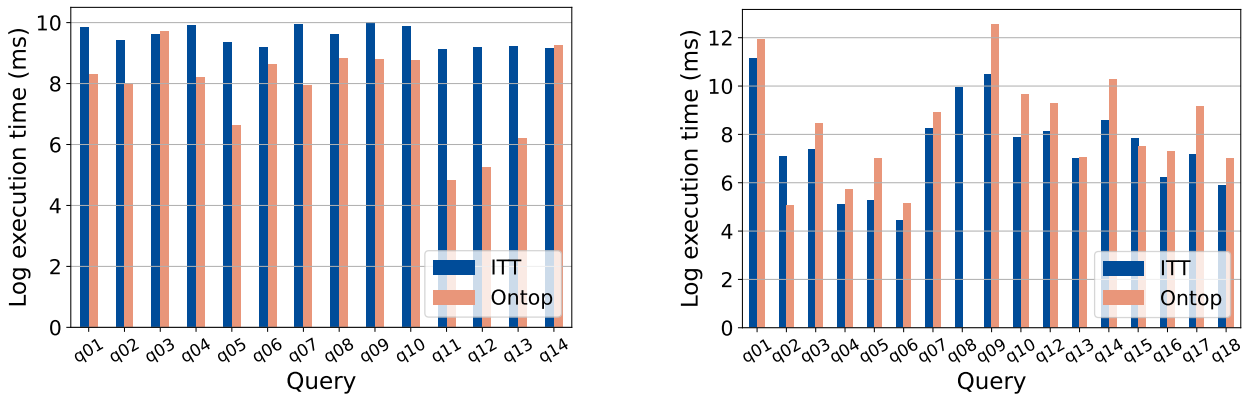
	MySQL		PostgreSQL		Mongo
	Ontop	ITT	Ontop	ITT	ITT
q01	17.29	**8.16**	15.11	**5.75**	**14.89**
q02	**0.17**	0.19	**0.15**	0.21	**0.36**
q03	1.05	**0.23**	0.87	**0.25**	**0.42**
q04	0.22	**0.13**	0.19	**0.16**	**0.16**
q05	3.08	**0.1**	0.26	**0.12**	**0.13**
q06	0.13	**0.07**	0.11	**0.08**	**0.08**
q07	2337.39	**0.55**	1.59	**0.64**	**1.07**
q08	3094.02	**3.56**	311.78	**1.22**	**1.26**
q09	5.53	**3.28**	27.33	**3.59**	**14.55**
q10	TO	**0.3**	1.61	**0.33**	**0.56**
q12	TO	**0.44**	1.54	**0.48**	**0.85**
q13	0.33	**0.16**	0.33	**0.19**	**0.34**
q14	TO	**0.49**	3.46	**0.55**	**0.94**
q15	0.56	**0.31**	0.56	**0.38**	**0.8**
q16	54.63	**0.18**	0.5	**0.22**	**0.27**
q17	TO	**0.21**	1.66	**0.26**	**0.35**
q18	0.35	**0.18**	0.33	**0.24**	**0.27**

Table 4.7: Results for GTFS-Madrid SF 100.

	MySQL		PostgreSQL		Mongo
	Ontop	ITT	Ontop	ITT	ITT
q01	173.88	**64.11**	150.77	**68.52**	**151.48**
q02	**0.19**	1.1	**0.16**	1.19	**2.74**
q03	5.0	**1.44**	4.66	**1.59**	**3.34**
q04	0.36	**0.14**	0.31	**0.17**	**0.19**
q05	269.27	**0.17**	1.13	**0.19**	**0.29**
q06	0.12	**0.08**	0.17	**0.09**	**0.09**
q07	TO	**3.9**	7.4	**3.9**	**8.62**
q08	TO	**23.46**	TO	**20.73**	**12.61**
q09	52.88	**34.3**	279.94	**35.66**	**147.11**
q10	TO	**2.49**	16.02	**2.61**	**5.1**
q12	TO	**3.26**	11.03	**3.44**	**7.28**
q13	1.05	**0.92**	1.15	**1.1**	**2.56**
q14	TO	**8.46**	28.59	**5.27**	**8.59**
q15	**1.27**	2.37	**1.84**	2.5	**7.47**
q16	3571.96	**0.48**	1.51	**0.5**	**1.01**
q17	TO	**1.32**	9.38	**1.33**	**2.15**
q18	0.9	**0.31**	1.12	**0.36**	**0.84**

Table 4.8: Results for GTFS-Madrid SF 1000.

	MySQL		PostgreSQL		Mongo
	Ontop	ITT	Ontop	ITT	ITT
q01	**1770.62**	OOM	**1483.32**	OOM	OOM
q02	**0.53**	11.81	**0.38**	12.56	**28.9**
q03	40.11	**16.55**	38.9	**15.84**	**34.77**
q04	1.0	**0.27**	1.02	**0.32**	**0.47**
q05	TO	**0.82**	5.08	**0.91**	**1.89**
q06	0.12	**0.11**	**0.12**	0.13	**0.2**
q07	TO	**38.62**	81.18	**40.99**	**91.6**
q08	TO	OOM	TO	OOM	OOM
q09	**860.02**	OOM	**2679.64**	OOM	OOM
q10	TO	**26.28**	201.75	**26.71**	**54.17**
q12	TO	**33.9**	134.33	**35.67**	**74.52**
q13	**7.27**	10.44	**6.75**	10.53	**27.12**
q14	TO	**44.97**	316.11	**52.02**	**91.41**
q15	**5.9**	26.83	**11.97**	28.23	**81.27**
q16	TO	**3.19**	7.54	**3.3**	**9.06**
q17	TO	**10.25**	103.79	**13.16**	**22.85**
q18	5.99	**1.83**	10.32	**1.79**	**7.01**



(a) LUBM4OBDA SF 100.

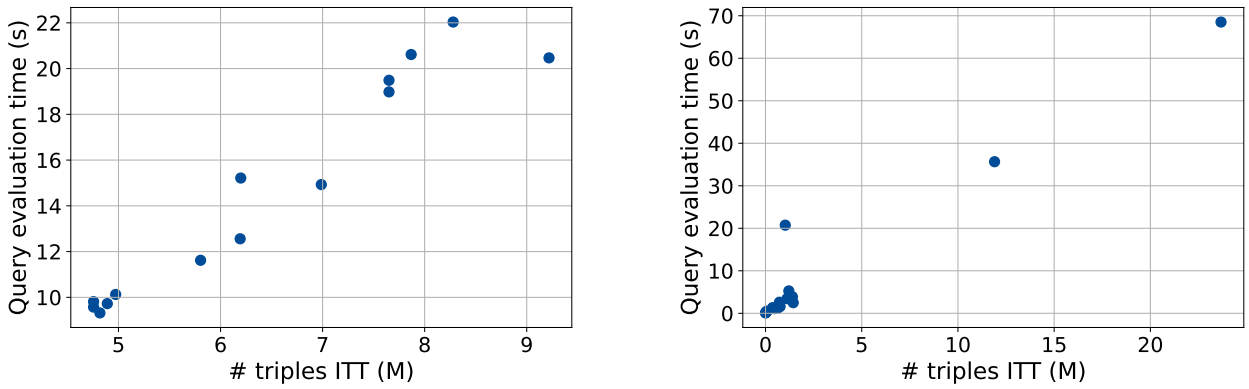
(b) GTFS-Madrid SF 100.

Figure 4.6: Log query execution times for LUBM4OBDA SF 100 and GTFS-Madrid SF 100 with PostgreSQL. The absence of the bar indicates a timeout.

of this SQL query is determined by that of the SPARQL query, since it is derived from it. The rationale behind the good results of our architecture with the GTFS-Madrid benchmark is that transactional RDBMSs, such as MySQL and PostgreSQL, are optimized to evaluate many simple queries, as needed to materialize the `ITT`, and because `ITT` is queried using an embedded analytical DBMS, optimized for evaluating a single computationally intensive query.

Query evaluation time in Ontop is dominated by the execution of the unfolded query over the underlying RDBMS. Here, a single but complex query (i.e., analytical) resulting from the aggregation of multiple logical views in the mappings is executed over the RDBMS. This becomes a bottleneck, since transactional RDBMSs are not optimized for analytical workloads. However, in the case of LUBM4OBDA, with simpler SPARQL queries, Ontop outperforms `ITT` given that the RDBMS can efficiently evaluate the unfolded query and there is no extra data transformation cost to create the `ITT`.

The behavior of `ITT` with operational- and analytical-like workloads is corroborated in Figure 4.5, which shows the ratio between `ITT` materialization time and query evaluation time over the `ITT` for both benchmarks with SF 100 and PostgreSQL. The results for LUBM4-OBDA queries are uniform, being data translation the bottleneck for all queries. In particular, materialization corresponds to 90%-95% of the query execution time except for query 9 (89.12%). The simplicity of LUBM4OBDA queries results in fast query executions over `ITT` and makes materialization the most time-consuming task. Nevertheless, the ratios for GTFS-Madrid present significant differences among queries. The relative time taken by materialization w.r.t. query execution ranges from 9.31% for query 8 to 96.53% for query 15, and in general, query evaluation takes significantly longer than in LUBM4OBDA. Query 18 is an outlier and the result is explained by the complexity of the query, which becomes the bottleneck; this is consistent with the result observed for Ontop (Table 4.7). On the contrary, the simplicity of query 15 w.r.t. the others makes it stand out as the one where query execution consumes less time, and materialization is the bottleneck.



(a) LUBM4OBDA SF 100.

(b) GTFS-Madrid SF 100.

Figure 4.7: Log query execution times for LUBM4OBDA SF 100 and GTFS-Madrid SF 100 with PostgreSQL. The absence of the bar indicates a timeout.

Figure 4.8: Relationship between the number of triples of the `ITT` (in millions) and the end-to-end query execution time (in seconds) for LUBM4OBDA SF 100 and GTFS-Madrid SF 100 with PostgreSQL.

Execution issues We observe that `ITT` produces errors only for the datasets with the highest SF while Ontop fails for smaller SFs. Basically, `ITT` is error-free for small and medium-sized datasets, while Ontop can produce errors even for reduced volumes of data. It must be noted that the types of errors differ among the data and query translation approaches. `ITT` keeps subsets of local data sources in memory, resulting in out-of-memory errors. Ontop produces timeouts because it generates analytical SQL queries that operational RDBMSs are not optimized for.

We take a closer look at the failed queries. Query 3 of LUBM4OBDA consists of two triple patterns. Ontop unfolds this query into an SQL query accessing 6 tables with CROSS JOINS (i.e., Cartesian product) and also includes a UNION clause. The high number of CROSS JOINS already results in timeouts with MySQL for data SF 10. However, `ITT` produces a simple query with a single INNER JOIN. `ITT` produces out-of-memory errors for three queries with SF 1.000 in which the translated data is too large to fit in memory. In the GTFS-Madrid benchmark, Ontop produces timeouts for several queries with MySQL, but successfully executes them for PostgreSQL. Query 8 is the only one that timeouts for Ontop over PostgreSQL for data SFs 100 and 1.000; we examined all unfolded queries, and this is likely the most complicated one. `ITT` also fails for query 8; it is able to create the triple table, but the embedded DBMS runs out of memory. In addition, our architecture runs out of memory when materializing the triple tables for queries 1 and 9.

Differences among queries Figure 4.6 shows the logarithmic query execution times for both benchmarks with SF 100 and PostgreSQL. The execution times of `ITT` are roughly in the same order of magnitude for all LUBM4OBDA queries. In contrast, execution times are more diverse for Ontop; some queries take several seconds to execute, and others just a fraction of a second. In the case of GTFS-Madrid benchmark, the execution times among

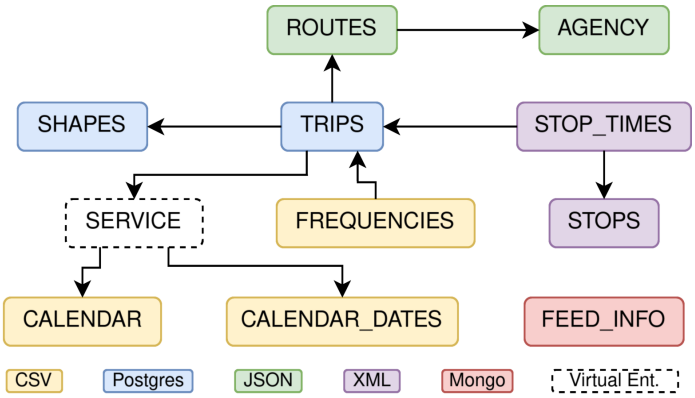


Figure 4.9: Distribution of GTFS-Madrid benchmark with mixed data formats (CSV, PostgreSQL, JSON, XML and MongoDB).

queries significantly differ for both systems.

The complexity of SPARQL queries has an impact on both VKG approaches, since a complex SPARQL query is likely to result in a more complicated query over the `ITT` and a more complicated SQL unfolding in the case of Ontop. In addition, end-to-end execution in `ITT` is also influenced by the amount of data to be translated. Figure 4.8 shows the relationship between the size of the `ITT` in terms of the number of triples for LUBM4OBDA and GTFS-Madrid with SF 100 and the execution time with PostgreSQL. It is clear that there is a correlation between both variables, with larger sizes of the `ITT` entailing higher query execution times. Thus, query evaluation with `ITT` is influenced by the amount of data to materialize.

Differences among RDBMSs It is observed in Tables 4.1-4.8 that `ITT` performs similar with MySQL and PostgreSQL, with most queries being slightly faster with MySQL. This is because only simple queries in the mappings are executed over the underlying database to ingest data, and most of the time is spent in other tasks such as data transformations (percent encoding, string escaping, duplicate and `NULL` removal, etc.). However, Ontop presents significant differences among RDBMSs, with PostgreSQL outperforming MySQL in the general case. PostgreSQL is well-known to be more optimized for complicated queries like those generated by Ontop. The differences between both RDBMSs are more noticeable in the GTFS-Madrid benchmark, with analytical-like queries that make MySQL struggle. Here, MySQL times out for some queries even with SF 10, and takes thousands of seconds to execute for others.

4.3.3 Validation of source independence

We tested `ITT` with GTFS-Madrid benchmark and a document store, namely MongoDB. The result sets obtained for all queries are correct and identical to those produced for RDBMSs. Query execution times (Tables 4.5-4.8) are generally slower than with RDBs, but without excessive differences. We go a step further and test `ITT` under a mix distribution of the GTFS-Madrid benchmark with multiple data formats (CSV, PostgreSQL, JSON, XML and MongoDB) with SF 100. Figure 4.9 shows the GTFS-Madrid distribution that we used. All

queries run successfully and the results are the same as those for Ontop and ITT for RDBs. This shows that ITT does not only work for data models beyond RDBs, but also operates over multiple heterogeneous data sources and evidences the generality and source independence of the architecture.

4.4 Discussion

This section discusses insights of the ITT architecture based on the conducted experimentation, potential improvements, and some general advantages and limitations.

Memory-intensive architecture A potential criticism of our architecture is that it makes intensive use of memory w.r.t. query translation. As shown in the experimentation, very large volumes of data may produce out-of-memory errors in our baseline implementation since the ITT is kept on memory. Nevertheless, larger-than-memory ITTs could be supported by spilling to disk. Similarly, queries whose ITT fits in memory but query evaluation over it makes intense use of memory can also be avoided with out-of-core processing. Indeed, DuckDB, the system on which our implementation is based to evaluate queries over the ITT, implements out-of-core operators in its latest version.

Polystore VKG Query translation VKG systems such as Ontop are bound to a data model and only work with a single database, since they delegate query execution to it. To evaluate queries over multiple RDBs in this approach, an additional federation layer is required. There are two possible approaches [135], (i) federating at the data source level with SQL federators such as Denodo[5] so as to query a unified relational view with a VKG query translation system such as Ontop, or (ii) federating at the SPARQL endpoint level by creating a VKG for each RDB with a query translation system and using an SPARQL federator such as FedX [116] or HeFQUIN [38]. ITT does not need an additional federation layer, and it is itself a polystore [52] that works not only with multiple RDBMSs, but also with NoSQL databases given that it is source independent.

Hybrid VKG systems We have shown that ITT is competitive w.r.t. query translation VKG systems. However, there are cases where the performance of query translation is superior. A natural step towards more optimized VKG systems is hybridization. A hybrid solution would select the most efficient technique given a particular query and local data source. For example, ITT can be applied with queries that are complex or produce restrained intermediate triple tables, and query translation can be selected for simpler queries or those that involve translating large volumes of data in the ITT architecture.

Declarative transformation functions In our work, we have focused on schema transformations with RML, i.e., the correspondences between the local sources and the target ontology of the VKG. However, some use cases also require data transformation, computation, and filtering [44] of the local sources. So far, query translation VKG systems used RML

[5] https://www.denodo.com

views[6] to push down these operations (with functions in the SQL queries) to the underlying database. However, each DBMS provides its own set of functions, which makes mappings source dependent. RML-FNML [46] is the RML module for the declarative definition of transformation functions. Function execution with RML-FNML is independent of the database that is being accessed by the VKG. Given that `ITT` is based on data translation, transformations can be performed during KG materialization, so that the evaluation of the query over `ITT` is not affected. This intrinsic support of declarative transformation functions (indeed, our `ITT` implementation supports RML-FNML with user-defined functions, see Section 7.3) is an advantage of `ITT` w.r.t. the query translation approach. We also propose an approach to execute RML-FNML by VKG query translation systems (see Section 7.4).

OLAP backends in VKGs Our experimental setting in Section 4.3 uses OLTP RDBMSs to store data. One could also study VKGs in cases where the underlying RDB is an OLAP system instead of OLTP. In such a case, the query translation approach can potentially perform better, since they are optimized for the analytical-like unfolded queries. However, this is not common for the following reason. VKGs are more adecuate for dynamic data with frequent CRUD (Create, Read, Update, and Delete) operations for which OLTP is optimized. This is consistent with VKG experimentation in previous works [37, 90, 114, 111] and major use cases [86, 82, 134, 99], which also use OLTP RDBMSs as backend. It is comprehensible that VKG systems must optimize for OLTP rather than OLAP backends.

