Programa de Doctorado de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos

PhD Thesis

Knowledge Graph Construction from Heterogeneous Data Sources exploiting Declarative Mapping Rules

Author: David Chaves Fraga
Supervisors: Oscar Corcho

XXXXXXX, 2021

Tribunal nombrado por el Sr. Rector Magfco. de la Universidad Politécnica de Madrid, el día XXX de Junio de XXXXX.

Presidente: Dr. Presidente Presidente Presidente

Vocal: Dr. Vocal Vocal Vocal

Vocal: Dr. Vocal2 Vocal2 Vocal2

Vocal: Dr. Vocal3 Vocal3 Vocal3

Secretario: Dr. Secretario Secretario Secretario

Suplente: Dr. Suplente Suplente Suplente

Suplente: Dr. Suplente2 Suplente2 Suplente2

Realizado el acto de defensa y lectura de la Tesis el día XX de MMMMM de YYYY en la Escuela Técnica Superior de Ingenieros Informáticos

Calificación: _____

EL PRESIDENTE VOCAL 1 VOCAL 2

VOCAL 3 EL SECRETARIO

Agradecimientos

Realizar un trabajo de investigación complejo, como es el caso de una tesis doctoral, no se puede entender si no como un trabajo colaborativo. Es por ello, que a lo largo de las lineas que siguen trataré agradecer y mencionar a todo aquel que ha contribuído, de una forma u otra, en el desarrollo de esta investigación.

Para comenzar, esta tesis habría sido imposible de realizar sin el apoyo y soporte continuo de dos personas que admiro profundamente desde el primer momento en el que las conocí y a las que considero, mis dos padrinos acádemicos, Oscar Corcho y Maria-Esther Vidal. Oscar, gracias por haberme dado la posibilidad de realizar este trabajo bajo tu supervisión en el OEG. Además de haber disfrutado enormemenete con las discusiones que hemos ido teniendo a lo largo de estos cinco años, me gustaría agradecerte la confianza depositada en mi y la libertad que me has otorgado, quizá un poco difícil de gestionar y comprender al comienzo, pero que sin ella, no habría crecido personal y profesionalmente de la manera que lo he hecho. Al final, el tiempo pone a cada uno en el lugar que le corresponde. Gracias por tu paciencia y continua positividad, que me han ayudado a *"ver siempre el lado bueno de la vida"* (como cantaban los *Monthy Python* en "La Vida de Brian"), tan necesario en los momentos complicados y los tiempos que corren. La pasión, el creer en lo que uno hace y el valor del trabajo de investigación lo aprendí de Maria-Esther. Nunca sabré cómo agradecerte el trato recibido, como uno más de tus estudiantes, el tiempo que estuve de estancia en SDM y por todo el trabajo que hemos realizado juntos durante la segunda parte de mi doctorado. Es complicado expresar en tan pocas lineas, el completo y desinteresado apoyo que he recibido por tu parte, así como los valores y enseñanzas sobre el esfuerzo, la perfección en el trabajo y tu entusiasmo en todo lo que haces. Simplemente, gracias.

El sentimiento de pertenencia a un grupo es otra de las claves para poder llegar a completar una tesis con éxito. Más que un grupo, creo que el Ontology Engineering Group es una familia, un gran familia. Las enriquecedoras discusiones y charlas que he tenido con mis "hermanos", Paola y Carlos, sobre el doctorado, la investigación y la vida en general. Gracias al resto de doctorandos: Alba, Serge,

Elvi, Pablo, Julia, etc, a los “seniors” del OEG y ex-OEGs Esteban, Raul Alcazar, Raul García, JARG, Ana Ibarrola, Elena, Miguel Ángel, Maria Póveda, Idafen, Jose Luis, etc. y a Álvaro, mi colega de festivales y cervezas. Pero en especial agradecer el apoyo a Patricia, mi compi de batallas en la vida, de viajes en el coche, de charlas interminables, del “siempre hay que decir si a todo”, de las idas y venidas y las noches madrileñas hasta el amanecer. Asimismo, ha sido un gusto compartir tiempo y experiencias con todos los integrantes del grupo de integración de datos. Especialmente me gustaría agradecer a Edna su ayuda en los artículos que hemos escrito juntos y su paciencia en la revisión esta tesis. Freddy, gracias por tus ideas locas y tu buen humor. Jhon, Ana, Luis, Dani, Julián, Ahmad, Marlene y Andrea he disfrutado enormemente discutiendo, definiendo y logrando retos con todos vosotros.

To the people from Ghent, specially to Pieter and Anastasia, my “postdocs” in the shadows. Thanks for giving me the possibility to work and collaborate with you and your teams. Hope that this is only a promising beginning. Lucie and Sam, I am extremely glad to meet you, first in Bertinoro, and then in Hannover. Lucie you were my support during the good and bad moments of my internship in Germany. I am really happy to have a friend like you in my life. Sam, I have never going to forget our legendary week in Rodhes and all the awesome things that come after it, I really admire your passion and I enjoyed to work with you. Finally, to the rest of the SDM-TIB family: Enrique, Kemele, Ahmad, Ariam, Maria Isabel...

E por último, pero non menos importante, quedariame agradecer a miña xente de sempre, os meus galegos. Meus pais que forón unha continua axuda durante todo esta longa viaxe, deles quédome coa sua forza para afrontar os momentos delicados e a paixón coa que enfrentan o día a día. Mamá, gracias por todas as discusións e o teu punta de vista sobre como debe ser un bo investigador, eres unha referente para min. Papá, ogallá algún día poida desfrutar de cada momento cao mesma intesidade e felicidade coa que o fas tí. Martín, o meu irmán, creo que ainda que en diferentes etapas da vida, durante estes anos, os nosos camiños se xuntaron pouco a pouco, e o teu apoio incondicional sempre foi de gran axuda. Gracias os meus amigos de sempre: Rafa, Marta, Clau, Paula, Erle, Ánxela, Carme, Olga e Minia... Pero tamén ós que me atopei en Madrid: Santi, Patricia, Pamela, Carolina...

Las última lineas van dedicadas a a

Abstract

Resumen

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> Your impact will be as big as the quality of your pitch to explain the solution
>
> *Pieter Colpaert*

Over the last years, a large and constant growth of data have been made available on the Web. These data are published in many different formats such as HTML tables, tabular formats, JSON, PDF documents and web services. Open data portals are one of the most important and successful digital infrastructures that allows governments, public institutions and non-profit organizations to provide a common and unique point where data can be accessible by key stakeholders (e.g, citizens, developers, private companies, etc.). For example, at the time of writing, the European Data Portal[1] aggregates approximately 1,2M datasets from EU countries in a diversity of domains. In this context, the World Wide Web Consortium (W3C) organism has proposed a set of technologies and recommendations, which are the basis for Semantic Web, Linked Data, or Knowledge Graphs (KG) data models. RDF (Brickley *et al.*, 1999) has been proposed as a standard format for data interchange on the Web, and RDF Schema (Brickley *et al.*, 2014) and OWL ontologies (McGuinness *et al.*, 2004) have begun to appear so as to provide shared models in some domains, while SPARQL (Pérez *et al.*, 2009) is defined as the standard query language for these data models. However, the amount of non-RDF data (e.g., CSV, JSON, XML) that are published in these open data portals continues to dominate the scene, and interoperability issues hinder their (re)use and consumption.

Knowledge graphs, as the most relevant exponent of the use of Semantic Web technologies, have gained momentum as a result of this explosion of available data and the demand of ex-

[1] `https://www.europeandataportal.eu/catalogue-statistics/Evolution`

pressive formalisms to integrate factual knowledge spread across various data sources (Hogan *et al.*, 2020). The number of hits per day of public knowledge graphs such as DBpedia[2] and Wikidata[3], as well as the amount of increasing scientific publications referencing these formal models[4], provide evidence of the spectrum of opportunities that they are bringing into the industrial, public administration and scientific landscape. Although these results endorse the success of Semantic Web technologies, they also exhort the development of computational tools to scale up knowledge graphs to the astronomical data growth expected for the next years. The definition of robust methodologies able to integrate these data sources across the Web is the first step that has to be solved for starting to see the Web as an integrated overall database.

Data integration is not a new problem. It was already identified and addressed several decades ago with an emphasis on data stored in relational databases (Wiederhold, 1992), but it is exacerbated by the availability of such amount of heterogeneous data on the Web. Different techniques and tools have been used to address this problem (Gruser *et al.*, 1998; Halevy, 2018; Lenzerini, 2002). In our work, we focus on approaches based on the use of ontologies as global common modes, and Semantic Web technologies, what has been traditionally name as Ontology-Based Data Access (Poggi *et al.*, 2008) and more recently as knowledge graph construction (KGC). Data consumers issue queries over a dataset according to a common unified view (an ontology). The relationship between the ontology and the data sources is usually available in the form of declarative mapping rules. In Ontology Based Data Integration (OBDI) (Poggi *et al.*, 2008), these techniques are expanded to address heterogeneous datasets, whose data need to be integrated to provide answers to these queries. In knowledge graph construction approaches, two different alternatives have traditionally existed to enable data access: (1) materialized KGC: where data are transformed taking into account the mappings and the ontologies (for example, data is converted into RDF and loaded into a triple store, so that it can be natively queried using SPARQL), and (2) virtual KGC: where the transformation is done on the queries using the mapping rules and ontologies, which can then be evaluated on the original data sources.

OBDA/I approaches have been traditionally focused on providing access to data stored in relational databases. With the emergence of the Web and the possibility of publishing data

[2] `https://wiki.dbpedia.org/blog/keep-using-dbpedia`

[3] `https://stats.wikimedia.org/`

[4] `https://pubmed.ncbi.nlm.nih.gov/?term=knowledge+graph`

in diverse formats, the focus is also expanded to other data publication formats, as aforementioned. Efforts in the last two decades have focused in how to define the relationships between the input data (source model) and the ontology (target model). From the very beginning, declarative rules have been proposed with this aim (Auer *et al.*, 2009; Barrasa *et al.*, 2004; Bizer & Seaborne, 2004), but it is not until the standardization of these rules, with the W3C recommendations R2RML (Das *et al.*, 2012a) and Direct Mapping (Arenas *et al.*, 2013), where the construction of knowledge graphs from relational databases attract wider attention from a research point of view. For example, engines and approaches such as Ontop (Calvanese *et al.*, 2017), Morph-RDB (Priyatna *et al.*, 2014) and Ultrawrap (Sequeda & Miranker, 2013) describe optimizations in the query translation process from SPARQL-to-SQL taking into account R2RML (or equivalent) mapping rules, usually based on the standard transformation between these two query languages proposed in (Chebotko *et al.*, 2009; Elliott *et al.*, 2009). The standardization of declarative mapping rules for relational databases also produces the appearance of other mapping specifications with the aim of providing coverage to other kinds of data formats and features. A bit after R2RML was recommended, and because of its use in different types of contexts, new needs and requirements arose, especially in relation to supporting other formats beyond relational databases, and this resulted in the creation of many new mapping languages, such as RML (Dimou *et al.*, 2014) (to deal with CSVs, JSON and XML data sources), xR2RML (Michel *et al.*, 2015) and KR2RML (Slepicka *et al.*, 2015) (to deal with nested data), CSVW[5] (to describe CSV files on the Web), or D2RML (Chortaras & Stamou, 2018a) (for XML, JSON and REST/SPARQL endpoints). In addition to declarative languages, non-declarative mapping languages have also been proposed, such as SPARQL-Generate (Lefrançois *et al.*, 2017), Helio[6] or Tarql[7]. A common situation for those performing the construction of a KG is that they need to provide access to a varied set of heterogeneous data sources, but there are so many different options that and it is difficult to determine which one is better for each situation. **Mapping languages are not necessarily interoperable, and many of them come associated with a very specific engine that supports them**.

One of the most used mapping languages to construct knowledge graphs from heterogeneous data sources is RML (Dimou *et al.*, 2014). There are a good number of materialized KGC engines that have been proposed for parsing RML mapping rules such as CARML[8],

[5] `https://www.w3.org/ns/csvw`
[6] `https://helio.linkeddata.es/`
[7] `https://github.com/tarql/tarql`
[8] `https://github.com/carml/carml`

RMLMapper[9] or RocketRML (Şimşek *et al.*, 2019), but **no one has addressed deeply typical problems that appear in this context, such as scalability and performance in complex data integration scenarios**. RML-based virtual KGC engines have been also proposed (Endris *et al.*, 2019; Mami *et al.*, 2019b), applying query translation techniques for different and heterogeneous data sources. **The proposed optimizations by latest virtual KGC engines are mainly focused on the distribution of the SPARQL queries exploiting the mapping rules**, and they delegate the execution of the queries to external platforms or databases that provide SQL wrappers such as Presto (Bershad *et al.*, 1988), SPARK[10] or Apache Drill (Hausenblas & Nadeau, 2013).

In this context, application developers need to understand the strengths and weaknesses of existing KGC engines, so as to determinate whether these engines cover the requirements of real-use-case scenarios. Evaluation approaches are needed to have a clear and up-to-date overview of these engines. Indeed, several benchmarks focused on testing performance and scalability already exist in the state of the art of virtual knowledge graph construction (Bizer & Schultz, 2009; Lanti *et al.*, 2015). The BSBM benchmark (Bizer & Schultz, 2009) is focused on comparing the performance of SPARQL-to-SQL query translation versus the performance of native RDF Stores, and only considers virtual KGC engines that access relational data stores. The NPD benchmark (Lanti *et al.*, 2015) specifically analyzes requirements related to relational databases as tje data source, query sets, mapping rules and query languages. However, **none of these benchmarks address the requirement of virtual KGC systems over multiple datasets available in heterogeneous formats**. The most recent engines have been evaluated in an ad-hoc manner (Endris *et al.*, 2019; Mami *et al.*, 2019a) and to the best of our knowledge, no benchmarks have been developed to evaluate these proposals in a systematic manner. In materialization techniques, to the best of our knowledge, not even a systematic analysis of the parameters that can affect the performance of these systems has been proposed. Apart from testing performance and scalability, it is important to define conformance test cases of the mapping language specifications. They are able to determine which processor or engine is the most suitable for a certain use case, or provide useful information to the developers for fixing possible issues over their KGC engines. R2RML and Direct Mapping have their test-cases (Villazón-Terrazas & Hausenblas, 2012). However, **no similar work has been proposed**

[9] `https://github.com/semantifyit/RocketRML`
[10] `https://spark.apache.org/`

for testing the conformance of engines that construct knowledge graphs from heterogeneous data sources using declarative mapping rules.

We organize the contributions of this thesis in two different groups. The first group is formed by a set of optimizations techniques for enhancing the construction of knowledge graphs (virtual and materialized) exploiting the information from mapping rules. We first describe the concept of *mapping translation*, together with its main desirable properties. *Mapping translation* describes how mapping rules can be interoperable among the different specifications and extensions. This concept defines the foundations for our proposed optimizations and improvements over knowledge graph construction processes. For materialization techniques, we propose the exploitation of different types of mapping rules to define and implement a set of physical operators to scale-up the construction of knowledge graphs. We also define a preprocessing framework for the efficient execution of functional mappings (mapping rules that contain declarative data transformation functions) formed by a set of data and mapping translation rules. For virtual KGC systems, we describe a set of additional steps over the typical virtual knowledge graph construction workflow for the effective application of domain and integrity constraints over tabular data. The aim of the proposal is to enhance the performance of SPARQL-to-SQL approaches when the input sources are tabular data (e.g., CSV or spreadsheet) together with the improvement of query completeness. Finally, we present a virtual KGC engine able to translate declarative mapping rules into programmed wrappers for allowing the access to legacy systems through non-semantic web query interfaces such as GraphQL.

The second group of the thesis contributions cover the gap of systematic and standard evaluation for knowledge graph construction systems over heterogeneous data sources. We propose the GTFS-Madrid-Bench as a way to evaluate the performance and scalability of different virtual KGC engines, including recent engines able to translate queries over semantic data lakes and most mature engines only focused on SPARQL-to-SQL translation techniques. We are able to evaluate multiple and heterogeneous approaches using the proposed benchmark, assessing the current status of these engines. Additionally, our proposal introduces several scenarios that aim at measuring the query capabilities, performance and scalability using an open data model from the transport domain, the General Transit Feed System of GTFS, covering the most import SPARQL features and generating data at scales with state-of-the-art techniques (Lanti *et al.*, 2017). Finally, we analyze what are the parameters that can impact in the performance of materialization KGC engine, together with the definition of a set of testbeds that include these parameters and an experimental evaluation over two well known RML engines. Additionally,

to cover the lack of test cases in the construction of KG from heterogeneous data sources, we extend the R2RML test cases to take into account other kinds of data formats using RML mappings. We also include the corresponding implementation report of that language[11].

1.1 Thesis Structure

The remainder of the thesis is organized as follows:

- In Chapter 2 we analyze the current state of the art directly aligned with the topics of this work. Semantic Web and its main standard technologies for the construction of knowledge graphs, using declarative mapping languages and ontology-based data integration systems. We identify the limitations of the current approaches, which conduct the contributions of the presented work.

- In Chapter 3 we describe the objectives and main contributions of this thesis. Additionally, we present the assumptions, hypothesis and restrictions of the work.

- Chapter 4 defines the concept and properties of a new generation of knowledge graph construction systems, based on the idea of support semantic interoperability among the different types of declarative mapping specifications, what we call *mapping translation*. Additionally, it provides a set of use cases where this idea has been already applied, with a special focus on the construction of KG in the statistics domain.

- In Chapter 5 a set of evaluation proposals for KG construction engines is described. First, we identify and evaluate what are the parameters that can affect the behavior of the KGC engines in materialization environments together with a set of representative test-cases and testbeds that help to perform this kind of evaluations. Finally, we present a comprehensive benchmark for virtual knowledge graph access, which considers multiple data formats and different data scales. Several engines from the state of the art are evaluated with this benchmark so as to assess the current status of virtual knowledge graph access.

- Chapter 6 describes two proposals to optimize the construction of virtual knowledge graphs exploiting the concept of mapping translation. We first describe a set of optimization techniques to enhance completeness and performance of virtual knowledge

[11] `https://rml.io/implementation-report/`

graph access over tabular data. Second, we present a system that translates declarative mapping rules to programmed query endpoints for providing access to heterogeneous data sources.

- In Chapter 7, we describe a set of optimizations techniques over knowledge graph materialization approaches to provide scalability to this process. First, we describe a set of physical data structures and operators for enhancing KG construction techniques and then, an approach for the efficient pre-processing of functional mappings, i.e., mappings that include declarative definition of transformation functions.

- Finally, Chapter 8 describes the main conclusions of this thesis and identified the future lines of research in the are of semantic data integration and knowledge graph construction.

1.2 Dissemination Results

Our work has been disseminated in the following international workshops, conferences and journals:

- Our contribution in Chapter 4 has been published in: Oscar Corcho, Freddy Priyatna and David Chaves-Fraga: Towards a New Generation of Ontology Based Data Access, Semantic Web Journal 2020: 153-160.

- The implementation over a real use case of the ideas proposed in Chapter 4 has been published in: David Chaves-Fraga, Freddy Priyatna, Idafen Santana-Pérez and Oscar Corcho: Virtual Statistics Knowledge Graph Generation from CSV files, in Proceedings of the 6th International Workshop on Semantic Statistics co-located with the 17th International Semantic Web Conference 2018 (ISWC2018). This paper received the **Best Paper Award** in the workshop.

- The first and second contributions to Chapter 5 have been published in: Pieter Heyvaert, David Chaves-Fraga, Freddy Priyatna, Oscar Corcho, Erick Mannens, Ruben Verborgh and Anastasia Dimou: Conformance test cases for the RDF mapping language (RML), Proceedings of 1st Iberoamerican Knowledge Graphs and Semantic Web Conference 2019 (KGSW2019); and in: David Chaves-Fraga, Kemele M. Endris, Enrique Iglesias,

Oscar Corcho and Maria-Esther Vidal: What are the Parameters that Affect the Construction of a Knowledge Graph?. In OTM Confederated International Conferences On the Move to Meaningful Internet Systems 2019 (OTM2019). These contributions are the results of joint collaborations with Ghent University and German National Library of Science and Technology (TIB), respectively, as a results of research stays in these institutions and further collaboration work in the context of the W3C[12].

- The last contribution of Chapter 5 has been published in: 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. Journal of Web Semantics 2020, 65: 100596. 2020[13]. Following Open Science principles all code, datasets and queries are available, to facilitate reproducibility and further extensions.

- The first part of our contribution to Chapter 6 has been published in: David Chaves-Fraga, Edna Ruckhaus, Freddy Priyatna, Maria-Esther Vidal and Oscar Corcho: Enhancing Virtual Ontology Based Access over Tabular Data with Morph-CSV, Semantic Web Journal, 2021.

- The second part of our contribution to Chapter 6 has been published in: Freddy Priyatna, David Chaves-Fraga, Ahmad Alobaid, and Oscar Corcho: morph-GraphQL: GraphQL Servers Generation from R2RML Mappings, Proceedings of the 31st International Conference on Software Engineering & Knowledge Engineering (SEKE2019). The extended version of this research, presented in this thesis, has been published in: David Chaves-Fraga, Freddy Priyatna, Ahmad Alobaid and Oscar Corcho: Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL. International Journal of Software Engineering and Knowledge Engineering, 2020: 785-803.

- The first contribution to Chapter 7 has been published in: 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, Proceedings of the 29th ACM International Conference on Information and Knowledge Management

[12] https://www.w3.org/community/kg-construct/
[13] https://doi.org/10.1016/j.websem.2020.100596

2020 (CIKM2020). These contribution is the results of joint collaborations with the German National Library of Science and Technology (TIB), as a results of research stay in the institution and the first three authors have contributed equally to the research.

- The second part of our contribution to Chapter 7 has been published in: Samaneh Jozashoori, David Chaves-Fraga, Enrique Iglesias, Maria-Esther Vidal and Oscar Corcho: FunMap: Efficient Execution of Functional Mappings for Knowledge Graph Creation, Proceedings of the 19th International Semantic Web Conference 2020 (ISWC2020). These contribution is the results of joint collaborations with the German National Library of Science and Technology (TIB), as a results of research stay in the institution and the first two authors have contributed equally to the research.

Chapter 2

State of the Art

In this chapter, we discuss the current state of the art in knowledge graph construction using declarative mapping rules. We provide an overview of approaches, techniques and methodologies for constructing and querying (virtual) knowledge graphs based on semantic technologies. We describe the declarative annotations and mapping languages specifications that have been proposed to construct these kind of data models together with their main features. We also present the current methodologies to evaluate the quality of knowledge graph construction engines such as benchmarkings and test-cases.

More in detail, in Section 2.1 we describe the foundations of data integration concept and its relation with the theoretical contributions where the global schema is defined by an ontology. Section 2.2 summarizes the languages used for representing and querying data on the Semantic Web (i.e., the RDF model and the SPARQL query language). Section 2.3 presents an overview of the most important contributions for representing declarative annotations with a focus on the representations of mapping rules and data constraints. Additionally, Section 2.4 describes the most relevant knowledge graph construction systems and their corresponding optimizations while Section 2.5 finally, describes evaluation methodologies used for these approaches. We conclude in Section 2.6 with a summary and conclusions on the current gaps, which will help to understand the contributions of this thesis.

2.1 Data Integration and OBDA

A data integration system (DIS) is defined as the process of making available a set of different sources through a common view (Lenzerini, 2002). Formally, it is described as $DIS = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$,

where:

- $\mathcal{G}$ is the *global schema*, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$.
- $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$.
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of assertions matching queries over $\mathcal{S}$ and $\mathcal{G}$, in order to establish correspondences between concepts in both schemes.

Different types of data integration systems have been proposed in the literature and are applied in research and data scenarios. Data warehouses (Vassiliadis, 2009) are proposed to integrate multiple and heterogeneous data sources in a centralized storage system. The process of feeding such system with data is usually known in enterprises ecosystems as an *extract-transform-load* process (ETL). In contrast, mediators (Wiederhold, 1992) have been proposed as another data integration approach where the data remains in the data sources. For accessing the data, queries defined over the global schema are translated to the source schema and executed. Multiple systems that implement these ideas with their corresponding optimizations (Chawathe *et al.*, 1994; Rajaraman *et al.*, 1996; Roth & Schwarz, 1997).

Ontology based data access (OBDA) and integration (OBDI) are data integration systems where the global schema is defined by an ontology (Poggi *et al.*, 2008). The formal framework presented in (Xiao *et al.*, 2018a) defines an OBDA specification as a tuple $P = \langle O, S, M \rangle$ where O is an ontology, S is the source schema, and M a set of mappings. Additionally, an OBDA instance is defined as a tuple $PI = \langle P, D \rangle$ where P is an OBDA specification and D is a data instance conforming to S. The main difference between OBDA and OBDI systems is that in OBDA D is fixed to a single data source in a specific data format, while OBDI extends D to cover heterogeneous data sources and formats. In both ontology-based approaches, two different alternatives exist to enable data access: (1) those where data are materialized taking into account the mappings and the ontologies, what can be seen as a ontology-based ETL process, and (2) those where the transformation is done on the queries, which can then be evaluated on the original data sources, so, it can be defined as a kind of mediator system. In this work we refer to the first alternative as a materialized knowledge graph construction process, while the second alternative is defined as a virtual knowledge graph construction one. As we are focused on creating KG from heterogeneous data sources, both alternatives can be described as an OBDI approach.

2.2 Representation and Query Languages for the Semantic Web

In this section, we provide an overview of the core semantic web languages that have a relevant role during a knowledge graph construction process. We discuss in detail the Resource Description Framework (RDF) (Cyganiak *et al.*, 2014), used for implementing the target data model in the DIS (i.e., the ontology), but also usually used for declaring mapping rules (Das *et al.*, 2012a; Dimou *et al.*, 2014; Michel *et al.*, 2015). We also provide a detailed explanation of the SPARQL query language (Harris & Seaborne, 2013) as it is an essential input for virtual knowledge graph construction techniques (e.g., translating SPARQL to SQL).

2.2.1 RDF: Resource Description Framework

The Resource Description Framework (RDF) (Cyganiak *et al.*, 2014) is the basic data model used in the Semantic Web. Basically, RDF is defined by a set of *triples*, where each triple is represented in the form of $< s, p, o >$ where s is the subject, p the predicate and o the object. One of the main features of RDF is the used of IRIs to represent concepts and the relationships between them. This allows having unique identifiers across the Web for each resource. Assuming that there are a pairwise disjoint infinite I, B and L (IRIs, blank nodes and literals respectively), we can formally defined an RDF triple as $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ (Pérez *et al.*, 2009). Additionally, we can define G as an RDF graph composed by a set of RDF triples.

Listing 2.1 shows an example of an RDF graph with a set of triples in the transport domain (shown in its graphical representation in Figure 2.1). As we can observe, there are, in total, 4 different triples (or statements), all of them with the same subject, defined using the URI *http://example.org/transport/trip/1*. We can also observe that all the predicates of the triples (e.g., http://vocab.gtfs.org/terms#stop) are also defined by an IRI. More in detail, the first triple declares that *http://example.org/transport/trip/1* is a trip defined in the LinkedGTFS vocabulary[14]. The predicate `rdf:type` is used in the RDF data model for classify the resources. Then, predicates `gtfs:shortName` and `gtfs:wheelchair` represent the information about the name of the trip and if it is accessible or not using a wheelchair. The objects of these two triples are literals in RDF that can be either simple string ("Puerta del Sur") or typed literal ("false"^^xsd:boolean). The typed literals contain data values with a tag for representing its

[14] `http://vocab.gtfs.org/terms`

Listing 2.1: Example of RDF graph

```
@prefix gtfs: <http://vocab.gtfs.org/terms#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/trip/1> rdf:type gtfs:Trip .
<http://example.org/trip/1> gtfs:stop <http://transport.org/stop/360> .
<http://example.org/trip/1> gtfs:shortName ”Puerta del Sur” .
<http://example.org/trip/1> gtfs:wheelchair ”false”^^xsd:boolean .
```

data type. Finally, `gtfs:stop` indicates one of the trips stops, using and IRI, which means that the stop is also a resource.

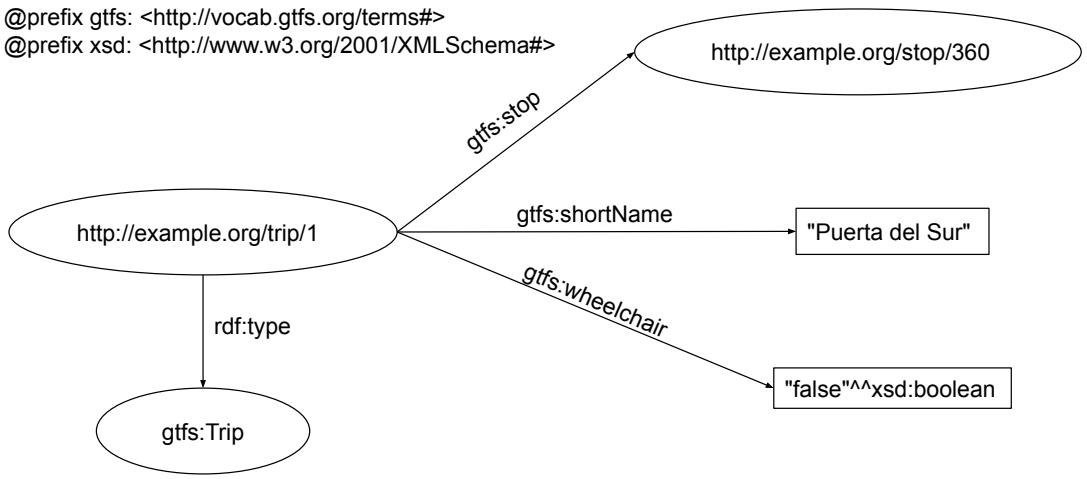


Figure 2.1: Graphical representation of an RDF Graph

Several formats can be used to serialize an RDF graph. RDF/XML[15] was the first serialization to be proposed and it is supported by XML. Notation 3 (N3) provides a human readable serialization, although currently is not used widely. N-Triples[16] and Turtle[17] are subsets of N3, more widely adopted. Finally, the last serialization that has been proposed as W3C recommendation is JSON-LD[18]. Since it is based on JSON, it facilitates the consumption of RDF data to developers and practitioners as it encapsulates the RDF in a standard JSON document. There

[15] `https://www.w3.org/TR/rdf-syntax-grammar/`
[16] `https://www.w3.org/TR/n-triples/`
[17] `https://www.w3.org/TR/turtle/`
[18] `https://www.w3.org/TR/json-ld11/`

Listing 2.2: Example of SPARQL query

```
PREFIX gtfs: <http://vocab.gtfs.org/terms#> .

SELECT ?name WHERE {
    ?trip rdf:type gtfs:Trip .
    ?trip gtfs:shortName ?name .
}
```

are other serialization of RDF that are not recommended by W3C but are highly adopted for specific purposes. For example, HDT (Fernández *et al.*, 2013) is a binary serialization of RDF for publishing and exchanging RDF data at large scale.

2.2.2 SPARQL

SPARQL is the W3C recommendation graph matching query language for RDF graphs. For defining the syntax of the query language we use the work presented in (Pérez *et al.*, 2009), which defines the SPARQL graph patterns recursively as follows:

- A tuple $(I \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a graph pattern (triple pattern if it is single), where I is an IRI, V is a variable and L a literal.
- If P_1 and P_2 are graph patterns, then expression (P_1 AND P_2), (P_1 OPT P_2) and (P_1 UNION P_2) are also graph patterns.
- If P is a graph pattern and R is a SPARQL built-in condition, then (P FILTER R) is also a graph pattern.
- If P is a graph pattern and T is a finite set of variables, then (SELECT T WHERE P) is a graph pattern.

Given the RDF graph shown in Figure 2.1, we may be interested in obtained the name of the trips of our graph. In Listing 2.2 we depict the SPARQL query used for obtaining the desirable result-set. The first part of the query defines the prefixes that can be used in the query. Then, the SELECT operator indicates the set of variables that will be projected from the graph pattern in query result-set (?name). Variables in SPARQL are represented using the question mark before the variable name (in the example ?name or ?trip are variables). During the query evaluation

process, the variables are bound to the different values of the RDF graph that match with the triples pattern (formal definition described in (Pérez *et al.*, 2009)). In the example, the variable ?trip is bound to *http://transport.org/trip/1* as there are two facts in the graph that match with the two triple patterns defined in the query. Then, variable ?name is bound to "Puerta del Sur" and retrieved in the SPARQL result-set, usually represented in the form of a table where each column contains the values of each projected variable (see Table 2.1).