ITT optimization Materialization, as derived from experimentation, is the task in our architecture that takes the longest time to execute, so further optimizations of `ITT` should focus on this task. This can be done in different ways. An alternative is optimizing RDF materialization itself, which multiple recent works recently addressed (see Section 3.1). Another possibility is to reduce the amount of triples that are materialized into the `ITT` so that a lower volume of data is translated and, as a result, the execution time decreases. In this sense, instead of generating all triples for a binding of a mapping, the *bind join* [62] optimization could be applied to generate fewer triples. *Caching* of triples in the `ITT` could also lower materialization costs in use cases that can assume slightly outdated query results.

4.5 Summary

In this chapter, we addressed the problem of creating VKGs over multiple and disparate data sources. We propose an architecture based on immediate data translation, rather than query translation, which is widely adopted in VKGs, but is data model-specific. Our architecture, `ITT`, uses an schema-oblivious data structure, a triple table, that intervenes between the local sources and the queries. A fundamental point for the effectiveness of the proposal is that the amount of data transformations is reduced given that the intermediate data structure conforms to the output of the mapping rules, whose output can be efficiently combined. Star-shaped mapping candidate selection is used before materializing the triple table to reduce the amount of translated data. When creating the associated triple table of a graph query,

[6] An RML view over an RDB is an SQL query (instead of a base table) limited in scope to a certain mapping rule.

the query is translated into an SQL query over the triple table, which is evaluated with an embedded analytical database.

We implement the architecture on top of the Morph-KGC system and experimentally demonstrate its feasibility and data model independence. We show that the proposal is competitive w.r.t. VKG systems based on query translation and that it is especially favorable for complex graph queries.

Chapter 5

Declarative Generation of RDF-star Graphs

RDF-star (originally, *RDF** [67]) was proposed as an extension of RDF to make statements about other statements (also known as reification [73]). RDF-star extends RDF's conceptual data model and concrete syntaxes by providing a compact alternative to other reification approaches, such as *standard reification* [72] or *singleton properties* [108]. Following the uptake of the initial version of RDF-star, the W3C RDF-DEV Community Group[1] released a W3C Final Community Group Report [69] and the RDF-star Working Group[2] was formed to extend related W3C Recommendations.

RDF-star introduces the concept of *quoted triple*. A quoted triple is an RDF term which is an RDF triple itself. `« :Angelica :jumps "4.80" » :date "2022-03-21"` is an RDF-star triple with a quoted triples appear in the subject. In this way, quoted triples are annotated with statement-level metadata. A quoted triple can appear in the subject or object of a triple, they can be deeply nested, and may be asserted or not (i.e., if the quoted triple is also an element of the graph, then it is also asserted). An RDF-star triple is recursively defined as follows [69]:

- any RDF triple is an RDF-star triple;
- if t and t' are RDF-star triples $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$, and $o \in (\mathcal{I} \cup \mathcal{B} \cup L)$, then (t, p, o), (s, p, t) and (t, p, t') are RDF-star triples.

Although several libraries and graph stores adopted RDF-star,[3] the generation of RDF-star graphs from diverse data remains largely unexplored. In this chapter, we extend RML for the generation of RDF-star graphs. Specifically, we (i) create a module to declare the creation of quoted triples, namely RML-star, (ii) provide an algorithm to process RML-star mappings and materialize RDF-star, (iii) implement the algorithm in the Morph-KGC system, and (iv) validate the implementation with test cases.

[1] https://www.w3.org/groups/cg/rdf-dev
[2] https://www.w3.org/groups/wg/rdf-star
[3] https://w3c.github.io/rdf-star/implementations

5.1 Related Work

Two alternatives have been proposed to systematically generate RDF-star from diverse data sources. R2RML-star [121] is an extension of R2RML for which the author provides an algorithm to translate SPARQL-star into SQL queries (i.e., for virtualization). SPARQL-Anything [19] is a SPARQL-based mapping language that uses the CONSTRUCT clause in SPARQL-star and Apache Jena. In the following, we compare these two alternatives with our RML-star proposal.

RML-star and SPARQL-Anything allow generating RDF-star from multiple diverse data sources and formats, while R2RML-star builds upon R2RML, which is limited to relational databases.

RML-star and SPARQL-Anything support joins and recursion. The R2RML-star extension enables recursion, but joins can only be performed with R2RML views. This occurs because the ranges of `star:subject` and `star:object` are `rr:ObjectMap` but `rr:RefObjectMap` is not foreseen, which is the one that allows joining with other data sources.

RML-star introduces a unique construct to define the quoted triples and "flags" if a quoted triple should be asserted. In R2RML-star only quoted triples are generated. If the corresponding asserted triples need to be generated, an extra triples map needs to be defined to assert the quoted triple. Similarly, to assert a quoted triple in SPARQL-Anything, an additional triple has to be specified in the query.

RML-star, R2RML-star and SPARQL-Anything are accompanied by implementations. In Section 5.3 we present a procedure to materialize KGs with RML-star, and Section 5.4 presents our implementation in Morph-KGC. R2RML-star is implemented as an extension of Ontop [137] for virtual RDF-star graphs [121], while the implementation of SPARQL-Anything carries the same name as the syntax. The implementations of RML-star and SPARQL-Anything follow the materialization approach, while the R2RML-star one follows the virtualization approach. Morph-KGC and SPARQL-Anything are open source but the source code of the R2RML-star implementation is not publicly available. It must be noted that, the current implementation of R2RML-star is limited: it does not support binding of quoted triples to variables in SPARQL-star queries.

5.2 RML-star

We designed RML-star as a complementary module of RML. Figure 5.1 depicts the RML-star module which introduces two new constructs to generate quoted triples: *star map* and *non-asserted triples map*. We describe these constructs below and exemplify their usage with the CSV data in Listing 5.1.

Star Map The star map (`rml:StarMap`) is the building block of RML-star. A star map references another triples map to declare the generation of quoted triples. Following the RDF-star data model, a quoted triple may appear only in the subject or object of a triple. Thus, star maps can only be used in subject and object maps. Star maps use the property

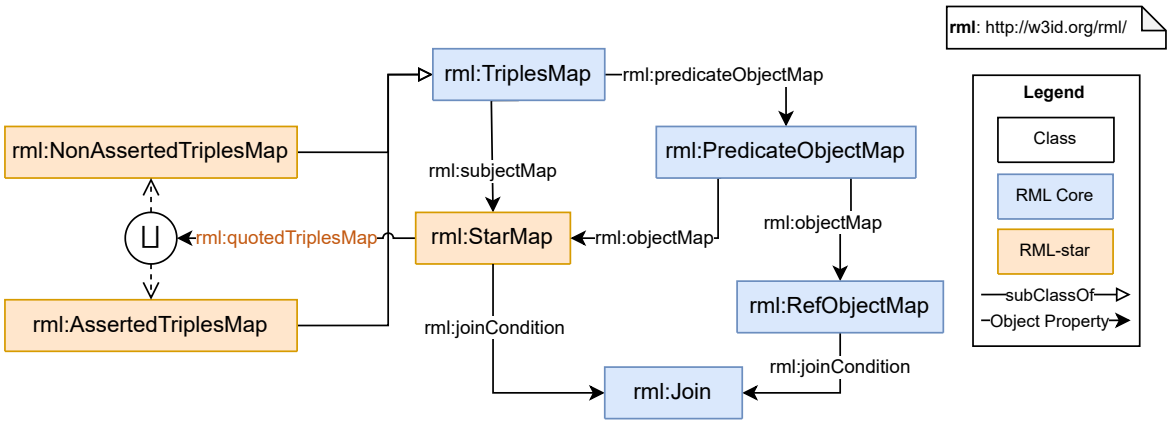


Figure 5.1: The RML-star module (represented using the Chowlk Visual Notation [55]). RML-star resources are highlighted in orange, while the rest of the represented ontology belongs to the RML-Core module.

`rml:quotedTriplesMap` to refer to the triples map that generates the quoted triples. An example of a star map can be observed in Listing 5.2, with the `#dateTM` triples map containing a star map in the subject (*line 24*) pointing to the `#jumpTM` triples map; the `#dateTM` triples map generates the quoted triples in *lines 2, 4, 6 and 8* of Listing 5.3. The recursive design of star maps allow chaining triples maps to create deeply nested quoted triples.

Non-Asserted Triples Map An *asserted RDF-star triple* is a triple that is an element of an RDF-star graph, as opposed to a quoted triple, that only appears in the subject or object of another triple. A quoted triple can optionally be asserted if it is also an element of the RDF-star graph. In RML-star, all generated triples, including quoted triples, are asserted by default. To allow the creation of quoted triples that are not to be asserted, we introduce the non-asserted triples map (`rml:NonAssertedTriplesMap`) as a subclass of triples map. To illustrate this, imagine that the `#jumpTM` triples map was a non-asserted triples map, then the triples in *lines 1, 3, 5 and 7* would not be generated, as the quoted triples would not be asserted. A non-asserted triples map has the same expressiveness as any other triples map, and it just adds the information of being non-asserted.

```
1 ID, DATE       , MARK, PERSON   , SCORE
2 1  , 2022-03-21, 4.80, Angelica , 1211
3 2  , 2022-03-19, 4.85, Katerina , 1224
```

Listing 5.1: Contents of the logical source `:marks` in CSV format. It must be noted that we have included blank spaces for a better visualization of the tabular structure.

```
<#jumpTM>
    a rml:AssertedTriplesMap;
    rml:logicalSource :marks;
    rml:subjectMap [
        rml:template ":{PERSON}"
    ];
    rml:predicateObjectMap [
        rml:predicate :jumps;
        rml:objectMap [
            rml:reference "MARK"
        ]
    ];
    rml:predicateObjectMap [
        rml:predicate :score;
        rml:objectMap [
            rml:reference "SCORE"
        ]
    ].

<#dateTM>
    a rml:TriplesMap ;
    rml:logicalSource :marks;
    rml:subjectMap [
        rml:quotedTriplesMap <#jumpTM>
    ];
    rml:predicateObjectMap [
        rml:predicate :date;
        rml:objectMap [
            rml:reference "DATE"
        ]
    ].
```

Listing 5.2: Example RML-star mapping that transforms data in Listing 5.1.

```
:Angelica :jumps "4.80" .
<< :Angelica :jumps "4.80" >> :date "2022-03-21" .
:Katerina :jumps "4.85" .
<< :Katerina :jumps "4.85" >> :date "2022-03-19" .
:Angelica :score "1211" .
<< :Angelica :score "1211" >> :date "2022-03-21" .
:Katerina :score "1224" .
<< :Katerina :score "1224" >> :date "2022-03-19" .
```

Listing 5.3: RDF-star triples generated by the mapping in Listing 5.2.

5.3 Materialization Procedure

The materialization of an RML-star mapping rule is presented in Algorithm 5. It takes the following parameters as input: (i) the mapping rule to be processed (m); (ii) the complete set of rules in the mapping document (M), which is needed to retrieve the nested rules; and (iii) the nesting level of a rule (*nestLevel*), with 0 referring to the lack of nesting. An RML-star

processor generates the output dataset of an RML-star document by applying Algorithm 5 to all the mapping rules in the document.

Algorithm 5: Materialization of an RML-star rule

Require: ***m***: mapping rule
Require: ***M***: set of rules in the mapping document
Require: ***nestLevel***: depth of recursion of the function call
▷ *SM*, *PM*, *OM* and *GM* refer to subject, predicate, object and graph map

```
1  Procedure materializeMappingRule(m, M, nestLevel = 0)
2      if isNonAsserted(m) and nestLevel == 0 then
3          return
4      if isSimpleTermMap(m.SM) then
5          subjects = materializeTermMap(m.SM)
6      else if isStarTermMap(m.SM) then
7          m_parent = getMappingRule(m.SM, M)
8          m_joint = joinMappingRules(m, m_parent)
9          subjects = materializeMappingRule(m_joint, M, nestLevel + 1)
10     if isSimpleTermMap(m.OM) then
11         objects = materializeTermMap(m.OM)
12     else if isRefTermMap(m.OM) then
13         objects = materializeRefTermMap(m.OM, M)
14     else if isStarTermMap(m.OM) then
15         m_parent = getMappingRule(m.OM, M)
16         m_joint = joinMappingRules(m, m_parent)
17         objects = materializeMappingRule(m_joint, M, nestLevel + 1)
18     predicates = materializeTermMap(m.PM)
19     if nestLevel == 0 then
20         if hasGraphMap(m) then
21             namesGraphs = materializeTermMap(m.GM)
22             return createQuads(subjects, predicates, objects, namedGraphs)
23         else
24             return createTriples(subjects, predicates, objects)
25     else if nestLevel > 0 then
26         return createStarTriples(subjects, predicates, objects)
```

Triples map The principal point to consider when processing an RML-star rule is that it resembles a binary tree in which the left and right children are given by the mapping rules referenced by star maps in the subject and object, respectively. The general idea of Algorithm 5 is traversing the tree of mapping rules in post-order: first, the left subtree (given by the star map in the subject) of the current mapping rule, then the right subtree (given by the star map in the object), and finally the current mapping rule is processed for generating the quoted triples. Hereinafter, we refer to the mapping rule at the root of the tree as the *outermost* mapping rule, and the rest as *inner* mapping rules. We use *level of nesting* to refer to the depth of a mapping rule in the tree. In the following, Algorithm 5 is explained in

detail together with a running example using the mapping in Listing 5.4, which contains the `#dateTM` triples map with a star map in the subject pointing to the `#jumpTM` non-asserted triples map.

```
1  <#jumpTM>
2      a rml:NonAssertedTriplesMap;
3      rml:logicalSource :marks;
4      rml:subjectMap [
5          rml:template ":{PERSON}"
6      ];
7      rml:predicateObjectMap [
8          rml:predicate :jumps;
9          rml:objectMap [
10             rml:reference "MARK"
11         ]
12     ];
13
14 <#dateTM>
15     a rml:TriplesMap;
16     rml:logicalSource :marks;
17     rml:subjectMap [
18         rml:quotedTriplesMap <#jumpTM>
19     ];
20     rml:predicateObjectMap [
21         rml:predicate :date;
22         rml:objectMap [
23             rml:reference "DATE"
24         ]
25     ].
```

Listing 5.4: Running example to describe the RML-star materialization procedure. The mapping transforms data in Listing 5.1 with the `#dateTM` triples map having a star map pointing to the `#jumpTM` non-asserted triples map.

Non-asserted triples maps First, non-asserted triples maps must not generate asserted triples (i.e., the triples must not be added to the output RDF-star graph). This entails that the mapping rules within a non-asserted triples map must only be processed when generating quoted triples. Algorithm 5 uses the `nestLevel` parameter to keep track of the level of nesting that is being processed, with 0 referring to the outermost mapping rule. When a mapping rule within a non-asserted triples map is in the outermost level of nesting, it is immediately discarded by Algorithm 5 (*lines 2-3*) as the triples that it generates should not be asserted. If `nestLevel` is not 0, the generated triples will be quoted and the mapping rule should be processed.

Running Example 1.1. To generate the output RDF-star graph, the algorithm is executed twice, once for each triples map in the input mapping document. First, when the algorithm is applied to the `#jumpTM` triples map, the rule is not executed (*lines 2-3*), as the triples map is non-asserted and should only generate quoted triples. The algorithm is then applied to the `#dateTM` triples map which is processed as it is asserted. In this execution, the `#jumpTM` triples map will be later processed, as it is a quoted triples map within the star map in the `#dateTM`

triples map.

Term maps Next, term maps are processed to generate the terms of the triples. There are three types of maps in RML-star that need to be differentiated for materialization: term maps, referencing object maps, and star maps. When evaluating a map, Algorithm 5 checks its type (for instance, for object maps it is checked in *lines 10, 12 & 14*, with *isSimpleTermMap(m.OM)* evaluating to `true` if the object map of a rule is a simple term map) and then processes the term map according to it. The handling of simple and referencing term maps (covered in *lines 5, 11, 13 & 18*) is already considered in R2RML and RML materialization procedures that are well reported in the literature and more details of their materialization can be found in the R2RML Recommendation. When the type of a map resolves to a star map (*lines 6 & 14*), it is managed as described next.

Running Example 1.2. The `#dateTM` rule has two simple term maps in the predicate (`rml:constant :date`) and object (`rml:reference "DATE"`) positions and one star map in the subject position; while `#jumpTM` contains only simple term maps. The simple term maps in `#dateTM` are evaluated after the star map, and the term maps in `#jumpTM` are evaluated in the recursive call of the procedure when evaluating the star map in `#dateTM`.

Star maps Star maps can occur in both the subject and object positions (*lines 6-9 & 14-17* respectively). Before generating the triples, the logical sources involved in the star map (a star map involves two triples maps) must be joined. In this way, the terms for the quoted triples and the annotation triple are generated from the same joint logical source, complying with the provided join condition. To achieve this, the parent mapping rule is retrieved from the set of mapping rules M (*lines 7 & 15*), and the logical sources of both triples maps are merged into a joint logical source (*lines 8 & 16*). When the logical sources of the triples maps are the same and no join condition is provided *lines 8 & 16* have no effect and any of the original logical sources (child or parent) can be used as the joint logical source. As star maps entail nested rules, processors should deal with any level of nesting. Considering the recursive nature of RML-star, the materialization of RDF-star graphs must also be implemented recursively. Algorithm 5 recursively calls `materializeMappingRule()` (*lines 9 & 17*) passing the joint mapping rule (i.e., with the joint logical source) and increasing `nestLevel`, as a deeper level of nesting will be processed. In this way, the triples generated by the inner mapping rule will be quoted in the subject or object of the triples generated by the mapping rule at the current level of nesting.

Running Example 1.3. In `#dateTM`, the subject map was previously evaluated as a star map in *line 6*. The mapping rule given by `#jumpTM` is retrieved as the parent rule in *line 7*. The join between the rules (*line 8*) has no effect as both of them have the same logical source, and the logical source of the parent is used as the joint logical source. Last, the quoted subject terms are generated in a recursive call of the procedure passing the joint mapping rule as an argument (*line 9*).

Combining subject, predicate and object terms to generate the output triples is done in a

similar way than in RML-Core. This combination is done row-wise, one triple is generated for each row in the (joint) logical source of the mapping rule.

Graph maps Finally, the generation of quads must be considered. In RDF-star, quads are never quoted. However, in RML-star, triples maps are not restricted from having a graph map (i.e., inner mapping rules can also have a graph map). To prevent the generation of *quoted quads* in RML-star, graph maps must only be processed in the outermost mapping rule (i.e., the level of nesting in which triples or quads are asserted) and ignored otherwise. *Lines 19-22* of Algorithm 5 process graph maps when `nestLevel` is 0. If the outermost mapping rule does not have a graph map, RDF-star triples are added to the default graph of the output dataset (*lines 23-24*). When processing an inner mapping rule, the generated triples must be quoted (*lines 25-26*), i.e., enclosed with "«" and "»".

Running Example 1.4. In the case of `#dateTM`, the level of nesting is 0 (*line 19*) and it has no graph map (*line 20*), so plain triples are created (*line 24*). When processing `#jumpTM` as a result of the aforementioned recursive call, the level of nesting is 1 (*line 25*), and star triples will be created (*line 26*) enclosed by "«" and "»".

5.4 Validation

We implemented Algorithm 5 in Morph-KGC. To validate the algorithm and the implementation, we created a set of RML-star test cases and a benchmark that are described below.

5.4.1 Test cases

We created a representative set of test cases for RML-star by using the N-Triples-star syntax tests[4] as a reference. For each N-Triples-star test case, we created two associated ones for RML-star that generate the original RDF-star dataset: one test case with a single input data source (i.e., the mapping does not include joins) and another with two input data sources (i.e., the mapping includes joins among triple maps). For each test case, we manually created the input source(s) in CSV format and the corresponding RML-star mappings. We obtained 16 test cases that cover corner cases such as deeply nested star maps in the subject and object position. Listing 5.5 shows the RML-star mapping for test case RMLSTARTC008b, which features multiple levels of star map nesting. The test cases are openly available in Zenodo [36], and can be reused by any engine to test its conformance with respect to RML-star. Morph-KGC successfully passes all the test cases.

```
1 @prefix rml: <http://w3id.org/rml/> .
2 @prefix ex: <http://example/> .
3 @prefix : <http://example.org/> .
4 @base <http://example.org/> .
5
6 :elementaryTM1 a rml:NonAssertedTriplesMap ;
```

[4] https://w3c.github.io/rdf-star/tests/nt/syntax

```
7     rml:logicalSource [
8         rml:source "test/rml-star/RMLSTARTC008b/data1.csv";
9         rml:referenceFormulation rml:CSV
10    ];
11    rml:subjectMap [
12        rml:template "http://example/{c1-1}"
13    ];
14    rml:predicateObjectMap [
15        rml:predicate ex:p1 ;
16        rml:objectMap [
17            rml:template "http://example/{c1-2}"
18        ]
19    ] .
20
21 :firstJoinTM a rml:NonAssertedTriplesMap ;
22    rml:logicalSource [
23        rml:source "test/rml-star/RMLSTARTC008b/data1.csv";
24        rml:referenceFormulation rml:CSV
25    ];
26    rml:subjectMap [
27            rml:quotedTriplesMap :elementaryTM1
28    ];
29    rml:predicateObjectMap [
30        rml:predicate ex:q1; ;
31        rml:objectMap [
32            rml:quotedTriplesMap :elementaryTM2
33        ]
34    ] .
35
36 :elementaryTM2 a rml:NonAssertedTriplesMap ;
37    rml:logicalSource [
38        rml:source "test/rml-star/RMLSTARTC008b/data1.csv";
39        rml:referenceFormulation rml:CSV
40    ];
41    rml:subjectMap [
42        rml:template "http://example/{c1-3}"
43    ];
44    rml:predicateObjectMap [
45        rml:predicate ex:p2 ;
46        rml:objectMap [
47            rml:template "http://example/{c1-4}"
48        ]
49    ] .
50
51 :centralJoinTM a rml:AssertedTriplesMap ;
52    rml:logicalSource [
53        rml:source "test/rml-star/RMLSTARTC008b/data2.csv";
54        rml:referenceFormulation rml:CSV
55    ];
56    rml:subjectMap [
57            rml:quotedTriplesMap :firstJoinTM ;
58            rml:joinCondition [
59              rml:child "c2-5" ;
60              rml:parent "c1-5" ;
```

```
61              ];
62      ];
63      rml:predicateObjectMap [
64          rml:predicate ex:q2;
65          rml:objectMap [
66              rml:quotedTriplesMap :secondJoinTM
67          ]
68      ] .
69
70  :elementaryTM3 a rml:NonAssertedTriplesMap ;
71      rml:logicalSource [
72          rml:source "test/rml-star/RMLSTARTC008b/data2.csv";
73          rml:referenceFormulation rml:CSV
74      ];
75      rml:subjectMap [
76          rml:template "http://example/{c2-1}"
77      ];
78      rml:predicateObjectMap [
79          rml:predicate ex:p3 ;
80          rml:objectMap [
81              rml:template "http://example/{c2-2}"
82          ]
83      ] .
84
85  :secondJoinTM a rml:NonAssertedTriplesMap ;
86      rml:logicalSource [
87          rml:source "test/rml-star/RMLSTARTC008b/data2.csv";
88          rml:referenceFormulation rml:CSV
89      ];
90      rml:subjectMap [
91              rml:quotedTriplesMap :elementaryTM3
92      ];
93      rml:predicateObjectMap [
94          rml:predicate ex:q3; ;
95          rml:objectMap [
96              rml:quotedTriplesMap :elementaryTM4
97          ]
98      ] .
99
100 :elementaryTM4 a rml:NonAssertedTriplesMap ;
101     rml:logicalSource [
102         rml:source "test/rml-star/RMLSTARTC008b/data2.csv";
103         rml:referenceFormulation rml:CSV
104     ];
105     rml:subjectMap [
106         rml:template "http://example/{c2-3}"
107     ];
108     rml:predicateObjectMap [
109         rml:predicate ex:p4 ;
110         rml:objectMap [
111             rml:template "http://example/{c2-4}"
112         ]
113     ] .
```

Listing 5.5: RML-star mapping for test case RMLSTARTC008b.

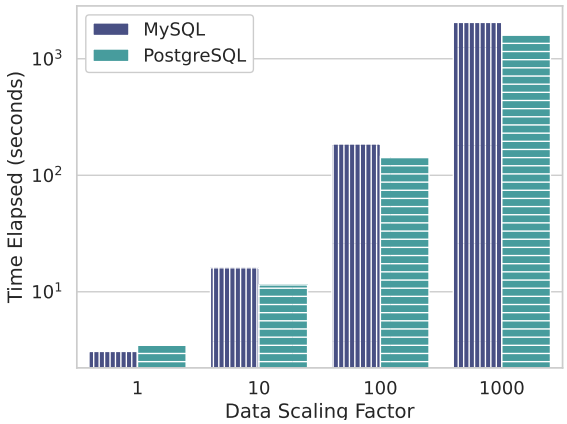


Figure 5.2: Execution times for LUBMM4OBDA benchmark with RML-star mappings using Morph-KGC and different configurations. Times are reported using a logarithmic scale.

5.4.2 Performance with the LUBM4OBDA Benchmark

In Section 4.3 we introduced the LUBM4OBDA benchmark that was used to evaluate our `ITT` architecture. The benchmark also allows to evaluate the creation of statement-level metadata with RML. Among other meta knowledge approaches, LUBM4OBDA considers the creation of RDF-star graphs with RML-star. Specifically, the benchmark annotates undergraduate degrees in the LUBM dataset with the year in which they were awarded.

We used LUBM4OBDA to evaluate the performance of our implementation using Morph-KGC v2.8.1 with MySQL v5.7 and PostgreSQL v14.17. We used SFs 1, 10, 100 and 1.000, with the latter comprising more than 150 million triples. Figure 5.2 shows the results. The materializations for all scaling factors were obtained in less than 1 hour, showing that large RDF-star graphs can be generated. The experiment was conducted on a system equipped with an Intel® Core™ i7-1165G7 (2.80GHz) and a memory of 40 GB RAM DDR4 (3200 MHz) utilizing mapping partitioning[5] and sequential processing.

5.5 Summary

In this chapter, we tackled the problem of RDF-star generation with declarative mappings. We introduced the RML-star extension of RML, which incorporates two new constructs for creating quoted triples: the star map and the non-asserted triples map. We devised an algorithm for materializing RDF-star graphs and implemented it in Morph-KGC. We validated our approach using test cases derived from the N-Triples-star syntax tests which cover complex scenarios with deeply nested quoted triples. Finally, we evaluated the performance of our implementation with the LUBM4OBDA benchmark, generating RDF-graphs of more than 150 million triples. RML-star is now part of the RML ecosystem [81] together with the

[5]The mapping partitioning algorithms presented in Chapter 3 can be applied to RML-star mappings by assigning an empty invariant to the star maps.

test cases, which are maintained by the W3C Knowledge Graph Construction Community Group.

Mapping partitioning can be applied to RML-star, albeit with some limitations. Partitioning based on term and literal types is possible, as well as partitioning by invariant for non-star maps. Since we did not define invariants for star maps, these are assigned an empty invariant, potentially resulting in fewer mapping groups. The partitioning of predicate and graph maps is not affected, as they cannot contain star maps.

We concentrated on materialization with RML-star, but did not address virtualization. To extend the `ITT` architecture presented in Chapter 4, further work is required on RDF-star graph data management. Specifically, triple tables may need to be extended to store RDF-star data, along with the unfolding of SPARQL-star queries into SQL over the extended data structure.

Automating the extraction of declarative mappings is particularly interesting because it is a time-consuming task. Although bootstrapping has traditionally focused on relational databases, edge properties in property graphs can now be conveniently encoded with RDF-star [92]. In the next chapter, we address the automatic extraction of RML-star mappings from property graphs, which contributes to the interoperability of both graph models.

Chapter 6

Mapping Bootstrapping from Property Graphs

There are two prevalent models for graph data management: RDF and property graphs (PGs). The interoperability between the two models is currently the focus of ongoing research [92]. One of the proposals that bring RDF closer to PGs is RDF-star [66], which we studied in the previous chapter. Specifically, PGs allow to assign properties to nodes and edges, but edge properties in RDF (i.e., statement-level metadata) are intricate [92]. RDF-star reconciles both graph data models with quoted triples, which is a more natural representation of edge properties in RDF.

Direct mappings between PGs and RDF-star [66] were proposed. However, transforming PGs to RDF-star typically involves using a domain ontology or vocabulary, which requires customized mappings. An established approach to express customized mappings is declarative languages. In Chapter 5 we proposed RML-star to declare the transformations from diverse data, including PGs, to RDF-star graphs organized in a domain vocabulary.

The flexibility of declarative mapping languages such as RML-star comes at the cost of time-consuming mapping development [83]. To mitigate this, a solution is to write mappings in a semi-automatic way. First, an initial version of the mapping is obtained with automatic extraction (i.e., bootstrap) from a source database; the generated mappings reflect the behavior of a direct mapping [104]. Then, this mapping is manually edited to structure it according to a domain vocabulary. Mapping bootstrapping from relational databases to RDF has been widely studied [33, 104, 83]. However, there are no works on the automatic extraction of declarative mappings from PGs to RDF-star, which we address in this chapter.

6.1 Related Work

6.1.1 Mapping bootstrapping from relational databases

Prior to the publication of the R2RML standard, D2RQ [27] already provided a tool to automatically generate a default mapping by analyzing the schema of a relational database.

The inception of R2RML motivated further works. BootOX [83] and MIRROR [104] generate R2RML mappings that produce RDF graphs according to the W3C Direct Mapping Recommendation [7]. Under some assumptions such as schema normalization, these systems additionally exploit relational patterns to derive further implicit information like subclass-of relationships. Calvanese et al. [33] recently contributed the most comprehensive and up-to-date mapping patterns catalog. The fact that our work is on graph-to-graph instead of relational-to-graph mapping facilitates preserving the structure of the source PG.

6.1.2 Property graph to RDF

Tomaszuk et al. [125] proposed the PGO ontology to describe PGs and an algorithm to transform them into RDF using PGO, without customized mappings. ProGOMap [54] automatically generates declarative mappings from a PG to an ontology. However, PGO and ProGOMap do not consider RDF-star as the work presented in this chapter. PRSC contexts [29] are user-defined mappings from PGs to RDF-star, but templated terms are not supported as in RML-star, and the automatic extraction of the PRSC contexts was not addressed. Hartig [66] provides a direct mapping from PGs to RDF-star which we reuse in our bootstrapping approach. There are also works in the opposite direction, from RDF to PG. Angles et al. [6] provides direct mappings that do not consider customized transformations, and G2GML [40] is a declarative mapping language from RDF to PG, but both proposals omit RDF-star. Abuoda et al. [2] explored methods for converting RDF-star into PGs; however, they do not consider customized mappings.

6.2 Illustrative Example

Figure 6.1 shows a PG (stored in the Kùzu graph database [138]) which consists of four nodes with the label `User`. Edge properties define the `Follows` relationship between users, for example, in the context of a social network. There are two node properties, `name` and `age`, and one edge property, `since`.

Listing 6.1 depicts the bootstrapped RML-star mappings[1] from the PG in the YARRRML [75] syntax. The mapping uses the Cypher[2] query language [43] to access the Kùzu database. The first set of rules in the `User` triples map transforms the nodes (`User`) into resources and node properties (`name` and `age`) into datatype properties. The `User_Follows_User` triples map transforms edges (`Follows`) into object properties. Finally, the `User_Follows_User_quoted` triples map transforms edge properties (`since`) into subject quoted triples.

Listing 6.2 shows the generated RDF-star graph. The triples in *lines 5, 7, 9* and *11* are generated by the `User` triples map; the triples in *lines 6, 8* and *10* by the `User_Follows_User` triples map; and the triples in *lines 12-15* by the `User_Follows_User_quoted` triples map.

```
1 mappings:
```

[1] It must be noted that here the generated triples maps contain multiple predicate-object maps (i.e., mappings are not normalized). This is because from a user perspective having rules grouped in triples maps facilitates mapping manipulation.

[2] Kùzu uses the Cypher query language, but any PG query language is valid.

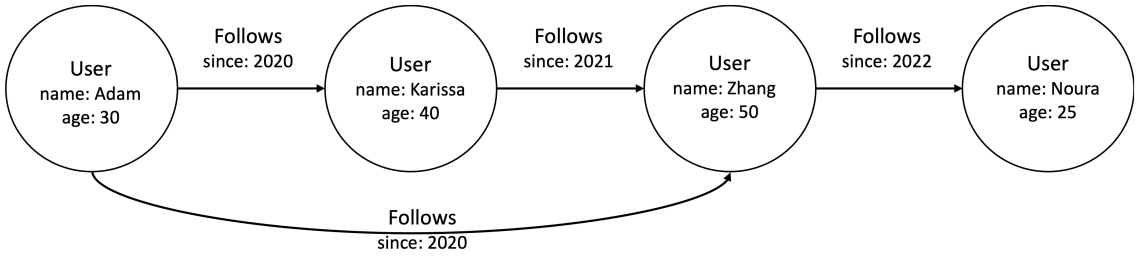


Figure 6.1: Source PG based on an example from the Kùzu documentation.

```
2
3   User:
4     sources:
5       query: MATCH (x:User) RETURN DISTINCT OFFSET(ID(x)) AS x_User_id, x.
     name AS name, x.age AS age
6       referenceFormulation: cypher
7     subject: ns:User/$(x_User_id)
8     predicateobjects:
9     - [rdf:type, ns:User]
10    - [ns:name, $(name)]
11    - [ns:age, $(age), xsd:integer]
12
13  User_Follows_User:
14    sources:
15      query: MATCH (x:User)-[r:Follows]->(y:User) RETURN DISTINCT OFFSET(
     ID(x)) AS x_User_id, OFFSET(ID(y)) AS y_User_id, r.since AS since
16      referenceFormulation: cypher
17    subject: ns:User/$(x_User_id)
18    predicateobjects:
19    - [ns:Follows, ns:User/$(y_User_id)]
20
21  User_Follows_User_quoted:
22    sources:
23      query: MATCH (x:User)-[r:Follows]->(y:User) RETURN DISTINCT OFFSET(
     ID(x)) AS x_User_id, OFFSET(ID(y)) AS y_User_id, r.since AS since
24      referenceFormulation: cypher
25    subject:
26      quoted: User_Follows_User
27    predicateobjects:
28    - [ns:since, $(since), xsd:integer]
```

Listing 6.1: Automatically extacted RML-star mapping in YARRRML syntax.

```
@base <http://example.com/ns#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<#User/0> rdf:type <#User>; <#name> "Adam"; <#age> "30"^^xsd:integer;
          <#Follows> <#User/1>, <#User/2> .
<#User/1> rdf:type <#User>; <#name> "Karissa"; <#age> "40"^^xsd:integer;
          <#Follows> <#User/2> .
<#User/2> rdf:type <#User>; <#name> "Zhang"; <#age> "50"^^xsd:integer;
          <#Follows> <#User/3> .
<#User/3> rdf:type <#User>; <#name> "Noura"; <#age> "25"^^xsd:integer .
<< <#User/0> <#Follows> <#User/1> >> <#since> "2020"^^xsd:integer .
<< <#User/0> <#Follows> <#User/2> >> <#since> "2020"^^xsd:integer .
<< <#User/1> <#Follows> <#User/2> >> <#since> "2021"^^xsd:integer .
<< <#User/2> <#Follows> <#User/3> >> <#since> "2022"^^xsd:integer .
```

Listing 6.2: Generated RDF-star graph with the bootstrapped mappings.

6.3 Automatic Mapping Extraction

The bootstrapping process consists of three types of mapping rule extraction, which are described below. The description of the steps are linked to the corresponding triples maps of the RML-star mapping of the illustrative example (Listing 6.1).