?name
"Puerta del Sur"

Table 2.1: Example of a SPARQL result-set

The SPARQL built-in conditions are constructed using the elements of the set ($I \cup U$) and constants. These constants can take different values such as logical connectives ($\neg, \wedge, \vee$), inequality or equality symbols ($\leq, \geq, <, >, =$) and unary predicates such as bound, isBlank, isIRI, etc. (Harris & Seaborne, 2013). The can be combined with the aforementioned SPARQL operator such as UNION, FILTER or OPTIONAL. For example, the OPTIONAL operator attracted much attention on virtual knowledge graph construction, due the difficulties on its efficient translation to SQL (Xiao *et al.*, 2018b).

In SPARQL there are many type of query forms based on graph pattern matching that allow to evaluate several query types. As we shown in Listing 2.2 the SELECT clause is used to select elements from the data. DESCRIBE query returns information about the resources that match a graph pattern in form of another RDF graph. CONSTRUCT is used to construct a new RDF graph based on the graph pattern indicated in the WHERE clause. ASK is a boolean-based query that evaluates true if there is at least one solution for the provided pattern. Additionally, we can include clauses such as SERVICE, that allows to query external RDF graphs, and GRAPH, that specifies the RDF graph where the query will be performed over the SPARQL endpoint.

2.3 Declarative Annotations for Knowledge Graph Construction

Annotations are one of the main components used for the construction of knowledge graphs. We define annotation as mapping rules, that relate the target model with the input sources, and constraints, which allow: i) defining ad-hoc transformation functions that permit the cleaning and preparation of the input data; and ii) describing the content of the input source. Constraints

are essential during a knowledge graph construction process as it is able to deal with the typical features of heterogeneous data sources such as the absence of a well-defined and fixed data schema, a normalized database instance or the non-explicit declarations of relations among the sources. We start this section discussing existing approaches for the design of mappings. Then, we describe the current mapping language specifications, some of which have been standardized by W3C. Finally, we present approaches to define, declaratively, constraints over a DIS.

2.3.1 Mapping Rules

In a DIS, the mapping layer contains information about how the input sources are related with the target model. There are two basic approaches for defining mapping rules in a data integration system: Local as a View (LAV) and Global as a View (GAV). In KG construction, the usual approach followed to define these rules is the Global as View one. We describe in detail each proposal and continue with the specific mapping languages, summarizing them in Table 2.3

2.3.1.1 Local as a View Mapping rules (LAV)

In (Ullman, 1997) the elements of the source schema S are mapped to a query Q_G over the target schema G. The main benefits of this approach is that it supports continuous changes of the source schema (e.g., adding new sources or modify their underlying representation) since there is no need to change the query processing component. Thus, LAV is usually useful when the global schema G is stable but the local schema S may suffer modifications over time. However, one of its main disadvantages is that it cannot represent the source S completely if it is not modeled in the global schema, hence, the approach usually provides partial answers for a query Q_G. Query translation following this approach is not a trivial process, as the Q_G has to be translated into an equivalent query over the source schema S. These techniques are usually known as query translation using views (Halevy, 2001).

Consider the following example of a LAV mapping:

- The global schema G is defined by a class $Player(ID)$ with two additional properties: $name(ID, playerName)$ and $sport(ID, sportName)$
- The source schema S is defined by two relations: $BasketballPlayer(BID, BName)$ and $TenisPlayer(TID, TName)$

- The LAV mapping between G and S is defined as:
  $BasketballPlayer(BID, BName) \rightsquigarrow \{< x,y > | Player(x) \wedge name(x,y) \wedge sport(x,''basketball'')\}$
  $TennisPlayer(TID, TName) \rightsquigarrow \{< x,y > | Player(x) \wedge name(x,y) \wedge sport(x,''tennis'')\}$

2.3.1.2 Global as a View Mapping Rules (GAV)

Global as view are defined in (Halevy, 2001), where each element of the global schema G is mapped to a query over Q_S the source schema S. Opposed to the LAV approach, the benefits of following a GAV approach is that it supports changes over the global schema G, as the queries are defined following the source schema S. Although there are no theoretical limitations to provide access to other data formats, ontology-based data integration processes have been traditionally focused on allowing the integration of relational databases as source schema, based on SPARQL-to-SQL translation techniques. Due to the aforementioned limitations in these techniques for LAV approaches, most of the semantic web mapping rules specifications following the GAV approach (e.g., R_2O, DR2Q, R2RML).

Consider the following example of a GAV mapping:

- The global schema G is defined by a class AmateurTenisPlayer(ID)
- The source schema S is defined by the relation TenisPlayer(ID,Name,Level)
- The GAV mapping between G and S is defined as:
  $AmateurTennisPlayer(ID) \rightsquigarrow \{< x > | TennisPlayer(x,y,''Amateur'')\}$

2.3.1.3 The W3C Recommendation R2RML

Since 2012, R2RML is a W3C recommendation to specify declarative mapping rules between RDF and RDB (Das *et al.*, 2012a). These rules are defined in an R2RML mapping document, which is formed by a set of Triples Maps (`rr:TriplesMap`). Usually, each Triple Map defines the rules for generating the entities and their properties of a defined class in the ontology and are defined as:

- one logical table (`rr:LogicalTable`) that specifies the source relational table/view
- one subject map (`rr:SubjectMap`) that specifies how to generate the subjects of the triples and the corresponding class.

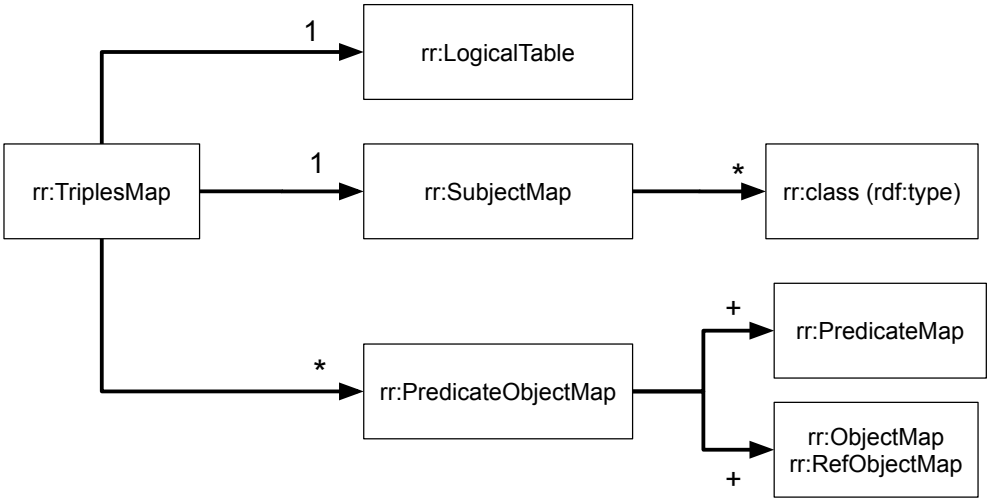


Figure 2.2: R2RML structure with its relevant properties (Das *et al.*, 2012a)

- any number of predicate-object maps (`rr:PredicateObjectMap`). A predicate-object map (POM) is formed by one or many predicate maps (`rr:PredicateMap`), and one to many object maps (`rr:ObjectMap`) or reference-object maps (`rr:RefObjectMap`). This last one is used when joins among logical sources are defined.

Figure 2.2 shows the basic structure of an R2RML Triples Map, with the relations between its main properties and their cardinalities. `rr:SubjectMap`, `rr:PredicateMap` and `rr:ObjectMap` are defined as term maps (`rr:TermMap`. This property is used to generate the desirable RDF terms, either as IRIs (`rr:IRI`) Blank Nodes (`rr:BlankNode`) or literal (`rr:Literal`). The values of the term maps can be specified using the following properties: `rr:Constant` for constant values, `rr:Column` for values obtained directly from a column of a table or `rr:Template` for the ones that are a concatenation between a string and a column reference, for example, to generate subject IRIs. Furthermore, additional information can be provided such as the language of a literal, using the `rr:Language` property, or its corresponding datatype with `rr:Datatype`. Figure 2.3 gives an overview of the R2RML term map.

We provide a complete example of the construction of a knowledge graph through an R2RML mapping. The input table (Figure 2.4) represents information about the stop times for a given trip of the Madrid's metro system. It provides the corresponding identifier for the trip and each stop together with the pickup time and type. Figure 2.5 shows the R2RML mapping for

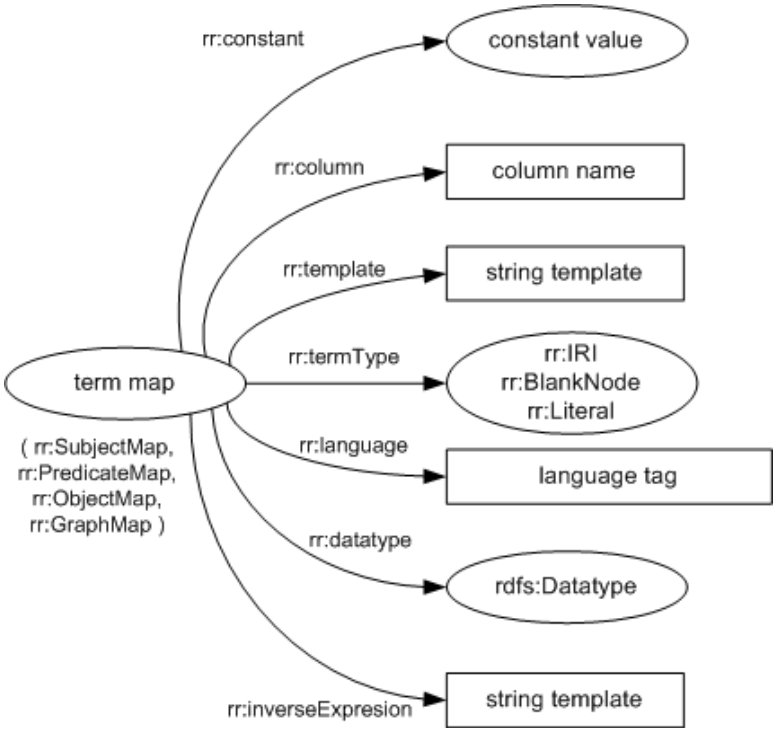


Figure 2.3: R2RML `rr:TermMap` overview (Das *et al.*, 2012a)

transforming the input table to a RDF dataset following the LinkedGTFS[19] ontology. We can observe that the subject map uses an `rr:template` with reference to three different columns of the table to generate unique identifiers for the IRIs. The set of `rr:PredicateObjectMap` (POM) defines the rules to generate three different properties of the `gtfs:StopTime` class. The `gtfs:arrivalTime` POM uses the `rr:dataType` property to declare the type of the input data, `gtfs:pickupType` uses the `rr:template` to generate an uri-based literal in the output dataset, and finally, `gtfs:trip` predicate has a `rr:RefObjectMap` to generate its corresponding object, referencing to the corresponding IRI trip defined in antoher TriplesMap. Finally, the output RDF dataset is shown in Figure 2.6.

2.3.1.4 RML: Extending R2RML for Heterogeneous Data

The RDF Mapping Language (RML) (Dimou *et al.*, 2014) extends the R2RML mapping specification to cover other kinds of data formats such as CSV, XML or JSON, hence, this specification is one of the most used for constructing knowledge graph from heterogeneous data

[19] `https://lov.linkeddata.es/dataset/lov/vocabs/gtfs`

trip_id	arrival_time	stop_id	pickup_type
4_I12-001_I12_1	00:00:00	par_4_263	0
4_I12-001_I12_1	00:03:10	par_4_262	0
4_I12-001_I12_1	00:05:17	par_4_261	0

Figure 2.4: Excerpt of a Stop-times table

```
@prefix gtfs: <http://vocab.gtfs.org/terms#>.

<TriplesMapStoptimes>
        rr:logicalTable [ rr:tableName "stop_times"; ];
        rr:subjectMap [
                rr:template "http://example.com/stoptimes/{trip_id}-{stop_id}-{arrival_time}";
                rr:termType rr:IRI; rr:class gtfs:StopTime;
        ];
        rr:predicateObjectMap [
                rr:predicateMap [ rr:constant gtfs:arrivalTime ];
                rr:objectMap[ rr:column "arrival_time"; rr:datatype xsd:time; ];
        ];
        rr:predicateObjectMap [
                rr:predicateMap [ rr:constant gtfs:pickupType ];
                rr:objectMap[ rr:template "http://example.com/PickupType/{pickup_type}"; ];
        ];
        rr:predicateObjectMap [
                rr:predicateMap [ rr:constant gtfs:trip ];
                rr:objectMap [
                        rr:parentTriplesMap <trips>;
                        rr:joinCondition [ rr:child "trip_id"; rr:parent "trip_id"; ];
                ];
        ];
```

Figure 2.5: R2RML mapping for stop-times table

sources. In this section we explain the main differences between RML and R2RML and then we describe another serialization of the RML mappings following a YAML syntax, and its main properties (Heyvaert *et al.*, 2018).

Summarized in Table 2.2, the main differences between R2RML and RML are:

Logical Source. A Logical Source extends R2RML's Logical Table and describes the input data source used to generate the RDF. The Logical Table is only able to describe relational

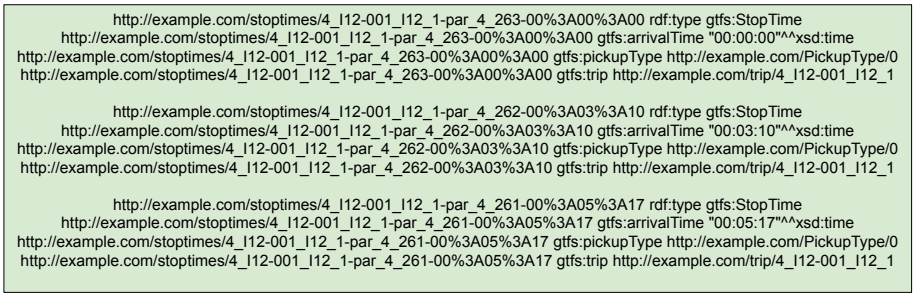


Figure 2.6: Output RDF dataset after evaluating the R2RML document on the original data source

	R2RML	**RML**
Input reference	Logical Table	Logical Source
Data source language	SQL (implicit)	Reference Formulation (explicit)
Value reference	column	Logical reference (valid expression acc. Reference Formulation)
Iteration	per row (implicit)	per record (explicit – valid expression acc. Reference Formulation)

Table 2.2: The differences between R2RML and RML (Anastasia, 2020)

databases,whereas the Logical Source defines different heterogeneous data sources, including relational databases.

Reference Formulation. As RML is designed to support heterogeneous data sources. Data in a specific format can be parsed according to the grammar of a certain formulation (e.g., path and query languages or custom grammars). For example, one can refer to data in an XML file via XPath and in a relational database via SQL. To this end, the *Reference Formulation* was introduced indicating the formulation used to refer to data in a certain data source.

Iterator. In R2RML processors iterate over each row to generate RDF. However, as RML is designed to support heterogeneous data sources, the iteration pattern cannot always be implicitly assumed. For example, iterating over a specific set of objects is done by selecting them via a JSONPath expression. To this end, the *Iterator* was introduced which determines the iteration pattern over the data source and specifies the extract of data used to generate RDF during each iteration. The iterator is not required to be specified if there is no need to iterate over the input data.

Logical Reference. When referring to values in a table or view of a relational database, R2RML relies on column names. However, as RML is designed to support heterogeneous data sources, rules may also refer to elements and objects, such as in the case of XML and JSON. Consequently, references to values should be valid with respect to the used reference formulation. For example, a reference to an attribute of a JSON object should be a valid JSON-Path expression. To this end, (i) the `rml:reference` is introduced to replace `rr:column`, (ii) when a template is used, via `rr:template`, the values between the curly brackets should have an expression that is valid with respect to the used reference formulation, and (iii) `rr:parent` and `rr:child` of a Join Condition should also have an expression that is valid with respect to the used reference formulation.

YARRRML (Heyvaert *et al.*, 2018) is a serialization of the RML mappings based on YAML syntax (Ben-Kiki *et al.*, 2001). The aim of YARRRML is to help users in the creation of mapping rules using a human-readable approach. Figure 2.7 shows an example of the mapping rules for constructing the KG based on the Stop Times CSV file show in Table 2.4. The main keys used in a YARRRML-based mapping are `prefixes` to define the used prefixes and `mappings` where the rules are specified. Each `rr:TriplesMap` is defined with a key defined by the user (stoptimes in the example), and then three main keys have to be declared: `sources` for the input sources, `s` for generating the subject and then `po` for the `rr:PredicateObjectMap` properties. In the same manner as in [R2]RML, the defined identifier for the `rr:TriplesMap` is used for the `rr:RefObjectMap` declarations (e.g., trips in Figure 2.7). Additionally, this approach provides a translator to RML[20], so that any RML-compliant engine can be used when the rules are defined following this serialization.

2.3.1.5 Other approaches to Construct Knowledge Graphs

There are a set of mapping languages and engines for constructing knowledge graphs that include special features. Instated of being a general solution to be applied in a declarative manner to any kind of data format, there are some proposals that focus on providing support to concrete issues that can appear in some data formats or extend other semantic web technologies for defining these rules such as SPARQL (Harris & Seaborne, 2013) or ShExML (Prud'hommeaux *et al.*, 2014).

xR2RML. The xR2RML proposal (Michel *et al.*, 2015) extends R2RML to describe mappings from NoSQL databases (e.g., MongoDB) to RDF. It also includes some of the properties from

[20] `https://rml.io/yarrrml/matey/`

Figure 2.7: YARRML mapping example based for transforming Stop Times table

RML such as the `rml:iterator`. It contains three distinctive features to deal with issues that usually appear when hierarchical documents (XML or JSON) have to be mapped to an RDF model:

- **Access to outer fields.** When a hierarchical structure has to parse to RDF, sometimes it is necessary to combine data from different levels of the tree. xR2RML incorporates the `xrr:pushDown` property with that aim. It allows the declaration of an external field outside the iterator defined, so it can be used in any part of the mapping. It is declared inside the logical source object with two properties. `xrr:reference` which defines the data reference of the value and `xrr:as` which declare the alias to be used within the mapping.

- **Dynamic language tag.** xR2RML extends the `rr:language` property from R2RML and replaces it by `xrr:languageReference`. This property allows to define the language of an object dynamically, using a reference that can get the value from the database.

Features/Mappings	R2RML	RML	xR2RML	SPARQ-Generate	ShExML
Format	RDB	Heterogeneous Datasources	Tree-based formats	Heterogeneous Datasources	Heterogeneous Datasources
Type	Declarative	Declarative	Declarative	Procedural	Declarative
Special Features	W3C Recommendation	Integration with CSVW and FnO	Access outer fields Dynamic Language RDF Lists	Native support for streaming data	-
Based on	-	RML	R2RML and RML	SPARQL	ShEx
Transformation Functions	-	Function Ontology	-	SPARQL Functions	-

Table 2.3: Summary of Mapping Languages specifications

- **RDF lists and containers.** In formats such as JSON it is very common to have values that are modeled as arrays of data. This kind of data structures are also allowed in RDF using `rdfs:Container` with its corresponding subclasses `rdf:Bag`, `rdf:Seq` and `rdf:Alt` when there is not order, and the `rdf:List` class is used when there is an order. xR2RML extends the term map types including `xrr:RdfList`, `xrr:RdfBag`, `xrr:RdfSeq` and `xrr:RdfAlt` that allow the construction of these data structures in RDF.

SPARQL-Generate. The solution proposed in (Lefrançois *et al.*, 2017) presents a template-based language to construct knowledge graphs extending SPARQL 1.1. It exploits the representation capabilities from SPARQL to declare the transformation rules inside the query. The approach of using SPARQL to define the transformation rules has been also proposed in previous approaches focused on specific data formats such as Tarql for CSV files[21]. However, this kind of solution follows a procedural approach to define rules, instead of a declarative one. Therefore, and in comparison with the rest of proposals, they cannot exploit the benefits of declarative definition of rules such as the maintainability, reproducibility and understandability.

ShExML. In (García-González *et al.*, 2020), the authors propose a solution based on the validation language for knowledge graphs, ShEx (Prud'hommeaux *et al.*, 2014). Although it is based on this specification, it uses its own syntax and grammar. In comparison to R2RML-based proposals, ShExML separates the declaration, how to extract the data from the input sources, from the shapes, how to generate the desirable RDF graph. The main objective of this proposal is to help users to create the rules, and it also provides a translation engine to transform ShExML rules to RML mappings.

[21] `https://tarql.github.io/`

2.3.2 Declarative Constraints: Transformation Functions and Metadata

Data constraints play a key role during a data integration process (Calì *et al.*, 2002). They allow validating an input dataset *D* against a schema *S*. Previous proposals of knowledge graph construction over relational database (OBDA) such as Morph-RDB (Priyatna *et al.*, 2014) and Ontop (Calvanese *et al.*, 2017), assume the existence of integrity constraints explicitly defined over the schema *S* to propose their optimizations in SPARQL-to-SQL transformation processes. Additionally, mapping recommendations (i.e., R2RML) for RDB2RDF approaches declare that cleaning or preparation steps are not part of the KG construction process and they have to be performed before running it. However, during the construction of a KG from heterogeneous data sources, these data may not be normalized, and information about relationships or column/key names are not always descriptive or homogeneous, among other possible issues. Hence, data consumers are usually forced to apply ad-hoc or manual data wrangling processes to consume these kind of data. In order to try to avoid manual and not reproducible cleaning/preparation steps, there are a set of declarative proposals to allow the description of constraints over data on the web. Specifically, we will be focused on two different ways to declare constraints: extensions of mapping specifications for including the possibility to define transformation functions inside the mapping rules and metadata to describe data content on the web.

2.3.2.1 Transformation Functions in Mappings

Rahm and Do (Rahm & Do, 2000) reported the relevance of data transformations expressed with functions during data curation and integration. Grounding on this statement, different approaches have been proposed for facilitating the definition of functions to enhance data curation (e.g., (Galhardas *et al.*, 2001; Gupta *et al.*, 2012; Raman & Hellerstein, 2001)). Similarly, declarative languages have been proposed to allow for the definition of functions in the mappings. An approach independent of a specific implementation context is described in (De Meester *et al.*, 2019). It enables the description, publication and exploration of functions and instantiation of associated implementations. The proposed model is the so called Function Ontology (De Meester *et al.*, 2016) and the publication method follows the Linked Data principles. It is used as an extension over RML mapping rules to allow declaring declaratively these transformation functions (De Meester *et al.*, 2017). More in detail, to integrate RML and FnO, the authors create a new type of term map `fnml:FunctionMap`. This new term

Listing 2.3: RDF graph generated by RML+FnO mapping

```
@prefix gtfs: <http://vocab.gtfs.org/terms#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://transport.org/trip/1> rdf:type gtfs:Trip .
<http://transport.org/trip/1> gtfs:stop <http://transport.org/stop/360> .
<http://transport.org/trip/1> gtfs:headsign ‘‘puerta del sur’’ .
<http://transport.org/trip/1> gtfs:wheelchair ‘‘false’’^^xsd:boolean .
```

map defines how a function has to be executed with its definition and parameters following the Function Ontology. The alignment between RML and FnO is possible as they are both declarative and described in RDF. Previous works related to this topic focus on developing ad-hoc and programmed functions. For example, R2RML-F (Debruyne & O'Sullivan, 2016) and FunUL (Junior *et al.*, 2016a,b) allow using functions in the value of the `rr:objectMap` property, so as to modify the value of the cells from a relational database first (R2RML-F) and other kinds of formats after (FunUL). KR2RML (Slepicka *et al.*, 2015), used in Karma, extends R2RML by adding transformation functions in order to deal with nested values. OpenRefine enables such transformations with the usage of GREL functions, which can be used in its RDF extension.

Figure 2.8 shows an example of the integration between the RML mapping language and the Function Ontology following the YARRRML specification. We can observe that the object of the predicate `gtfs:headsign` is defined as `fnml:FunctionMap` term map, where we declarative indicate the function that has to be executed (`grel:lowercase`), with its corresponding parameters. In this example, the function only needs one parameter that is the reference to the column `stop_headsign` from the Stop_Times input source. Listing 2.3 depicts how the output RDF graph after the application of the RML+FnO, where the `grel:lowercase` function has been applied over the object value of `gtfs:headsign`.

2.3.2.2 Metadata for data on the web

Providing metadata for describing the content of a data source (e.g., data on the web) is a useful manner to obtain relevant information of a specific dataset that may be relevant during the construction of a knowledge graph. For example, DCAT (Maali *et al.*, 2018) is a W3C recom-

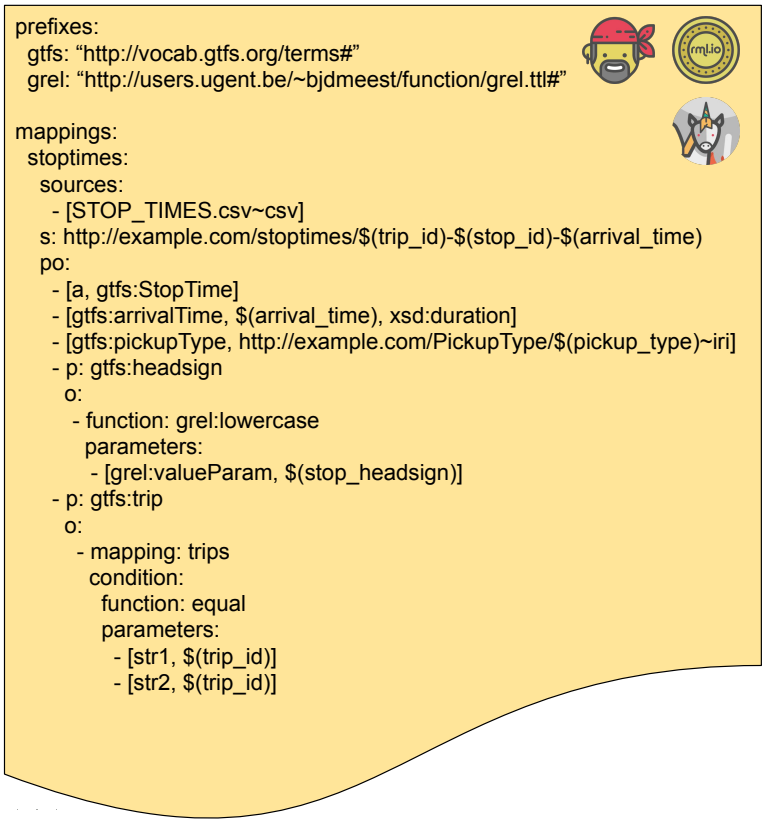


Figure 2.8: YARRML mapping integrated with transformation functions following the FnO

mendation to describe metadata from data catalogs. This standard is used, with some extension, in most of the open data portals across EU to describe the content of their datasets. Another relevant W3C recommendation for describing metadata is CSV on the Web (Tennison *et al.*, 2015). The recommendation[22] defines a model for providing metadata on CSV files and other tabular data, such as datatypes, valid values, data transformations, and primary and foreign key constraints. A related W3C proposal[23] defines a procedure and rules for the generation of RDF from tabular data and a few implementations that refer to this proposal are already available such as COW[24].

Taking into account this metadata during the process of KG construction from tabular datasets can help to enhance the process in terms of completeness and performance. It also allows dealing with the typical heterogeneity issues that appear in these kinds of data: data

[22] http://www.w3.org/TR/tabular-metadata/

[23] https://www.w3.org/TR/csv2rdf/

[24] `https://csvw-converter.readthedocs.io/`

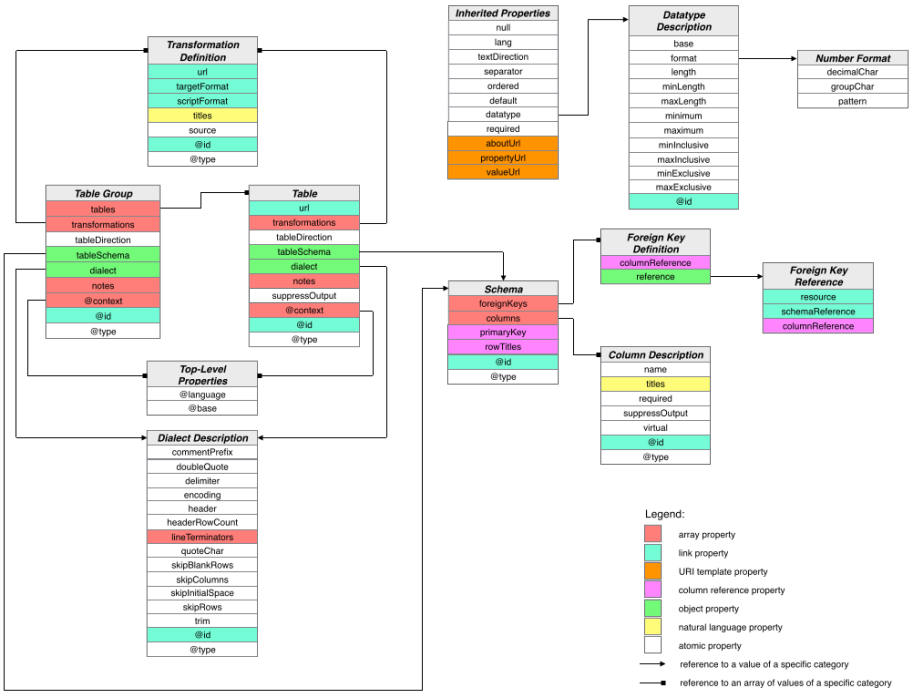


Figure 2.9: The CSV on the Web model(Tennison *et al.*, 2015)

may not be normalized, information about relationships or column names are not always descriptive or homogeneous, missing (meta)data, etc. Figure 2.9 shows the complete data model for describing tabular content on the web. It has been splinted in several parts that can describe the schema of the dataset, dialect annotations (e.g., delimiter value or the prefix used for comments) and information about datatype and related properties such as `csvw:null` (for null values) and `csvw:lang` (for the language). Each of these properties can be defined as an array, link, URI template, column reference or atomic value among others. The data model is materialized in a specific dialect of JSON-LD created for it, so that metadata annotations can be also expressed as an RDF graph, facilitating the interoperability of this proposal with others such as RML or R2RML. In Figure 2.10 we show an example of CSVW annotations for our running example following the JSON-LD syntax. It provides useful information about the datatypes, column names or boolean values for the trips of a GTFS feed.

Figure 2.10: An example of CSVW annotations over Transport Domain

2.4 Knowledge Graph Construction Engines

Academic and industrial solutions for knowledge graph construction from structured and semi-structured data are gaining momentum. Since 2012, with the appearance of the W3C recommendation R2RML (Das *et al.*, 2012a), diverse proposals, and optimization techniques have provided support for these transformations. We describe in detail engines for both groups of approaches, materialization and virtualization.

2.4.1 Materialized Knowledge Graph Construction Engines

With RML (Dimou *et al.*, 2014) proposed as an extension of R2RML for describing mapping rules over diverse data formats, multiple tools have been developed for the creation of knowledge graphs. There are multiple RML parsers[25]: CARML[26] executes RML rules and includes additional features like MultiTermMap (to deal with arrays) and XML namespace (to improve XPath expressions). RocketML (Şimşek *et al.*, 2019) is an RML engine implemented using

[25] `https://rml.io/implementation-report/`
[26] `https://github.com/carml/carml`

the NodeJS framework that pre-processed join conditions to enhance the performance of the process. RMLMapper[27] is another RML engine implemented in Java and extensively used to obtain high-quality KGs. Additionally, RMLStreamer (Haesendonck *et al.*, 2019) is an RML engine that is able to generate knowledge graphs from streaming data. There are other engines that are able to construct knowledge graphs from heterogeneous data sources without the support of mapping rules. For example, SPARQL-Generate (Lefrançois *et al.*, 2017) is an engine based on SPARQL 1.1 that includes transformation rules within a SPARQL query. Chimera (Scrocca *et al.*, 2020) is based on RMLMapper and it includes a lowering step to convert RDF datasets into other formats and data models. Although most of the R2RML-compliant engines are able to generate materialized KGs from a RDB, their scientific contributions are focused on optimizations over the SPARQL-to-SQL process, so we describe them in the following section.

The CSV2RDF tool is presented in (Mahmud *et al.*, 2018), where the authors define algorithms to transform CSV data into RDF using CSVW metadata annotations, and their experimental study uses datasets from the CSVW Implementation Report[28]. Another tool, COW (Converter for CSV on the Web[29]) allows the conversion of datasets in CSV format and uses a JSON schema expressed in an extended version of the CSVW recommendation.