Node to resource and **node property to datatype property** This extraction generates the `User` triples map. First, find all node labels (by posing queries to the PG); then, for each node label, find their property labels (with queries to the PG as well) and create a triples map as follows:

1. The logical source is a PG query that uses the node label and node property labels to retrieve the ids of the nodes together with the node property values. The reference formulation is the PG query language of the source database.
2. The subject map is a template-valued term map with the node label (constant) and the node ids (reference to the logical source).
3. Add a predicate-object map to create the resource, with `rdf:type` as the predicate value and the node label as the object value.
4. For each node property add a predicate-object map to create the associated datatype property. The predicate is the node property label and the object is the associated reference in the PG query, optionally accompanied by its XSD datatype (this is similar to the natural mapping of SQL values [44]).

Edge to object property This extraction generates the `User_Follows_User` triples map. For each node label found in the former step, use it as the source node label to get associated edge labels and target node labels and create a triples map as follows:

1. The logical source is a PG query that uses the edge label and the source and target

node labels to retrieve the ids of the source and target nodes together with the edge properties.

2. The subject map is a template with the source node label (constant) and the source node ids (reference to the logical source).
3. Add a predicate-object map to create the object property. The predicate is the edge label and the object is a template with the target node label (constant) and the target node ids (reference to the logical source).

Edge property to quoted triples This extraction generates the `User_Follows_User_-quoted` triples map. For each edge found in the previous step, create a triples map as follows (if it has edge properties):

1. The logical source is the same as the one in the associated *edge to object property* triples map.
2. The subject is a star map that references the associated *edge to object property* triples map, which produces the quoted triples. It is not necessary to specify a join condition since the logical sources are the same.
3. For each edge property add a predicate-object map to annotate the quoted triples. The predicate is the edge property label (constant) and the object is the associated reference in the PG query, optionally accompanied by its XSD datatype.

6.4 Implementation and Validation

6.4.1 Implementation

We implemented the automatic extraction of RML-star in Python for the Neo4j [57] and Kùzu graph databases. The syntax of the bootstrapped mappings is YARRRML, as it is more user-friendly than the turtle-based syntax of RML and the idea is to facilitate editing the mappings. To be able to use the bootstrapped mappings, we also implemented connectors for Neo4j and Kùzu in Morph-KGC.

6.4.2 Validation on the LDBC Social Network Benchmark

We employed the LDBC Social Network Benchmark [122] to validate the approach. The benchmark comes with a dataset generator that creates a social network graph which can be scaled in size. We used the dataset with scaling factor 1, loaded it into Kùzu, and extracted the RML-star mappings with our implementation. We then used the generated mappings with Morph-KGC to create the RDF-star graph and manually validated its correctness. The resulting mapping file is composed of 37 triples maps that are broken down as follows:

- 8 TMs correspond with the *node to resource* and *node property to datatype property* extraction.
- 23 TMs correspond with the *edge to object property* extraction.

- 6 TMs correspond with the *edge property to quoted triples* extraction.

Listing 6.3 shows three of the triples maps obtained. Specifically it contains the `Person` TM obtained from the homologous PG node, the `Person_person_likes_comment_Comment` TM obtained from an edge of that node, and finally the `Person_person_likes_comment_Comment_quoted` TM corresponding with the creation date property of the previous edge. The complete bootstrapped mapping is available in GitHub.[3]

```
prefixes:
  rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
  xsd: http://www.w3.org/2001/XMLSchema#
  ns: http://example.com/ns#

mappings:
  Person:
    sources:
      query: MATCH (x:Person) RETURN DISTINCT OFFSET(ID(x)) AS x_Person_id
    , x.id AS id, x.firstName AS firstName, x.lastName AS lastName, x.
    gender AS gender, x.birthday AS birthday, x.creationDate AS
    creationDate, x.locationIP AS locationIP, x.browserUsed AS browserUsed
      referenceFormulation: cypher
    subject: ns:Person/$(x_Person_id)
    predicateobjects:
    - predicates: rdf:type
      objects:
        value: ns:Person
        type: iri
    - predicates: ns:id
      objects:
        value: $(id)
        datatype: xsd:integer
    - predicates: ns:firstName
      objects:
        value: $(firstName)
    - predicates: ns:lastName
      objects:
        value: $(lastName)
    - predicates: ns:gender
      objects:
        value: $(gender)
    - predicates: ns:birthday
      objects:
        value: $(birthday)
    - predicates: ns:creationDate
      objects:
        value: $(creationDate)
        datatype: xsd:integer
    - predicates: ns:locationIP
      objects:
        value: $(locationIP)
    - predicates: ns:browserUsed
      objects:
```

[3] https://github.com/arenas-guerrero-julian/pg2rml-star/tree/main/test

```
42          value: $(browserUsed)
43
44    Person_person_likes_comment_Comment:
45      sources:
46        query: MATCH (x:Person)-[r:person_likes_comment]->(y:Comment) RETURN
        DISTINCT OFFSET(ID(x)) AS x_Person_id, OFFSET(ID(y)) AS y_Comment_id,
       r.creationDate AS creationDate
47        referenceFormulation: cypher
48      subject: ns:Person/$(x_Person_id)
49      predicateobjects:
50      - predicates: ns:person_likes_comment
51        objects:
52          value: ns:Comment/$(y_Comment_id)
53          type: iri
54
55    Person_person_likes_comment_Comment_quoted:
56      sources:
57        query: MATCH (x:Person)-[r:person_likes_comment]->(y:Comment) RETURN
        DISTINCT OFFSET(ID(x)) AS x_Person_id, OFFSET(ID(y)) AS y_Comment_id,
       r.creationDate AS creationDate
58        referenceFormulation: cypher
59      subject:
60        quoted: Person_person_likes_comment_Comment
61      predicateobjects:
62      - predicates: ns:creationDate
63        objects:
64          value: $(creationDate)
65          datatype: xsd:integer
```

Listing 6.3: Mapping with three of the triples maps bootstrapped from the LDBC Social Network Benchmark. Each triples map corresponds with a different extraction type described in Section 6.3.

6.5 Summary

The automatic extraction of declarative mappings was studied mainly for relational databases but not for other data models. RDF-star was proposed to bridge the gap between RDF and property graphs, making both data models more interoperable, as edge properties can be conveniently encoded with quoted triples. In the previous chapter, we introduced the RML-star mapping language to declaratively generate knowledge graphs. In this chapter, we extended the proposal to automatically extract RML-star mappings from property graphs.

The work presented here reduces the human effort in mapping development, which is a time-consuming and error-prone task. This advances in the automation and efficiency of mapping development. This automation can lead to more efficient and accurate generation of knowledge graphs. When necessary, the bootstrapped mappings can be adapted to generate custom knowledge graphs according to a target vocabulary. We implemented RML-star mapping bootstrapping for the Neo4j and Kùzu databases, and validated it with the LDBC Social Network Benchmark, showing the feasiblility of the proposal.

Chapter 7

Comprehensive Processing of Data Transformations

Data integration use cases normally involve messy data that need to be cleaned to ensure the quality of the KG exposed by the mappings. Apart from basic data transformations, aggregations, filtering, or custom procedures with user-defined functions might also be needed. Figure 7.1 motivates data transformations to create a KG of materials from a data source in German. It first shows a KG built without data transformations that contains errors due to the lack of data cleaning. In contrast, in the KG built with data transformations, data is not only clean, but it is also more complete, since a natural language translation function was applied to also include the names of the materials in English.

Computations beyond schema transformations were already devised by R2RML. A relational database is accessed in R2RML by referring to a base table or using an SQL query. The former is the basic alternative, since it is not necessary to be familiar with SQL; however, it does not allow computations (e.g., converting a column to lowercase). The latter allows to access the RDB using an SQL query in which data transformations can be declared. In this way, the execution of the transformations is pushed down to the underlying database. The construct for defining SQL queries within a mapping rule is known as an R2RML view.

RML, similar to R2RML, also supports views. Although views in RML solve data transformations for RDBs, they present limitations for diverse sources. In this chapter, we propose solutions to challenges regarding data transformations in RML. Specifically, we first extend RML views to tabular files (e.g., CSV or Apache Parquet), which do not only allow to execute data transformations, but also solve limitations of RML for tabular data such as complex joins or composite data values. Next, we incorporate Python user-defined functions into RML, which empowers the declarative construction of KGs with the flexibility of an interpreted programming language. Finally, we address the problem of executing data transformations defined in mappings with the RML-FNML module by VKG systems based on query translation. Our approach unfolds the data transformation into RML views, which can then be transparently executed by query translation VKG systems.

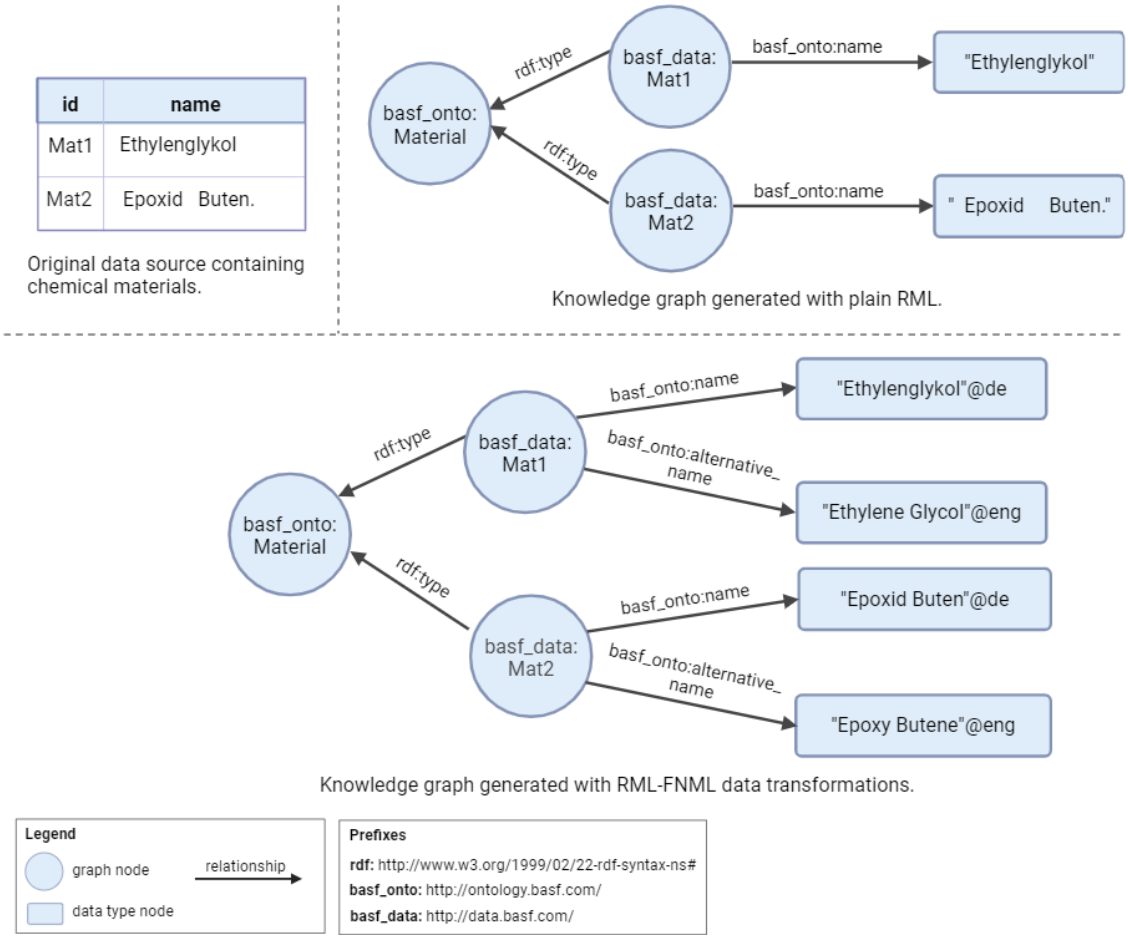


Figure 7.1: Example of KGs built from a data source of chemical materials (upper left). The first KG (upper right) was built without applying data transformations. The second KG (bottom) was constructed by applying a set of RML-FNML functions resulting in a refined KG with clean material names (without blank spaces and special characters) and with material names also in English.

7.1 Related Work

7.1.1 Data Transformation in Declarative Mappings

The SPARQL query language has been extended in several works, such as SPARQL-Anything [19], SPARQL-Generate [94] or Tarql [124], to generate RDF KGs from tabular data. Similarly to SQL in RML views, SPARQL functions allow data transformation using the GENERATE clause in SPARQL-Generate and the CONSTRUCT query form in SPARQL-Anything (by overloading the SERVICE clause) and Tarql (via the FROM clause). Complex joins are enabled through nested GENERATE and CONSTRUCT clauses, but mixed content is not supported. The main difference with respect to RML views is that SPARQL-based approaches use a query language over the target ontology, while RML views use a query language over the tabular sources (these approaches are known as local- and global-as-view respectively, see Section 2.1). Semantic Web practitioners may prefer SPARQL based alternatives, since they

are familiar with this query language, while data engineers who are used to SQL may lean towards RML views.

García-González et al. [59] proposed using Shape Expresions to map heterogeneous data to RDF. The Shape Expressions Mapping Language (ShExML) relies on a data validation language instead of a query language. ShExML is more limited regarding transformation functions, for which only matchers and string operations are supported, and filtering or mixed content is not possible. Also, limited join functionality can be achieved with shape linking and the JOIN clause.

Szekely et al. [123] proposed the T2WML language to map tabular data for layouts beyond the canonical representation (one column for each variable). T2WML maps on a cell-centric basis rather than the row-centric model of RML's base source. Although T2WML allows transformation functions, it does not support joins between different tabular sources or mixed content. YAML is used to write T2WML rules, similar to YARRRML, a popular human-readable serialization of RML.

Several mechanism extending RML have been proposed to increase the flexibility of the mapping language. RML has been aligned to FnO [46] and FunUL [42], which define functions in a generic and reusable way. While these approaches define functions within term maps, RML views define them directly in the logical source. RML fields introduce a nested iteration model to handle mixed-content, and the work presented in [105] proposes the concatenation of path expressions using the *mixed-syntax paths* constructs. RML+FnO and RML fields are now under the hood of the W3C Knowledge Graph Construction Community Group.

7.1.2 RML-FNML

RML-FNML is the consolidation of the RML+FnO proposal to declare data transformations in RML, and it stands out for its full integration in the RML ecosystem as a module [81]. Figure 7.2 depicts an overview of RML-FNML. The evaluation of a transformation function in RML-FNML is defined via a *function execution*, with the function given by a *function map*. An *input* links an FnO parameter (*parameter map*) to some input value (*input value map*).

7.2 RML Tabular Views

Views in RML are limited to RDBs and do not cover tabular data sources, which are restricted to RML's base source [50]. This reduces the capabilities of the mapping language and led to a number of proposals extending RML using additional constructs. In this section, we first analyze the limitations of RML extensions to handle tabular data. Then we propose to address them by extending RML views to tabular files. Finally, we implement RML tabular views on Morph-KGC and validate them with test cases and the LUBM4OBDA benchmark.

7.2.1 Limitations of RML for Tabular Data Sources

In the following, we analyze the limitations of RML for tabular data along with some extensions in the literature that address them.

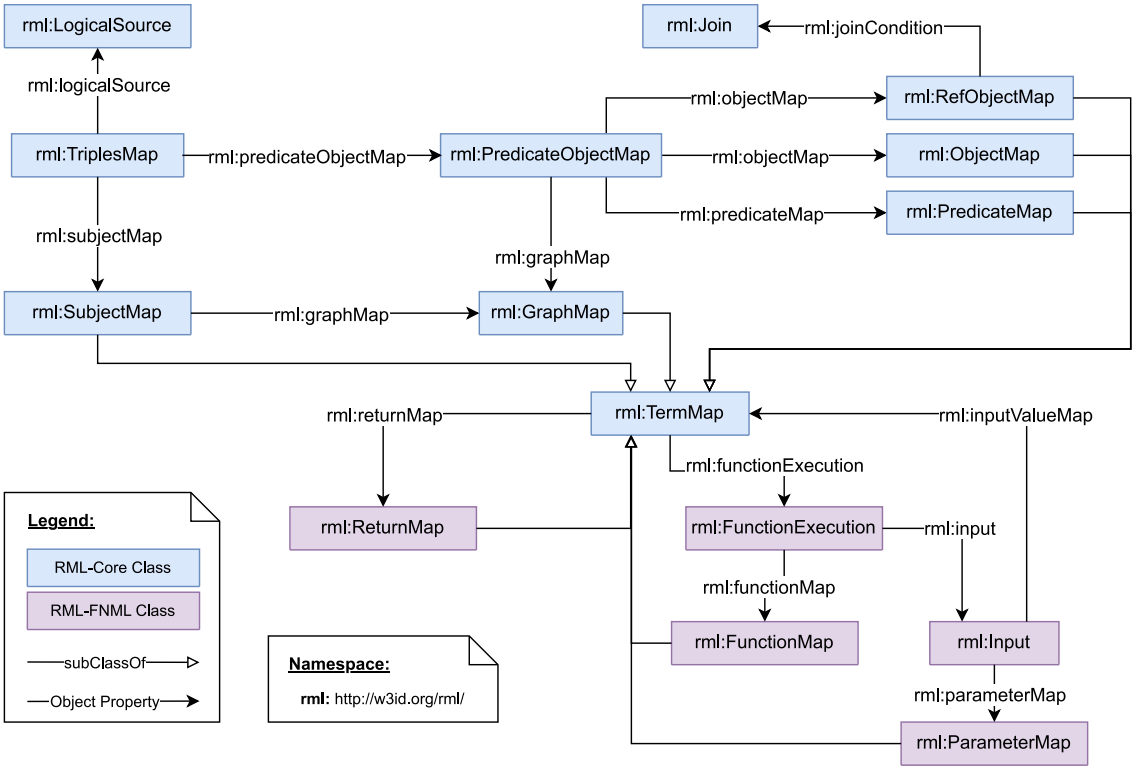


Figure 7.2: RML-Core and RML-FNML modules represented in the Chowlk Visual Notation [55].