2.4.2 Virtual Knowledge Graph Construction Engines

Most of the works proposed under this framework are focused on providing access to relational databases (Calvanese *et al.*, 2017; Priyatna *et al.*, 2014; Sequeda & Miranker, 2013) and optimizations on the SPARQL-to-SQL translation process. The first approach on the translation between these two query languages is proposed in (Chebotko *et al.*, 2009), where the authors define an algorithm to create an equivalent SQL query based on an input SPARQL query. With the appearance of the R2RML recommendation, both Morph-RDB (Priyatna *et al.*, 2014) and Ontop (Calvanese *et al.*, 2017) propose the use of these mapping rules and optimize the algorithm proposed in (Chebotko *et al.*, 2009). For example, there are recent studies and optimizations for an efficient translation of the `OPTIONAL` SPARQL operator in this process in (Xiao *et al.*, 2018b). Nowadays, Morph-RDB and Ontop are the two most well known open source engines that construct virtual knowledge graph from relational databases. Ultrawrap (Sequeda

[27] `https://github.com/RMLio/rmlmapper-java`
[28] https://w3c.github.io/csvw/tests/reports/index.html
[29] https://csvw-converter.readthedocs.io/en/latest/

& Miranker, 2013; Sequeda *et al.*, 2014) is another SPARQL-to-SQL engine that use SQL views to enhance the execution of the SQL queries after its translation from SPARQL. In (Sequeda *et al.*, 2014), the authors propose a cost function that helps to decide when a SQL view should be or not physically materialize in the RDB.

In this context, the term constraint has been used in (Hovland *et al.*, 2016), where the authors defined two new properties extending the concept of OBDA instance. They propose a set of optimizations during the SPARQL-to-SQL translation process with techniques that take into account these constraints. However, the main assumptions made over the OBDA framework (e.g, the data source is an RDB o has an RDB wrapper, or the schema contains a set of constraints) are maintained. There are other works such as (Botoeva *et al.*, 2019; Michel *et al.*, 2015) that apply the OBDA framework over document-based databases. For example, in Morph-xR2RML (Michel *et al.*, 2015), the authors formally define the translation from SPARQL to NoSQL databases.

There are two main proposals of the construction of virtual knowledge graphs beyond relational databases: Ontario and Squerall. Ontario (Endris *et al.*, 2019) is based on the concept of RDF molecule templates (Endris *et al.*, 2017) which aims to perform efficient source selection in a data lake composed of heterogeneous data sources in their original format. It creates a set of star-shaped sub-queries that match the RDF Molecule Templates (RDF-MT), and applies optimization techniques to define the query plan that will be executed. Similarly, Squerall (Mami *et al.*, 2019a) takes input data and mappings, and offers a middleware that is able to aggregate the intermediate results in a distributed manner. Finally, Polyweb (Khan *et al.*, 2019) is another proposal that is able to translate and distribute queries using RML mappings over relational databases and CSV files.

2.5 Knowledge Graph Construction Evaluation Resources

To facilitate knowledge graph construction, KG engineers need to understand the strengths and weaknesses of the varied set of KGC engines that were presented in Section 2.4. They may want to know whether their engines cover the requirements of their real-use-case scenarios. The challenge is to develop techniques and methodologies that cover the requirements for constructing knowledge graphs, and to ensure that is extensible and sustainable over time, when new approaches appear, or new versions of existing engines are released. In general, it is necessary to have an overview of state of the art engines that are tailored to different source formats,

accepting as input those mappings that are represented in a variety of declarative languages. In this section we first provide an overview of test-cases defined to evaluate the conformance of KGC engines over a mapping language specification. Then we describe the main contributions about benchmark the performance and scalability of KGC engines.

2.5.1 Test-Cases and Testbeds

Whenever a specification becomes a W3C recommendation, such as SPARQL (Harris & Seaborne, 2013), RDF (Cyganiak *et al.*, 2014), Direct Mapping of relational data to RDF (DM) (Arenas *et al.*, 2012), and R2RML (Das *et al.*, 2012a), several tools need to support them, and a set of test cases need to be defined for each of them (SPARQL test cases[30], RDF 1.1 test cases[31], and R2RML and Direct Mapping test cases[32]). These test cases provide useful information to choose the tool that covers better certain needs. It is also a relevant step in the standardization process of any technology or specification. We describe the R2RML in more details as it is the one that is most related to the scope of our work.

Determining the conformance of tools executing R2RML rules in the process of RDF generation is a step to provide objective information about the features of each tool. For this reason, the R2RML test cases (Villazón-Terrazas & Hausenblas, 2012) were proposed, with 63 test cases. Each test case is identified by a set of features, such as the SQL statements to load the database, title, purpose, specification reference, review status, expected result, and corresponding R2RML rules. All the test cases are semantically described using the RDB2RDF-test[33] and Test Metadata Vocabulary[34]. In 2021, several R2RML processors were assessed for their conformance with the R2RML specification running the test-cases. The results were available in the R2RML implementation-report (Das *et al.*, 2012b), and are also annotated semantically using the Evaluation and Report Language (EARL) 1.0 Schema[35].

2.5.2 Benchmarks

Several benchmarks have been developed to measure the performance of SPARQL-to-SQL query translation of OBDA engines. The main two proposals in this field are the Berlin

[30] `https://www.w3.org/2001/sw/DataAccess/tests/r2`
[31] `http://www.w3.org/TR/rdf11-testcases/`
[32] `https://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/`
[33] `http://purl.org/NET/rdb2rdf-test#`
[34] `https://www.w3.org/TR/2005/NOTE-test-metadata-20050914/`
[35] `https://www.w3.org/TR/EARL10/`

SPARQL Benchmark (BSBM) (Bizer & Schultz, 2009) and the Norwegian Petroleum Directorate Benchmark (NPD) (Lanti *et al.*, 2015). The BSBM benchmark sets its context in the e-commerce domain, and provides a configurable data generator and a set of SPARQL queries together with their equivalent SQL queries. This benchmark has been used to compare the query performance of native RDF stores with the performance of OBDA engines that execute virtualized SPARQL access against relational databases.

Specific OBDA requirements have been analyzed by the authors of the NPD benchmark in the setting of a real-world scenario from the oil industry. The nine proposed requirements are related to the datasets, query sets, mappings and query languages. The benchmark includes a data generator, VIG (Lanti *et al.*, 2017), to generate scaled RDB instances that obtain a number of expected triples from a SPARQL query, using as inputs an ontology, an R2RML mapping document, and the schema of the RDB together with its corresponding instance.

Simple queries defined in LSLOD (Hasnain *et al.*, 2017) have been used in order to evaluate the performance of the Ontario (Endris *et al.*, 2019) engine. In this work, all of the original RDF datasets were translated into RDB tables. The idea was to evaluate an heterogeneous setup consisting of RDF and RDB sources, focused on the source selection problem, and the generation of the corresponding optimized query plan. Additionally, the simple queries from LSLOD are focused on the evaluation of the distribution of star-shaped groups that do not exploit some of the features of the SPARQL language, such as FILTER, ORDER BY, GROUP BY and NOT EXISTS, which are relevant in the context of real-life use cases.

Many other benchmarks have been proposed in the Semantic Web area, but can be considered out of the scope of our work since they do not focus on KG construction. This includes for exmaple, FedBench (Schmidt *et al.*, 2011) and QFed (Rakhmawati *et al.*, 2014) for SPARQL query federation or SP2Bench (Schmidt *et al.*, 2009) and HOBIT (Röder *et al.*, 2020) for RDF triplestores.

2.6 Conclusions and limitations of the State of the Art

After the analysis and description of the state of the art in knowledge graph construction from heterogeneous data sources, we want to conclude this section with the identification of the strengths and weaknesses of the field that have motivated our work:

- **Overlapping among mapping language specifications.** There are many mapping languages that allow defining rules for constructing knowledge graphs from heterogeneous

data sources. Although they usually have their own specific features for solving different problems, they also share multiple features. Languages are not necessarily interoperable, and many of them come associated with a very specific engine that supports them. Having the possibility of translating among these different languages would allow practitioners to have the possibility of selecting a wider set of engines to implement their knowledge graph construction process.

- **The importance of declarative approaches.** The use of declarative and standardized mapping rules and metadata makes it possible to generalize a proposal, avoiding ad-hoc and manual steps. It also incorporates a set of important benefits for a KG construction process, such as the improvement of its maintainability, reproducibility readability, and understandability.

- **Constructing virtual KG over heterogeneous data sources.** Most of the data shared on the web is currently raw data in well known formats such as CSV, JSON, and XML. Semantic Web and more specifically, KG construction technologies, play a key role in starting to see the web as an integrated database that can be queried. Querying heterogeneous and flat data is: i) neither a trivial nor an easy task that can be delegated to naïve querying approaches and ii) optimizations and improvements can still be proposed taking advantage and exploiting current annotation proposals to not only enhance performance but also completeness. Most of the virtual knowledge graphs construction engines over heterogeneous data sources have focused on the adaptation of techniques proposed by federated SPARQL engines, but they do not support the majority of the SPARQL operators and they do not correctly execute the queries when the data source is beyond RDB instances.

- **Constructing materialized KGs at scale.** Despite the rich repertory of knowledge graph construction approaches using materialization techniques, optimizing the declarative description of complex data integration systems (i.e., heterogeneous data sources, high rate of duplicates, application of transformation functions, big data sizes, etc.) remains still open. The absence of frameworks capable of efficiently executing complex data integration systems negatively impacts on the global adoption of existing formalisms in real-world applications of knowledge graphs.

- **Comprehensive evaluations KG construction approaches.** The absence of testbeds, test-cases and benchmarks for knowledge graph construction has prevented the community from conducting fair evaluations of the existing engines. This resource deficiency has also impeded for a holistic understanding about the pros and cons of the state of the art, as well as for clear directions to advance the area. Given the expected growth rate of available data, these evaluation standard methodologies are demanded in order to devise the next generation of tools able to integrate data at scale.

Chapter 3

Objectives and Contributions

> As computer scientist the most important part of a research is the what, not the how
>
> *Maria-Esther Vidal*

As discussed in the previous chapters, in our work we focus on the exploitation of declarative mapping rules for the construction of knowledge graphs from heterogeneous data sources. Additionally, in order to evaluate these kinds of systems, we propose and design a set of methodologies and benchmarks. This chapter presents the objectives of our work and identifies our contributions to the current state of the art. We also enumerate the assumptions considered when we started this work, and describe the most relevant hypotheses and restrictions that delimit the scope of this thesis.

3.1 Objectives

The principal research question we want to answer in this thesis is: *How can we build more scalable and efficient KG construction engines based on declarative mapping rules, independently of the specification of the mapping language?*. To tackle this open research problem, we identify two main objectives. First, our aim is to describe a new generation of KGC engines that use declarative annotations to scale up this process over complex data integration scenarios. The exploitation of these declarative mapping rules and constraints allow the definition of structures and associated operators that enhance virtual and materialized construction of KGs from heterogeneous data sources. The second goal of this thesis is to define objective

methodologies to evaluate this new generation of KG construction engines. More in detail, our objective is to create an evaluation framework for materialized and virtual KG construction engines that is general enough to allow the comparison of these systems and also to detect the main limitations of all of them.

The first objective of this thesis (scalable and efficient construction of (virtual) KGs) can be decomposed into the following goals tackling open research problems:

- The current heterogeneity in the number of declarative mapping specifications reduces the benefits of this approach for constructing knowledge graphs. Based on the W3C recommendation R2RML (Das *et al.*, 2012a), and with the aim of providing support to other formats beyond relational databases, many new declarative mapping languages have been proposed such as RML (Dimou *et al.*, 2014), xR2RML (Michel *et al.*, 2015), KR2RML (Slepicka *et al.*, 2015), CSVW (Tennison *et al.*, 2015), or D2RML (Chortaras & Stamou, 2018b). When mappings are defined in a specific language, the choice of the KGC engine is more limited. Thus, KG engineers may not benefit from optimizations that are applicable to their use cases but are implemented for another languages. This situation negatively impacts on the global adoption of these data integration systems in real-world applications of knowledge graphs. Our goal is to provide support for the translation among mapping languages while their features are preserving.

- The construction of virtual knowledge graphs from heterogeneous data sources (e.g., CSV, JSON, XML or RDB) does not always exhibit good query evaluation performance nor completeness. The common technique is to use naïve SQL data wrappers that hidden the complexity of the input sources. However, in order to tackle the advantage of proposed SPARQL-to-SQL optimization techniques (Calvanese *et al.*, 2017; Priyatna *et al.*, 2014), well-formed RDB instances are required, including their corresponding domain and integrity constraints. Our goal is to define horizontal solutions that exploit declarative annotations for KGC (i.e., mapping rules and declarative constraints) together with mapping translation techniques to enhance the query evaluation performance and completeness of current SPARQL-to-SQL approaches.

- Current approaches for constructing materialized knowledge graphs (Lefrançois *et al.*, 2017; Şimşek *et al.*, 2019) do not scale-up in complex data integration scenarios that are common in the real world (e.g., high rate of duplicates or interoperability issues over the

input data sources, high data volume, etc.). The absence of efficient engines that exploit declarative mapping languages to construct materialized knowledge graphs hinders their global and industry adoption over real use cases. Our goal is to develop efficient and horizontal techniques for scaling-up the construction of KG from heterogeneous data sources exploiting declarative KGC annotations.

The second goal of this thesis (evaluation methodologies of KGC systems to understand what are their main limits) can be decomposed into the following goals with the associated open research problems:

- Current evaluations performed over materialization knowledge graph construction engines are neither representative nor general enough to provide a clear overview of the limitations of the applied techniques. Furthermore, some engines such as RocketRML (Şimşek *et al.*, 2019) and SPARQL-Generate (Lefrançois *et al.*, 2017) have only been evaluated using their own use cases, datasets and parameters. Our goal is to create an evaluation framework that is general enough and considers as many parameters as possible to allow fair comparisons among KGC engines and the definition of new benchmarks and testbeds including other relevant features .

- Virtual knowledge graph construction benchmarks (Bizer & Schultz, 2009; Lanti *et al.*, 2015) have mainly focused so far on relational databases, ignoring other types of data sources that are used for KG construction (e.g., CSV, JSON, XML). Hence, they are not enough to test the capabilities of the new generation of engines that are able to run or distribute SPARQL queries over data sources in several formats. Our goal is to define a benchmark for VKGC on a relevant domain allowing the comparison over multiple and heterogeneous virtual KGC engines using a single point, taking into account data sources beyond RDB and relevant parameters that can impact in the performance of the engines.

3.2 Contributions to the State of the Art

In this work, we aim to provide solutions to the goals and open research problems described in the previous section. The contributions for solving the first research problem are:

C1.1. The definition of the concept of *Mapping Translation* and the characterization of its main properties. We motivate the idea of making the different mapping language

specifications interoperable by a new step in a knowledge graph construction workflow, where mappings can be translated to a different specification. Additionally, we define the desirable properties of this concept based on the work of (Hartig, 2017) and we demonstrate their potential usage over a set of use cases extracted from the literature where it has been successfully applied.

C1.2. The **formalization and application of constraints extracted from mapping rules and metadata over tabular data** to enhance virtual KG construction approaches. Focused on tabular data sources, we extend a typical virtual KG construction workflow adding a set of additional pre-processing steps to handle common issues for querying tabular data sources. Exploiting information from mapping rules and metadata annotations, we are able to efficiently apply integrity and domain constraints, together with a set of new optimization steps, for enhancing both performance and completeness in the (virtual) KG construction over tabular sources.

C1.3. The exploitation of standard **declarative mapping rules to automate the generation of functional wrappers** for virtual KG construction. Following best practices on data publication on the web (Bizer *et al.*, 2011) and with the aim to help knowledge and software engineers to design and implement efficient functional wrappers, we propose a solution that exploits mapping rules for the automatic creation of programmed wrappers. More in detail, we use R2RML mappings and a new interpretation of the SPARQL-to-SQL baseline algorithm presented in (Chebotko *et al.*, 2009) for creating programmed and efficient GraphQL wrappers, which allows the translation from GraphQL to SQL queries for obtaining data from relational databases.

C1.4. Definition of **physical operators exploiting information from mapping rules** for constructing materialized KGs at scale. We define a set of data structures aligned with the different types of tasks that a materialization KGC engine has to perform to create the RDF graph. The data structures, together with their corresponding operators, are able to construct materialized KGs in complex data integration scenarios (big data sizes, high rate of duplicates, etc). We demonstrate that our proposal is able to generate KGs at scale in comparison with the state of the art engines.

C1.5. **Heuristic-based optimizations for functional mappings** (mapping rules with transformation functions) for scaling up the construction of materialized knowledge graphs.

Covering the gap of parsing mapping rules that contain ad-hoc transformation functions over the input data sources in a efficient way, we propose a set of mapping and data translation rules to enhance the construction of materialized KGs from this kind of mapping rules.

With regard to the second objective, this work presents new contributions in the following aspects:

C2.1. Definition of a set of **test cases to evaluate the conformance of KG construction engines in RML**. We extend the R2RML test cases (Villazón-Terrazas & Hausenblas, 2012) for covering other kinds of data formats and relational databases. Executed over the RML parsers in its corresponding RML implementation report, they give an overview of the current state of the engines in terms of language conformance.

C2.2. Identification and experimental evaluation of the **parameters that affect the behavior of the KGC engines**. We analyze the typical inputs and their corresponding features in a knowledge graph construction workflow and how they can impact over the performance and completeness of the process. Additionally, we empirically demonstrate the importance of taking into account these parameters (e.g., join selectivity, data partitioning, duplicate generation) for generating representative benchmark and testbeds for these engines.

C2.3. Design of a complete and **representative benchmark for evaluating virtual KGC engines**, based on open data from the transport domain. We create Madrid-GTFS-Bench, a comprehensive benchmark for virtual knowledge graph construction, which considers multiple data formats and different data scales. Several engines are evaluated with a single benchmark so as to assess the current status of virtual knowledge graph access.

Chapter 4 describes the mapping translation concept and its properties.Chapter 5 describes the GTFS-Madrid-Bench, a virtual knowledge graph construction benchmark in the transport domain, aligned with the second objective. In the same Chapter, two evaluation methods for testing mapping compliance and performance during the contruction of materialized KG are presented. Chapter 6 provides the solutions to enhance virtual knowledge graph access over heterogeneous data exploiting the mapping translation concept. Finally, Chapter 7 presents two approaches for scaling up the construction of materialized KGs.

3.3 Assumptions

Our work is based on the assumptions listed below. These assumptions provide a background to facilitate the comprehension of the decision taken during the development of this thesis.

A1 Mapping rules and metadata descriptions are declarative and follow W3C standards (or their extensions). Although the contributions proposed in the thesis may be generalized to any kind of mapping rules and metadata descriptions, we assume that these annotations are defined following best practices for describing content on the web of data, hence, using W3C standards.

A2 The target schema (ontology) for integrating the source data is available and is implemented in OWL. We also assume that in any of the data integration systems (*DIS*) defined in this thesis, the target model used for integrating the input sources has been already created, and it is modeled using the Web Ontology Language (OWL).

A3 Mapping rules and metadata descriptions are available. In the same manner as the ontology, the rules that provide the relations between target and source schemes and metadata that describe the content of the source input data are available. Therefore, we assume that the *DIS* presented in the thesis are complete and no user interaction is needed.

A4 Data are represented in formats that are not RDF. The input data sources that have to be integrated in a *DIS* are described in formats such as RDB, CSV, XML or JSON, but we assume that none of the datasets will be described following RDF.

A5 Datasets are static, not streams. We assume that the input data sources are static, although they can have a short update time (e.g., 5 minutes), the proposals of this thesis do not provide support for data streams.

3.4 Hypothesis

After the identification of the assumptions, we can describe the research hypothesis of this thesis. They cover the general characteristics of the contributions.

H1 It is possible to translate declarative mapping rules among different specifications allowing practitioners to have the possibility of selecting a wider set of engines to construct

their KG. Our hypothesis here is that mapping translation techniques can be used to enhance multiple features of a KG construction workflow such as mapping rules generation, maintainability, optimization techniques, etc.

H2 The exploitation of mapping rules and metadata descriptions for data constraint extraction and its application can enhance the performance, access and completeness of current virtual knowledge graph construction systems. This can be performed by extending the current virtual KG construction workflow, including a set of efficient additional steps for the application of the extracted constraints over the input sources.

H3 A benchmark based on `de-facto` standard models for publishing transport data on the web is able to stress and provide a full overview of the current state of different virtual types knowledge graph construction engines. The intrinsic features of these data models, a well defined target ontology and set of relevant queries covering SPARQL 1.1 features are able to define a representative benchmark that addresses the strengths and limitations of virtual KG construction engines and applied techniques.

H4 Using the information from the input mapping rules, physical data structures and their corresponding operators can be defined for scaling up the construction of materialized KGs from heterogeneous data sources. These structures can handle complex data integration environments where a large number of joins among sources and high rate of duplicates have an impact over the performance of the KG construction.

H5 Optimizations for processing functional mapping rules can be applied to scale up the construction of materialized knowledge graphs. The application of efficient techniques for pre-processing transformation functions during a KG construction process, together with mapping and data translation rules, can deal with complex data integration systems.

3.5 Restrictions

Finally, there is a set of restrictions that describe the limitations and define the future work objectives. The restrictions are:

R1 Input data sources must be located in the same physical place as the knowledge graph construction process is run. We do not consider network latencies for data transference or other similar parameters that may have impact over a process of constructing a KG.

R2 We do not have to consider data protection nor access restrictions. We assume that there are not limitations to access any of the input data sources of the *DIS*. The datasets are openly available through the web together with a license that allow its use.

R3 The size of datasets is defined in terms of Gigabytes. We assume each single input data source will be no bigger than 100Gb.

R4 Our proposal does not make use of the capabilities of SPARQL-to-SQL engines. We do not exploit the specific optimizations that each engine applies, we are only aware about the importance of indexes and integrity database constraints for the effectiveness of their translation process.

R5 Our proposal does not make use of the features of the transformation functions performed over the input data sources. Therefore, we are not aware about if the transformation functions are bijective or not, which can have an impact over query translation techniques.

Chapter 4

A New Generation of Knowledge Graph Construction Engines

> The primary goal of a researcher should be to continually challenge science with creative ideas
>
> *Oscar Corcho*

Database technologies play a vital role in the development of information systems for all sorts of organizations. So far, relational databases (RDB) are still the dominating type of structure and technology used for data management inside organizations, although other formats (e.g. JSON, spreadsheets, XML) and types of databases (e.g. noSQL, graph databases) have also emerged as alternatives for data representation and management in the last decades.

In the early days of information system development, it was natural for organizations to develop their own data models, which were strongly aligned with their activities. This led to a large heterogeneity across organizations, and even across different departments inside the same organization. Such heterogeneity was especially evident in the case of organizational changes, merges, etc. Similarly, data warehouses were also used in order to align and materialize data from different sources, normally from the same organization, so as to provide support for analytical queries and for the generation of reports. These situations made researchers and professionals start working on solutions for data integration, where data from several sources needed to be accessible according to a unified and global view over such local heterogeneous data sources. Popular technologies used in production systems worldwide included the use of

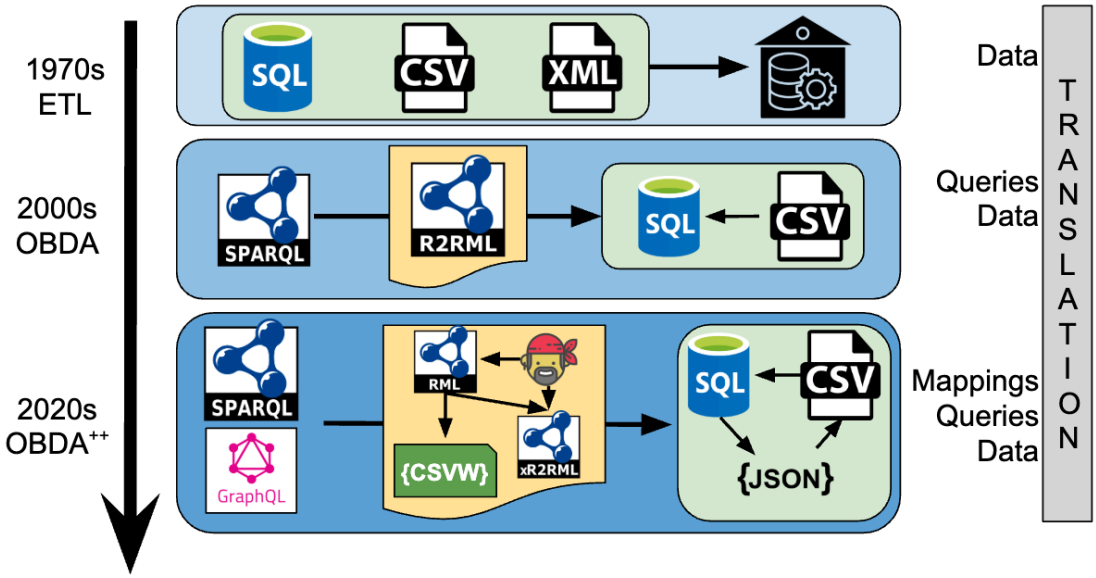


Figure 4.1: Timeline of data integration techniques. During the 1970s ETL approaches started with data translation techniques, current generation of KGC incorporated techniques for query translation and next generation of KGC systems which mapping translation approaches are to be applied.

Extract-Transform-Load (ETL) workflows to overcome heterogeneity and ensure the availability of data in such data warehouses or on integrated databases. Indeed, these approaches are still strongly used nowadays.

In the meantime, data integration challenges became even more relevant since two decades ago, when organizations started using Web technologies to provide access to their data (via Web Services, REST APIs or using Semantic Web and Linked Data approaches), both for their own information system development as well as for data sharing, and later on when public administrations started publishing open data according to public-sector information reuse initiatives. Availability and heterogeneity of data (both in terms of content and format) is nowadays present at an unprecedented level. Following the aforementioned ETL approaches, the term data lake has been rather recently coined to refer to an evolution of data warehouses that considers not only structured data but also the other types of (semi-)structured and unstructured formats in which data is made available nowadays, as discussed above.

Over these decades, several approaches have been proposed to tackle data integration challenges. We are specially interested in those that fall under the area of Ontology Based Data Access (OBDA) and Integration (OBDI) (Poggi *et al.*, 2008), which is recently also defined as

Knowledge Graph Construction (KGC) methods. From now on we will refer to all of them, in a general manner, as KGC. As we have already mentioned in previous chapters, in KGC, ontologies are used as a global view over heterogeneous data sources and mappings are used to describe such relationships in a declarative manner. Many different types of KGC mapping languages have been proposed over the last decades, with a large variety of syntax and formats especially in the early ones. Since the standardization of languages like RDF and OWL, several languages were proposed focused on the transformation from relational databases into RDF (e.g. D2R, R2O (Barrasa *et al.*, 2004)). This led to the creation of the RDB2RDF W3C Working Group, which published two recommendations for transforming the content of relational databases into RDF: Direct Mapping (Arenas *et al.*, 2013) and R2RML (Das *et al.*, 2012a).

A bit after R2RML was recommended, and because of its use in different types of contexts, new needs and requirements arose, especially in relation to supporting other formats beyond relational databases, and this resulted in the creation of many new mapping languages, such as RML (Dimou *et al.*, 2014) (to deal with CSVs, JSON and XML data sources), xR2RML (Michel *et al.*, 2015) (to deal with MongoDB), KR2RML (Slepicka *et al.*, 2015) (to deal with nested data), CSVW[36] (to describe CSV files on the Web), or D2RML (Chortaras & Stamou, 2018a) (for XML, JSON and REST/SPARQL endpoints). In addition to declarative languages, non-declarative mapping languages have also been proposed, such as SPARQL-Generate (Lefrançois *et al.*, 2017), Helio[37], Tarql[38] or Triplify (Auer *et al.*, 2009).

There are several reasons why new mapping languages are needed. The first and main reason is that a typical mapping language is designed to work with a specific **data format** (e.g. R2RML is focused on relational databases). Even for a more generic purpose mapping language, such as RML, there may still be a need to extend it to support a more specific technology, such as xR2RML. Another reason is **readability and compactness**. Most mapping languages are designed in a format to be parsed by machines and they do not take into account human readability. Examples of languages created to account for this are R2RML-Iterator (for statistical CSV files) (Chaves-Fraga *et al.*, 2018) or YARRRML (Heyvaert *et al.*, 2018). Lastly, many of existing mapping languages **lack formalization**, making it difficult to apply,

[36] `https://www.w3.org/ns/csvw`
[37] `https://helio.linkeddata.es/`
[38] `https://github.com/tarql/tarql`

for example, query translation techniques. Therefore, the current situation of an KG construction practitioner that needs to provide access to a varied set of heterogeneous data sources is that there are many different options to select from, and it is difficult to determine which one is better for each situation. Languages are not necessarily interoperable, and many of them come associated with a very specific engine that supports them. However, at the same time, it is clear that most of these languages share many common aspects, such as the description of where the data comes from, how URIs can be created for resources, how triples need to be generated (in a materialized or virtual way), etc. Having the possibility of translating among these different languages, covering at least those common characteristics that are shared across languages, would allow practitioners to have the possibility of selecting a wider set of engines to implement their KG construction system.

In this chapter, we lay out our vision that the next generation of KGC systems should take advantage of this proliferation of mapping languages. In other words, in addition to the data translation and query translation techniques that have been widely addressed in the state of the art of KGC so far, the KGC research community will need to think carefully about how to address mapping translation (See Figure 4.1).

4.1 Mapping Translation: Concept Definition

We define the mapping translation concept as a function that transforms a set of mappings described in one language (we call them original mappings) into a set of mappings described in another language (we call them target mappings).

Our next step is to attach desirable properties for such a function. In this line, we propose to use and adapt some properties that have been described by (Sequeda *et al.*, 2012) and (Hartig, 2017) in their works. To be more specific, those properties are *information preservation* and *query result preservation* (Figure 4.2).

The **Information Preservation Property (IPP)** applied to a mapping translation function states that at least there is a function so that its application over the information generated by the application of the target mappings over the original data source returns the same information generated by the application of the original mappings over the same data source.

The **Query Result Preservation Property (QRPP)** applied to a mapping translation function states that for any query that can be evaluated over the information generated from the application of the original mappings to the original data source, there is at least a function to

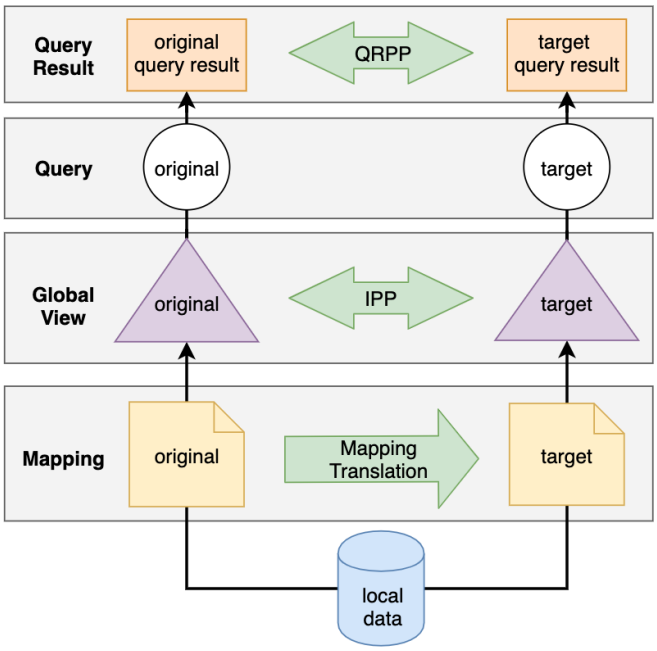


Figure 4.2: Mapping Translator Properties. The results (triangles) may satisfy the IPP property after the application of the source and target mappings over the same data. In the same way, query results (rectangles) may satisfy the QRPP property when equivalent queries are evaluated over the source and target results.

generate another query that can be evaluated over the information generated by the application of the target mappings to the same data source, in such a way that both queries return the same results.

Finally, using these two properties, we define the concepts of weak and strong semantics preservation for a mapping translation function, as follows: a mapping translation function exhibits **weak semantics preservation** if only IPP holds. If both IPP and QRPP are satisfied, then we say that it holds the **strong semantics preservation** property.

4.2 Mapping Translation by example

In this section we identify a set of scenarios and challenges in the creation and use of KGC mapping languages, where mapping translation is relevant. We describe the challenge and provide some references to some of the work presented in the literature addressing or acknowledging it. The presented use cases are summarized in Table 4.1.