Transformation Functions [46] Transformations of the data need to be defined in the mapping to handle data cleaning and computations such as reshaping, aggregating, or filtering. RML's base source uses the data source as is, without additional modifications. Projections of the source may still be computed using the references in the term maps to avoid processing unnecessary data [85]. To allow declarative transformation functions in mappings, RML has been extended with additional constructs such as RML-FNML.

Joins RML is restricted regarding join operations over tabular sources. Referencing object maps join two triples maps (with their associated tabular base sources) for which join conditions can be defined. Some of the limitations of referencing object maps are:

- *Multiple joins.* A referencing object map involves two triples maps, consequently, RML does not support joining three or more tabular sources. To the best of our knowledge, no specific solution addressed this. A workaround to enable multiple joins is to create a relational schema for the tabular sources, load the data to an RDBMS and use an RML mapping with views to encode the joins in SQL queries. This implies increased complexity, due to the cost of defining the SQL schema, the addition of an RDBMS to the data integration pipeline, and the overhead of loading the data to it.
- *Theta joins* [87]. The class of join conditions in RML (i.e., `rml:Join`) only allows for equality conditions. This is shown in the R2RML Recommendation, where the *joint*

SQL query[1] resulting from a referencing object map is an equijoin. However, some data integration scenarios require theta joins (or inequality joins). We are not aware of specific proposals tackling this limitation, for which the workaround described for multiple joins could be used.

- *Literal generation with joins* [47]. Referencing object maps use the subject map of parent triples maps to generate the object RDF terms of the output triples. Given that the term type of a subject map cannot be a literal (enforced by the RDF data model), it is not possible in RML to generate literals with RML's base source. Recently, Debruyne [47] proposed an extension of RML enabling the generation of literals with joins.

Mixed Content [105] Tabular sources in real data integration use cases usually present composite data values: values such as JSON or lists are embedded in cells. This has been referred to as *mixed content* [105]. RML does not allow for mixed content, although solutions such as *fields* [49] or *mixed-syntax paths* [105] addressed this limitation.

7.2.2 RML Views over Tabular Data

The approach of R2RML to solve the limitations above is R2RML views, which uses the SQL query language to push down computations to RDBMSs. It must be noted that R2RML views are different from *materialized views* [60]; the former is an SQL query that is executed once and whose results are not persisted in the RDBMS, the latter is a table in the database resulting from the execution of a query for which refreshing policies apply (incremental, at regular time intervals, or on demand).

The RML specification [50] currently limits the scope of RML views to databases (the `rr:R2RMLView` class is not extended). The seminal work of RML [51] already devised that logical sources could be extended to support views over other data sources to allow data cleaning and transformation. However, views have not yet been considered for tabular sources.

We extend RML views to tabular data, which address the limitations of RML's base source. An RML view over tabular sources is a logical table populated with data resulting from the execution of an SQL query against the input tabular sources. It is represented by a resource with:

- One `rml:query` property (which extends R2RML's `rr:sqlQuery`), whose value is a literal with a lexical form that is a valid SELECT SQL query. The query result set cannot have duplicate column names, and projected columns resulting from an expression (e.g., aggregates) must be aliased to allow for referenciation from term maps.
- Zero or one `rml:referenceFormulation` property. Because of backwards compatibility with R2RML the property `rr:sqlVersion` can also be used. RML predefines `ql:CSV` to refer to CSV files using columns. In the case of RML views, the default is `rr:SQL2008` but others could be used.[2] This reference formulation applies to CSV and other tabular

[1] https://www.w3.org/TR/r2rml/#dfn-joint-sql-query
[2] https://www.w3.org/2001/sw/wiki/RDB2RDF/SQL_Version_IRIs

data formats such as Apache Parquet.

- Zero or one `rml:iterator` property. This is optional, since the default per-row iteration is assumed.

R2RML processors require an SQL connection[3] to access the input database. RML views over tabular sources do not need this connection, being the input sources directly referenced in the SQL query (using absolute or relative paths to the tabular files in the system or a URL for a remote file), which can be conveniently aliased. Since tabular sources are referenced as they are (in an RDBMS two tables with the same name can coexist in different databases), a default catalog and schema are used.

Tabular views are illustrated in Listing 7.1 which maps the CSV data in Listing 5.1. In the example, an SQL query with a filter is used to select the scores higher than a threshold to generate triples with the persons that obtained a high score.

```
<#TMTV1>
    rml:logicalSource [
        rml:query """
                    SELECT PERSON
                    FROM 'path/to/marks.csv'
                    WHERE SCORE > 1220
        """
    ];
    rml:subjectMap [
        rml:template ":{PERSON}"
    ];
    rml:predicateObjectMap [
        rml:predicate :hasHighScore;
        rml:objectMap [
            rml:constant "true";
            rml:datatype xsd:boolean
        ]
    ].
```

Listing 7.1: Example of an RML tabular view that accesses the CSV in Listing 5.1. The SQL query contains a filter to obtain scores higher than 1220 and generate triples with persons with a high score.

7.2.3 Implementation

We implemented RML tabular views in Morph-KGC by integrating with DuckDB. When an RML tabular view is provided in the mappings, the execution of the SQL query is delegated to DuckDB, and the query result is converted to a Pandas DataFrame, just as with any other data source.

RML tabular views allow pushing down some operations in the mappings, such as duplicate removal (using the DISTINCT clause), NULL elimination (IS NOT NULL statement) or joins (replacing referencing object maps) that can improve its performance. The flexibility of views

[3] https://www.w3.org/TR/r2rml/#dfn-sql-connection

also enables the creation of identifiers when they are missing from tabular sources[4] and can potentially solve more limitations of base RML that may arise.

It must be noted that RML-FNML prevents good mapping partitioning (i.e., obtaining a mapping partition with a high number of groups), while RML views do not affect the partitioning. This is because to obtain a mapping partition, the constant part of term maps (in `rr:constant` and `rr:template`) is used; however, for function maps, it is not possible to make assumptions of constant parts of the generated RDF terms to retrieve the invariants. By encoding transformation functions in RML views, we avoid the need of function maps in Morph-KGC, thus obtaining better mapping partitions and consequently, ensuring scalability.

7.2.4 Testing

Test cases The R2RML test cases [130] are a companion of the R2RML Recommendation to validate the compliance of systems with respect to the mapping language. Later, they were extended to RML [74]. However, given the lack of RML views for tabular data sources at that time, test cases with R2RML views were only considered for RDBs and excluded for the CSV data format. In order to validate the compliance of tabular views in Morph-KGC, we extended these R2RML test cases to RML views. Table 7.1 lists them along with a description and an alternative solution using RML's base source with additional constructs. It must be noted that the test cases RMLTVTC0015a, RMLTVTC0015b and RMLTVTC0019a were included in the RML test cases for CSV [74], but the tabular files were preprocessed to enable RML's base source. Here, we maintain the original structure of the data in the R2RML test cases. We also created four additional test cases (RMLTVTC0026a, RMLTVTC0027a, RMLTVTC0028a, RMLTVTC0029a) to further validate some of the limitations previously described in this section. Morph-KGC successfully passes all test cases that are openly available in Zenodo [9].

LUBM4OBDA benchmark We used the LUBM4OBDA benchmark, whose mappings contain SQL queries with multiple joins, to show how our extension of Morph-KGC can handle complex joins efficiently even in the presence of large volumes of data. As mentioned in previous chapters, the benchmark provides the data as SQL dumps, and we exported here its tables as tabular data in CSV and Apache Parquet (in a similar manner as done by GTFS-Madrid-Bench) formats. The benchmark consists of 14 tabular files that result in an output KG of more than 150 million triples for scaling factor 1.000. These tabular files and the mappings with RML tabular view are openly available in Zenodo [10]. As a baseline, we considered a setup in which a relational representation of the tabular sources is created, the tabular data is loaded into an RDBMS, and mappings with RML views over the RDB are used. We just take into account the materialization times, ignoring the additional cost of creating the relational representation and the overhead of loading the data into an RDBMS. We use PostgreSQL v15.0, MySQL v8.0.31 and data SFs 1, 10, 100 and 1.000 of LUBM4OBDA. We utilized Morph-KGC v2.3.1 on a system equipped with an Intel® Core™ i7-1165G7 (2.80GHz) and a memory of 40 GB RAM DDR4 (3200 MHz).

Figure 7.3 shows the materialization times obtained. It is observed that Morph-KGC is

[4] https://github.com/morph-kgc/morph-kgc/discussions/102

Table 7.1: Test cases that are not supported by RML's base source.

Test Case	Description	RML Support
RMLTVTC0002d	Concatenation of a column and a string.	RML-FNML
RMLTVTC0002g	Tests the presence of an invalid SQL query.	N/A
RMLTVTC0002h	Tests the presence of duplicate column names in the SELECT list of the SQL query.	N/A
RMLTVTC0002i	Two columns mapping, SQL version identifier.	N/A
RMLTVTC0002j	Two columns mapping, qualified column names.	N/A
RMLTVTC0003b	Concatenation of two columns and a string.	RML-FNML
RMLTVTC0009c	Concatenation of two columns and a string.	RML-FNML
RMLTVTC0009d	Aggregation of a column.	RML-FNML
RMLTVTC0011a	M to M relation, by using an SQL query.	Yes (by using an additional Triples Map)
RMLTVTC0014d	Replacement of data values.	RML-FNML
RMLTVTC0015a	Filtering.	RML-FNML
RMLTVTC0015b	Filtering.	RML-FNML
RMLTVTC0019a	Filtering.	RML-FNML
RMLTVTC0026a	Embedded list in a column.	RML-FNML
RMLTVTC0027a	Embedded JSON in a column.	RML+Fields
RMLTVTC0028a	Generation of literals with joins.	RML+[47]
RMLTVTC0029a	Join of multiple sources.	No

faster when materializing directly from tabular sources compared to relying on RDBMSs. Differences are appreciated between RDBMSs, in particular, while PostgreSQL is not far from the materialization times obtained for tabular data, times significantly increase for MySQL when the scaling factor is large. This shows that our implementation supports multiple joins and that it is even more efficient than relying on RDBMSs.

7.3 Python User-defined Functions

RML-FNML materialization systems typically offer built-in function sets such as the General Refine Expression Language[5] (GREL). However, built-in functions in RML systems are not enough to handle the heterogeneity in input data and address data processing tasks where more flexibility is needed. This additional expressive power is usually achieved with user-defined functions (UDFs) which can be written in languages such as Java, C++, or Python. Many

[5] https://openrefine.org/docs/manual/grelfunctions

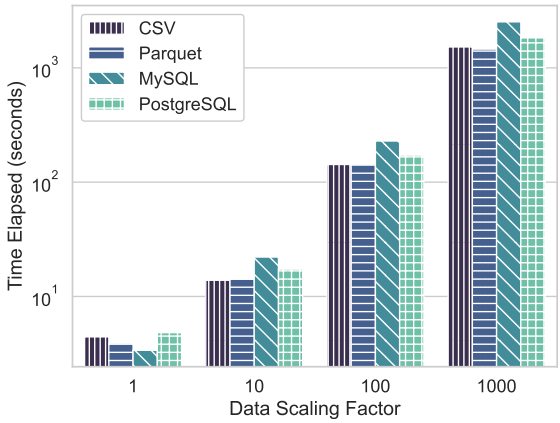


Figure 7.3: Execution times for LUBMM4OBDA benchmark with Morph-KGC and different configurations. Times are reported using a logarithmic scale.

researchers and data scientists typically rely on interpreted languages such as Python [112], as it is high-level, easy to learn and use, with features such as dynamic typing or automatic memory management, and comes with an extensive set of data processing packages. For these reasons, many modern data processing systems such as databases [56] offer Python UDFs. Therefore, our hypothesis is that making Python UDFs available in RML systems would empower KG creation.

In this section, we describe the implementation of an RML-FNML module for Morph-KGC that supports Python UDFs. The development of this module was motivated by requirements in use cases in BASF,[6] a large multinational chemical company, and many other organizations, where Python is the language of choice in data processing pipelines. We first describe the implementation and then provide an illustrative example of the use of Python user-defined functions at BASF.

7.3.1 Implementation

Built-in and user-defined functions Our RML-FNML implementation in Morph-KGC supports built-in and user-defined functions. Both of them use Python decorators to specify the function identifier and to link the parameters of the FnO function to the procedural Python parameters. Built-in functions use the `bif` decorator (this is transparent to the user since the function does not have to be provided), and UDFs use the `udf` decorator with `fun_id` defining the identifier to refer to the function from the RML-FNML mappings. An example of the latter is shown in Listing 7.2, which presents a UDF that removes blank spaces and special characters; the Python function receives `material` as input parameter, which is linked in the decorator to the FnO `grel:valueParam` parameter. To filter values in a UDF, `None` can be returned, in which case the RML-FNML module will not generate triples for those values.

[6] https://www.basf.com

```
import re

@udf(
    fun_id="http://ontology.basf.com/BASF_Functions/format_name",
    material="http://users.ugent.be/bjdmeest/function/grel.ttl#valueParam"
   )
def format_name(material):
    material = material.strip()
    clean_material = " ".join(re.split(r"\s+", material))
    return re.sub(r"[^\w\s]", "", clean_material)
```

Listing 7.2: Example of a Python UDF that removes blank spaces and special characters from a string representing a material in BASF.

Materialization procedure The system processes one mapping rule at a time. It first loads the data from the logical source, converting it to a Pandas DataFrame. Data manipulation, such as percent-encoding, `NULL` removal, and string escaping and concatenation, are performed over the created DataFrame. The procedure handles the different kinds of term maps and produces the RDF terms that are aggregated to create the final output triples. In the case of functions, the procedure identifies function-valued term maps and delegates its execution to the associated operator, which is described next.

Function executer The RML-FNML interpreter first uses the identifier of the function to retrieve it along with its decorator. When the function is user-defined, this entails loading it from a script provided by the user. The columns of the internal DataFrame are the input values for the function, and a new column is produced to store the results of its evaluation (the RDF term specifying the function execution in the mapping is used as the name of the new column). Chaining (or nesting) of functions is achieved following *operator-at-a-time* processing: the individual operators (i.e., the functions) of a mapping rule are executed over entire columns before moving to the next operator. Because the results of the function are stored in the DataFrame, the output of one function can be used as the input of subsequent functions, hence enabling chaining. It must be noted that, in accordance with the R2RML standard, `NULL` values resulting from the evaluation of a function are filtered after its execution.

RML-FNML compliant The compliance of the implementation with respect to the RML-FNML module has been validated with the test cases from the W3C Community Group on KG Construction.

7.3.2 Illustrative Example

BASF offers the world's largest portfolio and variety of chemical materials to meet the specific needs of several industries such as pharmaceuticals, automotive, agricultural, among others. Most of these materials are the product of in-house inventions that have been refined over more than a century of work. In this refinement process, there are several legacy systems that handle data related to these materials in different laboratories around the world. Unfortunately, some of the names of these materials include incorrect characters because legacy systems allowed

names to be added as a free text entry, limiting searchability. In addition, the names of most of the materials are provided only in German, which hinders its exploitation by non-German experts in laboratories across the world, who typically use English.

To format and translate materials from BASF relational sources, UDFs are applied to create a refined KG (see Figure 7.1). Listing 7.3 shows the RML-FNML rules that address these transformations. Here, the `basf_function:format_name`, presented in Listing 7.2, takes the `$(name)` value as input parameter and removes blank spaces, dots, colons, among other characters that were erroneously added to the name. Additionally, to generate material names in English, a nested function is applied to (i) first clean the material name using the `basf_function:format_name` function, and then (ii) translate the cleaned name into English using the `basf_function:translate_name` function, which calls an external translation service. Language tags are specified in the mapping rules according to the language of the generated material names.

```
mappings:
  materials:
    sources:
      table: materials
    s: basf_data:$(id)
    po:
      - [rdf:type, basf_onto:Material]
      - p: basf_onto:name
        o:
        - function: basf_function:format_name
          parameters:
          - parameter: grel:valueParam
            value: $(name)
          language: de
      - p: basf_onto:alternative_name
        o:
        - function: basf_function:translate_name
          parameters:
          - parameter: grel:valueParam
            value:
              function: basf_function:format_name
              parameters:
              - parameter: grel:valueParam
                value: $(name)
          language: en
```

Listing 7.3: Mapping to format and translate material names into English.

7.4 RML-FNML to RML View Unfolding

The execution of data transformation defined in RML-FNML has already been incorporated into materialization procedures by executing the transformations locally. However, query translation VKG systems push down computations to RDBs and avoid local computations as much as possible. To our knowledge, the execution of RML-FNML by VKG systems based

on query translation remains unexplored.[7] In this section, we propose to address this problem by unfolding RML-FNML to RML views, which can be executed by VKG systems. This approach follows the traditional idea of query translation in which computations are delegated to the underlying RDB.

Figure 7.4 depicts the overview of our proposed solution which consists in unfolding data transformations specified in RML-FNML into RML views which can then be used by VKG systems. Mapping translation from RML-FNML to RML views is performed as a preprocessing step in the VKG pipeline. This can be implemented as an auxiliary task in the *starting phase* of VKG systems [90]. The unfolded mappings with RML views can then be used as input for VKG systems. The approach is transparent for systems that do not need to interpret RML-FNML. It also presents the advantage of pushing down computations to RDBs, which are widely studied systems likely to efficiently perform the data transformations.

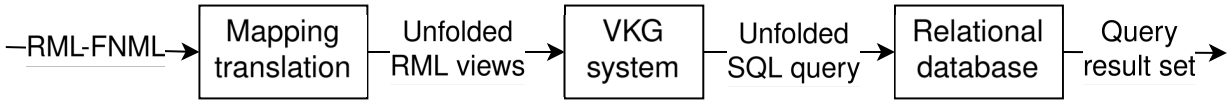


Figure 7.4: Overview of the proposal to execute RML-FNML mappings in VKG systems based on query translation. Data transformations defined in RML-FNML are unfolded to data transformations defined in RML views. The mappings with RML views are then used as input for VKG systems which perform SPARQL-to-SQL query translation.

The mapping translation step is similar to the execution of RML-FNML in materialization systems, but instead of transforming the data, the functions transform the effective SQL queries in the mappings. Functions can be divided into two main categories: (i) *transformations* (e.g., converting strings to lowercase), and (ii) *filtering*, which correspond to the selection operator (σ) in the relational algebra.

It is worth mentioning the FunMap [84] approach for executing data transformation in the materialization approach. FunMap applies the data transformation in the mappings to the heterogeneous data sources and removes the functions in the mappings. Function-free mappings and transformed data sources can then be used with an RML materialization system (which does not need to support RML-FNML) to perform the schema transformations. The FunMap approach transforms mappings and data (which may become a bottleneck when processing large volumes of data) before using an RML materialization system, while ours only translates mappings before using the RML virtualization system. Our proposal focuses on virtualization (since many materialization systems already support RML-FNML), and can also be applied in materialization.

7.4.1 Illustrative Example

```
1 <#frequencies>
2     rml:logicalSource [
3       rml:source "FREQUENCIES"
```

[7] VKG systems based on data translation can execute RML-FNML with materialization procedures, as our `ITT` implementation in Chapter 4.

```
4     ];
5     rml:subjectMap [
6         rml:template "http://linkeddata.es/madrid/metro/
7                         frequency/{trip_id}-{start_time}";
8     ];
9     rml:predicateObjectMap [
10        rml:predicate gtfs:exactTimes;
11        rml:objectMap [
12            fnml:execution <#frequenciesExe1>;
13            rml:datatype xsd:boolean
14        ];
15    ].
16
17 <#frequenciesExe1>
18    fnml:function grel:string_replace;
19    fnml:input
20        [ fnml:parameter grel:valueParam;
21          fnml:valueMap [
22              fnml:execution <#frequenciesExe2>] ],
23        [ fnml:parameter grel:value_find;
24          fnml:value "1" ],
25        [ fnml:parameter grel:value_replace;
26          fnml:value "true" ].
27
28 <#frequenciesExe2>
29    fnml:function grel:controls_filter;
30    fnml:input
31        [ fnml:parameter grel:param_a;
32          fnml:valueMap [
33          rml:reference "exact_times" ] ],
34        [ fnml:parameter grel:uri_value;
35          fnml:value "1" ].
```

Listing 7.4: RML-FNML mapping that transforms boolean "*1s*" into the RDF "*true*" form.

```
1 <#frequencies>
2     rml:logicalSource [
3     rml:query """
4        SELECT trip_id, start_time,
5           REPLACE(
6             CAST(exact_times AS VARCHAR),
7             '1', 'true')
8           AS v2d2bb5b65c
9        FROM FREQUENCIES
10       WHERE exact_times='1'
11    """ ];
12    rml:subjectMap [
13        rml:template "http://linkeddata.es/madrid/metro/
                        frequency/{trip_id}-{start_time}";
14    ];
15    rml:predicateObjectMap [
16        rml:predicate gtfs:exactTimes;
17        rml:objectMap [
18            rml:reference "v2d2bb5b65c";
```

```
19              rml:datatype xsd:boolean
20          ];
21      ].
```

Listing 7.5: Translated mapping in Listing 7.4 with RML views. It unfolds `grel:string_replace` into the `REPLACE` SQL function and `grel:controls_filter` into the `WHERE` clause in SQL.

We use GTFS data to illustrate our proposal. Boolean values in GTFS are encoded as "*0s*" and "*1s*", which must be transformed into the value space of `xsd:boolean` ("*true*" and "*false*" values). This can be done with data transformations in RML-FNML. Listing 7.4 depicts an RML-FNML mapping that transforms "*1s*" in the `exact_times` column of the `FREQUENCIES` table to "*true*". To do this, the values in a column are first filtered (with `grel:controls_filter`) to keep only those values equal to "*1*"; then these values are replaced (`grel:string_replace`) with "*true*". Listing 7.5 shows the unfolded mapping with RML views. The function `grel:string_replace` is unfolded into the `REPLACE` SQL function and `grel:controls_filter` into the `WHERE` clause in SQL. The translation of "*0s*" to "*false*" can be done in a similar manner.

7.5 Summary

In this chapter, we addressed challenges related to data transformations in RML mappings. Data transformations are crucial in data integration scenarios, as they are used to clean and homogenize messy data. The significance of data transformations led to the RML-FNML module in the RML ecosystem. Consequently, we proposed solutions to enhance the usability and flexibility of data transformations in declarative mappings, thereby improving the efficiency and effectiveness of integrating data into a KG.

First, we proposed RML tabular views as an alternative to RML-FNML for expressing data transformations using SQL over tabular sources such as CSV and Apache Parquet. RML views offer a more compact syntax for defining transformations compared to the verbose RML-FNML. Views are particularly appealing to users familiar with SQL, as they eliminate the need to learn the RML-FNML syntax. Additionally, our proposal addresses the limitations of RML for tabular data regarding joins, enabling (i) multiple joins, (ii) theta joins, and (iii) literal generation with joins. We validated RML tabular views with test cases derived from the ones of R2RML, and analyzed their performance with the LUBM4OBDA benchmark, demonstrating their effectiveness in handling multiple joins.

Next, we integrated Python user-defined functions with RML-FNML. The flexibility of user-defined functions is crucial in scenarios involving messy data that requires cleaning, and in which built-in functions are not enough. Programming functions in Python not only enhances flexibility in constructing knowledge graphs in these scenarios but also ensures usability, as Python is a high-level, easy-to-learn language widely used for data processing.

Finally, we devised a strategy for executing RML-FNML mappings in virtual knowledge graph systems based on query translation. The `ITT` architecture presented in Chapter 4 natively supports the execution of RML-FNML mappings, as data transformations can

be performed during the materialzation of the intermediate triple table. However, query translation systems do not materialize any RDF data, necessitating an alternative execution approach for RML-FNML. We proposed unfolding RML-FNML into RML views, where transformations in RML-FNML are converted into transformations defined in SQL queries. The resulting mappings do not include RML-FNML and can then be directly executed by any virtual knowledge system without further modification. This is the first proposal to execute RML-FNML mappings in virtual knowledge graph systems based on query translation.

Chapter 8

Conclusions

How to optimize declarative KG materialization from large heterogeneous data? How to generalize KG virtualization to multiple and diverse sources? This thesis demonstrates that it is possible to exploit RML's declarative nature to advance in these research problems. It also addresses other challenges in KG construction arising from the emergence of the RDF-star graph model and the need for flexible data transformations. The main contributions and results of this dissertation can be summarized as follows:

Materialization Optimization: Mapping Partitioning

Conditions for term map disjointness were identified based on term map types, invariants, and literal types. These conditions allow for the identification of disjoint mapping rules, making it theoretically possible to create a mapping partition with groups of rules with non-overlapping outputs. The experimental results demonstrate the positive impact of this optimization on materialization:

- For several benchmarks and real-world use cases, both partitioning algorithms—partial aggregations, and maximal—achieved high numbers of groups. Occasionally, maximal partitioning outperformed partial aggregations by retrieving more groups and reducing the number of rules per group. The results suggest that mapping partitioning can generally be applied to optimize materialization.
- Parallel processing of groups in a mapping partition significantly accelerates materialization. This strategy was proven to reduce execution times with negligible impact on memory compared to a baseline without partitioning.
- Sequential processing reduces memory needs to that of the larger mapping group in the partition, in terms of the amount of triples generated. This strategy showed a meaningful reduction in memory usage compared to the baseline and prevented out-of-memory errors for large datasets.
- It was also observed that materialization with mapping partitioning produced small overheads. Sequential processing exhibited a slight delay compared to the baseline, but this overhead is inconsequential in parallel processing. Maximal partitioning also

showed negligible delays compared to partial partitioning due to reaching the processor's parallelization capacity and the negative effect of the overhead—introduced by a larger number of groups—and the higher computational cost of maximal partitioning.

Furthermore, we empirically demonstrated the negative impact on materialization of redundant self-joins within mappings. This effect is avoided through canonicalization, where the mapping is first normalized and then redundant self-joins are eliminated.

Generic Virtualization: `ITT`

We devised the data translation-based `ITT` architecture for generic VKGs. Star-shaped mapping groups were formalized and employed for binding star-shaped subqueries. Star-shaped mapping candidate selection permits the identification of a relevant subset of mappings for a query, reducing the amount of materialized mapping rules and, in consequence, the size of intermediate results. Mapping partitioning accelerates materialization and ensures duplicate-free intermediate results with minimal impact on memory. Intermediate results conform to the output of mappings—triples—and are directly stored in a triple table. In this way, heterogeneous data is uniformly converged into a generic, schema-oblivious structure that is homogeneously queried using an analytical in-process SQL database. The source-agnostic essence and performance of the architecture were examined:

- The experimentation with relational databases, document stores, and mixed data—CSV, PostgreSQL, JSON, XML, MongoDB—demonstrate the feasibility of `ITT` for virtualizing over multiple, heterogeneous data sources. `ITT` behaves as a semantic polystore.
- End-to-end query execution time is dominated by the materialization of the intermediate triple table and the evaluation of the unfolded SQL query over it. Materialization is the main bottleneck, generally taking more than 75% of the overall execution time. This causes query evaluation times to increase with the number of intermediate triples.
- Sporadic out-of-memory errors were observed for large data. This intensive use of memory is expected, as the architecture is based on data translation and retains large intermediate results in memory. This issue is not observed for query translation using Ontop. However, Ontop occasionally produced very complex queries that timed out. In contrast, `ITT` did not experience any timeouts. `ITT` is more steady for small and medium-sized data, for which it successfully evaluates all queries.
- `ITT` is less sensitive to the performance of underlying databases compared to the query translation approach. This is because queries against local databases are simple in `ITT`, but complex in Ontop. The results for PostgreSQL and MySQL with `ITT` are very similar, but Ontop exhibits significantly different results for both RDBMSs.

RDF-star Graph Generation: RML-star

We extended RML to generate RDF-star graphs and introduced a baseline materialization algorithm. The generation of deeply nested quotes and non-asserted triples with RML-star was examined. We also devised a procedure to automatically extract RML-star mappings from property graphs. The results are summarized as follows:

- The generation of quoted triples can be declared by referencing mapping rules—triples maps—within term maps. This is done in RML-star with star maps, which are recursively defined. Deeply nested quoted triples can be generated through deeply nested triples maps.
- Whether a triples map asserts triples in the graph can be controlled with a flag in the mapping rule. In RML-star this is done defining two types of triples maps: asserted and non-asserted triples maps.
- A mapping rule can be processed as a binary tree, where the left and right children are determined by the rules referenced by star maps in the subject and object, respectively. As verified through unit tests, the tree can be traversed—in post-order in our case—to materialize the triples.
- RML-star is generic, unlike the former R2RML-star language, which restricts joins to R2RML views and is therefore specific to relational data.
- The automatic extraction of a default RML-star mapping is possible and can be employed for direct mapping to RDF-star and to mitigate the labor-intensive process of creating an initial mapping for subsequent customization. The results show the effectiveness of the procedure.

Enhanced Data Transformations: Harnessing RML Views

We also addressed limitations of data transformations in RML. RML views were extended to handle tabular files. The unfolding of RML-FNML into RML views was studied to permit its execution by query translation-based VKG systems. Python user-defined functions were implemented to allow more flexible integration of heterogeneous data. The results are summarized as follows:

- RML views can be extended to tabular files to define data transformations using SQL. This was confirmed by extending R2RML views test cases to tabular files. The effectiveness of RML tabular views in handling composite data values was also demonstrated.
- RML tabular views enable complex joining of tabular files, including multiple and theta joins, as well as the generation of literals with joins. It has been demonstrated that joins in RML tabular views are efficient by delegating their execution to an in-process SQL database.
- Preliminary results of RML-FNML unfolding to RML views suggest the feasibility of this approach. This permits query translation-based systems to execute RML-FNML with prior unfolding of the mapping.
- RML-FNML is inherently supported in data translation-based virtualization with `ITT` by executing transformations during the materialization of the intermediate triple table.
- The integration of Python-user defined functions in RML allows for more flexible data transformations by leveraging a high-level programming language. The positive impact of Python UDFs in complex data integration use cases has been demonstrated at an

industry scale in the chemical sector [14].

8.1 Broader Impact

Many of the techniques presented in this thesis are embodied in the materialization system Morph-KGC. The first release of the system in October 2021 was a research prototype that implemented mapping partitioning. Since then, we have implemented more novel techniques, making the system feature-rich and production-ready. This has led to its adoption by many organizations across various sectors, including transport, biomedicine, chemistry, telecommunications, and manufacturing. Some of the applications have been reported in the literature [101, 93, 30, 21, 100, 4, 45, 127].

The broader impact of Morph-KGC is particularly evident in the implementation of Python UDFs. This originated from the needs in data integration use cases at BASF, and was carried out as a collaboration between the company and Universidad Politécnica de Madrid.

8.2 Future Work

The body of our work addressed problems in the area of declarative KG construction. However, it has certain limitations that could be interesting to explore in future works. The main **limitations** and areas for expansion include:

- **Mapping partitioning beyond core term maps:** The mapping partitioning optimization can be applied with RML-star and RML-FNML, but it presents some limitations that were not addressed yet in our proposal. Specifically, when considering these RML modules, it is possible to partition by term and literal types, but the invariant presents complications that were not addressed. This might reduce the number of groups in the partitions and consequently the performance gain of the optimization. In this regard, it is necessary to further extend invariants to star and function maps. Both of them present the challenge of chaining, and the latter presents the additional challenge of obtaining an invariant from a function that, in many cases, might yield the empty invariant. Identifying invariants may be even more challenging for UDFs.
- **Larger-than-memory workloads:** The materialization techniques presented in this thesis maintain intermediate results in memory. If the memory limit is exceeded, materialization fails. This is mitigated with sequential processing of a mapping partition, but again, if the larger mapping group does not fit into the machines' main memory, materialization is not possible. This also affects `ITT` when the intermediate triple table does not fit in main memory. This limitation could be avoided with out-of-core processing by spilling to disk.
- **RDF 1.2:** We thoroughly addressed the declarative generation of RDF-star from diverse data. However, during the course of this work, the W3C RDF-star Working Group has been standardizing this data model—the upcoming RDF 1.2—and has introduced significant changes. For instance, in the current draft specification [70] quoted triples—now called *triple terms*—have an associated *reifier* and can only occur in the object

position of another triple. These changes require a revision of RML-star.

- **Generic RML views:** Albeit we addressed limitations of RML for tabular files with RML tabular views, this solution is still specific to relational and tabular data. Although in some cases it is possible to use them for other formats—such as JSON in Morph-KGC—it is convenient to generalize views. Indeed, the initial publication of our proposal [13] motivated the need for this, and researchers are working on *RML logical views* [131] and their integration in the RML ecosystem.[1]
- **User-defined aggregate functions:** Our implementation of Python UDFs only supports scalar functions. However, there may be scenarios where aggregate functions are required.

In addition to these potential opportunities for expansion, our work paves the way for the study of other challenges in the area of declarative KG construction and RDF graph management. Some important **open research problems** are:

- **Efficient updates:** Updates to local sources do not affect virtualization because the data is accessed on the fly. However, they cause materialized graphs to become obsolete. To keep materialized graphs up to date, changes imply re-materializing or using selective update algorithms for partial materialization. Initial research in this direction has recently been proposed [128]. A potential line of work for this problem could involve exploiting mapping partitioning to optimize updates.
- **`ITT` optimization:** We proposed a basis for generic VKGs, but further research is necessary to optimize our plain `ITT` architecture. As it was observed, end-to-end query evaluation times are significantly influenced by the volume of translated data. Therefore, exploring bind joins [68] could be beneficial, as they may reduce the amount of intermediate results and, consequently, execution time and memory consumption. Another line of work is hybrid querying. We observed that various factors have different impact on the query and data shipping approaches. A detailed study on these impacts could lead to the development of hybrid approaches that dynamically select the execution strategy. On a more advanced level, this could be extended for hybrid processing of different parts of a query [8].
- **RDF-star virtualization:** It is not clear how to virtualize with RDF-star graphs. One challenge towards this is the binding of SPARQL-star queries and RML-star mappings: Can star-shaped mapping candidate selection be extended to RML-star? Another challenge is the representation of intermediate results: Is a plain triple table sufficient for efficiently querying RDF-star graphs? Can SPARQL-star be unfolded to SQL over a triple table? These challenges are crucial for extending `ITT` to RDF-star. Moreover, extending query unfolding to SPARQL-star in Ontop has also proven challenging [121].

[1] https://w3id.org/rml/lv/spec

References

[1] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. "Scalable semantic web data management using vertical partitioning". In: *Proc. of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 2007, pp. 411–422. URL: https://www.vldb.org/conf/2007/papers/research/p411-abadi.pdf.

[2] Ghadeer Abuoda, Daniele Dell'Aglio, Arthur Keen, and Katja Hose. "Transforming RDF-star to Property Graphs: A Preliminary Analysis of Transformation Approaches". In: *Proc. of the 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs*. Vol. 3279. CEUR Workshop Proceedings, 2022, pp. 17–32. URL: https://ceur-ws.org/Vol-3279/paper2.pdf.

[3] Waqas Ali, Muhammad Saleem, Bin Yao, Aidan Hogan, and Axel-Cyrille Ngonga Ngomo. "A survey of RDF stores & SPARQL engines for querying knowledge graphs". In: *The VLDB Journal* (2021). DOI: 10.1007/s00778-021-00711-3.

[4] Ginés Almagro-Hernández, Juan Mulero-Hernández, Prashant Deshmukh, José Antonio Bernabé-Díaz, José Luis Sánchez-Fernández, Paola Espinoza-Arias, Juergen Mueller, and Jesualdo Tomás Fernández-Breis. "Evaluation of alignment methods to support the assessment of similarity between e-commerce knowledge graphs". In: *Knowledge-Based Systems* 315 (2025), p. 113283. DOI: 10.1016/j.knosys.2025.113283.