Use Case	Engine	Translation
Maintenance	Mapeathor	Excel-to-[R2]RML
Maintenance	yarrrml-parser	YARRRML-to-RML
Maintenance	r2rml-statistic	R2MLIterator-to-R2RML
Maintenance	ShExML	ShExML-to-RML
Declarative to Programmed	morph-graphQL	R2RML-to-GraphQLResolver
Access	Morph-CSV	RML(+FnO)-to-R2RML
Access	FunMap	RML(+FnO)-to-RML
Optimizations/Semantics	ontop	R2RML-to-OBDA

Table 4.1: Summary of Mapping Translation Approaches

4.2.1 Improving mapping creation and maintenance.

Creating and maintaining KGC mappings is usually difficult, since mapping languages have been created so that they can be consumed by the corresponding engines, and they commonly suffer from readability and compactness problems. With respect to **readability**, several approaches have focused on providing mapping editors (e.g. (Heyvaert *et al.*, 2016)) so that mappings are easier to create by non experts. However, these editors are usually limited to some features of the mapping language or to a specific version of the mapping language specification, and in general they still require knowledge about the underlying mapping language syntax. With respect to **compactness**, there are cases where the generated mapping documents are very long and repetitive, making it difficult to create and maintain (Chaves-Fraga *et al.*, 2018). For instance, this is the case when an KGC approach is used to provide access to multidimensional data sources, such as the ones commonly used to publish statistical data. We describe three cases where mapping translation ideas are already being applied to address these issues.

YARRRML (Heyvaert *et al.*, 2018) is a serialisation of RML mappings that uses the YAML (a human-readable data serialization language) format[39]. It is designed with the objective to reduce the size and verbosity of RML. There is no specific engine or parser to exploit YARRRML mappings in an KGC setting. Instead, the tool Matey[40] is in charge of translating YARRRML mappings into RML, so that any RML-compliant OBDA engine can be used to exploit them. ShExML (García-González *et al.*, 2020) presents a similar work, where the authors extends the syntax of ShEx shapes constraints (Prud'hommeaux *et al.*, 2014) to support the construction of KGs from heterogeneous data sources. They conduct an user evaluation and observe that their

[39] `https://yaml.org/`

[40] http://rml.io/yarrrml/matey/

approach is easy to use and maintain by the users. To exploit all the benefits from the RML engines, they provide a translator engine from ShExML to RML.

In the case of multidimensional data (e.g. official statistics data), the W3C RDF DataCube recommendation is the ontology that is commonly used as a global view in an KGC setting. In most cases, the amount of mappings that would need to be created to link the original data source with the ontology will be rather large and with similar structure. Therefore, there is a high risk that the [R2]RML mapping document(s) generated in the end will contain clerical errors due to copy&paste&edit operations. Furthermore, they will be difficult to maintain. As a result, R2RML-Iterator (Chaves-Fraga *et al.*, 2018) is proposed as a simplified mapping language specifically designed for this type of data. The proposal is detailed in Section 4.3.

4.2.2 From declarative mappings to programmed adapters

Introduced in 2000, REST (Fielding & Taylor, 2000) has become now the most popular architecture for the provision of web services and the implementation of Web-based applications. However, the complexity of software development continues evolving, and aspects that received little attention, such as the size of data being exchanged/transmitted or the number of API calls being made, are now becoming more relevant in the context of mobile application development. As a result, problems like *over-fetching* (a REST endpoint returns more data than what is required by the client) and *under-fetching* (a single REST endpoint does not provide sufficient information requested by the client) are now being discussed. In order to address these problems, Facebook proposed the GraphQL query language (Facebook, Inc., 2018), used internally since 2012 and released for public use in 2015. Since then it has been increasingly adopted, and GraphQL is now supported by multiple GraphQL engines for major programming languages (e.g. JavaScript, Python, Java, Golang, Ruby).

The two main components of a GraphQL server are the **schema** and the **resolvers**. The GraphQL schema specifies the type of an object together with the fields that can be queried. GraphQL resolvers are data extraction functions implemented in a programming language that are responsible to translate GraphQL queries into queries supported by the underlying datasets (e.g. GraphQL to SQL). In addition, query planning tools have been developed in order to translate GraphQL queries into other query languages (e.g. dataloader[41], joinmonster[42]).

[41] `https://github.com/facebook/dataloader`
[42] `https://join-monster.readthedocs.io/en/latest/`

In a recent paper (Priyatna *et al.*, 2019) we proposed the use of the mapping translation concept to facilitate the generation of GraphQL resolvers. We propose specifying mappings in R2RML, which is a well-defined and formalized mapping language, and apply a mapping translation technique to generate automatically the corresponding GraphQL schemes and resolvers in different programming languages. Our intuition is that following this approach, GraphQL resolver will be easier to maintain, as they are declarative and independent from any programming language. The details of this approach are presented in Section 6.2.

4.2.3 Providing access to semi-structured data

Semi-structured data formats are one of the most widely used formats to publish data on the Web. Although existing mapping languages provide support for this type of data sources, existing engines are mostly focused on the generation (materialization) of RDF-based knowledge graphs, with only a few proposals (e.g. xR2RML (Michel *et al.*, 2015)) focused on the application of query-translation techniques (virtualization) over such types of data sources.

In the specific case of spreadsheets (CSV), providing access to this format is difficult for two main reasons: (i) CSV does not provide its own query language, (ii) there are some transformations that are commonly needed when treating data available in CSVs. For solving the first issue, query translation techniques have been applied over such data format by considering a CSV file as a single table that can be loaded in an RDB. For the second issue, some extensions of well-known mapping languages (RML together with the Function Ontology (De Meester *et al.*, 2017)) and annotations following the CSVW specification (Tennison *et al.*, 2015) can be used.

Morph-CSV applies the concept of mapping translation for enhancing OBDA query translation over CSV files from SPARQL. It exploits the information of CSVW annotations and RML+FnO mappings to create an enriched RDB representation of the CSV files together with the corresponding R2RML mappings, allowing the use of existing query translation (SPARQL-to-SQL) techniques implemented in R2RML-compliant OBDA engines. This contribution is detailed in Section 6.1.

4.2.4 Understanding the semantics of mapping languages

To the best of our knowledge, there has not been yet any formal study of the relationship between R2RML and the Direct Mapping recommendations, and among the many different

mapping languages that have arisen recently.

For the first case (R2RML and Direct Mapping), intuitively we may consider the Direct Mapping is a subset of R2RML, given the expressive power provided by the latter. However, it would be interesting to know how expressive Direct Mapping may be in case that views are generated for the underlying data sources, for instance. Our intuition is that given the possibility of creating a database view from an existing database, there exists a fragment of R2RML that can be translated into Direct Mapping, such that the application of Direct Mapping over the view generates equivalent results as the application of R2RML mappings over the original database. Finding such fragment brings a practical implication because it would lower down the barrier for transforming data into RDF and enable people to use Direct Mapping engines, which are in general easier to use than R2RML engines for those people who are used to manage databases.

Similarly, this analysis may be extended to other combinations of mapping languages, so as to allow mapping translations among them that would allow exploiting the specific characteristics of each associated implementation, as well as describing formally their semantics, especially for those cases where no formal specification of the semantics has been provided yet.

Ontop (Rodriguez-Muro & Rezk, 2015) is an OBDA system that comes with both data and query translation techniques. Ontop translates R2RML mappings into its own mapping called ”OBDA mappings”. These mappings are represented as datalog rules, allowing the formalisation and semantic optimisation techniques to be performed, and generating a more efficient SQL queries (e.g. self-join elimination) that can be evaluated in less time by the underlying databases.

4.3 Use Case: Virtual Knowledge Graph Construction in the Statistics Domain

Statistics data is one of the most common ways of sharing public information nowadays. PDF, HTML and, especially, CSV format, are some of the most used formats of tabular data being published on the web by statistics agencies. Whereas this is still the main trend, many agencies worldwide are embracing semantic technologies for publishing their resources as Statistics Knowledge Graph (SKG), which is the knowledge graph on the web that stores and allows

Features	Ad-Hoc	R2RML
Processor Type	Solution Specific	General Purpose
# Processors	1	Many
Materialization	Yes	Yes
Virtualization	No	Yes

Table 4.2: Comparison of Approaches for Transforming Statistics Data to SKG

access to statistics linked data. In many cases both formats co-exist, allowing to access the information in different ways.

Due to high volume and variability of data, the transformation from tabular to SKG-oriented formats requires a process that is standard and maintainable. We identify two main approaches for transforming tabular data to Statistics Knowledge Graph (SKG). The first approach is an ad-hoc approach, such as one that has been reported in (Corcho *et al.*, 2017), in which CSV data is converted into RDF Data Cube (Cyganiak *et al.*, 2012) using a set of custom rules. The second approach is defined on the basis of mapping languages, such as the RDB2RDF W3C Recommendation, R2RML, in which transformations are codified in an standard language, with several tools available for applying them.

In an ad-hoc approach (Corcho *et al.*, 2017), a processor which is the main component responsible for the transformation process, is developed for a specific purpose. Those processors are not commonly used for other solutions. The SKG resulting from the transformation process is stored in a triple store and there is a need for repeating the transformation process whenever the original statistics data changes to ensure that generated SKG is synchronised with the original statistics data. On the other hand, using R2RML to generate SKG brings several benefits in comparison to the ad-hoc approach. There are many R2RML processors (Calvanese *et al.*, 2017; Priyatna *et al.*, 2014) available which means that one is not restricted to use a particular processor and may easily use another if necessary. Furthermore, many of those processors have incorporated techniques for keeping the desired SKG virtual, thus eliminating the need of synchronization between the tabular data and SKG. A virtualization process is highly recommended when the data published is volatile because it ensures that the retrieved data is updated. This is achieved by translating SPARQL queries posed to the virtual SKG into another query supported by the underlying data source and evaluated on the original dataset (Poggi *et al.*, 2008). Table 4.2 summarizes our discussion regarding the approaches on transforming statistics data into SKG.

Listing 4.1: Data source mapping

```
rr:logicalTable [ rr:tableName ”Statistics2016” ];
```

The size of the R2RML mapping documents depends on the number of columns in the tabular data and number of dimensions to be generated. For example, a CSV file that contains data from all the European countries will need an R2RML mapping with one section defined the required dimensions for each country. Considering the correlation between the size of a mapping document and its complexity, i.e., the more lines it has, the more difficult it is to maintain, a crucial task is to reduce the size of the mappings to ease the mapping maintenance task. We address this challenge by proposing an approach to reduce the size of the mapping documents using an iterator that has been incorporated into R2RML.

4.3.1 From statistic data to SKG

In this section we discuss and exemplify two different approaches for transforming tabular data to SKG, the first one being the baseline which allows to illustrate the advantages of our approach.

4.3.1.1 Approach 1 - Naive

Given a CSV file containing statistics data, a typical way to transform it to SKG using R2RML mappings is to create one TriplesMap for each column corresponding to a slice of a dimension, for example January in Time dimension or Male in Gender dimension. The TriplesMap will have the following properties:

- Its `rr:logicalTable` property specifies the source CSV file (Listing 4.1)
- Its Subject Map specifies `qb:Observation` as the generated triples' RDF type (Listing 4.2)
- It will have a pair of PredicateObjectMap mappings that specify a slice of a dimension and its values (Listing 4.3)
- It will have a PredicateObjectMap mapping that specifies which dataset the generated triples belong to (Listing 4.4)

Listing 4.2: Observation mapping

```
rr:subjectMap [
    a rr:Subject; rr:template "...";
    rr:class qb:Observation;
];
```

Listing 4.3: Dimension slice mapping

```
rr:predicateObjectMap[
    rr:predicate ex:month;
    rr:objectMap [ rr:constant "interval:January"; ];
];
rr:predicateObjectMap[
    rr:predicate ex:numberOfArrivals;
    rr:objectMap [ rr:column "Jan"; ];
];
```

Listing 4.4: Dataset mapping

```
rr:predicateObjectMap[
    rr:predicate qb:dataSet;
    rr:objectMap [ rr:constant "ex:Arrivals"; ];
];
```

Listing 4.5: Columns and dictionary properties in R2RML-Iterator

```
rr:logicalTable [
    rr:tableName "\"2016−P21\"";
    rmlc:columns ["Jan","Oct","Dec"];
    rmlc:dictionary {"Jan":"January","Oct":"October","Dec":"December"};
];
```

Considering that a typical statistics CSV file contains many columns, the size of the R2RML mapping document grows linearly with the number of columns. The main issue with the first approach is the size of the R2RML/RML mapping, where the mapping expert has to create a TriplesMap for each column to answer the desirable SPARQL queries, as we will show in the Evaluation section.

4.3.1.2 Approach 2 - R2RML-iterator

In this approach, we aim to reduce the size of a R2RML mapping for statistics data by incorporating an iterator variable the mapping rules. We identify that the only difference between those TriplesMaps is the name of the column, which provides a unique identifier to the TriplesMap object and the access to the data of each column. By incorporating a variable that references the target columns, R2RML-Iterator reduces the size of the mapping while maintaining the semantics of the R2RML mapping. This variable is formalized in the mapping language by incorporating four new properties (Figure 4.3) to the Logical Table object:

- the `rmlc:columns` whose value must be an array of column names that exist in the CSV file.
- the `rmlc:columnRange` whose value must be an array with cardinality two and the values have to be column names that exist in the CSV file.
- the `rmlc:dictionaryFile` whose value must a path to a JSON file that defines the correlation between the columns and their aliases in JSON format.
- the `rmlc:dictionary` whose value must be a JSON schema that defines a correlation between the columns and their aliases in JSON format.

Listing 4.6: Iterator variables in the extension

```
<TriplesMap2016{$column}>

rr:subjectMap [
    a rr:Subject;
    rr:template ”http://ex.org/2016{$column}”;
    rr:termType rr:IRI;
    rr:class qb:Observation;
];
rr:predicateObjectMap[
        rr:predicate sltsv:month;
    rr:objectMap [
        rr:termType rr:IRI;
        rr:constant ”http://reference.data.gov.uk/def/intervals/{$alias}”;
    ];
];
rr:predicateObjectMap[
    rr:predicate sltsv:numberOfArrivals;
    rr:objectMap [
        rr:termType rr:Literal;
        rr:column {$alias};
        rr:datatype xsd:integer;
    ];
];
```

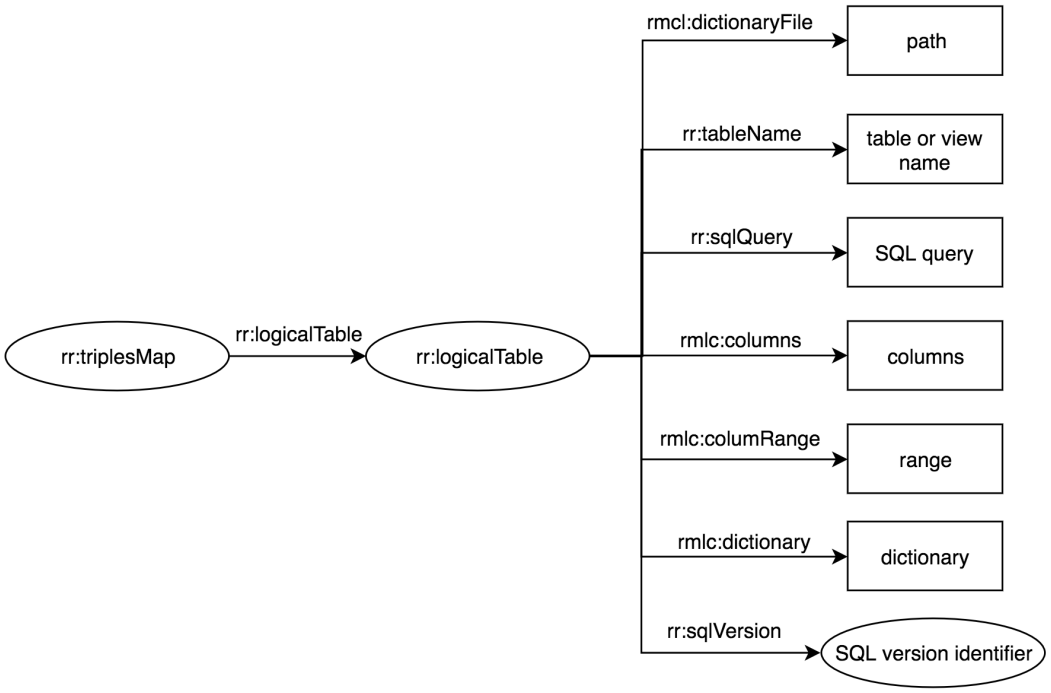


Figure 4.3: R2RML extension for statistics tabular data

Listing 4.5 depicts an example of the usage of the `rmlc:columns` property, in which the subset of the CSV columns to be used in the transformation process is specified. This property is defined using a LogicalTable object. We use the `rmlc:dictionary` (or its corresponding `rmlc:dictionaryFile`) property to establish a correlation between a column and its alias, that is useful in some parts of the mappings. These properties are based on JSON syntax for easing their creation to the mapping editors.

During the mapping translation process, the variables are replaced with the name of each column or its alias, as defined in the LogicalTable object. The resulting R2RML mapping contains as many TriplesMap objects as the number of columns specified in the R2RML-iterator properties. In this example, the identifier of each TriplesMap, the URI of the subjects and the object of the `sltsv:numberOfArrivals` predicates, include the variables so they will be replaced with the name of a column or its alias. The R2RML mapping will contain three TriplesMap objects, one for each defined column. All the iterator variables can be specified anywhere in the mapping, as shown in Listing 4.6, using the {$column} or {$alias} syntax.

In order to ensure that the iterator proposal aligns with R2RML, we have developed a mapping translator that converts its mapping document with an iterator variable to a R2RML mapping with multiple TriplesMap objects. In this way, we reduce the size of the R2RML mapping document, minimizing the time of the mapping creation and supporting an easier maintenance. Besides, as it is aligned to R2RML, any R2RML available processor is able to deal with our approach.

4.3.2 R2RML iterator: Use Cases

In this section we describe our experiment of transforming two statistics datasets from two different domains and agencies: a tourism statistics dataset from the Sri Lanka Tourism Development Authority (SLTDA) and a immigration statistics from the EuroStat. For each dataset, we use both R2RML and R2RML-Iterator mappings with Morph-RDB to generate virtual SKGs and evaluate three SPARQL queries for each one.

4.3.2.1 Case 1: Statistics from the Sri Lanka Tourism Development Authority

Dataset and Queries The Sri Lanka Tourism Development Authority performs data collection and market research about tourism in Sri Lanka and publishes comprehensive statistics as PDF files[43]. We use tabula-java[44] to extract these statistics as CSV files and make them available online. For example, the CSV file that contains the number of passengers grouped by countries (y-axis) and the arrival months (x-axis) in 2016.

Our intention is to transform that CSV file into a virtual SKG. Because this SKG is not materialized, there is no need to store it in a dedicated triple store. Any R2RML engine with query translation support will be able to answer SPARQL queries posed to the dataset. For example, consider the following three queries:

- Q1: Retrieve observations of the number of incoming tourist originated from Spain in 2016.
- Q2: Retrieve observation of the number of incoming tourists in May 2016.
- Q3: Can be seen as the combination between Q1 and Q2, i.e., retrieve observations of the number of incoming tourists from Spain in May 2016.

[43] `http://www.sltda.lk/statistics`
[44] `https://github.com/tabulapdf/tabula-java`

Features	R2RML	R2RML-iterator
Total Lines	∼700	74
#TriplesMaps / #SubjectMaps	12	1
#PredicateObjectMaps	60	5

Table 4.3: Comparison between R2RML and R2RML-iterator mappings used in the Sri Lanka Tourism dataset example.

Mappings The R2RML mapping document generated using the naive approach described in the previous section contains 12 TriplesMaps and around 700 lines. Each TriplesMap describes the transformation rules of the number of arriving passengers every month of the year. Except for those PredicateObjectMap properties corresponding to the name the CSV columns, all TripleMaps have identical values.

On the contrary, the corresponding proposal mappings, either with `rmlc:column` property or with `rmlc:range` property have only 74 lines in total with 1 TriplesMap and 5 PredicateObject mappings. The Table 4.3 summarizes the characteristics of the mappings.

4.3.2.2 Case 2: EuroStat - Immigration Statistics

Dataset and Queries Eurostat[45] the statistics office of the European Union. Its main responsibility is to provide statistics information about European Union such as economy, finance, population, industry, etc. In this example, we consider a dataset containing the number of immigrants that have arrived in European countries available online[46]. We downloaded the aforementioned dataset as a CSV file containing the number of immigrants grouped by countries (x-asis) and years (y-axis). We have created three SPARQL queries, similar to the ones in the previous example:

- Q1: Retrieve the number of immigrants arriving in Spain.
- Q2: Retrieve the number of immigrants arriving in any of European Countries in the year of 2015.
- Q3: Retrieve the number of immigrants arriving in Spain in the year of 2015.

[45] `http://ec.europa.eu/eurostat`
[46] `http://ec.europa.eu/eurostat/product?code=tps00176&mode=view`

Features	R2RML	R2RML-iterator
Total Lines	>2800	<70
#TriplesMaps / #SubjectMaps	>40	1
#PredicateObjectMaps	>170	4

Table 4.4: Comparison between R2RML and R2RML-iterator mappings used in the Eurostat Immigration dataset example.

Mappings Generating the naive mapping described in the first approach is not feasible in this case, as there is a need to generate a TriplesMap for each column that represents a country and the dataset contains more than 40 columns. Instead, we generate two R2RML-iterator mapping documents,one with `rmlc:columns` property and the other with `rmlc:range` property. Then we use the our mapping translator engine to generate the naive version of R2RML.

Our proposals mapping document (either version) contains only 1 TriplesMap and 4 PredicateObjectMap mappings, totalling less than 70 lines. On the contrary, the generated R2RML mapping document has more than 40 TriplesMap, more than 170 PredicateObjectMap mappings, totalling more than 2800 lines. The Table 4.4 summarizes the characteristics of the mappings.

Chapter 5

Systematic Evaluation for Knowledge Graph Construction Engines

In this chapter, we present the contributions related with the evaluation of knowledge graph construction systems. To facilitate data exploitation in this context, application developers need to understand the strengths and weaknesses of existing knowledge graph construction tools. Additionally, tool developers may want to know if their engines cover the requirements of real use-case scenarios. In general, it is necessary to have an overview of state of the art engines that are tailored to different source formats, accepting as input mappings that are represented in a variety of declarative languages.

In summary, Section 5.1 describes a preliminary set of test cases for testing the conformance of RML mapping language over its available processors, Section 5.2 describes and analyzes the impact of several parameters for the evaluation of materialized KGC engines together with the definition of a set of representative testbeds and their evaluation over two compliant RML engines. Finally, Section 5.3 presents a virtual knowledge graph construction benchmark over the transport domain. Additionally to the benchmark proposal, the contribution is evaluated over five different engines from the state of the art.

5.1 Conformance Test Cases for the RDF Mapping Language (RML)

Unlike R2RML, there are no test cases available to determine the conformance of the processors to the RML specification. As a result, the processors are either not tested or only tested with custom test cases, which do not necessarily assess every aspect of the specification. Con-

sequently, no implementation report is available that allows comparing the different processors that generate knowledge graphs from heterogeneous data sources based on the conformance to the specification. This way it is hard to determine the most suitable processor for a certain use case.

In this section, we focused on RML and introduce an initial set of RML test cases, which contains 297 test cases based on the existing R2RML test cases. However, instead of only considering relational databases as data sources, as it occurs for the R2RML test cases, we also consider data in CSV, XML, and JSON format. Furthermore, we tested the conformance of the RMLMapper, RocketRML, SDM-RDFizer and CARML: every test case is executed by each processor and we noted if the generated knowledge graph matches the expected one. The corresponding implementation report is available at `http://rml.io/implementation-report`. This allows to determine which processor is the most suitable for a certain use case. For example, do users want a processor that supports the complete specification, or do they prefer a processor that does not support certain aspects of the specification, but executes the rules faster?

5.1.1 RML Test Cases

We propose test cases to determine the conformance of RML processors to the RML specification. The proposed test cases are based on the R2RML test cases, but they take into account different heterogeneous data sources and the corresponding differences in RML. Our preliminary set of test cases includes (i) adjusted R2RML test cases for relational databases (including MySQL[47], PostgreSQL[48], and SQL Server[49]) and (ii) new test cases for files in the CSV, XML (with XPath as the reference formulation), and JSON format (with JSONPath as the reference formulation). The test cases are described at `http://rml.io/test-cases/` and the corresponding files are available at `https://github.com/rmlio/rml-test-cases`.

5.1.1.1 Data model

We describe the test cases semantically to increase their reusability and sharability. To this end, we created a semantic data model[50], with as main entity the test case (see Figure 5.1). For

[47] `https://www.mysql.com/`
[48] `https://www.postgresql.org/`
[49] `https://www.microsoft.com/en-us/sql-server/`
[50] `http://rml.io/test-cases/#datamodel`

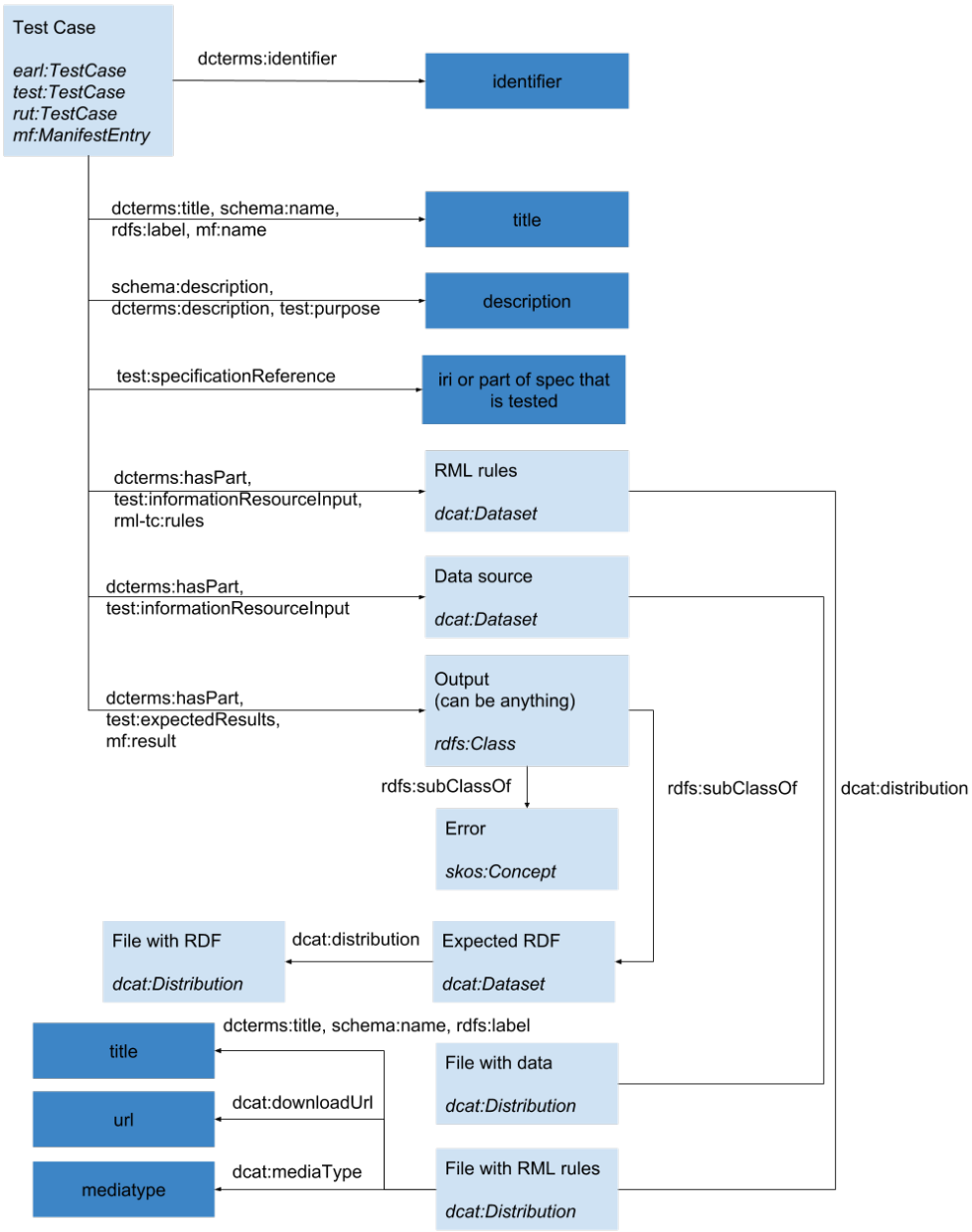


Figure 5.1: Data model of the RML test cases

each test case, the following details are described: unique identifier, title, description, relevant aspect of the RML specification, data sources (optional), expected knowledge graph or error, and RML rules.

To provide the corresponding semantic descriptions, the model uses mostly the Evalua-

tion and Report Language (EARL) 1.0 Schema[51], the Test case manifest vocabulary[52], the Test Metadata vocabulary[53], and the Data Catalog Vocabulary[54]. A test case is annotated with the classes `earl:TestCase`, `test:TestCase`, and `mf:ManifestEntry`. The identifier, title, description, and the specific aspect of the RML specification that is being tested are added as datatype properties. The files that are provided as input to the tools are linked to the test cases via `test:informationResourceInput` and `dcterms:hasPart`. The file with the RML rules is also linked via `rml-tc:rules`[55]. The objects of these properties are of the class `dcat:Dataset`,which in turn link to a `dcat:Distribution` that includes a link to a file. The expected output, whether that is a knowledge graph or an error, is linked via `test:expectedResults`, `mf:result`, and `dcterms:hasPart`. In the case of a knowledge graph, the object of these properties is a `dcat:Dataset`, linked to a `dcat:Distribution`, to describe the file containing the graph. In the case of an error, we link to the expected error.

5.1.1.2 Test case files

Each test case consists of a set of files that contain the input data sources, the RML rules, and the expected RDF output. In practice, the files are organized as follows: all files for a single test case are contained in a single folder. There are three types of files for each test case:

- 0 or more **data source files** for CSV (with extension .csv), XML (with extension .xml), and SON (with extension .json), or 1 file with SQL statements to create the necessary tables for relational databases (called `resource.sql`);
- 1 **file with the RML rules** (in Turtle format, called `mapping.ttl`); and
- 0 or 1 **file with the expected RDF** (in N-Quads format, called `output.nq`).

Distinct test cases assess different behaviors of the processors. Certain test cases assess the behavior of the tools when (i) the required data sources are not available, and others when (ii) an error occurs and no output is generated. In the former, no data sources files or SQL statements are provided. In the latter, no file with the expected RDF is provided. The test cases

[51] `https://www.w3.org/TR/EARL10/`, with prefix `earl`
[52] `http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#`, with prefix `mf`
[53] `https://www.w3.org/2006/03/test-description#`, with prefix `test`
[54] `https://www.w3.org/TR/vocab-dcat/`, with prefix `dcat`
[55] `http://rml.io/ns/test-cases`, with prefix `rml-tc`

are independent of how the processors materialize the knowledge graph: a data dump, as done by the RMLMapper, or on the fly, as done by Ontario (Endris *et al.*, 2019).

5.1.1.3 Differences with R2RML test cases

For most R2RML test cases, we created an RML variant for CSV, XML, JSON, MySQL, PostgreSQL, and SQL Server, leading to 6 RML test cases per R2RML test case. For R2RML test cases that focus on specific features of SQL queries, we only created 3 RML test cases, i.e., for MySQL, PostgreSQL, and SQL Server.

For test cases with CSV, XML, and JSON files as data sources, we created the corresponding files with the data based on the tables of the relational databases. For CSV, we used the table created by the SQL statements of the R2RML test case and stored it as a CSV file. For XML, the name of the table was used for the root of the XML document and every row of the table was used to create an XML element. Within this element, elements were created for each column and their values are the values of the corresponding columns in the table. For JSON, we followed a similar approach as XML. The file contains a JSON object at the root with the name of the table as the only attribute. This attribute has as value an array, where each element of the array corresponds with a row in the table. For each row, attributes were created for each column and their values are the values of the corresponding columns in the table.

Data errors. 2 of the R2RML test cases expect a data error to happen, e.g., when the subject IRI of an entity cannot be generated. In this case, an error is thrown and no knowledge graph is generated. With RML for entities where no subject IRI can be generated there is also no output generated, but, in contrast to R2RML, for the other entities the corresponding output is still generated. Therefore, for the corresponding RML test cases the processors can still throw an error, but the generation of the knowledge graph must not be halted.

Inverse expressions. 3 of the R2RML test cases are designed to test the use of inverse expressions[56]. However, inverse expressions are only used to optimize the knowledge graph generation and no differences are observed in the generated knowledge graph. Thus, whether inverse expressions are used by a processor or not cannot be verified by such test cases. Thus, we do not include them for RML.

[56] `https://www.w3.org/TR/r2rml/#inverse`

SQL-specific features. 18 of the R2RML test cases focus on specific features of SQL queries, e.g., a duplicate column name in a SELECT query. As there are no corresponding RML test cases for CSV files, XML files with XPath, and JSON files with JSONPath, we only provide 54 corresponding test cases for MySQL, PostgreSQL, and SQL Server.

Null values. 1 of the R2RML test cases tests null values in the rows. However, a corresponding RML test case cannot be provided for the CSV and XML format, because both formats do not support null values.

Spaces in columns. 1 of the R2RML test cases is designed to test the behaviour when dealing with spaces in the columns of the SQL tables. However, a corresponding RML test case cannot be provided for the XML format, because it does not allow spaces in names.

In total, we have 297 test cases: 39 for CSV, 38 for XML, 41 for JSON, and 180 for relational databases. Of these 297, 255 test cases expect an knowledge graph to be generated, while 36 expect an error that halts the generation.

5.1.2 Conformance of RML Parsers

In this section, we describe the executing of the test cases and their results for four RML processors: the RMLMapper, CARML, RocketRML and SDM-RDFizer. We detail on (i) the method for running the test cases and obtaining the results, which are annotated semantically, and (ii) the implementation report similar as proposed for R2RML[57].

5.1.2.1 Method

We define a method to run the RML test cases over RML processors and generate the corresponding results. The main goal is to facilitate the testing process and provide a general solution for running the test cases over other RML processors that can be developed in the future. The method consists of two main steps: (i) assessing if a processor passes every test case and (ii) annotating the obtained results as RDF using the EARL Schema through a set of YARRRML rules (Heyvaert *et al.*, 2018).

We implemented a Java-based tool for checking the conformance of the test cases over different RML processors. At this moment, it is able to evaluate the test cases for JSON, CSV

[57] `https://www.w3.org/TR/rdb2rdf-implementations/`

RMLMapper	**CSV**	**JSON**	**XML**	**MySQL**	**PostgreSQL**	**SQL Server**	**Total**
passed	39	41	38	55	55	55	283
failed	0	0	0	5	5	5	15
inapplicable	0	0	0	0	0	0	0

Table 5.1: Results of the RMLMapper

CARML	**CSV**	**JSON**	**XML**	**MySQL**	**PostgreSQL**	**SQL Server**	**Total**
passed	29	28	28	0	0	0	85
failed	10	10	13	0	0	0	33
inapplicable	0	0	0	60	60	60	180

Table 5.2: RML test-cases results for CARML

and XML formats. We relied on the test framework of each processor to execute the RDB test cases. If a new processor wants to be added to test its conformance, the framework only needs to have access to the corresponding output of each test-case. Then it checks, using the same method for all processors, if the output is the same as the correct one, if an error that is expected has really thrown, etc. It generates as output two CSV files with the results and the metadata needed to generate the corresponding RDF.

The results are semantically annotated, as the test cases, using the EARL Schema. Each test case evaluated by a tool is annotated as an `earl:Assertion` with the properties: `earl:subject` with the URI of the tool, `earl:assertedBy` with the identifier of who performed the evaluation, `earl:test` with the URI of the test and `earl:result` with the result of the test-case.

Three types of results are possible: "passed", "failed", and "inapplicable".The option **Passed** (`earl:passed`) is used either when the actual output matches the expected output when no error is expected, or when the tool throws an error when an error is expected. **Failed** (`earl:failed`) is used either when the actual output does not match the expected output if no error is expected, the processor returns an error trying to execute a test or the tool does

RocketRML	**CSV**	**JSON**	**XML**	**MySQL**	**PostgreSQL**	**SQL Server**	**Total**
passed	25	26	24	0	0	0	65
failed	14	15	14	0	0	0	43
inapplicable	0	0	0	60	60	60	180

Table 5.3: RML test-cases results for RocketRML

SDM-RDFizer	CSV	JSON	XML	MySQL	PostgreSQL	SQL Server	Total
passed	39	41	38	60	60	60	298
failed	0	0	0	0	0	0	0
inapplicable	0	0	0	0	0	0	0

Table 5.4: RML test-cases results for SDM-RDFizer

not throw error if an error is expected. **Inapplicable** (`earl:inapplicable`) is used when the tool clearly states that specific features used in a test case are not supported. The results also provide its type (`earl:TestResult`) and its mode (`earl:automatic` for all these cases). We created a set of YARRRML rules to generate these annotations following the EARL Schema that, using the outputs of the test cases.

5.1.2.2 Results

We perform the test-cases over four processors. In Table 5.1 we show the results for the RMLMapper processor. It passes all CSV, JSON and XML test cases, but fails in the same 5 test cases for the RDBs. The failures are related to the automatic datatyping of literal for RDBs specified by R2RML[58]. RMLMapper expects to pass the failed test-cases in next versions of the processor. The effort prediction to pass these test-cases is not very much since the failures depend on the general processor, not on the used RDBMS. Once they have been solved for one RDBMS they will automatically pass over the rest.

In Table 5.2 we show the results for the CARML processor. It partially passes the CSV, JSON and XML test cases, but it does not provide support for any of the RDBs test cases. The failures are related to the unsupported for multiple Subject Maps, multiple Predicate Maps, and Named Graphs. The developers of the tool declare that CARML will support these features in next versions of the processor. However, at the moment of writing, we do not have any information about if CARML will provide support for RDBs.

Table 5.3 shows the results for the RocketRML processor. Its behavior is similar to CARML, as it partially passes the CSV, JSON and XML test cases, but it does not provide support for any of the RDBs test cases. Finally, Table 5.4 show the results for the SDM-RDFizer processor, which passes all the proposed test cases, having a similar behavior as the RMLMapper engine.

[58] `https://www.w3.org/TR/r2rml/#dfn-natural-rdf-literal`

Finally, we can declare that testing a RML processor with the defined cases and analyzing the obtained results offers a general view of the current status of it. These results also give useful information to the tool developers on knowing where they should put their effort to improve the conformance of the processor.

5.2 Parameters that Affect the Construction of a Knowledge Graph

Despite these developments, the absence of testbeds has prevented the community from conducting fair evaluations of the existing tools for knowledge graph construction. This testbed deficiency has also impeded for a holistic understanding about the pros and cons of the state of the art, as well as for clear directions to advance the area. Given the expected growth rate of available data, testbeds are demanded in order to devise the next generation of tools able to integrate data at scale.

In this section, we study the process of knowledge graph construction and analyze various variables and configurations that can impact on the performance of materialization techniques. The relevant parameters studied in this work include selectivity of the joins between mapping rules, types of relations, and percentage of duplicates. We also present diverse examples that evidence the heterogeneous behavior that each engine may exhibit whenever small changes are conducted to the variables and the configurations of a testbed.

We devise a set of parameters involved in a knowledge graph construction process and we empirically show how they can impact on the behavior of two existing engines: RMLMapper[59] and SDM-RDFizer[60]; these engines are compliant with the RML specification according to a set of defined test-cases[61]. We develop a synthetic data generator for the generation of (semi)-structured data and RML mapping rules, that consider the identified set of parameters. The results of our empirical study provide evidence of the importance of the proposed set of variables and configurations during the evaluation of these tools. The testbeds used to conduct this evaluation are available online[62].

Our main contribution includes the definition of various dimensions and set of variables to be considered during the creation of testbeds or to be measured while the evaluation of knowledge graph construction tools. Another contribution represents the empirical evaluation of the

[59] `https://github.com/RMLio/rmlmapper-java`
[60] `https://github.com/SDM-TIB/SDM-RDFizer`
[61] `http://rml.io/implementation-report/`
[62] `https://github.com/SDM-TIB/KGC-Param-Eval`

effects that the variables and configurations have on the tasks of knowledge graph construction. Furthermore, the results of the experimental study contribute to the understanding of the pros and cons of the studied engines, and the directions that need to be followed in order to devise tools able to scale up to real-world scenarios.

5.2.1 Motivation Example

We motivate our work by analyzing different scenarios where the performance of RMLMapper and SDM-RDFizer may be affected by changing the configuration of the testbeds utilized for empirically evaluating these engines. We aim at remarking the importance of taking into account different parameters during the definition of a testbed. We first describe a scenario where naïve parameters (size and format) leads to wrong decisions during the comparing of SDM-RDFizer and RMLMapper. The testbeds include a data source with one thousand rows, different number of predicate-object (POM) in RML triple maps, and diverse configurations of selectivity of triple map joins.

In this example, we execute a testbed where three different configurations of an RML mapping rule: Two-POM, Five-POM, and Ten-POM, i.e. they correspond to three mapping rules with two, five, and ten Predicate-Object Maps, respectively. Both engines exhibit a similar behavior while the number of predicate-object maps varies from two to five POMs, as shown in Table 5.5. However, when more complex mapping rules with more POMs are considered, the behavior of the SDM-RDFizer and RMLMapper is not impacted equally. Moreover, the results suggest that RMLMapper execution time increases with the number of POMs, while the SDM-RDFizer seems to be slightly affected. We now consider another parameter, the join selectivity, i.e. the cardinality of matching values from outer to the inner table (relation), in a referencing object map between two RML mapping rules. The join selectivity varies from **High Selectivity**, **Medium Selectivity**, and **Low Selectivity**, and Table 5.6 reports on the results of RMLMapper and SDM-RDFizer. First, it can be observed that the RMLMapper execution time increases by around 8 seconds, while the SDM-RDFizer behavior is not equally affected by the selectivity of the join condition. As can be seen in Table 5.6, the SDM-RDFizer execution time (in seconds) increases from high to medium selectivity by 0.04 (from 2.16 to 2.20), then decreases from medium to low selectivity by 0.01 (from 2.20 to 2.19). On the other hand, the RMLMapper execution time increases by 1.83 (from 38.6 to 40.43), and 5.63 (from 40.43 to 46.06) seconds from high to medium, and medium to low selectivity, respectively. As in the previous example, both engines are not equally affected by the complexity of the testbed.

Engine	Execution time (secs.)	Number of results
	Two POM	
RMLMapper	0.92	2,000
SDM-RDFizer	1.72	2,000
	Five POM	
RMLMapper	1.84	5,000
SDM-RDFizer	1.85	5,000
	Ten POM	
RMLMapper	3.36	10,000
SDM-RDFizer	1.98	10,000

Table 5.5: Impact of Number of Predicate-Object Maps. Various predicate object maps (POM) specified in the mapping rules. The behavior of the two engines is similar when the mapping rules are simple (less than 5 POM) but it is different when more complex mappings are running (10 POM); time in seconds.

Engine	Execution time (secs.)	Number of results
	High Selectivity	
RMLMapper	38.6	2,100
SDM-RDFizer	2.16	2,100
	Medium Selectivity	
RMLMapper	40.43	23,000
SDM-RDFizer	2.20	23,000
	Low Selectivity	
RMLMapper	46.06	30,000
SDM-RDFizer	2.19	30,000

Table 5.6: Impact of Join Selectivity. Impact of the join selectivity variable over the engines with high, medium and low percentage of selectivity. While RMLMapper engine behavior increases in terms of execution time when the selectivity decreases, the SDM-RDFizer behavior is maintained, i.e. this variable affects to the first engine but it does not impact equality to the second one.

The uncorrelated behavior of studied engines shows clearly the need to considering diverse variables and configurations during the definition of testbeds, and thus, uncovering characteristics of these engines. In this work, we analyze the parameters that might affect a knowledge

graph construction process and evaluate some of the most problematic ones (e.g. partitioning, relation type) to remark the importance of setting them during testbed design.

5.2.2 Relevant Parameters for Testbed Design

Independent Variables		Observed Variables	
		Execution Time	Completeness
Mapping	mapping order	✓	
	# triplesMap	✓	✓
	# predicateObjectMaps	✓	✓
	# predicates	✓	✓
	# objects	✓	✓
	# joins	✓	✓
	# named graphs	✓	✓
	join selectivity	✓	✓
	relation type	✓	✓
	object TermMap type	✓	
Data	dataset size	✓	
	data frequency distribution	✓	
	type of partitioning	✓	✓
	data format	✓	✓
Platform	cache on/off	✓	
	RAM available	✓	
	# processors	✓	
Source	distribution data transfer	✓	✓
	initial delay	✓	
	access limitation	✓	✓
Output	Serialization	✓	✓
	Duplicates	✓	✓
	Generation type	✓	✓

Table 5.7: Variables and Configurations. Set of variables and configurations that impact on the behavior of the tools for knowledge graph construction. Independent variables are divided into five groups and the impact on the observed variables is depicted.

In this section, we perform a study of the parameters that have impact on the knowledge

graph construction engines. First, we identify the generic groups of parameters involved and the effect they produce in this process. Second, we provide a list of specific variables that influence the construction of knowledge graphs and determine the relationships among them. Finally, we describe each parameter in detail given the reasons why it might affect the performance of the engines. Together with these descriptions, we provide use cases over a set of parameters to illustrate the importance of involving them in a testbed definition.

As in every empirical study, we consider two groups of variables: independent and observed. The independent variables are those features that need to be specified in a benchmark to ensure that the performed evaluation is reproducible. These variables are grouped in five dimensions: mapping, data, platform, source, and output. On the other hand, observed variables correspond to those characteristics that are measured during the evaluation of the testbed and that may be influenced by independent variables. The observed variables are as follows:

- *Execution time:* The variable is in turn comprised of: *i) Time for the first triple* (elapsed time between the engine starts and the first triple), *ii) total execution time* required to produce all the triples of the knowledge graph.

- *Completeness:* Number of returned triples in relation to all the RDF triples that should be created according to the data and input mappings.

The relations among independent and observed variables are presented in Table 5.7. These variables are described in detail in the next section.

5.2.2.1 Mapping Dimension

This dimension involves the variables that characterise the mappings in terms of their structure and evaluation. Regarding the structure, there are various aspects to be considered: mapping order, the complexity of the mapping in terms of number of predicates, objects, and the join type and selectivity.

Mapping Order. Although the mappings are usually defined using an RDF serialisation, where the order is not relevant, the features of each `rr:tripleMap` (e.g. joins) can affect the execution plan generated by each tool, having, thus, a potential negative impact on the total execution time.

Mapping complexity. The number of properties defined in a rule mapping, e.g. number of predicates, objects, or named graphs may affect the observed variables because the number

Engine	Execution time (secs.)	Number of results
	1-1	
RMLMapper	42.86	25,000
SDM-RDFizer	2.19	25,000
	1-N	
RMLMapper	43.34	22,490
SDM-RDFizer	2.19	22,490
	N-1	
RMLMapper	43.26	22,490
SDM-RDFizer	2.15	22,490
	N-M	
RMLMapper	78.64	25,200
SDM-RDFizer	2.33	25,200

Table 5.8: Impact of Relation types. Various relation types in a join specified in the mapping rules. N corresponds to 15 values in the case of 1-N and N-1 relations, N and M has 10 values in the last case. RMLMapper execution time is not affected by 1-N and N-1 relation types while it is affected by N-M relations. SDM-RDFizer performs better in N-1 than 1-N but the time increases in N-M.

of triples to be generated, is related to what is specified in the mappings. Additionally, the `rr:termtype` of the `rr:objectMap` can affect the total execution time because the cost of generating a constant or a template is not the same. Finally, the join selectivity and types of relation have also impact on the performance of an engine. In Table 5.8, we illustrate how the relation type affects the total execution time of the studied engines. In this case, the behavior of the RMLMapper only occurs when the relation type is N-M. However, the SDM-RDFizer behavior is impacted during the evaluation of 1-N and N-M joins. Additionally, during the join evaluation, there are many cases when duplicates are generated, then the engines have to eliminate them. Table 5.9 reports on how the generation of the duplicates –during the join condition evaluation– affects the total execution time. RMLMapper decreases its performance while the percentage of duplicates increases. However, SDM-RDFizer implements optimised data structured that allow for efficiently eliminating duplicates, and seems not to be equally affected by the complexity of these configurations, e.g. number of duplicates.

Engine	Execution time (secs.)	Number of results
	Low percentage of duplicates	
RMLMapper	37.94	20,027
SDM-RDFizer	2.01	20,027
	Medium percentage of duplicates	
RMLMapper	39.201	20,105
SDM-RDFizer	1.87	20,105
	High percentage of duplicates	
RMLMapper	40.81	20,263
SDM-RDFizer	1.89	20,263

Table 5.9: Impact of duplicates generation during join evaluation. Various configurations of duplicates generated during the evaluation of a join between two triple maps. While the complexity of the configuration increases (more percentage of duplicates), the RMLmapper decreases its performance. Surprisingly, the SDM-RDFizer seems not to be affected by the complexity of the testbeds, and improves its performance even when the complexity of testbeds increases.

5.2.2.2 Data Dimension

We describe the independent variables related with the original data that are required for the generation of a knowledge graph. Each dataset can be defined in terms of **size** and **total number of sources**. The first characteristic impacts on the number of triples that will be generated, affecting, thus, the total execution time. Additionally, the total number of sources that have to be processed to generate a knowledge graph may also affect the total execution time.

Partitioning and **distribution** are important variables considered in the construction of a knowledge graph. Partitioning refers to the way that a dataset is fragmented, and distribution is the format (e.g. CSV, JSON) of each partition. A dataset can be presented in only one format or in multiples formats, and this variable affects not only the total execution time but also the completeness of the results. A dataset may be fragmented into disjointed partitions; the partition may be horizontal, vertical or a combination of both. Horizontal partitioning fragments the dataset, so that, they represent different instances of the same resource (equal *TripleMaps* with different sources). Vertical partitioning produces fragments that contain at least one property of the same resources (*TriplesMaps* with *JoinCondition*). The horizontal partitioning may affect the completeness of a knowledge graph while the vertical partitioning has an influence on the execution time. Table 5.10 compares the behavior of the RMLMapper and SDM-RDFizer

Engine	Execution time (secs.)	Number of results
	Horizontal Partitioning without Replication	
RMLMapper	1,904.31	310,000
SDM-RDFizer	4.84	310,000
	Vertical Partitioning without Replication	
RMLMapper	2,067.77	310,000
SDM-RDFizer	4.73	310,000
	Horizontal Partitioning with Replication	
RMLMapper	2,276.98	310,000
SDM-RDFizer	5.86	310,000
	Vertical Partitioning with Replication	
RMLMapper	2,024.66	310,000
SDM-RDFizer	4.98	310,000

Table 5.10: Impact of Partitioning: Various configurations of vertical and horizontal partitioning with and without duplicates. The two engines perform similar with the two cases of the horizontal partitioning but they have different behaviors in vertical partitioning.

with different configurations. The two engines increase their execution time when the horizontal partitioning is compared with and without including replication. However, RMLMapper decreases its execution time when the vertical partitions with and without replication are compared, while SDM-RDFizer execution time increases. Thus, even SDM-RDFizer is tailored towards efficient duplicate elimination, data partitioning– with and without replication – seems to affect the SDM-RDFizer performance.

5.2.2.3 Platform Dimension

The platform dimension comprises variables related with the hardware used to create a knowledge graph. We include a set of variables related with the system cache, the available RAM memory for running the tool, and the number of processors of the machine. The **cache** and the **available RAM memory** may affect the total time execution. We recommend that other parameters, like the versions of operating system and processor, should be specified in the evaluation setup. To conclude, during testbed design, the platform and hardware specification requires attention and needs to be defined in detail.

5.2.2.4 Source Dimension

In this dimension, we consider different variables related with the original sources defined in the mapping rules. The **distribution data transfer**, which corresponds to the transfer time of a file by a Web service–in case the data is not in a local machine– will definitely influence the total execution time. Additionally, the **initial delay** of each engine to configure the corresponding wrappers for each data format and the **limit access** for example, a database, also strikes out the execution time and the completeness of the results.

5.2.2.5 Output Dimension

In this dimension, we consider the variables related with the output of the generation process. The **serialization** impacts on the total execution time; the effect will depend on the size of the output and the number of times the processor has to access the disk to store the output. **Generation type** represents how an engine constructs a knowledge graph. The generation can be continuous, e.g. the SDM-RDFizer stores each RDF triple in a file once it is generated. Contrary, the generation can be in-memory, e.g. RMLMapper stores the output when the knowledge graph is created completely. Finally, the engines usually can have a flag for removing **duplicates**; this operation has to be specified in the setup because it strikes out the completeness and also the total execution time. The efficiency of the engines components that eliminate duplicates, can be captured by observing the variables of this dimension.

As can be observed in the results reported in this section, the behavior of the studied engines is not equally affected by the different independent variables. Thus, benchmarks need to include all these variables in order to provide a holistic overview of the performance of the studied engines, and ensure general and reproducible evaluations.

5.2.3 Evaluation of affected parameters in KGC

The goal of our experiment is to assess the impact of the discussed variables and configurations during the evaluation of existing knowledge graph construction tools. We aim at answering the following research questions: **RQ1**:What is the effect of mixing different variables in one testbed?; **RQ2**: What is the impact of considering configurations of different complexity of the same variable in one testbed?; **RQ3**: Do the different variables and configurations influence in the behavior of existing knowledge graph construction tools? To answer these research questions, we set up the following experimental studies:

Dataset	#rows	#columns	#tables
1K	1,000	2	2
10K	10,000	2	2
50K	50,000	2	2

Table 5.11: Datasets. Properties of Datasets used in the Empirical Evaluations.

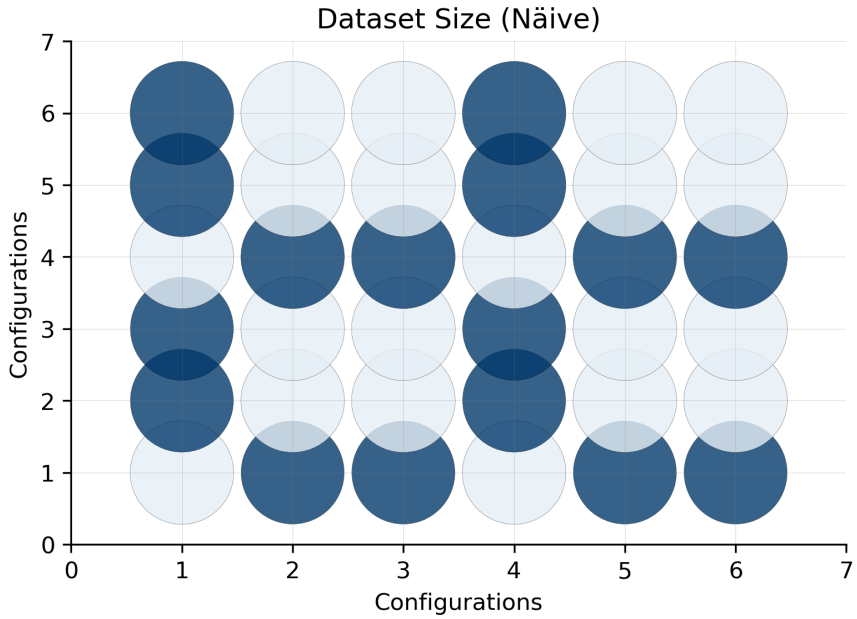


Figure 5.2: Comparison of Knowledge Graph Construction Tools on Different Dataset Sizes (Naïve). The first three configurations, i.e. 1, 2, and 3 in x-axis and y-axis, correspond to SDM-RDFizer on datasets 1K, 10K, and 50K, respectively. The last three configurations, i.e. 4, 5, and 6 on x-axis and y-axis, correspond to RMLMapper 1K, 10K, and 50K, respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation. The number of blue bubbles suggests that both systems exhibit similar behavior.

Datasets. For this evaluation, we generated three different datasets with 1,000 (1K), 10,000 (10K), and 50,000 (50K) rows, and various number of columns based on the tested parameters; Table 5.11 shows the properties of the datasets generated for `Relation Type`, `Join Duplicates`, and `Join Selectivity` evaluations. For the *Dataset Size (Näive)* parameter, we generated the same number of rows as in Table 5.11, but with 30 columns. During the experiments, we only considered the CSV file format to represent the generated tables.

Configurations. We consider different configurations for the above-discussed variables in

each dimension. `Dataset Size Configurations:` 1) SDM-RDFizer 1K; 2) SDM-RDFizer 10K; 3) SDM-RDFizer 50K; 4) RMLMapper 1K; 5) RMLMapper 10K; and 6) RMLMapper 50K. In each configuration of this parameter, we only use one data file. `Relation Type configurations:` 1) SDM-RDFizer 1-N; 2) SDM-RDFizer N-1; 3) SDM-RDFizer N-M; 4) SDM-RDFizer Combinations (all relation types); 5) RMLMapper 1-N; 6) RMLMapper N-1; 7) RMLMapper N-M; and 8) RMLMapper Combinations (all relation types). For relation cardinality, we evaluated $N = \{1, 5, 10, 15\}$ and $M = \{1, 3, 5, 10\}$. In addition, we set the percentage of rows that involve in those relation types to 25%, i.e. 25% of the overall rows from outer table have a matching join value to inner table, and 50%, respectively. `Join Duplicate configurations:` 1) SDM-RDFizer Low, 2) SDM-RDFizer High, 3) RMLMapper Low, 4) RMLMapper High. `Low` Join Duplicates refer to datasets with low percentage of duplicates, i.e. from 5% to 20% of data generated could have duplicates due to the join conditions, similarly `High` Join Duplicates refer to higher percentage of duplicates, i.e. from 30% to 50% of data generated could be duplicated. `Join Selectivity Configurations:` 1) SDM-RDFizer High; 2) SDM-RDFizer Low; 3) RMLMapper High; and 4) RMLMapper Low. In this case, the join selectivity `High` represents how many time the join condition matches the values in the inner join file from 5% to 20% of the overall rows, while `Low` means that the join condition matches range from 60% to 100% of the overall number of rows. As previously shown, we hypothesise that these configurations allow us to uncover patterns in the behavior of these engines that could not be observed if only naïve variables were studied.

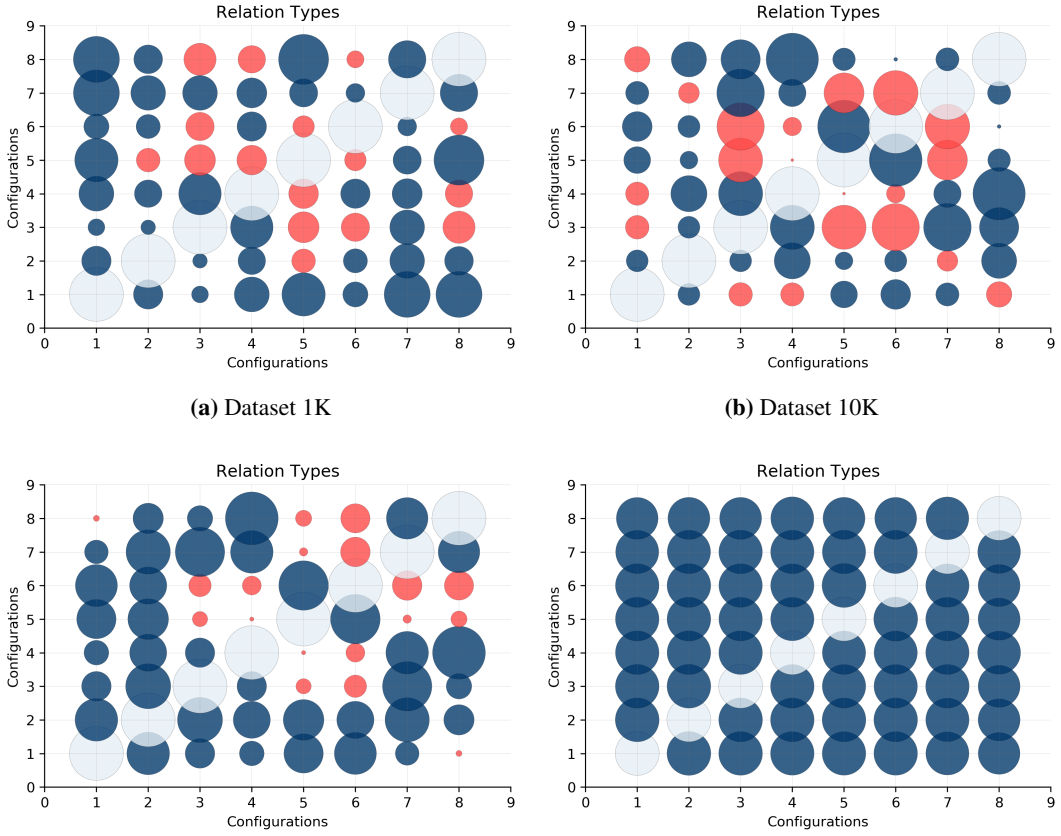
Metrics We report on the following metrics or observed variables: *Execution Time*: Elapsed time between execution of an engine and the delivery of the results. *Number of Results*: Number of triples generated by the KGC engine.

Implementations. The SDM-RDFizer and the testbeds are implemented in Python 3.6; the SDM-RDFizer is publicly available[63]. Furthermore, Jupiter Notebooks are available to generate the data and plot the results. Additionally, we have created a Docker image to run the testbeds and reproduce the experimental results[64]. The experiments were run in an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60GHz 20 cores, 100G memory with Ubuntu 16.04LTS.

Testbeds. Results of each configurations are ordered from lower to higher complexity and compared using the Pearson's correlation. A high positive value of correlation between two

[63] `https://github.com/SDM-TIB/SDM-RDFizer`

[64] `https://github.com/SDM-TIB/KGC-Param-Eval`



(a) Dataset 1K

(b) Dataset 10K

(c) Dataset 50K

(d) Combination of 1K, 10K, and 50K

Figure 5.3: Comparison of Knowledge Graph Construction Tools on Different Types of Relations. The first four (4) configurations, i.e. 1-4 in both x-axis and y-axis, represent results of SDM-RDFizer on *1-N*, *N-1*, *N-M*, and combination of all relations types, respectively. The later configurations, 5-8 both in x-axis and y-axis, shows results of RMLMapper on *1-N*, *N-1*, *N-M*, and combination of all relations types, respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation while red bubbles show a negative correlation. The plots reveal that both type of relations and size of the dataset need to be taken into account to uncover patterns in the behavior of the engines.

configurations indicates that the corresponding engines had a similar behavior, i.e. the trends of execution time of the tools are similar; they are represented with blue bubbles in our plots. When a configuration is compared to itself, the Pearson's correlation reaches the highest value (1.0), represented with grey bubbles in our plots. On the other hand, a negative value indicates that there is an inverse correlation between the engines, i.e. they exhibit an opposite behavior; they are represented with red bubbles.

Discussion of the Observed Results

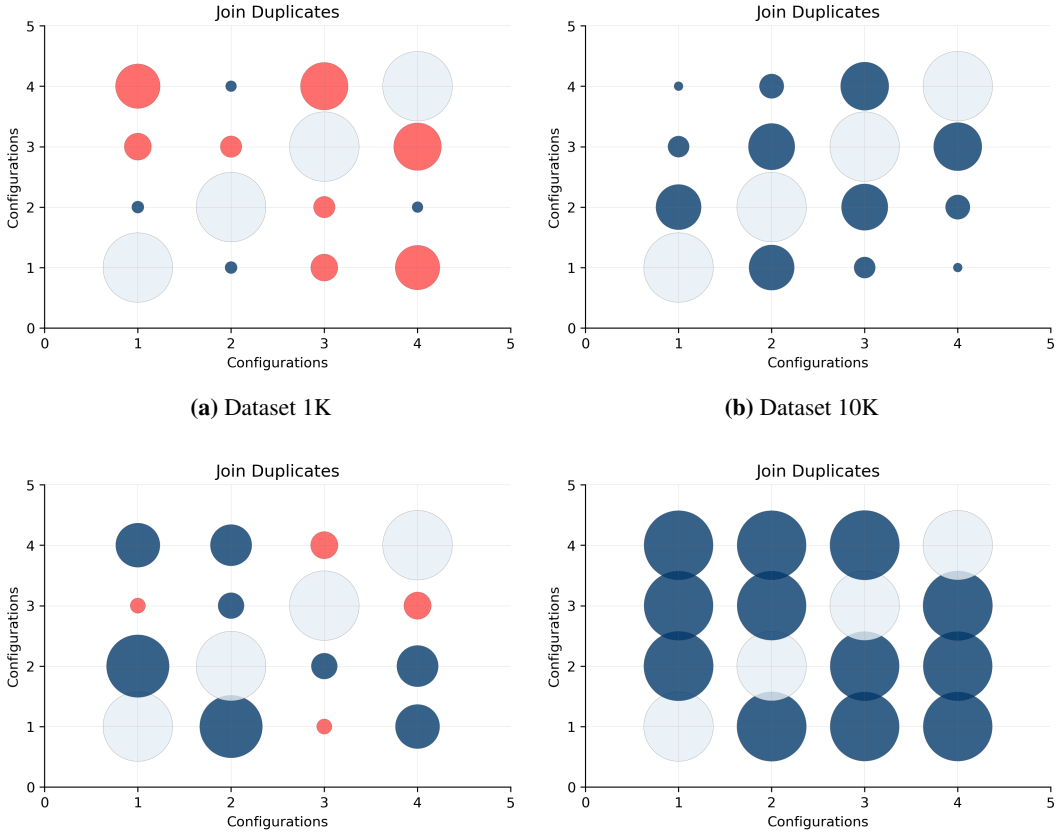
We observe that the behavior of the engines can be affected when multiple variables are involved in a testbed (e.g. size and relation type) or when different levels of complexity of a variables (e.g. low, high join selectivity). We discuss the obtained results during our evaluation over the different configurations and parameters involved in each experiment:

Dataset Size (Naïve): Figure 5.2 depicts the comparison of engines when the dataset size is considered. When `configuration 1` is compared to itself, the Pearson's correlation value is 1.0; additionally, it is high and positive when it is compared to `configurations 2, 3, 5, and 6` (large blue bubbles). Using this parameter, the correlation analysis suggests that both engines behave similarly in all configurations. Moreover, this indicates that only considering the data size is not enough to uncovered the properties of the studied engines.