[5] Renzo Angles, Carlos Buil Aranda, Aidan Hogan, Carlos Rojas, and Domagoj Vrgoč. "WDBench: A Wikidata Graph Query Benchmark". In: *Proc. of the 21st International Semantic Web Conference*. Springer International Publishing, 2022, pp. 714–731. DOI: 10.1007/978-3-031-19433-7_41.

[6] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. "Mapping RDF Databases to Property Graph Databases". In: *IEEE Access* 8 (2020), pp. 86091–86110. DOI: 10.1109/ACCESS.2020.2993117.

[7] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. World Wide Web Consortium (W3C), 2012. URL: https://www.w3.org/TR/rdb-direct-mapping/.

[8] Julián Arenas-Guerrero. "Physical Knowledge Graph Design for Efficient Federated Query Processing". MA thesis. Universidad Politécnica de Madrid, 2020. URL: http://oa.upm.es/63647/.

[9] Julián Arenas-Guerrero. *RML Tabular Views Test Cases*. Version 1.0. Zenodo, 2022. DOI: 10.5281/zenodo.7389760.

[10] Julián Arenas-Guerrero. *The LUBM4OBDA Benchmark for Tabular Sources*. Zenodo, 2022. DOI: 10.5281/zenodo.7389705.

[11] Julián Arenas-Guerrero. *ITT*. 2024. DOI: 10.5281/zenodo.11096735.
[12] Julián Arenas-Guerrero. *morph-kgc/morph-kgc*. 2024. DOI: 10.5281/zenodo.5543552.
[13] Julián Arenas-Guerrero, Ahmad Alobaid, María Navas-Loro, María S. Pérez, and Oscar Corcho. "Boosting Knowledge Graph Generation from Tabular Data with RML Views". In: *Proceedings of the 20th Extended Semantic Web Conference*. Springer Nature Switzerland, 2023, pp. 484–501. DOI: 10.1007/978-3-031-33455-9_29.
[14] Julián Arenas-Guerrero, Paola Espinoza-Arias, José Antonio Bernabé-Diaz, Prashant Deshmukh, José Luis Sánchez-Fernández, and Oscar Corcho. "An RML-FNML module for Python user-defined functions in Morph-KGC". In: *SoftwareX* 26 (2024), p. 101709. DOI: 10.1016/j.softx.2024.101709.
[15] Julián Arenas-Guerrero, María S. Pérez, and Oscar Corcho. "LUBM4OBDA: Benchmarking OBDA Systems with Inference and Meta Knowledge". In: *Journal of Web Engineering* 22.8 (2024), pp. 1163–1186. DOI: 10.13052/jwe1540-9589.2284.
[16] Julián Arenas-Guerrero, Mario Scrocca, Ana Iglesias-Molina, Jhon Toledo, Luis Pozo-Gilo, Daniel Doña, Oscar Corcho, and David Chaves-Fraga. "Knowledge Graph Construction with R2RML and RML: An ETL System-based Overview". In: *Proc. of the 2nd International Workshop on Knowledge Graph Construction*. Vol. 2873. CEUR Workshop Proceedings, 2021. URL: http://ceur-ws.org/Vol-2873/paper11.pdf.
[17] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. "Spark SQL: Relational Data Processing in Spark". In: *Proc. of the 2015 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2015, pp. 1383–1394. DOI: 10.1145/2723372.2742797.
[18] Chebotko Artem, Lu Shiyong, and Fotouhi Farshad. "Semantics preserving SPARQL-to-SQL translation". In: *Data & Knowledge Engineering* 68.10 (2009), pp. 973–1000. DOI: 10.1016/j.datak.2009.04.001.
[19] Luigi Asprino, Enrico Daga, Aldo Gangemi, and Paul Mulholland. "Knowledge Graph Construction with a Façade: A Unified Method to Access Heterogeneous Data Sources on the Web". In: *ACM Transactions on Internet Technology* 23.1 (2023). DOI: 10.1145/3555312.
[20] Magnus Bakken. "maplib: Interactive, Literal RDF Model Mapping for Industry". In: *IEEE Access* 11 (2023), pp. 39990–40005. DOI: 10.1109/ACCESS.2023.3269093.
[21] Sebastian Barzaghi, Ivan Heibi, Arianna Moretti, and Silvio Peroni. "Developing Application Profiles for Enhancing Data and Workflows in Cultural Heritage Digitisation Processes". In: *Proc. of the 23rd International Semantic Web Conference*. Springer Nature Switzerland, 2025, pp. 197–217. DOI: 10.1007/978-3-031-77847-6_11.
[22] Bayer, Michael. "SQLAlchemy". In: *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012. URL: http://aosabook.org/en/sqlalchemy.html.
[23] David Beckett. *RDF 1.1 N-Triples*. W3C Recommendation. World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/n-triples/.
[24] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. *RDF 1.1 Turtle*. W3C Recommendation. World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/turtle/.

[25] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". In: *Scientific American* 284.5 (2001). URL: https://www.scientificamerican.com/article/the-semantic-web/.

[26] Christian Bizer and Andreas Schultz. "The Berlin SPARQL Benchmark". In: *International Journal on Semantic Web and Information Systems, IJSWIS* 5.2 (2009), pp. 1–24. DOI: 10.4018/jswis.2009040101.

[27] Christian Bizer and Andy Seaborne. "D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs". In: *ISWC*. 2004. URL: http://iswc2004.semanticweb.org/posters/PID-SMCVRKBT-1089637165.pdf.

[28] Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Julien Corman, and Guohui Xiao. "Ontology-based data access – Beyond relational sources". In: *Intelligenza Artificiale* 13.1 (2019), pp. 21–36. DOI: 10.3233/IA-190023.

[29] Julian Bruyat, Pierre-Antoine Champin, Lionel Médini, and Frédérique Laforest. "PRSC: From PG to RDF and back, using schemas". In: *Semantic Web* 15.6 (2024), pp. 2555–2595. DOI: 10.3233/SW-243675.

[30] Pablo Calleja Ibañez and Elea Giménez-Toledo. "Exploring named-entity recognition techniques for academic books". In: *Learned Publishing* 37.3 (2024), e1610. DOI: 10.1002/leap.1610.

[31] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. "Ontop: Answering SPARQL queries over relational databases". In: *Semantic Web* 8.3 (2017), pp. 471–487. DOI: 10.3233/SW-160217.

[32] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. "The MASTRO system for ontology-based data access". In: *Semantic Web* 2.1 (2011), pp. 43–53. DOI: 10.3233/SW-2011-0029.

[33] Diego Calvanese, Avigdor Gal, Davide Lanti, Marco Montali, Alessandro Mosca, and Roee Shraga. "Conceptually-grounded mapping patterns for Virtual Knowledge Graphs". In: *Data & Knowledge Engineering* 145 (2023), p. 102157. DOI: 10.1016/j.datak.2023.102157.

[34] Upen S. Chakravarthy, John Grant, and Jack Minker. "Logic-Based Approach to Semantic Query Optimization". In: *ACM Transactions on Database Systems* 15.2 (1990), pp. 162–207. DOI: 10.1145/78922.78924.

[35] Miloš Chaloupka and Martin Nečaský. "Efficient SPARQL to SQL Translation with User Defined Mapping". In: *Knowledge Engineering and Semantic Web*. Springer International Publishing, 2016, pp. 215–229. DOI: 10.1007/978-3-319-45880-9_17.

[36] David Chaves, Ana Iglesias, Daniel Garijo, and Julián Arenas Guerrero. *kg-construct/rml-star-test-cases*. Version 1.1. 2022. DOI: 10.5281/zenodo.6518802.

[37] David Chaves-Fraga, Freddy Priyatna, Andrea Cimmino, Jhon Toledo, Edna Ruckhaus, and Oscar Corcho. "GTFS-Madrid-Bench: A Benchmark for Virtual Knowledge Graph Access in the Transport Domain". In: *Journal of Web Semantics* 65 (2020), p. 100596. DOI: 10.1016/j.websem.2020.100596.

[38] Sijin Cheng and Olaf Hartig. "FedQPL: A Language for Logical Query Plans over Heterogeneous Federations of RDF Data Sources". In: *Proc. of the 22nd International Conference on Information Integration and Web-Based Applications & Services, iiWAS.*

Association for Computing Machinery, 2021, pp. 436–445. DOI: 10.1145/3428757.3429120.

[39] Sijin Cheng and Olaf Hartig. "LinGBM: A Performance Benchmark for Approaches to Build GraphQL Servers". In: *Proc. of the 23rd International Conference on Web Information Systems Engineering (WISE)*. Springer International Publishing, 2022, pp. 209–224. DOI: 10.1007/978-3-031-20891-1_16.

[40] Hirokazu Chiba, Ryota Yamanaka, and Shota Matsumoto. "G2GML: Graph to Graph Mapping Language for Bridging RDF and Property Graphs". In: *ISWC*. Springer International Publishing, 2020, pp. 160–175. DOI: 10.1007/978-3-030-62466-8_11.

[41] Oscar Corcho, David Chaves-Fraga, Jhon Toledo, Julián Arenas-Guerrero, Carlos Badenes-Olmedo, Mingxue Wang, Hu Peng, Nicholas Burrett, José Mora, and Puchao Zhang. "A High-Level Ontology Network for ICT Infrastructures". In: *Proc. of the 20th International Semantic Web Conference, ISWC*. Springer International Publishing, 2021, pp. 446–462. DOI: 10.1007/978-3-030-88361-4_26.

[42] Ademar Crotti Junior, Christophe Debruyne, Rob Brennan, and Declan O'Sullivan. "An evaluation of uplift mapping languages". In: *International Journal of Web Information Systems* 13.4 (2017), pp. 405–424. DOI: 10.1108/IJWIS-04-2017-0036.

[43] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[44] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. World Wide Web Consortium (W3C), 2012. URL: http://www.w3.org/TR/r2rml/.

[45] Ioannis Dasoulas, Duo Yang, and Anastasia Dimou. "MLSea: A Semantic Layer for Discoverable Machine Learning". In: *Proc. of the 21st Extended Semantic Web Conference, ESWC*. Springer Nature Switzerland, 2024, pp. 178–198. DOI: 10.1007/978-3-031-60635-9_11.

[46] Ben De Meester, Tom Seymoens, Anastasia Dimou, and Ruben Verborgh. "Implementation-independent function reuse". In: *Future Generation Computer Systems* 110 (2020), pp. 946–959. DOI: 10.1016/j.future.2019.10.006.

[47] Christophe Debruyne. "Supporting Relational Database Joins for Generating Literals in R2RML". In: *Proc. of the 3rd International Workshop on Knowledge Graph Construction*. Vol. 3141. CEUR Workshop Proceedings, 2022. URL: http://ceur-ws.org/Vol-3141/paper7.pdf.

[48] Christophe Debruyne and Declan O'Sullivan. "R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings". In: *Proc. of the 9th Workshop on Linked Data on the Web*. Vol. 1593. CEUR Workshop Proceedings, 2016. URL: http://ceur-ws.org/Vol-1593/article-13.pdf.

[49] Thomas Delva, Dylan Van Assche, Pieter Heyvaert, Ben De Meester, and Anastasia Dimou. "Integrating Nested Data into Knowledge Graphs with RML Fields". In: *Proc. of the 2nd International Workshop on Knowledge Graph Construction*. Vol. 2873. CEUR Workshop Proceedings, 2021. URL: http://ceur-ws.org/Vol-2873/paper9.pdf.

[50] Anastasia Dimou and Miel Vander Sande. *RDF Mapping Language (RML)*. Tech. rep. World Wide Web Consortium (W3C), 2022. URL: https://rml.io/specs/rml/.

[51] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *Proc. of the 7th Workshop on Linked Data on the Web*. Vol. 1184. CEUR Workshop Proceedings, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[52] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. "The BigDAWG Polystore System". In: *SIGMOD Record* 44.2 (2015), pp. 11–16. DOI: 10.1145/2814710.2814713.

[53] Orri Erling and Ivan Mikhailov. "RDF Support in the Virtuoso DBMS". In: *Networked Knowledge - Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems*. Springer Berlin Heidelberg, 2009, pp. 7–24. DOI: 10.1007/978-3-642-02184-8_2.

[54] Naglaa Fathy, Walaa Gad, Nagwa Badr, and Mohamed Hashem. "ProGOMap: Automatic Generation of Mappings From Property Graphs to Ontologies". In: *IEEE Access* 9 (2021), pp. 113100–113116. DOI: 10.1109/ACCESS.2021.3104293.

[55] Serge Chávez Feria, Raúl García-Castro, and María Poveda-Villalón. "Chowlk: from UML-Based Ontology Conceptualizations to OWL". In: *Proc. of the 19th Extended Semantic Web Conference*. Springer International Publishing, 2022, pp. 338–352. DOI: 10.1007/978-3-031-06981-9_20.

[56] Yannis Foufoulas, Alkis Simitsis, Lefteris Stamatogiannakis, and Yannis Ioannidis. "YeSQL: "You Extend SQL" with Rich and Highly Performant User-Defined Functions in Relational Databases". In: *Proc. VLDB Endow.* 15.10 (2022), pp. 2270–2283. DOI: 10.14778/3547305.3547328.

[57] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. "Cypher: An Evolving Query Language for Property Graphs". In: *Proceedings of the 2018 International Conference on Management of Data*. Association for Computing Machinery, 2018, pp. 1433–1445. DOI: 10.1145/3183713.3190657.

[58] Kevin P. Gaffney, Martin Prammer, Larry Brasfield, D. Richard Hipp, Dan Kennedy, and Jignesh M. Patel. "SQLite: Past, Present, and Future". In: *Proc. VLDB Endow.* 15.12 (2022), pp. 3535–3547. DOI: 10.14778/3554821.3554842.

[59] Herminio García-González, Iovka Boneva, Sławek Staworko, José Emilio Labra-Gayo, and Juan Manuel Cueva Lovelle. "ShExML: improving the usability of heterogeneous data mapping languages for first-time users". In: *PeerJ Computer Science* 6 (2020), e318. DOI: 10.7717/peerj-cs.318.

[60] Jonathan Goldstein and Per-Åke Larson. "Optimizing Queries Using Materialized Views: A Practical, Scalable Solution". In: *SIGMOD Record* 30.2 (2001), pp. 331–342. DOI: 10.1145/376284.375706.

[61] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems". In: *Journal of Web Semantics* 3.2 (2005), pp. 158–182. DOI: 10.1016/j.websem.2005.06.005.

[62] Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. "Optimizing Queries Across Diverse Data Sources". In: *Proc. of the 23rd International Conference on Very Large Data Bases*. 1997. URL: https://www.vldb.org/conf/1997/P276.PDF.

[63] Gerald Haesendonck, Wouter Maroy, Pieter Heyvaert, Ruben Verborgh, and Anastasia Dimou. "Parallel RDF Generation from Heterogeneous Big Data". In: *Proc. of the International Workshop on Semantic Big Data*. Association for Computing Machinery, 2019, pp. 1–6. DOI: 10.1145/3323878.3325802.

[64] Stefan Hagedorn, Steffen Kläbe, and Kai-Uwe Sattler. "Putting Pandas in a Box". In: *Proc. of the Conference on Innovative Data Systems Research*. 2021. URL: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper07.pdf.

[65] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation. World Wide Web Consortium (W3C), 2013. URL: https://www.w3.org/TR/sparql11-query/.

[66] Olaf Hartig. "Reconciliation of RDF* and property graphs". In: *CoRR* (2014). arXiv preprint arXiv:1409.3288. DOI: 10.48550/arXiv.1409.3288.

[67] Olaf Hartig. "Foundations of RDF* and SPARQL* (An Alternative Approach to Statement-Level Metadata in RDF)". In: *Proc. of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web*. Vol. 1912. CEUR Workshop Proceedings, 2017. URL: http://ceur-ws.org/Vol-1912/paper12.pdf.

[68] Olaf Hartig and Carlos Buil-Aranda. "Bindings-Restricted Triple Pattern Fragments". In: *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Springer International Publishing, 2016, pp. 762–779. DOI: 10.1007/978-3-319-48472-3_48.

[69] Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg, and Andy Seaborne. *RDF-star and SPARQL-star*. W3C Final Community Group Report. 2021. URL: https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html.

[70] Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg, and Andy Seaborne. *RDF 1.2 Concepts and Abstract Syntax*. W3C Working Draft. World Wide Web Consortium, 2025. URL: https://www.w3.org/TR/2025/WD-rdf12-concepts-20250225/.

[71] Ali Hasnain, Qaiser Mehmood, Syeda Sana e Zainab, Muhammad Saleem, Claude Warren, Durre Zehra, Stefan Decker, and Dietrich Rebholz-Schuhmann. "BioFed: federated query processing over life sciences linked open data". In: *Journal of Biomedical Semantics* 8 (1 2017), p. 13. DOI: 10.1186/s13326-017-0118-0.

[72] Patrik J. Hayes and Peter F. Patel-Schneider. *RDF 1.1 Semantics*. W3C Recommendation. World Wide Web Consortium, 2014. URL: http://www.w3.org/TR/rdf11-mt/.

[73] Daniel Hernández, Aidan Hogan, and Markus Krötzsch. "Reifying RDF: What Works Well With Wikidata?" In: *Proc. of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems*. Vol. 1457. CEUR Workshop Proceedings, 2015, pp. 32–47. URL: http://ceur-ws.org/Vol-1457/SSWS2015_paper3.pdf.

[74] Pieter Heyvaert, David Chaves-Fraga, Freddy Priyatna, Oscar Corcho, Erik Mannens, Ruben Verborgh, and Anastasia Dimou. "Conformance Test Cases for the RDF Mapping Language (RML)". In: *Proc. of the 1st Iberoamerican Knowledge Graphs and Semantic Web Conference*. Springer International Publishing, 2019, pp. 162–173. DOI: 10.1007/978-3-030-21395-4_12.

[75] Pieter Heyvaert, Ben De Meester, Anastasia Dimou, and Ruben Verborgh. "Declarative Rules for Linked Data Generation at Your Fingertips!" In: *Extended Semantic Web Conference, ESWC*. Springer International Publishing, 2018, pp. 213–217. DOI: 10.1007/978-3-319-98192-5_40.

[76] Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. "Everything you always wanted to know about blank nodes". In: *Journal of Web Semantics* 27-28 (2014), pp. 42–69. DOI: 10.1016/j.websem.2014.06.004.

[77] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. "Knowledge Graphs". In: *ACM Comput. Surv.* 54.4 (2021). DOI: 10.1145/3447772.

[78] Sayed Hoseini, Johannes Theissen-Lipp, and Christoph Quix. "A survey on semantic data management as intersection of ontology-based data access, semantic modeling and data lakes". In: *Journal of Web Semantics* 81 (2024), p. 100819. DOI: 10.1016/j.websem.2024.100819.

[79] Enrique Iglesias, Samaneh Jozashoori, David Chaves-Fraga, Diego Collarana, and Maria-Esther Vidal. "SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs". In: *Proc. of the 29th ACM International Conference on Information and Knowledge Management, CIKM*. Association for Computing Machinery, 2020, pp. 3039–3046. DOI: 10.1145/3340531.3412881.

[80] Enrique Iglesias, Samaneh Jozashoori, and Vidal Maria-Esther. "Scaling up knowledge graph creation to large and heterogeneous data sources". In: *Journal of Web Semantics* 75 (2023), p. 100755. DOI: 10.1016/j.websem.2022.100755.

[81] Ana Iglesias-Molina, Dylan Van Assche, Julián Arenas-Guerrero, Ben De Meester, Christophe Debruyne, Samaneh Jozashoori, Pano Maria, Franck Michel, David Chaves-Fraga, and Anastasia Dimou. "The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF". In: *Proc. of the 22nd International Semantic Web Conference, ISWC*. Springer Nature Switzerland, 2023, pp. 152–175. DOI: 10.1007/978-3-031-47243-5_9.

[82] Yahlieel Jafta, Louise Leenen, and Thomas Meyer. "Investigating Ontology-Based Data Access with GitHub". In: *Proc. of the 20th Extended Semantic Web Conference, ESWC*. Springer Nature Switzerland, 2023, pp. 644–660. DOI: 10.1007/978-3-031-33455-9_38.

[83] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and Jose Mora. "BootOX: Practical Mapping of RDBs to OWL 2". In: *ISWC*. Springer International Publishing, 2015, pp. 113–132. DOI: 10.1007/978-3-319-25010-6_7.

[84] Samaneh Jozashoori, David Chaves-Fraga, Enrique Iglesias, Maria-Esther Vidal, and Oscar Corcho. "FunMap: Efficient Execution of Functional Mappings for Knowledge Graph Creation". In: *Proc. of the 19th International Semantic Web Conference*. Springer International Publishing, 2020, pp. 276–293. DOI: 10.1007/978-3-030-62419-4_16.

[85] Samaneh Jozashoori and Maria-Esther Vidal. "MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation". In: *Proc. of the Confederated International Conferences*. Springer International Publishing, 2019, pp. 58–75. DOI: 10.1007/978-3-030-33246-4_4.

[86] Elem Güzel Kalaycı, Irlan Grangel González, Felix Lösch, Guohui Xiao, Anees ul-Mehdi, Evgeny Kharlamov, and Diego Calvanese. "Semantic Integration of Bosch Manufacturing Data Using Virtual Knowledge Graphs". In: *Proc. of the 19th Inter-*

national Semantic Web Conference, ISWC. Springer International Publishing, 2020, pp. 464–481. DOI: 10.1007/978-3-030-62466-8_29.

[87] Zuhair Khayyat, William Lucia, Meghna Singh, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz Jorge-Arnulfo, Nan Tang, and Panos Kalnis. "Fast and scalable inequality joins". In: *The VLDB Journal* 26.1 (2017), pp. 125–150. DOI: 10.1007/s00778-016-0441-6.

[88] Craig A. Knoblock and Pedro Szekely. "Exploiting Semantics for Big Data Integration". In: *AI Magazine* 36.1 (2015), pp. 276–293. DOI: 10.1609/aimag.v36i1.2565.

[89] Donald Kossmann. "The State of the Art in Distributed Query Processing". In: *ACM Computing Surveys* 32.4 (2000), pp. 422–469. DOI: 10.1145/371578.371598.

[90] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. "The NPD Benchmark: Reality Check for OBDA Systems". In: *Proc. of the 18th International Conference on Extending Database Technology, EDBT*. OpenProceedings.org, 2015, pp. 617–628. DOI: 10.5441/002/edbt.2015.62.

[91] Davide Lanti, Guohui Xiao, and Diego Calvanese. "VIG: Data scaling for OBDA benchmarks". In: *Semantic Web* 10.2 (2019), pp. 413–433. DOI: 10.3233/SW-180336.

[92] Ora Lassila, Michael Schmidt, Olaf Hartig, Brad Bebee, Dave Bechberger, Willem Broekema, Ankesh Khandelwal, Kelvin Lawrence, Carlos Manuel Lopez Enriquez, Ronak Sharda, and Bryan Thompson. "The OneGraph vision: Challenges of breaking the graph model lock-in 1". In: *Semantic Web* 14.1 (2023), pp. 125–134. DOI: 10.3233/SW-223273.

[93] Nicolas Le Guillarme and Wilfried Thuiller. "A practical approach to constructing a knowledge graph for soil ecological research". In: *European Journal of Soil Biology* 117 (2023), p. 103497. DOI: 10.1016/j.ejsobi.2023.103497.

[94] Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. "A SPARQL Extension for Generating RDF from Heterogeneous Formats". In: *Proc. of the 14th Extended Semantic Web Conference*. Ed. by Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rinke Hoekstra, Pascal Hitzler, and Olaf Hartig. Springer International Publishing, 2017, pp. 35–50. DOI: 10.1007/978-3-319-58068-5_3.

[95] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. "DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia". In: *Semantic Web* 6.2 (2015), pp. 167–195. DOI: 10.3233/SW-140134.

[96] Maurizio Lenzerini. "Data Integration: A Theoretical Perspective". In: *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*. Association for Computing Machinery, 2002, pp. 233–246. DOI: 10.1145/543613.543644.

[97] Akifumi Makinouchi. "A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model". In: *Proceedings of the 3rd International Conference on Very Large Data Bases*. VLDB Endowment, 1977, pp. 447–453.

[98] Mohamed Nadjib Mami, Damien Graux, Simon Scerri, Hajira Jabeen, Sören Auer, and Jens Lehmann. "Squerall: Virtual Ontology-Based Access to Heterogeneous and Large Data Sources". In: *Proc. of the 18th International Semantic Web Conference, ISWC*. Springer International Publishing, 2019, pp. 229–245. DOI: 10.1007/978-3-030-30796-7_15.

[99] Martin Mansfield, Valentina Tamma, Phil Goddard, and Frans Coenen. "Capturing Expert Knowledge for Building Enterprise SME Knowledge Graphs". In: *Proc. of the 11th Knowledge Capture Conference, K-CAP*. Association for Computing Machinery, 2021, pp. 129–136. DOI: 10.1145/3460210.3493569.

[100] Milan Markovic, Daniel Garijo, Stefano Germano, and Iman Naja. "TEC: Transparent Emissions Calculation Toolkit". In: *Proceedings of the 22nd International Semantic Web Conference*. Springer Nature Switzerland, 2023, pp. 76–93. DOI: 10.1007/978-3-031-47243-5_5.

[101] Ignacio D. Martinez-Casanueva, Luis Bellido, Daniel González-Sánchez, and Diego Lopez. "CANDIL: A federated data fabric for network analytics". In: *Future Generation Computer Systems* 158 (2024), pp. 98–109. DOI: 10.1016/j.future.2024.04.013.

[102] Maroua Masmoudi, Sana Ben Abdallah Ben Lamine, Mohamed Hedi Karray, Bernard Archimede, and Hajer Baazaoui Zghal. "Semantic data integration and querying: a survey and challenges". In: *ACM Computing Surveys* (2024). DOI: 10.1145/3653317.

[103] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proc. of the 9th Python in Science Conference*. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[104] Luciano Frontino de Medeiros, Freddy Priyatna, and Oscar Corcho. "MIRROR: Automatic R2RML Mapping Generation from Relational Databases". In: *ICWE*. Springer International Publishing, 2015, pp. 326–343. DOI: 10.1007/978-3-319-19890-3_21.

[105] Franck Michel, Loıc Djimenou, Catherine Faron Zucker, and Johan Montagnat. "Translation of Relational and Non-Relational Databases into RDF with xR2RML". In: *Proc. of the 11th International Conference on Web Information Systems and Technologies*. Vol. 1. SciTePress, 2015, pp. 443–454. DOI: 10.5220/0005448304430454.

[106] Franck Michel, Catherine Faron-Zucker, and Johan Montagnat. "A Mapping-Based Method to Query MongoDB Documents with SPARQL". In: *Proc. of the 27th International Conference on Database and Expert Systems Applications*. Springer International Publishing, 2016, pp. 52–67. DOI: 10.1007/978-3-319-44406-2_6.

[107] Thomas Neumann and Gerhard Weikum. "The RDF-3X engine for scalable management of RDF data". In: *The VLDB Journal* 19.1 (2010), pp. 91–113. DOI: 10.1007/s00778-009-0165-y.

[108] Vinh Nguyen, Olivier Bodenreider, and Amit Sheth. "Don't like RDF Reification? Making Statements about Statements Using Singleton Property". In: *Proc. of the 23rd International Conference on World Wide Web*. Association for Computing Machinery, 2014, pp. 759–770. DOI: 10.1145/2566486.2567973.

[109] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. "Industry-scale Knowledge Graphs: Lessons and Challenges: Five diverse technology companies show how it's done". In: *Queue* 17.2 (2019), pp. 48–75. DOI: 10.1145/3329781.3332266.

[110] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. "YAGO 4: A Reason-able Knowledge Base". In: *Proceedings of the 17th Extended Semantic Web Conference*. Springer International Publishing, 2020, pp. 583–596. DOI: 10.1007/978-3-030-49461-2_34.

[111] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. "Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph". In: *Proc. of the 23rd*

International Conference on World Wide Web. Association for Computing Machinery, 2014, pp. 479–490. DOI: 10.1145/2566486.2567981.

[112] Mark Raasveldt and Hannes Mühleisen. "Vectorized UDFs in Column-Stores". In: *Proc. of the 28th International Conference on Scientific and Statistical Database Management*. Association for Computing Machinery, 2016. DOI: 10.1145/2949689.2949703.

[113] Mark Raasveldt and Hannes Mühleisen. "DuckDB: An Embeddable Analytical Database". In: *Proc. of the 2019 International Conference on Management of Data*. Association for Computing Machinery, 2019, pp. 1981–1984. DOI: 10.1145/3299869.3320212.

[114] Mariano Rodríguez-Muro and Martin Rezk. "Efficient SPARQL-to-SQL with R2RML Mappings". In: *Journal of Web Semantics* 33 (2015), pp. 141–169. DOI: 10.1016/j.websem.2015.03.001.

[115] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. "FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework". In: *Proc. of the 14th International Semantic Web Conference, ISWC*. Springer International Publishing, 2015, pp. 52–69. DOI: 10.1007/978-3-319-25007-6_4.

[116] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. "FedX: Optimization Techniques for Federated Query Processing on Linked Data". In: *Proc. of the 10th International Semantic Web Conference, ISWC*. Springer Berlin Heidelberg, 2011, pp. 601–616. DOI: 10.1007/978-3-642-25073-6_38.

[117] Mario Scrocca, Marco Comerio, Alessio Carenini, and Irene Celino. "Turning Transport Data to Comply with EU Standards While Enabling a Multimodal Transport Knowledge Graph". In: *Proc. of the 19th International Semantic Web Conference, ISWC*. Springer International Publishing, 2020, pp. 411–429. DOI: 10.1007/978-3-030-62466-8_26.

[118] Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. "OBDA: Query Rewriting or Materialization? In Practice, Both!" In: *Proc. of the 13th International Semantic Web Conference, ISWC*. Springer International Publishing, 2014, pp. 535–551. DOI: 10.1007/978-3-319-11964-9_34.

[119] Juan F. Sequeda and Daniel P. Miranker. "Ultrawrap: SPARQL execution on relational data". In: *Journal of Web Semantics* 22 (2013), pp. 19–39. DOI: 10.1016/j.websem.2013.08.002.

[120] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. 2012. URL: https://blog.google/products/search/introducing-knowledge-graph-things-not/.

[121] Lukas Sundqvist. *Extending VKG Systems with RDF-star Support*. 2022. URL: https://ontop-vkg.org/publications/2022-sundqvist-rdf-star-ontop-msc-thesis.pdf.

[122] Gábor Szárnyas, Jack Waudby, Benjamin A. Steer, Dávid Szakállas, Altan Birler, Mingxi Wu, Yuchen Zhang, and Peter Boncz. "The LDBC Social Network Benchmark: Business Intelligence Workload". In: *Proc. VLDB Endow.* 16.4 (2022), pp. 877–890. DOI: 10.14778/3574245.3574270.

[123] Pedro Szekely, Daniel Garijo, Divij Bhatia, Jiasheng Wu, Yixiang Yao, and Jay Pujara. "T2WML: Table To Wikidata Mapping Language". In: *Proc. of the 10th International Conference on Knowledge Capture*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 267–270. DOI: 10.1145/3360901.3364448.

[124] *Tarql: SPARQL for Tables*. 2019. URL: https://tarql.github.io/.

[125] Dominik Tomaszuk, Renzo Angles, and Harsh Thakkar. "PGO: Describing Property Graphs in RDF". In: *IEEE Access* 8 (2020), pp. 118355–118369. DOI: 10.1109/ACCESS.2020.3002018.

[126] Jörg Unbehauen, Claus Stadler, and Sören Auer. "Accessing Relational Data on the Web with SparqlMap". In: *Proc. of the 2nd Joint International Conference, JIST*. Springer Berlin Heidelberg, 2013, pp. 65–80. DOI: 10.1007/978-3-642-37996-3_5.

[127] Mikel Val-Calvo, Mikel Egaña Aranguren, Juan Mulero-Hernández, Ginés Almagro-Hernández, Prashant Deshmukh, José Antonio Bernabé-Díaz, Paola Espinoza-Arias, José Luis Sánchez-Fernández, Juergen Mueller, and Jesualdo Tomás Fernández-Breis. "OntoGenix: Leveraging Large Language Models for enhanced ontology engineering from datasets". In: *Information Processing Management* 62.3 (2025), p. 104042. DOI: 10.1016/j.ipm.2024.104042.

[128] Dylan Van Assche, Julián Andrés Rojas, Ben De Meester, and Pieter Colpaert. "Incremental Knowledge Graph Construction from Heterogeneous Data Sources". In: *Semantic Web* (2024). Under Review. URL: https://www.semantic-web-journal.net/system/files/swj3790.pdf.

[129] María-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. "Efficiently Joining Group Patterns in SPARQL Queries". In: *Proc. of the 7th Extended Semantic Web Conference, ESWC*. Springer Berlin Heidelberg, 2010, pp. 228–242. DOI: 10.1007/978-3-642-13486-9_16.

[130] Boris Villazón-Terrazas and Michael Hausenblas. *R2RML and Direct Mapping Test Cases*. W3C Note. World Wide Web Consortium (W3C), 2012. URL: http://www.w3.org/TR/rdb2rdf-test-cases/.

[131] Els de Vleeschauwer, Pano Maria, Ben De Meester, and Pieter Colpaert. "RML-view-to-CSV: A Proof-of-Concept Implementation for RML Logical Views". In: *Proc. of the 5th International Workshop on Knowledge Graph Construction*. Vol. 3718. CEUR Workshop Proceedings, 2024. URL: https://ceur-ws.org/Vol-3718/paper2.pdf.

[132] Denny Vrandečić and Markus Krötzsch. "Wikidata: A Free Collaborative Knowledgebase". In: *Communications of the ACM* 57.10 (2014), pp. 78–85. DOI: 10.1145/2629489.

[133] Xiaoying Wang, Weiyuan Wu, Jinze Wu, Yizhou Chen, Nick Zrymiak, Changbo Qu, Lampros Flokas, George Chow, Jiannan Wang, Tianzheng Wang, Eugene Wu, and Qingqing Zhou. "ConnectorX: Accelerating Data Loading from Databases to Dataframes". In: *Proc. VLDB Endow.* 15.11 (2022), pp. 2994–3003. DOI: 10.14778/3551793.3551847.

[134] Jiantao Wu, Fabrizio Orlandi, Declan O'Sullivan, and Soumyabrata Dev. "Publishing Climate Data as Linked Data Via Virtual Knowledge Graphs". In: *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*. 2022, pp. 4090–4093. DOI: 10.1109/IGARSS46834.2022.9884226.

[135] Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. "Virtual Knowledge Graphs: An Overview of Systems and Use Cases". In: *Data Intelligence* 1.3 (2019), pp. 201–223. DOI: 10.1162/dint_a_00011.

[136] Guohui Xiao, Roman Kontchakov, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. "Efficient Handling of SPARQL OPTIONAL for OBDA". In: *Proc. of the*

17th International Semantic Web Conference, ISWC. Springer International Publishing, 2018, pp. 354–373. DOI: 10.1007/978-3-030-00671-6_21.

[137] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel-Kalaycı, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. "The Virtual Knowledge Graph System Ontop". In: *Proc. of the 19th International Semantic Web Conference, ISWC*. Springer International Publishing, 2020, pp. 259–277. DOI: 10.1007/978-3-030-62466-8_17.

[138] Feng Xiyang, Jin Guodong, Chen Ziyi, Liu Chang, and Salihoğlu Semih. "Kùzu Graph Database Management System". In: *Conference on Innovative Data Systems Research, CIDR*. 2023. URL: https://www.cidrdb.org/cidr2023/papers/p48-jin.pdf.