Relation Types: Figure 5.3 reports on the correlation of different configurations for various join relation types. We can observe in Figures 5.3a, 5.3b, and 5.3c several red bubbles, indicating a negative correlation in the behavior of the compared configurations and engines. Contrary, Figure 5.3d does not depict any red bubble, suggesting thus, that the two engines in all the configurations exhibit the same behavior. These results clearly illustrate the need of considering different configurations and parameters in order to avoid drawing wrong conclusions about the main characteristics of existing tools.

Join Duplicates: Figure 5.4 depicts the correlation between different configurations when different setting of duplicates are produced during the execution of joins between triple maps. As can be observed, Figures 5.4a, and 5.4c include several red bubbles that indicate an opposite behavior of the engines. Contrary, Figures 5.4b, and 5.4d suggest that both engines behave similarly.

Join Selectivity: Figure 5.5 shows the correlation between different configurations for the selectivity of join conditions. Similarly, these testbeds reveal contradicting patterns in the behaviors of the studied engines. On one hand, Figures 5.5a, 5.5b, and 5.5c are composed of several red bubbles and indicate that these engines perform differently whenever the selectivity of the join condition is changed. Surprisingly, when the size of these datasets are also taken into account in the testbed (Figure 5.5d), these patterns are hidden, and the results of the evaluation suggest that both engines perform similarly whenever the selectivity of the join condition is changed.



(a) Dataset 1K

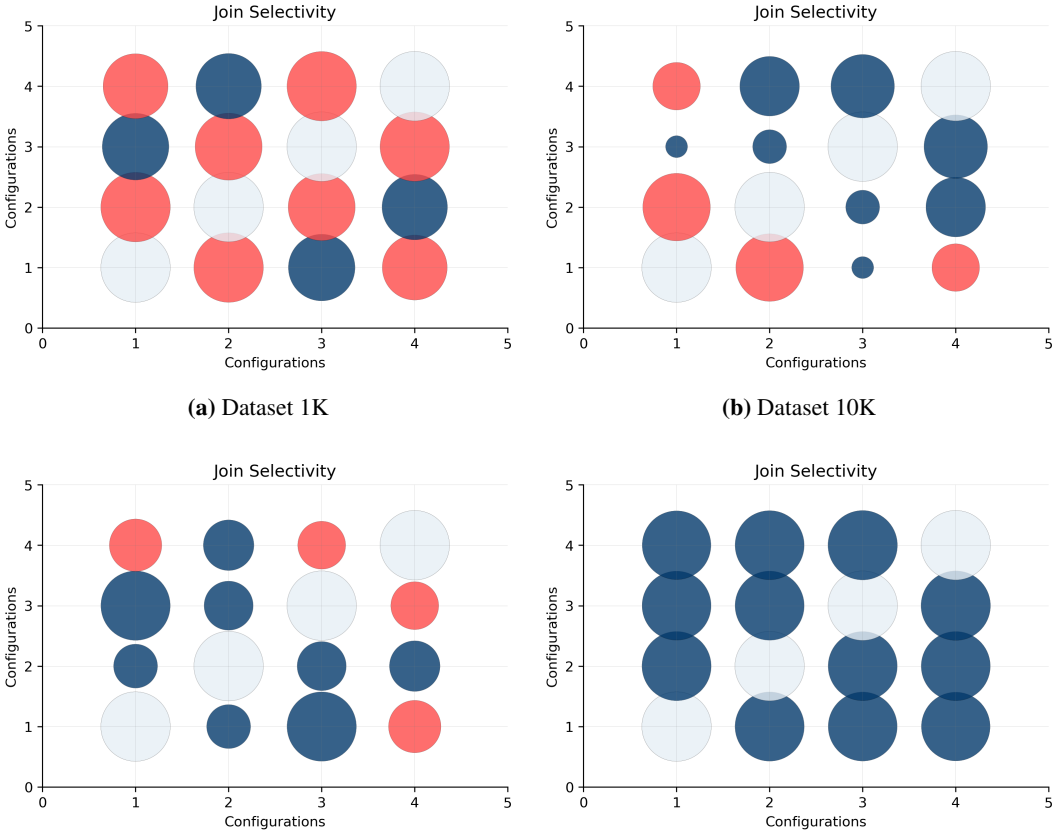
(b) Dataset 10K

(c) Dataset 50K

(d) Combination of 1K, 10K, and 50K

Figure 5.4: Comparison of Knowledge Graph Construction Tools on Duplicates during Join. The first two (2) configurations, i.e., 1-2 on x-axis and y-axis, represent results of SDM-RDFizer on datasets with *low* (5%-20% of data) number of duplicates and *high* (30%-50% of data) number of duplicates generated during joins, respectively. The last two configurations, i.e., 3-4 on x-axis and y-axis, represent results of RMLMapper on datasets with *low* number of duplicates and *high* number of duplicates generated during joins, respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation while red bubbles show a negative correlation. Results evidence that both join duplicates and dataset size are needed for characterising an engine performance.

The results reported in this experimental study provide clear evidence of the importance of the variables and configurations that composed the methodology devised in this work. Actually, in the four studied cases, they reveal important patterns that could not be observed whenever other parameters were studied simultaneously. Based on these observations, we can conclude that these variables and configurations should be included in the benchmarks in order to ensure



(a) Dataset 1K

(b) Dataset 10K

(c) Dataset 50K

(d) Combination of 1K, 10K, and 50K

Figure 5.5: Comparison of Knowledge Graph Tools on Join Selectivity. The first two configurations, i.e., 1-2 on x- and y-axis represent SDM-RDFizer on joins with *high* selectivity (5%-20% of data) and joins with *low* selectivity (60%-100% of data), respectively. Configurations 3 and 4 represent RMLMapper on joins with *high* selectivity (5%-20% of data) and joins with *low* selectivity (60%-100% of data), respectively. Grey bubbles correspond to correlation value of 1.0; blue bubbles show a positive correlation while red bubbles show a negative correlation. Dataset size and join selectivity affect both engines differently.

that the characteristics of knowledge graph construction engines are uncovered. Thus, these observations allow us to answer our three research questions: **RQ1**, **RQ2**, and **RQ3**. We encourage developers and users of knowledge graph construction tools to bear in mind them during benchmarking in order to draw clear conclusions about the performance of their tools.

5.2.4 Conclusions

In this section, we performed an in-depth analysis of the variables and configurations that impact on the behavior of two engines. The observation that existing engines exhibit heterogeneous behaviors whenever small changes in the testbeds are conducted, motivated the need of conducting this study involving a set of parameters that can reveal patterns in the behavior of the studied engines. Additionally, the lack of testbeds encouraged us to acquit the definition of variables and configurations that enable for the characterization of the pitfalls of existing engines and for identifying the list of challenges and research directions in the state of the art. With the proposed analysis and the results of the experimental study, we contribute with an empirical configuration that can be reused for the evaluation of other knowledge graph construction tools and mapping languages (e.g., SPARQL-Generate, TARQL, or R2RML). Furthermore, our set of variables and configurations can be utilized as a guideline during testing and benchmarking. One of the main lessons learned during the definition and evaluation of our approach, is that none of the evaluated engines behaves consistently whenever the complexity of the testbeds increases. Our ambition is that the reported results inspire the community to define general testbeds that facilitate the understanding of the state of the art and the development of novel tools for the construction of knowledge graphs at large scale. In the future, we plan to define testbeds and conduct a more detailed analysis of other engines and mapping languages. Moreover, we envision to motivate the community to conduct a joint effort in the definition of benchmarks that enable for fair evaluations of knowledge graph construction tools with replicable and generalizable results.

5.3 GTFS-Madrid-Bench: Evaluation of Virtual Knowledge Graph Construction Engines

We have identified several challenges for the development of a benchmark for virtual knowledge graph access that can be grouped into data, queries and mappings dimensions. The data challenges refer to having multiple data sources in an assortment of formats based on real-world data, and that can scale to large sizes. The challenges referred to queries point to SPARQL queries where different sources can be identified, where relations among sources (according to the specific data model) are exploited, and where necessary features of SPARQL are included to represent real-life use cases. Finally, the main mappings challenge is to include the relevant

parameters that affect in the generation of the virtual knowledge graph (Chaves-Fraga *et al.*, 2019) and give support to a set of declarative mapping languages.

To address these challenges, in this section, we describe a "virtual knowledge graph access" benchmark, GTFS-Madrid-Bench, that serves several purposes: (i) to evaluate and compare the performance of a mix of KGC engines that access several (homogeneous) sources in the same format, but where the mapping language used by each engine is specific to the data format considered; (ii) to evaluate virtual KGC engines over heterogeneous data sources; and (iii) to evaluate the strengths and weaknesses of both approaches. The general case of GTFS-Madrid-Bench is the comparison of the performance of KGC engines. The proposed GTFS-Madrid-Bench is composed of the following elements:

- Several collections of sources in different formats (e.g. CSV, JSON, SQL, XML), which derive from the GTFS[65] feed from the metro of the city of Madrid. These collections are scaled up so as to allow scalability testing.
- A set of mappings represented in the family of declarative languages that address different source formats (RML, R2RML, xR2RML, ontop OBDA mappings) that map the GTFS-based data sources into the Linked GTFS ontology[66].
- A set of 18 SPARQL queries of varied complexity.
- A set of well-established measurements (Lanti *et al.*, 2015; Mora & Corcho, 2013) that can be taken during the different phases of the KGC workflow (Acosta *et al.*, 2011; Lanti *et al.*, 2015), such as query rewriting, query translation, query execution and query aggregation time.

GTFS-Madrid-Bench offers a fair environment for the comparison of different KGC engines, regardless of the mapping language that they have implemented, as long as the new mappings follow the same restrictions and specifications defined in the benchmark. Thus, newly released tools may be evaluated with the benchmark. Additionally, although we have generated our

[65]The General Transit Feed Specification (GTFS) is a de-facto standard developed by Google for the description of public transport planning, routes and fares, among others. In recent years its popularity has increased thanks to its simplicity and the fact that it has not only been adopted by Google Maps, but also by other route planning systems such as Open Trip Planner or navitia.io.: `https://developers.google.com/transit/gtfs/`

[66]`https://github.com/OpenTransport/linked-gtfs`

datasets from the GTFS feed of the city of Madrid metro system, any other city's GTFS feed may be used as data in the benchmark.

We provide a data generator to scale up the original data in terms of size, and distribute the datasets over different formats (e.g. JSON, XML, CSV, RDB). We demonstrate the use of GTFS-Madrid-Bench with five open source engines: Morph-RDB[67], Ontop[68], Ontario[69], Morph-CSV[70] and Morph-xR2RML[71].

In summary, the main contributions of this work are:

1. C1: The proposal of a comprehensive and representative benchmark that includes a set of data sources, queries and mappings that allow evaluating and comparing multiple KGC engines for virtual knowledge graph access.

2. C2: The extension of existing OBDA benchmark requirements to take into account (i) metrics that are commonly used in federated query processing benchmarks; and (ii) steps defined in new KGC engines (Corcho *et al.*, 2019).

3. C3: A data generation process where single and mixed data formats are scaled-up based on the features of the original data model, integrating state of the art data generator proposals for benchmark OBDA engines (Lanti *et al.*, 2017).

4. C4: Evaluation of the proposed benchmark over five different engines, discussion of the obtained results and identification of the current limitations in the state of the art and future lines of work.

5.3.1 Concepts and Definitions

In this section, we introduce the main concepts and definitions that are later used to explain our work. Besides this, well-known concepts from the literature such as SPARQL queries and result sets (The W3C SPARQL Working Group, 2013), or ontologies (McGuinness *et al.*, 2004) will be used throughout this work.

[67] `https://github.com/oeg-upm/morph-rdb`
[68] `https://github.com/ontop/ontop`
[69] `https://github.com/SDM-TIB/Ontario`
[70] `https://github.com/oeg-upm/morph-csv`
[71] `https://github.com/frmichel/morph-xr2rml`

Sources & Dataset: we define a source as a tuple $\gamma = (\varphi, \Sigma, f)$ where φ is the data of any entity from our domain, Σ is the model of the data, e.g. the columns of a CSV or the schema of a database table for SQL, and f is a specific data format such as CSV, JSON, XML, or SQL, among others. We define a dataset as a set of *Sources*, i.e., $\mathcal{D} = \{\gamma_1, \gamma_2, ..., \gamma_n\}$.

Example 1. We define the following dataset $\mathcal{D}_1 = \{(Routes, \Sigma_1, SQL), (Stops, \Sigma_2, JSON)\}$ that involves the data of the metro routes (13 instances) and metro stops (1262 instances) in SQL and JSON formats, respectively. Both sources rely on different schemata Σ_1 and Σ_2, the first specifies the columns of a table and the second the keys of a JSON.

Dataset Generator: we define a dataset generator as a function δ that takes as input a tuple ($\mathcal{D}$, s) where $\mathcal{D}$ is a dataset and s is a non-negative number that specifies a scale factor. The output of δ is a dataset $\mathcal{D}'$ containing enlarged versions, according to s, of the data (φ) within the sources of $\mathcal{D}$.

Example 2. Assuming $\mathcal{D}_1$ from *Example 1* and a scale factor s of 2.5, a dataset generator may produce the following $\mathcal{D}' = \{(Routes{-}2.5, \Sigma_1, SQL), (Stops{-}2.5, \Sigma_2, JSON)\}$. Notice that the schematas and the formats are the same, but the data of $Routes2.5$ and $Stops2.5$ has been scaled up from their versions in $\mathcal{D}_1$, containing 189 and 3536 instances respectively.

Mapping: a mapping m is a set of rules that specify the relationship between an ontology and the model of one or more sources. A mapping rule relates the elements within the schema of a source with elements from an ontology, including constants. In other words, a mapping rule r contains the correspondences between an element e within a schema of a source Σ and an element e_* of an ontology Σ_*. The ontology is known as unified view since it is the output of translating heterogeneous sources into the same model, i.e., the ontology.

Example 3. Given the LinkedGTFS ontology and a CSV file with the columns "id" and "route", a mapping may state that each row generates a subject that includes the value of the column "id', the predicate *foaf:name* and its object with the corresponding value in the column "route".

Experiment configuration: we define an experiment configuration c as $(\mathcal{D}, q, M)$ where $\mathcal{D}$ is a dataset, q is an SPARQL query and M is a set of mappings.

Variable	Requirement
Ontology	The **ontology** should include classes with data and object properties
Dataset	The **virtual instance** should maintain the constraints defined in the original dataset
Dataset	The **virtual instance** should be based on real world data
Dataset	The **virtual instance** should be distributed in different data formats
Mappings	The **mappings** should be able to indicate the format of the source
Mappings	The **mappings** should be expressed using well known mapping languages
Queries	The **query set** should be based on actual user queries
Queries	The **query set** should be complex enough with relations among same but also different data sources
Metrics	The **metrics** should provide relevant general information but also specific measures for each defined phase

Table 5.12: Virtual Knowledge Graph Access Benchmark Requirements

Example 4. We can specify the following experiment configuration $(\mathcal{D}_1, q1, \{shapes, trips\})$, where $\mathcal{D}_1$ is the dataset specified in *Example 1*, q_1 is the SPARQL query reported in Table 5.15, and M is the set of mappings $\{shapes, trips\}$ reported in Table 5.14.

Processor: Given an experiment configuration c and an ontology Σ_*, a processor represents a software component that encodes the function ϕ that takes as input a pair (c, Σ_*) and outputs a SPARQL result set R (The W3C SPARQL Working Group, 2013).

Internally, the processor translates the SPARQL query q into one or more queries expressed in different languages, depending on the formats within the dataset of c, using the mappings M. Then, the processor distributes and evaluates the queries and gathers the results. As a result, a unified result set is provided as output. This task is known as **Virtual Knowledge Graph Access**. We distinguish two kinds of processors: OBDA and OBDI. The former ones are able to handle only experiment configurations where all the data sources have the same data format. The latter ones are able to handle any experiment configuration.

5.3.2 Benchmark Proposal

The GTFS-Madrid Benchmark consists of an ontology, an initial dataset of the metro system of Madrid following the GTFS model, a set of mappings in several specifications, a set of queries according to the ontology that cover relevant features of the SPARQL query language, a data generator based on a state of the art proposal (Lanti *et al.*, 2017), and a set of relevant metrics. In the following sections we describe in detail the resources of our virtual knowledge graph access benchmark. They are aligned with an extension of the requirements detailed in (Lanti *et al.*, 2015) (focused on benchmarks for OBDA) that we tailor to our context (Table 5.12). All the resources described in this section are available online[72].

5.3.2.1 The Linked GTFS Ontology

GTFS is a *de-facto standard* developed by Google for the description of public transport schedules, routes, fares, etc. The specification defines the headers of 13 types of CSV files and a set of rules. Each file, as well as their headers, can be mandatory or optional and they have relations among them.

The Linked GTFS vocabulary[73] can be seen as an ontology that represents the entities, properties and relationships described in the GTFS specification. The GTFS-Madrid-Bench mappings have been aligned to a subset of this vocabulary as the subway feed provides only the mandatory CSV files from the GTFS specification. Its conceptual model is shown in Figure 5.6 and a description of its classes is given in Table 5.13. The ontology usually defines one class for each of the sources in the GTFS specification with the corresponding data and object properties, but there are some additions. The `gtfs:Service` class represents information of the dates when a service (represented in GTFS in the files calendar and calendar_dates) is available for one or more routes, the ontology also adds the `gtfs:ServiceRule` class together with its two subclasses (`gtfs:CalendarRule` and `gtfs:CalendarDateRule`) to represent the service rules specified in the calendar and calendar_dates files. Finally, the class defined as `gtfs:WheelchairBoardingStatus` and its three possible values (instances) have also been added to represent the corresponding field definitions in stops and trips.

In general, all of the ontology classes have been populated except for `gtfs:FareClass` and `gtfs:FareRule` because the Madrid GTFS data does not contain information on these

[72] `https://github.com/oeg-upm/gtfs-bench`
[73] `https://github.com/OpenTransport/linked-gtfs`

Class	Description
Agency	Agency that operates a certain transport mode
Stop	Physical location where a vehicle stops or leaves. Multiple routes may use the same stop. A stop may be wheelchair accessible.
Route	Collection of one or more trips. Usually two trips in each direction.
Trips	A trip in a certain direction passes by stops. A trip is associated with a shape.
StopTimes	An ordered sequence of stops. Includes their arrival and departure times.
Service	Set of dates when a service is available. A Service follows a rule that may have exceptions.
ServiceRule	May be a calendar rule or a calendar date rule.
CalendarRule	For a certain period, weekdays where active.
CalendarDateRule	Date to add or delete a service.
Shape	A polygon associated to a trip.
Frequency	Frequency of a trip.
WheelchairBoardingStatus	Indicates whether wheelchair boarding is possible. Available for a trip or a stop.

Table 5.13: The LinkedGTFS Ontology: Classes and their Descriptions

two entities. The `gtfs:RouteType` class is not considered because the data covers only the Metro system.

5.3.2.2 Dataset Generation

Dataset generation for a virtual knowledge graph access benchmark should be focused on the two main variables that allow testing the capabilities of the engines: (i) data size, and (ii) formats in which data can be expressed. In the context of data generation for OBDA, VIG (Lanti *et al.*, 2017) proposes the use of R2RML mappings for an efficient scale up of the size of an instance of an RDB dataset. In this case, only one data format (SQL) is involved in the process.

We use GTFS as the original data source for several reasons: First, GTFS has been the *de-facto* standard for publishing transport data on the web, it also comes with a clear specification, making it easy to understand. Second, the GTFS model comprises several entities that are related through a variety of relationships. In addition it includes different data types such as

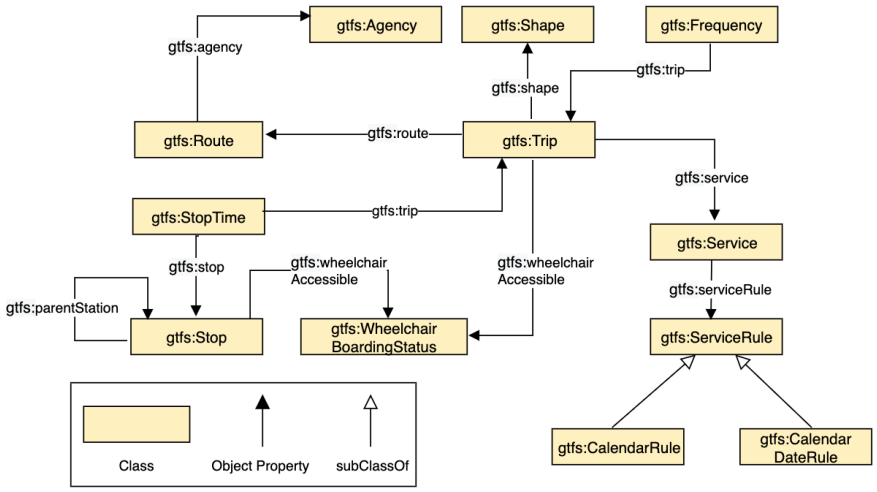


Figure 5.6: LinkedGTFS Ontology. Subset of the LinkedGTFS ontology used in the GTFS-Madrid-Bench for virtual knowledge graph access. There are eleven object property relations among the classes, and two subClassOf relations.

strings, integers, and booleans. Finally, many cities have adopted the GTFS data model and have published their GTFS data online. Although in our benchmark we propose the use of the GTFS Madrid subway data, GTFS data from a different city could be used as the original data source.

The GTFS-Madrid-Bench proposes an extended workflow using VIG as the Dataset Generator engine for the generation of the datasets taking into account multiple data formats (see an example in Figure 5.9). We describe the detailed steps of the proposed data generation workflow together with examples:

1) **Data preparation.** The original data source, GTFS, is in CSV format. VIG requires an instance of an RDB and an R2RML mapping for scaling up the data source. We use Morph-CSV[74], which takes as inputs a set of spreadsheets in the form of CSV files, their corresponding annotations using CSVW (Tennison *et al.*, 2015) and an RML mapping (Dimou *et al.*, 2014). It automatically produces the corresponding schema of an RDB (identifying typical constraints such as datatypes, PK/FK, indexes and NULLs) and an R2RML mapping document, which are the inputs for VIG.

   For the Madrid-GTFS-Bench, we use as input an open data dataset $\text{GTFS}^{CSV}_{mad} = (\text{GTFS}_{mad}$, GTFS, CSV). GTFS_{mad} is the set of data sources of the subway network of Madrid that

[74] `https://github.com/oeg-upm/morph-csv`

has been provided by its transport authority according to the schema GTFS as described in its specification[75]. This dataset is composed of a set of CSV files containing data of Agency, Route, Shape, Frequency, Trip, StopTime, Stop, Calendar and CalendarRule. This input is fixed during all of the data generation process, which means that the generated datasets are defined by the same schema and all of the generated data is obtained from this initial dataset. We create the corresponding RML mapping rules and CSVW metadata annotations and, using the Morph-CSV engine, we generate the corresponding RDB instance GTFS-SQL-1=(GTFS$^{SQL}_{mad}$,1) dataset and the R2RML mapping rules.

2) **Data creation.** VIG (Lanti *et al.*, 2017) takes into account the ontology and the set of R2RML mappings to generate each dataset. This engine also receives as input a scale value *s* that indicates that the size of each table of the database increases *s* times. The output of VIG is a set of CSV files, one file for each table of the RDB. In this step the dataset GTFS-CSV-s=(GTFS$^{CSV}_{mad}$,s) is generated, where *s* is the selected scale value.

3) **Data distribution.** Finally, each dataset generated using VIG is distributed in several formats. We use open source tools to perform this step such as csv2json, from Python CSVKit[76] and di-csv2xml[77], depending on the data formats (JSON and XML). We divide the distribution into two categories in order to cover both OBDA and OBDI approaches:

   In the first category, focused on providing support to OBDA techniques, the sources of each dataset are transformed into a single format (e.g., CSV files are transformed into JSON files). The datasets are transformed to the corresponding ones in JSON, XML, SQL and MongoDB obtaining the following datasets: GTFS-F-s=(GTFS$^{F}_{mad}$,s) where *s* is the scale value and F $\in$ {JSON, XML, SQL, MongoDB}.

   In the second category, focused on virtual KGC approaches, the sources of each dataset have to be transformed from the CSV files into multiple formats (e.g. CALENDAR is a JSON document, AGENCY is a XML file, etc.). To distribute the files and based on the GTFS model, the user may select the sources associated to each format and then the benchmark generates the dataset and the corresponding set of mapping rules.

   Additionally, the benchmark provides by default two datasets taking into account the selected formats (JSON, CSV, XML, SQL and MongoDB) where the number of relations

[75] `https://developers.google.com/transit/gtfs/`
[76] `https://csvkit.readthedocs.io/en/1.0.3/scripts/csvjson.html`
[77] `https://github.com/blue-yonder/di-csv2xml`

(joins) among sources in different formats is minimized or maximized. The formats of the sources in these datasets is also configurable during the generation data process, in order to allow users to analyze the impact of joins over formats with different features. For example, the join selectivity between shapes and trips is different than the join selectivity between routes and agencies, and depending on the formats of each source, the total query execution time of a processor may be impacted. We describe each dataset in more detail:

- Best Dataset: The number of joins among sources in different formats is minimised but ensuring that all of the formats are covered. The aim of this configuration is to study the behavior of the engines when they have to deal with different data sources but where most of the joins are done between sources in the same format. Hence they may delegate their treatment to the underlying data source manager (e.g. MySQL in RDB) and apply common optimisation techniques in query translation approaches (Priyatna *et al.*, 2014). To meet this requirement and, having 5 possible formats for the data sources, the proposed groups for best dataset are: trips, shapes, calendar and calendar_dates sources in one group, routes and agency in another, frequencies in the third group, stop and stop_times in the fourth one and feed_info in the last one. This composition generates the GTFS^{B}_{mad} dataset. We show a possible example of a best dataset in Figure 5.7.
- Worst Dataset: The number of joins among sources in different formats is maximised and the five formats are covered. In this distribution, all the possible joins are among sources in different formats. This means that the virtual KGC engine may be enforced to perform the joins after the execution of the translated queries over the original data sources. In the same manner as the best dataset, the groups of sources are: shapes and stops in one group, trips and feed_info in another, calendar and agency in the third group, routes and stop_times in the fourth and calendar_dates and frequencies in the last one. This composition generates the GTFS^{W}_{mad} dataset. We show a possible example of the worst dataset in Figure 5.8.

We want to be able to compare the results obtained by processors with the results obtained by the materialized graph in RDF. For this purpose, we take the output of VIG (e.g., GTFS-CSV-5, GTFS-CSV-10) and we run a materialized KGC process using the SDM-RDFizer[78]

[78] `https://github.com/SDM-TIB/SDM-RDFizer`

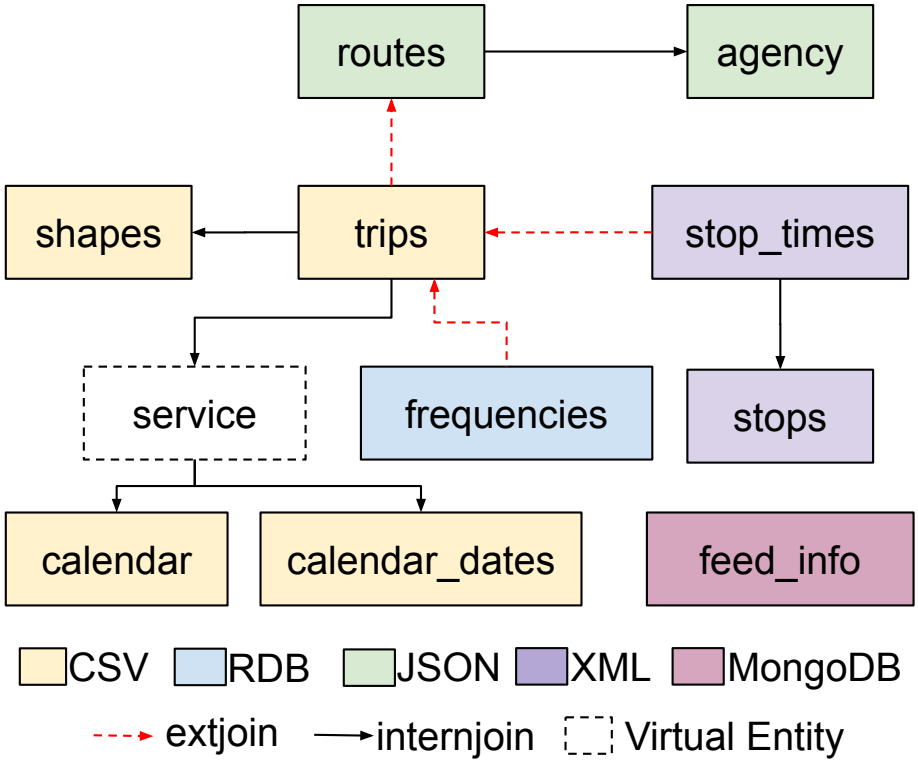


Figure 5.7: Example of Best Dataset. Best dataset distributes the formats over the data sources ensuring that at least there is one source per each format and the joins among different formats are minimised. *extjoin* means that there is a relation between sources in different formats and *internjoin* means that the joins are between sources in the same format.

engine, which generates the KG in RDF using RML mapping rules. We select this tool because it passed all the RML Test Cases (Heyvaert *et al.*, 2019) for CSV files[79], hence we assume that the generation is correct, and that it provides a set of techniques to optimize the generation of RDF at scale.

5.3.2.3 Mappings

Mappings play one of the most important roles in the benchmark since they are the main element used for the query translation process. In the state of the art there are multiple engines and tools that use different mapping languages. We select a set of the most relevant declarative mapping languages in the state of the art and we generate the corresponding mapping rules. In more detail, the GTFS-Madrid-Bench provides:

[79] `http://rml.io/implementation-report/`

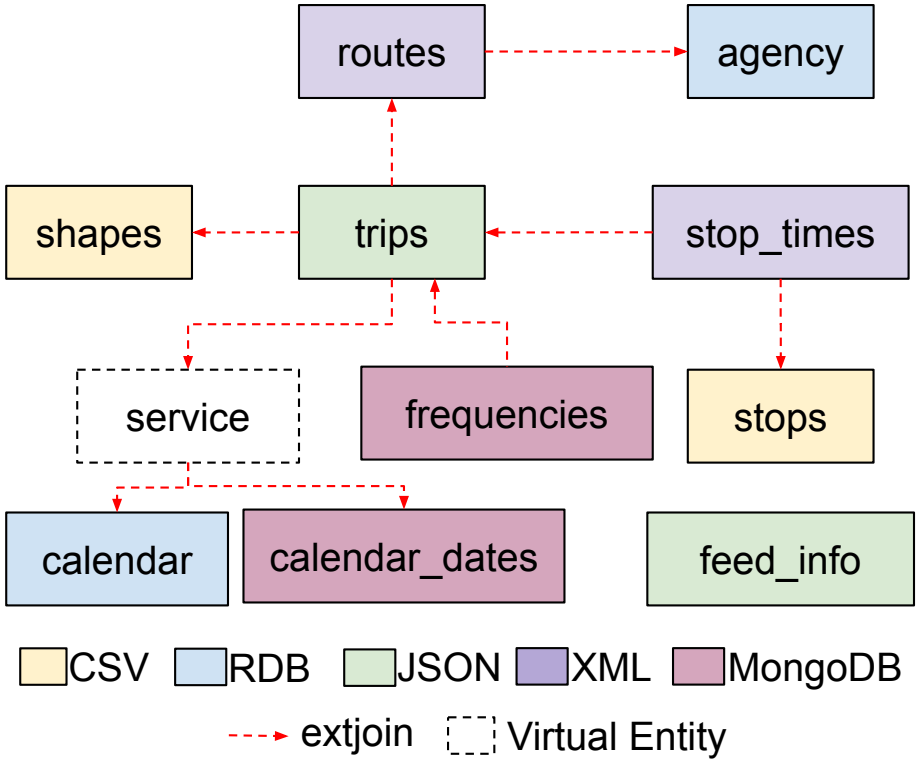


Figure 5.8: Example of Worst Dataset. Worst dataset distributes the formats over the data sources ensuring that at least there is one source per each format and the joins among different formats are maximised. *extjoin* means that there is a relation between sources in different formats.

- One R2RML mapping document for accessing SQL datasets.
- One xR2RML mapping document for accessing MongoDB datasets.
- Seven RML mapping documents[80] for accessing CSV, JSON, XML, SQL, MongoDB, Best and Worst datasets.
- One CSVW metadata file to provide annotations for the CSV datasets.

Conceptually, all the mappings represent the same relations among the concepts of the ontology and the concepts of the GTFS model, but each one has been developed according to a specification that handles the characteristics of each data format. The mappings are composed by a set of rules representing the relation of one element in the ontology with the corresponding schema element from a source. An overview of the rules within the mappings developed for

[80]Provided in RML and YARRRML serializations

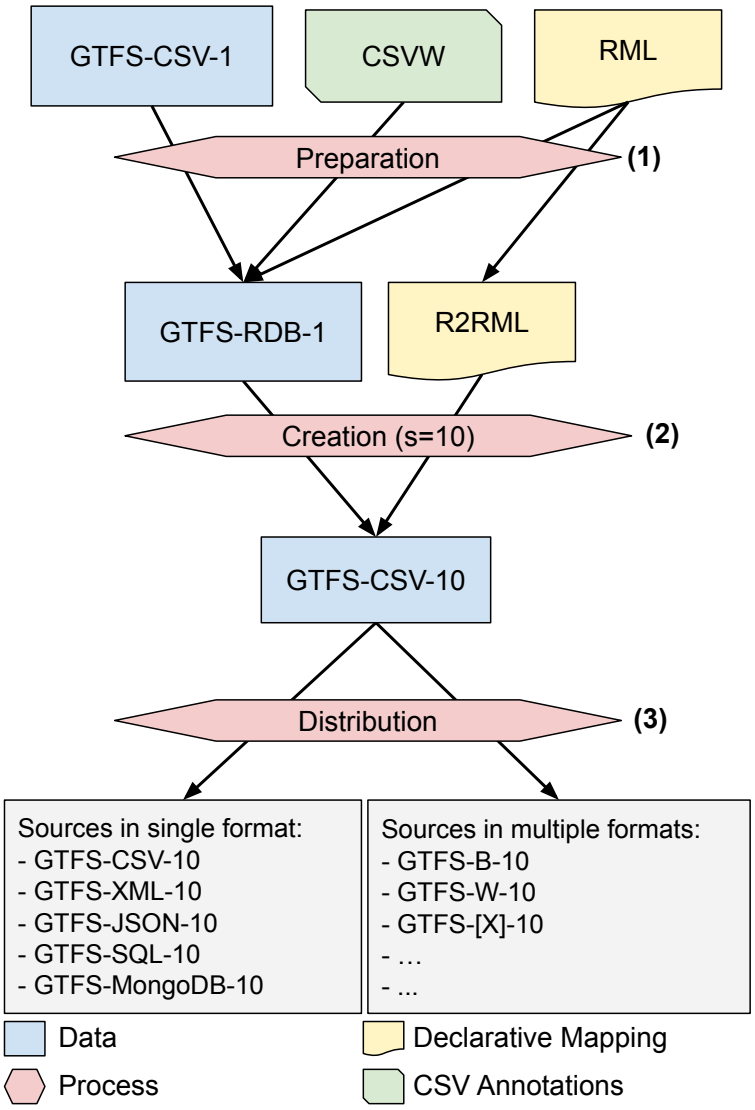


Figure 5.9: GTFS-Madrid-Bench Generation Workflow with scale value 10. From the original 10 CSV files of Madrid Metro GTFS we use (i) Morph-CSV to generate the corresponding RDB instance and an R2RML mapping that are the required inputs for (ii) scaling up the data using VIG and (iii) distributing the generated CSV dataset to the different formats.

TriplesMap	Source	Classes	#PredicateObjectMap	#Predicates	#Objects	#RefObjectMap
shapes	shapes	gtfs:Shape	4	4	4	0
trips	trips	gtfs:Trip	8	8	5	4
calendar_rules	calendar	gtfs:CalendarRule	9	9	9	0
calendar_date_rules	calendar_dates	gtfs:CalendarDateRule	2	2	2	0
stops	stops	gtfs:Stop	12	12	11	1
stoptimes	stop_times	gtfs:StopTime	9	9	7	2
routes	routes	gtfs:Route	8	8	7	1
agency	agency	gtfs:Agency	6	6	6	0
frequencies	frequencies	gtfs:Frequency	5	5	4	1
feed	feed_info	gtfs:Feed	6	6	6	0
service1	calendar	gtfs:Service	1	1	0	1
service2	calendar_dates	gtfs:Service	1	1	0	1
Total	10	11	71	71	60	11

Table 5.14: Mapping Features for transforming LinkedGTFS to GTFS. Each TriplesMap of the GTFS mapping file and its corresponding features: the related source, number of Classes, PredicateObjectMaps, Predicates, Objects and RefObjectMaps (joins).

this benchmark is shown in Table 5.14. The rules of the mappings are very relevant since they contain many parameters that impact on the performance of the virtual knowledge graph access engines (Chaves-Fraga *et al.*, 2019).

More in detail, each source of the GTFS feed has one associated TriplesMap with a rule to associate the generated entities to the class defined in the ontology, and a set of rules for the object and data properties. Additionally, there is a (virtual) entity, Service, in the data model with no corresponding source, what implies the definition of a set of mapping rules to generate the instances of the corresponding class (`gtfs:Service`). Following the GTFS specification, the identifier of Service can be found either in calendar or calendar_dates sources. This means that to be aligned with standard declarative mapping specifications (e.g. RML and R2RML only allow one source per TriplesMap), the mapping document needs to define two TriplesMap, one for calendar (service1) and another for calendar_dates (service2). This also implies that the trips TriplesMap has one predicate (`gtfs:service`) with two associated refObjectMaps, where the parentTriplesMap are service1 and service2; this allows generating all `gtfs:Service` defined in the original data source. Because the instances of `gtfs:WheelchairBoardingStatus` class are only objects in `gtfs:Trips` and `gtfs:Stops` triples, they are generated using the template property in trips and stops TriplesMap. In summary, the mapping contains rules to generate instances of 12 classes, 71 PredicateObjectMaps and Predicates, 60 Objects and 11 RefObjectMaps, covering the main features defined in state-of-the-art mapping specifications for KGC. All of the GTFS mappings are detailed in Appendix C using the YARRRML (Heyvaert *et al.*, 2018) serialisation.

5.3.2.4 Queries

Table 5.15 presents all the variables considered for the 18 queries in our benchmark. We have developed queries that are based on the Linked GTFS ontology, and are aligned with user stories in Madrid's transport domain, together with different combinations of values for the variables. It should be mentioned that the queries cover all of the data sources that were generated by the Madrid's transport authority as GTFS data from the metro system. These include agencies, routes, stops, trips, frequencies, shapes, calendar, and calendar dates. Although in the benchmark we have defined mappings to translate queries into the underlying query language of the source, these are independent from the queries (we have used these mappings to generate the materialized knowledge graph in the data generation step).

We have defined two sets of 18 queries with identical templates but differences in the constants that appear in subjects or objects of bounded triple patterns: (1) Baseline queries with constants that belong to Madrid's GTFS Linked Data, and (2) VIG queries with constants that belong to the datasets generated by the tool; these queries are executed in the evaluation.

5.3.2.5 Metrics

In this section we define the metrics that are used to evaluate the performance of Virtual Knowledge Graph access engines. The metrics consider the workflow followed by Virtual Knowledge Graph systems, and for each of the steps identified in the workflow we introduce a set of metrics to be measured and reported.

The workflow extends the OBDA phases identified by Mora and Corcho (Mora & Corcho, 2013), and Lanti et al. (Lanti *et al.*, 2015). In addition, it includes some of the steps that are defined by proposals that federate queries (Schwarte *et al.*, 2011). General metrics to be captured are **overall execution time**, **completeness of answers** and **initial delay**. Other metrics may be considered when the engine generates answers following a continuous behavior (Sharaf *et al.*, 2008), such as **dief@k** or **dief@t** proposed in (Acosta *et al.*, 2017). Additionally, for each phase of a workflow, a virtual knowledge graph construction engine may capture specific metrics that allow the identification of bottlenecks in the implementations. This relevant set of metrics for each phase are: (i) **loading time** during the starting phase when the ontology, mappings and query are loaded; (ii) **total number of requests** and **source selection time** during the source selection phase (the engine identifies the sources that can be used to answer the query); (iii) **query generation time** when the set of sub-queries to be evaluated over

Query	Description	#Triple Patterns	#Sources	OPTIONAL	Aggregation	Other features	FILTER		#Star-shaped groups	
							equal to	relational	w/o constants	w/constants
q1	All shapes	4	1						1	0
q2	All stops where the latitude is larger than a specific value	5	1	✓				✓	0	1
q3	Accessibility information of all stations	5	1	✓			✓		0	1
q4	All agencies and their routes	9	2	✓					2	0
q5	Services that have been added after a specific date	5	2					✓	1	1
q6	Number of routes covered by a specific agency	3	2		✓		✓		0	2
q7	All wheelchair-accessible stops in a specific route	15	4	✓		DISTINCT	✓		1	3
q8	Routes and their related trips, services, stops and stop times	14	5	✓					5	0
q9	Trips and associated shapes where latitude is larger than a specific value	7	2	✓				✓	1	1
q10	Number of trips that have a duration over a number of minutes	4	2		✓	DISTINCT		✓	1	1
q11	Trips that are available on a certain date	12	3			NOT EXISTS		✓	3	2
q12	Number of stops that are wheelchair-accessible grouped by route	10	4		✓	GROUP BY			3	1
q13	The accesses of all stations	6	1	✓					0	1
q14	All stops times and their related routes and stops ordered by their sequence, in a specific direction and service	8	3	✓		ORDER BY			3	0
q15	For all properties, triples that contain a specific word in the object placeholder	3	1				✓		0	1
q16	For all routes, all calendar changes in a specific month	8	3					✓	2	1
q17	Trips with their start and end time of the frequencies and associated routes	9	3						3	0
q18	All routes that have trips on Sunday	8	5			UNION			4	1

Table 5.15: Description of the GTFS-Madrid Benchmark Queries

Metric	Type or Phase	Dimension
General Metrics		
Total execution time	General	D, Q, M
# answers	General	D, Q, M
Initial delay	General	D, Q, M
Dief@k	C. Behaviour	D, Q, M
Dief@t	C. Behaviour	D, Q, M
Specific Metrics (Phases)		
Loading time	Starting	Q, M
Mapping trans. time	Starting	M
# requests	Distribution	Q
Source selec. time	Distribution	Q, M
Query gen. time	Distribution	Q
Query rewrit. time	Rewriting	Q
Query trans. time	Translation	Q, M
Query exec. time	Execution	Q, D
Query aggreg. time	Finishing	D

Table 5.16: Comparison between Benchmark Metrics and Dimensions. Relation between each relevant metric for the Madrid-GTFS-Bench and the dimensions that can impact over that metric. In the Dimension column, Q means query, M mappings and D data.

each data source is created, and the query plan is generated; (iv) **mapping translation time** when the engine requires to translate a provided mapping into another one in in a different language, maintaining a set of properties between them (Corcho *et al.*, 2019); (v) **query rewriting time** when the generated sub-queries are rewritten to other queries, taking into account potential inferences from the ontology and information in the mapping (Mora *et al.*, 2014); (vi) **query translation time** when the engine, taking the mapping into account, translates each sub-query to another one in the query language supported by the underlying data sources such as SPARQL-to-SQL (Chebotko *et al.*, 2009); (vii) **query execution time** when the translated queries are evaluated against the underlying data sources and the results are translated to RDF or as SPARQL bindings using the rules provided in the mappings; and (viii) **query aggregation time** when the results obtained for each sub-query are aggregated, including the removal of duplicates and the linking of resources. Variables that have an impact on the metrics have been grouped into three dimensions: Query, Data, and Mappings. The relation between each metric considered and the dimensions that can impact over that metric is shown in Table 5.16.

Query. The Query dimension variables refer to the structure of the queries, e.g. #triple patterns, #sources, and #star-shaped groups. A Star-shaped group is a group of triple patterns that are "joined" over the same subject or object variable (Vidal *et al.*, 2010). The most common case in real-world scenarios are subject star-shaped groups that represent properties that describe one source. The benchmark considers an increasing number of triple patterns, from 3 to 15, also the number of sources vary from 1 to 5. In particular we have several queries on 1 source with a varying number of triple patterns, and queries that have a large number of triple patterns combined with 4 and 5 sources. With respect to these two variables, our aim is to balance real-life use cases where several properties in the specification need to be combined and retrieved, and query complexity (worst case scenario is the "Worst Dataset" when the five sources are represented in the five available formats and the number of joins among sources is maximized).Furthermore, a large number of sources or triple patterns combined with a large number of non-instantiated star-shaped groups should impact overall execution time and also specifically impact query generation, query rewriting, query translation, and query execution times.

In general, queries in GTFS-Bench-Madrid combine those that contain single star-shaped groups ($q1$,$q2$, $q3$, $q15$) with those that contain chains of star-shaped groups, that is, where the object of a pattern in a group is the subject in the next group (with joins across different

sources): $q4$, $q5$, $q6$, $q7$, $q9$, $q10$, $q11$, $q12$, $q16$, $q17$, $q18$. According to the ontology structure shown in Figure 5.6, `gtfs:StopTime` relates to stops and trips and may lead to hybrid shapes such as $q8$ and $q14$. There is also the case of query $q13$, which refers one source and contains a self-join that relates an access to a station to its "parent" station.

Besides, as mentioned in (Montoya *et al.*, 2012), query plans generated by query evaluation systems during the subquery generation phase may be affected by the structural properties of a query. If the sources in the dataset are all represented in the same format, then query plans will be generated by the underlying engine (either an RDB engine or a NoSQL engine), and execution time will be affected by the number of joins within star-shaped groups and among these groups. When the sources of the dataset are not in the same format, the engine has to create the query plan. The performance will be affected by the plan proposed by the engine. Different combinations of these variables are considered in GTFS-Madrid-Bench queries: on the one hand we have a large number of triple patterns, sources and star-shaped groups in $q7$ and $q8$, and on the other hand queries like $q18$ combine a large number of sources and star-shaped groups with a medium-sized query (8 triple patterns).

Complexity of SPARQL queries is presented in (Pérez *et al.*, 2009), considering the SPARQL fragment with only AND and FILTER operators. Complexity is linear on the product of the dataset size and the size of the query (# triple patterns), and evaluation is NP-complete for queries constructed with AND, FILTER and UNION operators. Several queries in GTFS-Madrid-Bench have FILTER clauses and specifically, $q18$ contains a UNION of two triple patterns.

The evaluation problem becomes harder when the OPTIONAL operator is added (Pérez *et al.*, 2009). Additionally, the work described in (Xiao *et al.*, 2018b) presents optimization techniques applied in an OBDA setting specifically for queries that have to deal with OPTIONAL triple patterns, claiming that the underlying database systems do not optimize adequately these class of queries. Similar problems may be expected for querying CSV, XML and JSON data sources. We have designed eight queries that use OPTIONAL graph patterns (according to the corresponding non-mandatory attributes in the specification).

Constants in triple patterns together with FILTER with equality operators increase the selectivity of queries and are likely to reduce the cost of evaluating the query. According to (Montoya *et al.*, 2012), instantiated triple patterns have an important impact on the potential number of join intermediate results that may be generated throughout query execution. However, using a FILTER relational operator specially in the case of open ranges, e.g. a FILTER with a $>$

operator, may generate a large number of answers. We have considered several combinations of number of star-shaped groups with and without constants, $q8$ has no constants whereas in $q4$ both star-shaped groups in the query have bindings. An example of an intermediate case occurs in $q12$ with 1 out of 4 instantiated star-shaped groups.

Three queries contain the aggregated COUNT function, and one of these queries contains additionally the GROUP BY modifier. Other queries use language features like DISTINCT and ORDER, what will impact on the query execution time metric because all of them require an ordering of the tuples/entries of the underlying sources. We cover the impact of these variables in $q7$ and $q10$ with DISTINCT, and $q12$ and $q14$ with GROUP BY and ORDER BY respectively. Finally, having unbounded predicates in a query ($q15$) increases its complexity because the search space during query evaluation may be large.

The work in (Angles & Gutiérrez, 2016) studies the impact of negation in the computational complexity of SPARQL queries, it distinguishes four types of negation: negation of filter constraints, negation as failure, negation by MINUS and negation by NOT EXISTS. The use of NOT EXISTS introduces similar issues to sub-query evaluation because of the presence of correlated variables and the use of a nested iteration method to evaluate queries that contain this type of negation. Hence $q11$ contains negation with NOT EXISTS.

Mappings. Features of mappings are relevant because they may impact the performance of the engines. Previous work by (Chaves-Fraga *et al.*, 2019) evaluates different mapping variables that impact in the construction of a knowledge graph. Similarly, we consider that the following mapping variables influence overall query execution time and specifically query translation and query rewriting times. Regarding structure, we have considered the variables #Classes, #PredicateObjectMaps, #Predicates, #Objects, and #RefObjectMap that are presented in Table 5.14. Another variable is relation type, the mappings of the Madrid-GTFS-Bench include 1-1, 1-N, N-1 and N-M relation types. In general mappings for sources that represent N-M relationships (e.g. stop_times) are more complex and thus time consuming for query execution. Additionally, the variable `rr:termtype` of the `rr:objectMap` may also have an effect because the cost of generating a constant, a reference or a template is not the same.

Dataset. Variables in this dimension include dataset size and the formats of its sources. As already mentioned, datasets with different scale factors are generated in GTFS-Madrid-Bench. Size has an impact on the overall execution time, on the initial delay, and specifically on query

execution time because of the larger number of intermediate results. It also influences query aggregation time because in the benchmark, queries against larger datasets generate a larger number of answers.

The format variable may take a single value for datasets in only one format (RDB, CSV, XML, MongoDB, JSON) or multiple formats (Best, Worst and Random). This variable has an impact on the overall execution time, specifically on the query translation and query execution times, as well as on the number of answers because different formats have different access methods and different underlying query languages.

The work in (Montoya *et al.*, 2012) presents partitioning and data distribution in this dimension. In GTFS-Bench-Madrid there are fixed values for these variables: the partitioning is vertical and datasets and databases are loaded in local machines.

5.3.3 Sustainability and extensibility

The Madrid-GTFS-Bench is supported by a set of robust resources in order to ensure its sustainability. The benchmark can be adapted to any other virtual knowledge graph access engine that uses other mapping rules languages, or to other non-declarative proposals. The developers or users only have to create the mapping documents according to that specification. Additionally, virtual knowledge graph access engines that work with other graph query languages (e.g. Morph-GraphQL (Priyatna *et al.*, 2019)) can take advantage of our proposed benchmark.

A set of improvements for the data generation that we have identified are based on VIG, a robust and efficient engine for the generation of scalable datasets. Additionally, all the generated resources are available online [81] and their deployment (engines and databases) is done using docker images to ensure the reproducibility of the obtained results. Finally, because we define the dimensions of mappings and datasets taking into account the relevant parameters in the process of constructing knowledge graphs (Chaves-Fraga *et al.*, 2019), this benchmark can be also used to test the materialization KGC engines such as RMLMapper [82], RocketRML [83] or SDM-RDFizer [84] since, at this moment, there is no proposal to evaluate the performance and completeness of these engines in an objective manner.

The possibility to extend this benchmark is also one of the main points that differentiates this proposal to previous ones. First, there are multiple benefits obtained from relying on an

[81] `https://github.com/oeg-upm/gtfs-bench/`
[82] `https://github.com/RMLio/rmlmapper-java`
[83] `https://github.com/semantifyit/RocketRML/`
[84] `https://github.com/SDM-TIB/SDM-RDFizer`

open data model from the transport domain, such as linking this data with other data from the city and also having other GTFS transport systems feeds (e.g. metro and train datasets). In addition to queries that take into account the specific characteristics of the selected datasets, it is also possible to incorporate more complex mapping rules with extended features such as specific transformation functions (De Meester *et al.*, 2016), something that is difficult to address by previous proposals as their data model is usually relational database oriented (Bizer & Schultz, 2009; Lanti *et al.*, 2015). The incorporation of these features will ensure that we cover new characteristics of the new generation of virtual knowledge graph access engines without the need of creating a benchmark from scratch.

5.3.4 Experimental Evaluation

In this section we describe the evaluation performed using our benchmark. We first describe the selected virtual KGC engines involved in the evaluation, we describe the evaluation methodology and infrastructure, based on the use of docker images to ensure the reproducibility of the experiments, and finally, we provide the obtained results. All the resources used in this evaluation, such as queries, data, mappings, running scripts, results and docker images for engines and databases are publicly available online[85].

5.3.4.1 Tools

We selected the most relevant open source virutal KGC engines in the state of the art:

Ontario. Ontario (Endris *et al.*, 2019)[86] is an virtual KGC engine from heterogeneous data sources that is based on the concept of RDF molecule templates (RDF-MT) (Endris *et al.*, 2017). Ontario exploits the information provided by the mapping rules for creating the corresponding RDF-MT over the data sources. After the source selection and sub-query generation processes, Ontario translates the SPARQL query into the corresponding query language of the original data source. It supports the following formats: RDF, MySQL, CSV, TSV, JSON, XML, MongoDB and Neo4j.

Ontop. Ontop (Rodriguez-Muro & Rezk, 2015)[87] is an KGC system from RDB instances that includes both materialization and virtualization techniques. Ontop translates R2RML mappings into its own mapping language, called "OBDA mappings". These mappings, and a

[85] `https://github.com/oeg-upm/gtfs-bench`
[86] `https://github.com/SDM-TIB/Ontario`
[87] `https://github.com/ontop/ontop`

SPARQL query if available, are transformed into datalog rules, allowing semantic optimization techniques to be applied, and generating efficient SQL queries (e.g., self-join elimination). It only supports the SQL format.

Morph-RDB. Morph-RDB (Priyatna *et al.*, 2014)[88] is an R2RML engine that also includes materialization and virtualization techniques. The formalization of its query translation technique is based on the R2RML-based extension of SPARQL-to-SQL query translation algorithm proposed by (Chebotko *et al.*, 2009), originally designed to work with RDB-backed triples store. Similar to Ontop, several optimization techniques are also incorporated in order to generate more efficient SQL queries. It supports SQL and CSV files.

Morph-xR2RML. Morph-xR2RML (Michel *et al.*, 2015)[89] uses the xR2RML mapping language to support the generation of RDF lists, and to query data stored in NoSQL databases such as MongoDB.

Morph-CSV. Morph-CSV (Chaves-Fraga *et al.*, 2020b)[90] exploits the information of CSVW annotations and RML mappings to enforce implicit constraints over tabular data, explicitly declared in these annotations. It can be integrated on top of any existing SPARQL-to-SQL engine in order to enhance query completeness and performance.

We also intended to include other engines such as Squerall (Mami *et al.*, 2019a) or Polyweb (Khan *et al.*, 2019). In both cases, either the code is not available as open source or it was not feasible to run the engine due to the lack of documentation. Issues have been reported in their corresponding repositories, with the intention of alerting the authors and maintainers about the current limitations.

5.3.4.2 Setup

In this section we describe how we use our benchmark to evaluate several processors/engines. We have setup several experiment configurations for evaluating the selected processors. As an example, the experiment configurations for query $q4$ can be seen in Table 5.17. These experiment configurations have a fixed set of mappings with routes and agencies. The processor used to evaluate this query depends on the dataset, for example, Ontario in the case of the JSON dataset or Morph-RDB, Ontario and Ontop for SQL.

[88] `https://github.com/oeg-upm/morph-rdb`
[89] `https://github.com/frmichel/morph-xr2rml`
[90] `https://github.com/oeg-upm/morph-csv-sparql`

Query q	**Dataset** D	**TriplesMap** M	**Processor** ϕ
q4	$GTFS_{mad}$-CSV-s	$\{routes,agency\}_{RML}$	Morph-CSV
		$\{routes,agency\}_{R2RML}$	Morph-RDB
		$\{routes,agency\}_{RML}$	Ontario
	$GTFS_{mad}$-SQL-s	$\{routes,agency\}_{R2RML}$	Morph-RDB
		$\{routes,agency\}_{RML}$	Ontario
		$\{routes,agency\}_{OBDA}$	Ontop
	$GTFS_{mad}$-MongoDB-s	$\{routes,agency\}_{xR2RML}$	Morph-xR2RML
	$GTFS_{mad}$-XML-s-s	$\{routes,agency\}_{RML}$	Ontario
	$GTFS_{mad}$-JSON-s	$\{routes,agency\}_{RML}$	Ontario
	$GTFS_{mad}$-B-s	$\{routes,agency\}_{RML}$	Ontario
	$GTFS_{mad}$-W-s	$\{routes,agency\}_{RML}$	Ontario

Table 5.17: Experiment configuration example set. List of experimental configurations and processors for q4. D is a dataset where s is the scaling factor (i.e., 1, 5, 10, 50, 100, 500), M is the set of mappings, q is the SPARQL query, ϕ is a processor. q is a SPARQL query defined in the Appendix Section.

All the experiment configurations are loaded into a machine with the following characteristics: 2GHz CPU with 15 cores, 32 RAM, 200 GB HDD with Ubuntu 18.04 as its operating system. The machine contains a docker image for each of the processors: Morph-RDB v3.12.5, Ontop v3.0.0, Morph-CSV v1.0.0, Ontario v.0.3, Morph-xR2RML-1.1-RC2. All the engines are configured with the recommended settings provided in the corresponding online repository.

In terms of data size, we decide to evaluate the engines over the scale values (5, 10, 50, 100 and 500). After some preliminary tests, we observed that these values provide a good overview of the current state of the engines in terms of query evaluation performance. For each SQL dataset size, we create two docker images where the data is loaded, one as an instance of the MySQL Database Server v5.5 and another as an instance of the MySQL Community Server v8.0. Similarly, for each MongoDB dataset size, we create a docker image of an instance of the MongoDB Community Server v3.4 with the dataset is loaded. The rest of the datasets, which correspond to raw data (CSV, XML and JSON), are loaded into the machine and are accessible to all the processors.

In the case of Morph-RDB, we use it together with the docker images containing the instances of the MySQL Community Server v5.5, according to the corresponding documentation. As for Morph-xR2RML, we use it together with the docker images containing the instances of MongoDB server version v3.4. For these experimental configuration and processors, we eval-

Dataset	Processor		Query																
	Cache	Name	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-SQL-1	Warm	Morph-RDB	5.85	2.07	E	1.82	W	1.86	1.97	E	26.02	1.80	E	1.81	2.06	W	1.89	E	2.11	E
	Cold	Ontario	18.02	E	TO	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
		Morph-RDB	7.14	2.65	E	2.42	W	2.36	2.43	E	28.65	2.38	E	2.41	2.69	W	2.58	E	2.68	E
		Ontop	8.37	5.04	5.18	E	W	E	W	E	16.56	E	E	E	5.06	W	5.10	W	5.00	E
GTFS-MongoDB-1	Warm	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	28.67	W	W	6.52	W
	Cold	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	28.17	W	W	6.96	W
GTFS-CSV-1	Cold	Morph-RDB	6.94	3.04	E	2.78	E	2.78	TO	E	TO	2.97	E	6.23	3.97	E	E	E	3.14	E
		Morph-CSV	15.11	10.88	E	10.72	E	9.95	10.84	E	40.90	10.70	E	11.60	11.82	E	E	E	11.48	W
		Ontario	W	E	17.34	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-XML-1	Cold	Ontario	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GTFS-JSON-1	Cold	Ontario	18.04	E	17.14	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-B-1	Cold	Ontario	W	E	17.14	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-W-1	Cold	Ontario	W	E	17.14	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E

Table 5.18: Overall execution time (in seconds) of benchmark queries in experiment configurations with original size datasets. W means that the engine obtained a different number of results in comparison to the baseline. E means that the processor is not able to execute the query. TO means that the processor is not able to evaluate the query within the timeout duration (3600 seconds).

uate all the 18 queries both in warm and in cold mode. Each query is run five times. In warm mode we want to analyze how the cache mechanism may affect the performance. In order to do so, we first evaluate the query, discard its result and then run the query again five times, we then compute the average query execution time. On the contrary, in cold mode, we want to study the performance of the processors without the effect of the cache. In order to do so, we run the query five times and we always restart the database server after each run, so as to clean all the caches.

Additionally, we use Ontario and Ontop with the docker images containing the instances of MySQL server v8.0, the latest version at the time of writing. Note that the use of cache is not supported anymore in MySQL v8.0 so that we only evaluate our queries in cold mode. We perform the rest of the experiment configurations with Ontario against the CSV and JSON datasets, and Morph-CSV against CSV datasets.

5.3.4.3 Results

In this section we report the results obtained through our experimental configurations. Table 5.18 presents the results obtained for all of the datasets and all the processors with scale 1 and a timeout of 3600s (1 hour). The rest of the Tables (5.19,5.20,5.21,5.22,5.23) report the results for the other scale values (5, 10, 50, 100 and 500) with the same timeout. When an engine reports an error (e.g. a SPARQL query parsing error, memory overhead, etc) we represent it with an `E` in the table. When the engine does not report any error but the number of results obtained differs with respect to the baseline (RDF materialised graph), we represent the cell

with a W. We do not report the total execution time of those queries because in general these cases report 0 results in the execution but without error, so the time is not relevant. The tables comparing the number of results obtained by the baseline and the evaluated engines is reported in Annex A.

In terms of the comparison among different data formats, we can observe that CSV and SQL data formats are the ones best supported by the available engines. In these cases, most of the engines are able to answer a significant number of queries. As to the effect of cache, as it is expected, evaluation in the warm mode needed less time, yet, the difference is insignificant due to the relatively small size of the datasets.

We can also see that in general, it takes more time to evaluate queries over CSV datasets than over SQL datasets. This is expected because available engines need to first load the CSV dataset in a SQL database server in order to be able to query the dataset.

This is not the case of other data formats such as JSON and MongoDB, where the engines are only able to answer one or two queries. This is even worse in the case of the XML format, where the only engine that supports it is not able to answer any query. Similarly in the distributed format, the only query that can be answered by the virtual KGC engine is a query that is evaluated against a JSON dataset.

This trend holds in the other scale factors up to 100. In the scale factor 500, only those engines that use SQL datasets are able to answer queries.

Analysing the results in general, the errors obtained in the execution of the queries (E in the tables) over the tested engines may be due to two main reasons: (i) the engine does not support a SPARQL operator in the original query (ii) the engine is not able to manage large (intermediate) results, for example, maintaining them in memory. Additionally, the differences obtained in terms of query completeness (W in the tables) may be due because: (i) the engine supports the SPARQL operator but it does not translate it correctly to an operator of the underlying database, hence, the query is executed but the number of results obtained are different; (ii) the interpretation of the mapping rules is not performing correctly, hence, the semantics of the original query is not preserved in the translated query.

5.3.4.4 Discussion

In this section we provide a general analysis of the design, implementation and execution of Madrid-GTFS-Bench. It should be pointed out that our aim is to have a proposal that follows the benchmark requirements and give a general overview of the results and problems we

Dataset	Processor		Query																	
	Cache	Name	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-SQL-5	Warm	Morph-RDB	12.65	2.47	E	1.89	2.06	1.78	1.93	E	E	1.74	E	1.88	2.14	4.58	2.88	E	2.61	E
	Cold	Ontario	117.00	E	TO	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
		Morph-RDB	15.14	3.24	E	2.40	2.71	2.34	2.62	E	E	2.41	E	2.70	2.82	5.59	3.89	E	3.39	E
		Ontop	13.87	5.40	5.31	E	W	E	W	E	W	E	E	E	5.24	6.61	W	W	5.37	E
GTFS-MongoDB-5	Warm	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
	Cold	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
GTFS-CSV-5	Cold	Morph-RDB	14.42	4.38	E	3.81	E	3.64	TO	E	TO	6.57	E	TO	12.45	E	E	E	9.25	E
		Morph-CSV	43.41	W	E	33.51	E	34.44	W	E	TO	33.86	E	36.08	34.90	E	E	E	35.26	E
		Ontario	W	E	18.34	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-XML-5	Cold	Ontario	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GTFS-JSON-5	Cold	Ontario	W	E	15.66	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-B-5	Cold	Ontario	W	E	15.66	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-W-5	Cold	Ontario	W	E	15.66	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E

Table 5.19: Overall execution time (in seconds) of benchmark queries in experiment configurations with size 5 datasets. W means that the engine obtained a different number of results in comparison to the baseline. E means that the processor is not able to execute the query. TO means that the processor is not able to evaluate the query within the timeout duration (3600 seconds).

Dataset	Processor		Query																	
	Cache	Name	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-SQL-10	Warm	Morph-RDB	23.78	2.88	E	1.93	W	1.75	1.97	E	E	1.85	E	1.94	2.46	6.61	3.46	E	3.07	E
	Cold	Ontario	415.60	E	TO	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
		Morph-RDB	27.25	3.72	E	2.54	W	2.36	2.55	E	E	2.38	E	2.50	3.22	8.16	4.48	E	3.77	E
		Ontop	24.05	5.56	5.57	E	W	E	W	E	W	E	E	E	5.29	7.58	W	W	5.62	E
GTFS-MongoDB-10	Warm	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
	Cold	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
GTFS-CSV-10	Cold	Morph-RDB	25.90	6.06	E	5.20	E	4.89	TO	E	TO	16.06	E	TO	38.15	E	E	E	38.90	E
		Morph-CSV	97.00	W	E	69.39	E	68.78	W	E	TO	69.28	E	71.01	68.79	E	E	E	72.29	W
		Ontario	W	E	19.51	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-XML-10	Cold	Ontario	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GTFS-JSON-10	Cold	Ontario	W	E	17.21	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-B-10	Cold	Ontario	W	E	17.21	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-W-10	Cold	Ontario	W	E	17.21	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E

Table 5.20: Overall execution time (in seconds) of benchmark queries in experiment configurations with size 10 datasets. W means that the engine obtained a different number of results in comparison to the baseline. E means that the processor is not able to execute the query. TO means that the processor is not able to evaluate the query within the timeout duration (3600 seconds).

Dataset	Processor		Query																	
	Cache	Name	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-SQL-50	Warm	Morph-RDB	108.42	4.91	E	2.08	W	1.75	1.97	E	E	1.89	E	2.29	3.69	22.55	8.27	E	5.56	E
	Cold	Ontario	TO	E	TO	E	E	E	E	W	E	E	E	E	E	W	E	E	W	E
		Morph-RDB	121.31	6.01	E	2.68	W	2.31	2.63	E	E	2.59	E	2.91	4.54	27.02	10.00	E	6.89	E
		Ontop	119.89	6.92	6.61	E	W	E	W	E	W	E	E	E	6.05	15.69	W	W	7.31	E
GTFS-MongoDB-50	Warm	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
	Cold	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
GTFS-CSV-50	Cold	Morph-RDB	128.40	22.17	E	19.85	E	19.60	TO	E	TO	351.23	E	TO	1,039.29	E	E	E	TO	E
		Morph-CSV	575.15	449.54	E	442.60	E	436.06	W	E	TO	444.84	E	443.12	447.74	E	E	E	443.47	W
		Ontario	W	E	35.16	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-XML-50	Cold	Ontario	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GTFS-JSON-50	Cold	Ontario	W	E	23.74	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-B-50	Cold	Ontario	W	E	23.74	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-W-50	Cold	Ontario	W	E	23.74	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E

Table 5.21: Overall execution time (in seconds) of benchmark queries in experiment configurations with size 50 datasets. W means that the engine obtained a different number of results in comparison to the baseline. E means that the processor is not able to execute the query. TO means that the processor is not able to evaluate the query within the timeout duration (3600 seconds).

Dataset	Processor		Query																	
	Cache	Name	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-SQL-100	Warm	Morph-RDB	221.11	7.48	E	2.30	W	1.75	1.96	E	E	1.99	E	2.65	4.68	42.44	15.51	E	8.54	E
	Cold	Ontario	TO	E	TO	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
		Morph-RDB	245.98	8.83	E	3.05	W	2.33	2.52	E	E	2.63	E	3.38	5.76	50.99	19.45	E	10.38	E
		Ontop	1,477.38	8.87	8.25	E	W	E	W	E	W	E	E	E	6.80	27.18	W	W	9.20	E
GTFS-MongoDB-100	Warm	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
	Cold	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
GTFS-CSV-100	Cold	Morph-RDB	E	43.59	E	38.52	E	38.43	TO	E	TO	1582.52	E	TO	TO	E	E	E	TO	E
		Morph-CSV	1,254.19	W	E	958.43	E	933.69	W	E	TO	957.95	E	951.53	952.93	E	E	E	947.82	W
		Ontario	W	E	85.59	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-XML-100	Cold	Ontario	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GTFS-JSON-100	Cold	Ontario	W	E	33.56	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-B-100	Cold	Ontario	W	E	33.56	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-W-100	Cold	Ontario	W	E	33.56	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E

Table 5.22: Overall execution time (in seconds) of benchmark queries in experiment configurations with size 100 datasets. W means that the engine obtained a different number of results in comparison to the baseline. E means that the processor is not able to execute the query. TO means that the processor is not able to evaluate the query within the timeout duration (3600 seconds).

Dataset	Processor		Query																	
	Cache	Name	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-SQL-500	Warm	Morph-RDB	TO	29.85	E	3.39	W	1.81	1.96	E	E	3.19	E	6.34	13.60	220.35	93.72	E	33.64	E
	Cold	Ontario	TO	E	TO	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
		Morph-RDB	TO	32.71	E	3.92	W	2.09	2.30	E	E	3.62	E	6.95	14.69	218.00	99.00	E	35.77	E
		Ontop	W	20.93	17.17	E	W	E	W	E	W	E	E	E	10.82	114.59	W	W	23.95	E
GTFS-MongoDB-500	Warm	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W	W	TO	W
	Cold	Morph-xR2RML	W	W	W	W	W	W	W	W	W	W	W	W	W	TO	W	W	TO	W
GTFS-CSV-500	Cold	Morph-RDB	E	TO	E	TO	E	TO	TO	E	TO	TO	E	TO	TO	E	E	E	TO	E
		Morph-CSV	TO	TO	E	TO	E	TO	TO	E	TO	TO	E	TO	TO	E	E	E	TO	TO
		Ontario	W	E	E	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-XML-500	Cold	Ontario	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
GTFS-JSON-500	Cold	Ontario	W	E	E	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-B-500	Cold	Ontario	W	E	E	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E
GTFS-W-500	Cold	Ontario	W	E	E	E	E	E	E	W	E	E	E	E	E	W	E	E	E	E

Table 5.23: Overall execution time (in seconds) of benchmark queries in experiment configurations with size 500 datasets. W means that the engine obtained a different number of results in comparison to the baseline. E means that the processor is not able to execute the query. TO means that the processor is not able to evaluate the query within the timeout duration (3600 seconds).

observed during the development of the GTFS-Madrid-Bench. We do not intend to rank the performance of the evaluated engines but rather to identify current limitations in the state of the art in terms of the capabilities of the engines, so as to provide useful information to the developers of each engine as well as to general practitioners.

We also describe the process of creation of a benchmark for virtual knowledge graph access and depict the problems and limitations of the tools employed for creating its resources. This analysis can be used to solve open issues, propose improvements and identify future work in the field.

In terms of the capabilities of KGC engines, the main issue we observe is that many of them do not support some of commonly used SPARQL operators such as UNION, ORDER BY and NOT EXISTS. The engines that cover a wider range of SPARQL operators are the ones that execute a SPARQL-to-SQL query translation, due to the fact that this technique has been widely studied in the state of the art (Calvanese *et al.*, 2017; Priyatna *et al.*, 2014). The engines that perform query translation over raw data (e.g. CSV, JSON) or over a NoSQL database (MongoDB in this case), produce a lot of errors in the query translation and evaluation processes. For example, in the case of Ontario, the engine is more focused on the generation of an efficient query plan (i.e., distributing star-shaped groups (SSG) taking into account the molecule templates) than performing a correct translation and execution of each SSG over the raw data. The engine does not give support to most of the SPARQL operators and that is the main reason why it is not able to answer most of the queries. The same happens in terms of query evaluation time, some SPARQL-to-SQL approaches include several optimization techniques (Xiao *et al.*, 2018b) so that they can evaluate the translated queries efficiently, while the other translation techniques that target non SQL query languages are not as efficient. These observations point out the need of a deeper analysis of the techniques that perform efficient query translation from SPARQL to non SQL query languages and raw data (CSV, JSON, XML).

Our main conclusions of the results obtained from the tested engines are:

- Only the SPARQL-to-SQL engines provide an acceptable support for SPARQL operators, although there are still some operators that are not included (e.g., FILTER NOT EXIST in Morph-RDB).

- Virtual KGC proposals beyond relational databases are not mature enough and more research is needed in order to, for example, provide wider support of SPARQL operators

or generate efficient query plans that take into account parameters such as data format or join selectivity.

- The problem of translating SPARQL queries for querying raw data (CSV, JSON, XML) should not be understood as a technical case of SPARQL-to-SQL where the management of the data is delegated to RDB wrappers such as Presto, Spark and Apache Drill. Techniques and optimizations, and the analysis of features uniquely associated to these data sources have to be proposed.

- The distribution of SPARQL queries over heterogeneous sources exploiting mapping rules, and their translation and execution over different query languages, are the two main points for developing robust virtual KGC engines. Although the adaptation of current techniques proposed by federated SPARQL engines to these engines has been successfully proved in (Endris *et al.*, 2019; Mami *et al.*, 2019a), they do not support the majority of the SPARQL operators and they do not correctly execute the queries when the data source is beyond RDB instances. New investigation should be performed to address these issues.

Additionally, in our evaluation we only have the possibility to obtain the total execution time of each engine. Other metrics are proposed in the benchmark, such as initial delay, loading time or query translation time. However, they are only available in some of the engines. We point out the importance of providing all these metrics to identify possible bottlenecks in the evaluation process.

We have also found possible improvements in terms of data and mapping generation in the process of creation of the resources in this benchmark. In the data generation process, one of the main improvements may be the incorporation of semantics. For example, in our benchmark we have a file that represents the calendar of the trips, which has a start and an end date. The data generator should validate that the start date must be earlier than the end date, so that queries can be created to exploit this constraint. Another example that would improve with the inclusion of semantics is the scaling of dataset sources that are related and may be "joined". Currently, even if each dataset is scaled, the number of tuples per join attribute value does not change. Ideally, this should be scaled only in certain cases. Additionally, the inclusion of a set of constraints or validation rules may improve this process (e.g. define a range of possible values for a column).

With respect to the mapping generation process, we find two main issues. First, as mappings need to relate the ontology with the data source, the raw data needs some changes in a pre-processing step in order to be aligned with the features of the ontology (e.g classes or properties). There are some proposals to include these transformation functions in mappings such as the Function Ontology (De Meester *et al.*, 2016) or R2RML-F (Debruyne & O'Sullivan, 2016) but at the moment of writing only Squerall and Morph-CSV are able to parse RML mappings with functions. Finally, we have to create manually all of the mapping documents required to test the engines. Following the proposal in (Corcho *et al.*, 2019), an improvement will be to be able to define the mappings conceptually, independently of the language, and then, to have techniques to translate them to a specific language. With this approach we will ensure the correctness of all the mapping rules.

5.3.5 Conclusions

In this section we propose a benchmark for virtual knowledge graph construction using real data from the transport domain. The benchmark design considers variables that span all of its resources (queries, mappings and data) in order to test the capabilities and performance of the processors. GTFS-Madrid-Bench satisfies requirements that are an extension of those already identified in existing OBDA benchmarks. Besides, metrics have been established for each step of the workflow of virtualized knowledge graph access.

As already discussed, the main objective of this benchmark is not to provide a ranking of engines, but to provide a set of resources that can be useful for: (i) practitioners who choose the engine that best fits their use cases and (ii) developers of virtual knowledge graph access engines to improve their tools and compare their results with other proposals. As such, we expect this benchmark to be a stepping stone in this area where much research and development has been done for decades, but there is a need for more mature applications to be used in real-world environments. Indeed, our experimental study has shown that there are still relevant open issues, such as SPARQL conformance, semantic preservation in the translation from SPARQL queries to the query languages used to query raw data (CSV, JSON, XML), and the application of query evaluation optimization techniques. With the GTFS-Madrid-Bench we intend to contribute to the community by providing not only the baseline that can be used to improve the development of the current engines, but also the possibility to use it to test new approaches and techniques over the next years.

The design of this benchmark has been a complex task, since it had to cover all of the identified requirements and, at the same time, work on a very general scenario with a mix of KGC approaches. On the one hand, some of the current OBDA proposals work with SQL datasets and in general conform to most of the features of the SPARQL language. Throughout the experiments we realized that the OBDA proposals that are designed to work with other formats support fewer features of the query language, and in general they have issues in their query translation process. There is a lot of room for improvement in these proposals, such as generating more efficient queries, which has been done in the SQL-based OBDA proposals. On the other hand, we were only able to include in the benchmark the Ontario KGC proposal, even though other KGC proposals have been published in the literature since it was not feasible to execute them because of lack of documentation. In all cases, the evaluation of our benchmark queries with the different engines exposed the need for improvements in their current releases, in terms of efficiency and correctness of the results.

It is also worth mentioning that the benchmark is easily extensible to be used with other data formats and engines. That is, if in the future there is a requirement to evaluate an engine that supports a different data format, the only need is to create a script that translates the data sources from CSV to the corresponding format.

Future work includes the development of mapping translation techniques that involve the different mapping language specifications. In the context of virtual knowledge graph construction it would be very useful to help in the development and maintenance of mappings, so as to avoid inconsistencies among mappings and errors in the evaluation. Another line of work is to improve the data generation process to ensure that scaled data is well aligned with the domain data model.

Chapter 6

Exploiting Declarative Annotations for Virtual Knowledge Graph Construction

> Computer scientists do not make research, they solve real problems
>
> *Isabel Fraga*

In this Chapter, we introduce our contributions that exploit declarative annotations over data on the web for enhancing the construction of virtual knowledge graphs. The two frameworks presented take the advantage of the ideas about the *mapping translation* concept defined in Chapter 4 for improving the current proposals.

Section 6.1 presents Morph-CSV, a constraint-based approach for ensuring the effectiveness of SPARQL-to-SQL when the input tabular data is not a relational database instance. It uses RML+FnO mapping rules and CSVW metadata descriptions to explicitly declare implicit constraints over the input sources. Section 6.2 describes Morph-GraphQL, a framework that adapts SPARQL-to-SQL algorithm presented in (Chebotko *et al.*, 2009) to automating the generation of programmer data wrappers from declarative mapping rules. More in detail, it is focused on generating GraphQL resolvers for virtual access to relational databases using R2RML mappings as inputs.

6.1 Virtual Knowledge Graph Construction over Tabular Data

Albeit extensively utilized, tabular representations imposed various data management challenges to advanced users (e.g., developers, data scientists). The lack of a unified way to query tabular data, something available in other formats (e.g., RDB, JSON, XML), hinders the integration of sources, especially those having datatype inconsistencies. Moreover, data may not be normalized, and information about relationships or column names are not always descriptive or homogeneous. Hence, data consumers are usually forced to apply ad-hoc or manual data wrangling processes to consume data via open data portals.

Traditional virtual KGC approaches, usually, rely on loading tabular data into SQL-based systems[91,92](e.g., MySQL, Apache Drill, Spark SQL, Presto) to perform query translation techniques. However, the correctness and optimization of these techniques are supported by the main assumption about the existence of constraints over the source data (i.e., a good physical design of the relational database instance). Their absence during a virtual KGC process over tabular data directly impacts completeness and performance of these techniques. Completeness is affected because of heterogeneity issues in data sources (e.g., datatype CSV columns are simply treated as string-type SQL columns). Furthermore, performance is impacted because indexes are not created based on basic relational constraints, i.e., primary and foreign key constraints are not defined in the schema. Consequently, query translation optimization techniques that commonly exploit indexes (e.g., (Priyatna *et al.*, 2014; Rodriguez-Muro & Rezk, 2015)) may not produce the expected results whenever the constraints have not been applied, or the indexes have not been created.

KGC annotations such as the W3C recommendation to annotate tabular data, CSVW (Tennison *et al.*, 2015) and some extensions of standard mapping rules (e.g., RML+FnO (De Meester *et al.*, 2017)) are commonly used to describe constraints over a tabular dataset. For example, we can standardize a column indicating its format, define integrity constraints, or declare data types. The majority of virtual KGC engines (Endris *et al.*, 2019; Priyatna *et al.*, 2014) do not include this information. Those engines that have partially included the constraints (e.g., Squerall (Mami *et al.*, 2019b) parses RML+FnO mapping rules) are not fully documented; i.e., there is no explanation of how these constraints are taken into account. The definition of

[91] `https://github.com/oeg-upm/morph-rdb/wiki/Usage#csv-files`
[92] `https://ontop-vkg.org/tutorial/mapping/primary-keys.html`

a workflow that includes the exploitation of these tabular annotations during a virtual KGC process will ensure correct and optimized SPARQL-to-SQL translations.

Problem Statement: We address the limitations of current SPARQL-to-SQL query translation techniques over tabular data, which enforce and demand lots of unreproducible and hard manual work for the application of constraints to ensure efficient query processing and query completeness[93,94,95]. Our goals are to (i) define a framework that includes the application of a set of constraints over tabular data, and (ii) define a set of efficient operators that apply each type of constraint to improve query completeness and performance (e.g., removal of duplicates, normalization of input sources or application of transformation functions).

Proposed Solution: We propose a set of new steps to be aligned with the current KGC workflow. Further, we implement Morph-CSV, and evaluate its behavior embedded on top of two well known open source SPARQL-to-SQL engines, in comparison with previous approaches.

Contributions: Our main contributions are as follows:

1. Definition of the concept of Virtual Tabular Dataset (VTD) composed by a tabular dataset and its corresponding annotations, as well as its alignment with the current definition and assumptions of the OBDA framework (Xiao *et al.*, 2018a).

2. Morph-CSV, a framework that implements a constraint-based KGC workflow for tabular datasets; it receives a VTD and a SPARQL query as inputs and outputs an OBDA instance. Morph-CSV performs the following steps: (i) generation of the constraints based on information on the VTD; (ii) selection of sources and attributes needed to answer the query; (iii) pre-processing of the selected sources applying some of the constraints; and (iv) physical implementation of the corresponding RDB instance and associated schema, ensuring effectiveness of the SPARQL-to-SQL translations and optimizations. Morph-CSV is engine agnostic, i.e., it can be embedded on top of any SPARQL-to-SQL engine.

3. Evaluation of Morph-CSV re-using in the backend two well-known open source SPARQL-to-SQL engines: Morph-RDB (Priyatna *et al.*, 2014) and Ontop (Calvanese *et al.*, 2017); two benchmarks (BSBM (Bizer & Schultz, 2009) and GTFS-Madrid-Bench (Chaves-Fraga *et al.*, 2020a)), and a real-world testbed from the Bio2RDF project (Belleau *et al.*, 2008) are used in the study.

[93] `https://github.com/oeg-upm/morph-rdb/wiki/Usage#csv-files`
[94] `https://ontop-vkg.org/tutorial/mapping/primary-keys.html`
[95] `https://ontop-vkg.org/tutorial/mapping/foreign-keys.html`

6.1.1 Motivating Example

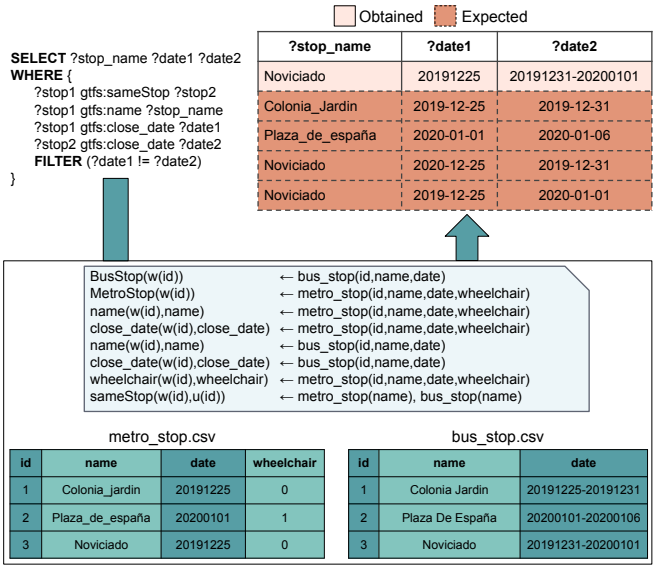


Figure 6.1: Motivating Example. SPARQL query evaluation over two tabular data files in the transport domain through a common virtual KGC approach. It loads the files as single tables in an SQL-based system and uses the mapping rules for query translation. The number of results differs with respect to the expected results due the heterogeneity of the raw data. Additionally, query performance may be affected by the join condition between the two tables, the absence of indexes and the loading of columns that are not needed to answer the input query (wheelchair).

Since May 2017, the publication of a new directive by the EU Commission on discoverability and access to public transport data across Europe[96] has motivated the development of solutions for multi-modal travel information services. This document states that transport data should be available through national access points (NAP), e.g., databases, data warehouses, and repositories. Consider the *de-facto standard* for publishing open data in the transport domain, GTFS[97]. This model enables the representation of transport-related concepts such as *schedules*, *stops,* and *routes*, using 15 different inter-related CSV files called GTFS feed. Following best modeling practices recommended in this model specification, each feed comprises entities of one type of transportation mode (e.g., metro, train, and tram). Linking these feeds based on their stops enables route planners to offer multi-modal routes, a route that can be created using various transportation types. Albeit straightforward and simple to use, GTFS feeds do not allow

[96] `https://ec.europa.eu/transport/themes/its/road/action_plan/nap`
[97] `https://developers.google.com/transit/gtfs/reference/`

for the definition of integrity constraints such as primary or foreign keys. As a consequence, data integrity cannot always be guaranteed.

Consider the GTFS feeds from the metro and buses of Madrid's city; they have several stops and stations in common. Different transport authorities create them, and the names of their stops are defined in various manners. Although these types of entities can be represented, the unique identification and relationships among them cannot be explicitly expressed. Figure 6.1 depicts a portion of these two GTFS feeds. As it is usual in open datasets, stop names do not follow a standard structure (e.g., "Colonia Jardin" in *bus_stops.csv* and "Colonia_jardin" in *metro_stops.csv*). A similar issue is present in closing dates, where there are multi-valued cells, and their format is not the standard one (e.g., yyyy-MM-dd). Suppose a user is interested in collecting information about bus and metro stops with the same name and information related to their closing dates during holidays; Figure 6.1 presents the SPARQL query describing this request. Following the approach commonly employed by typical KGC engines, the two files would be loaded into an SQL-system and treated as single tables. The obtained result set only contains one answer where the stop names in the two data sources are identical ("Noviciado"). However, the expected result set should include more answers by joining among the bus and metro's stop names through the normalization of multi-valued date columns.

Query's performance may also be affected whenever a join condition is executed between the stop names of both files. Furthermore, the absence of possible indexes in these joining columns makes ineffective the typical optimizations applied in a SPARQL-to-SQL process. Nonetheless, to effectively exploit the indexes to scale-up the execution of the translated queries, the satisfaction of the unique and foreign integrity constraints should be ensured. The manual and ad-hoc definition of the relational schema representing these tables and the corresponding integrity constraints will overcome this problem. Nevertheless, this task is time-consuming, and reproducibility is not ensured. In this section, we propose Morph-CSV, a constraint-based KGC framework capable of exploiting standard tabular data annotations (e.g., RML or CSVW) to generate the required constraints ensure the integrity of the tabular schema in terms of unique identifiers and foreign keys. Moreover, Morph-CSV applies metadata annotation from CSVW to generate domain-specific constraints. As a result, Morph-CSV enhances query completeness and performance of SPARQL-to-SQL techniques, in compliance with OBDA assumptions.

6.1.2 Virtual KGC over Tabular Data

This section describes a set of challenges demanded be addressed whenever tabular data is queried in a virtual KGC framework. Further, we describe relevant proposals for annotating tabular datasets and their alignment with the identified challenges.

6.1.2.1 Querying challenges under virtual KGC for tabular data

There are specific challenges on querying tabular datasets using an virtual KGC approach that have not been tackled by existing techniques. We will describe those challenges and explain how they may have a negative effect in terms of completeness and performance of query-translation approaches:

- **Updated results:** Existing frameworks load all of the tabular input files that are specified as sources in the mapping rules into a SQL database before executing the query-translation process. This step has to be repeated whenever a SPARQL query is evaluated to ensure up-to-date results, resulting in unnecessary longer loading time, affecting, thus, the performance.

- **Normalization:** Tabular data formats do not provide restrictions on how to structure data. As a result, cells may contain multiple values, and one file may represent multiple entities. Having non-normalized tables may affect the completeness of the query. When a tabular source with multiple-valued cells is loaded into an RDB table, the cell's value is interpreted by the RDBMS as an atomic value, reducing, thus, completeness for queries that filter or "join" on the corresponding column. Representing several entities in a single file may lead to duplicate answers, and in turn, decrease query answering performance.

- **Heterogeneity:** Tabular data normally contain values that need to be transformed before query evaluation (e.g., column default values or normalization of date formats). Since there may be different formats for the same datatype or default values that may have not been included in the dataset, query completeness can be affected.

- **Lightweight Schema:** Most of the tabular data only provide minimal information about their underlying schema in the form of column names in the header, if at all present. Also, although there is implicit information on keys and relationships among sources, there is no way to specify primary key or foreign key constraints. The same can be said on

General Challenge	Detailed Challenges	Relevant Properties
Updated results	Select relevant sources and columns	SPARQL + RML+FnO
Lightweight Schema	Describe the corresponding concept	rr:class
	Describe the corresponding property	rr:predicateMap
	Specify NOT NULL constraint	csvw:required
	Column datatype	csvw:datatype
Heterogeneity	Domain values	csvw:minimum, csvw:maximum
	Specify the format of a column	csvw:format
	Transform value	fnml:functionValue
	Default for missing values	csvw:default
	Specify NULL values	csvw:null
	Add header to a CSV file	csvw:rowTitles
Normalization	Primary Key	csvw:primaryKey
	Foreign Key	csvw:foreignKey
	Relationships between columns	rr:parentTriplesMap + rr:joinCondition
	Mutiple entities in one source	rr:TriplesMap + rml:logicalSource
	Support for multiple values in one cell	csvw:separator

Table 6.1: Properties of CSVW and RML+FnO that can be used to address the challenges of dealing with tabular data in a virtual KGC approach

indexes and datatypes. The existence of this type of information is assumed (Xiao *et al.*, 2018a) in an virtual KGC approach for performing optimizations in query evaluation techniques. Therefore, the lack of this information affects the performance of these engines.

Although some of the aforementioned challenges are not only specific to tabular datasets and are proposed in several data integration approaches (Doan *et al.*, 2012; Golshan *et al.*, 2017; Halevy *et al.*, 2006) there are two main reasons why it is important to address these problems in this context: first, the number of tabular datasets available in the web of data is enormous and still growing and these challenges were not taken into account in previous virtual KGC proposals; second, although there are declarative proposals to handle these issues in the state of the art like CSV on the Web (Tennison *et al.*, 2015) for metadata annotations, or mapping languages that include transformation functions to deal with heterogeneity (e.g., RML+FnO (De Meester *et al.*, 2017) or R2RML-F (Debruyne & O'Sullivan, 2016)), there is not yet a proposal that exploits the information from these inputs including their application in the form of constraints into a common virtual KGC workflow.

6.1.2.2 KGC annotations for tabular data

In Table 6.1, we summarize the relevant properties from RML+FnO and CSVW that can be used to address the challenges identified in the previous section. Additionally, we provide a detailed description of these properties:

- **Metadata.** The property `csvw:rowTitles` can be used to specify column names in case the first row is not used to specify them.

- **Transformation functions.** String concatenation functions are supported by both CSVW (`csvw:aboutUrl`, `csvw:valueUrl`) and the RML property (`rr:template`). In addition, more complex functions can be declaratively specified using RML+FnO, specifically, with the `fnml:functionValue` property. Finally, two special cases of transformation functions in the context of OBDA are related to how default values and NULL representations have to be generated in the RDB instance. These two cases can be handled by CSVW properties: `csvw:defaultValue` and `cvwv:null`.

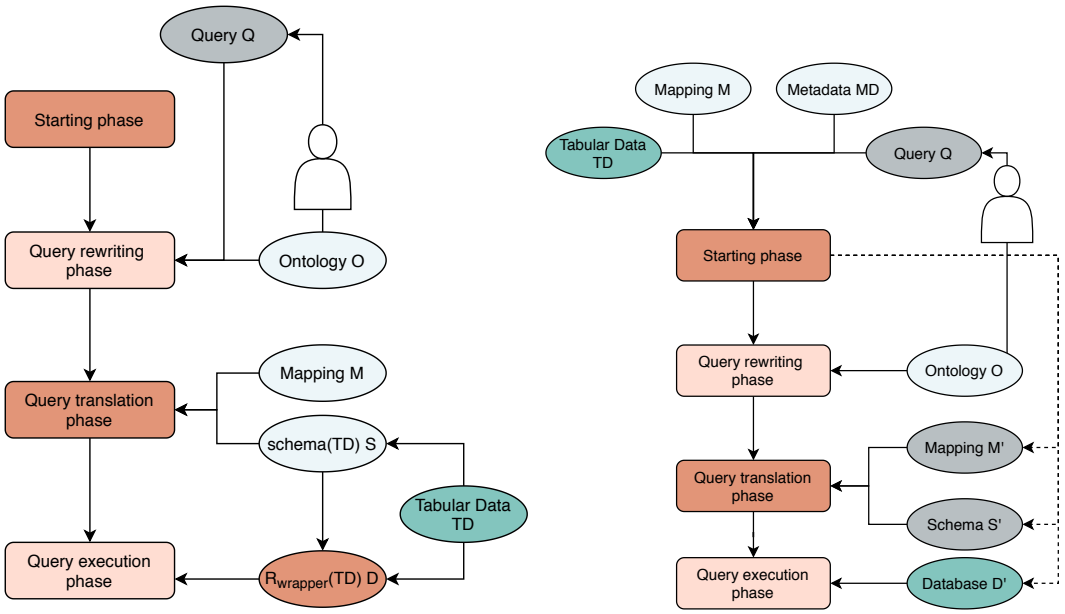
- **Domain Constraints.** CSVW allows for the specification of the datatype (`csvw:datatype` property) and format (`csvw:format` property) of tabular columns. CSVW also provides a couple of properties (e.g., `csvw:mininum` or `csvw:maximum`) to specify the range of numerical columns and a property `csvw:required` to specify the NOT NULL constraint over the column of a table.

- **Integrity Constraints.** In CSVW the property `csvw:primaryKey` can be used to declare explicitly the primary key of a table. As for the foreign key, the use of RML's properties `rr:parentTriplesMap` together with the property `rr:joinCondition` can be seen as an indication that the parent column used over this rule could be a foreign key, or at least that a relation exists. CSVW provides an explicit way to declare whether a column is a foreign key, using the `csvw:foreignKeys` property.

- **Normalization.** The property `csvw:separator` from CSVW indicates the character used to separate multiple values in the cells of a CSV column, which is relevant when a CSV file is in 1NF. Multiple RML TriplesMap using the same data source can be used as an indication that the source contains multiple concepts (2NF).

6.1.3 The Morph-CSV Framework

The formal framework presented in (Xiao *et al.*, 2018a) defines an OBDA[98] specification as a tuple $P = \langle O, S, M \rangle$ where O is an ontology, S is the source schema, and M a set of mappings. Additionally, an OBDA instance is defined as a tuple $PI = \langle P, D \rangle$ where P is an OBDA specification and D is a data instance conforming to S. In a virtual OBDA framework, queries are posed over a conceptual layer and then translated to queries over the data layer using information in the mappings. There is a set of assumptions over the framework that support the possibility of doing query translation and ensuring semantic preservation in the process, together with the application of optimization techniques proposed in the state of the art. To motivate our proposal, we have to establish what are the main assumptions made in previous proposals and their impact when data is represented in tabular form.



(a) Baseline approach

(b) Enhanced virtual OBDA workflow.

Figure 6.2: Virtual OBDA approaches for tabular data. The baseline approach creates the schema and relational database instance extracting file and columns names from the tabular dataset. The proposed workflow exploits the information from the mapping rules and metadata to extracted a set of constraints and applying them over the tabular data to generate the schema and the relational database instance.

[98] In this section we use OBDA as synonym of virtual KGC

6.1.3.1 OBDA assumptions

Analyzing the definition of OBDA in (Xiao *et al.*, 2018a) and its extension for NoSQL databases defined in (Botoeva *et al.*, 2019) we identified a set of assumptions made over the framework and their impact when the dataset is tabular:

- There is a native query language QL for D. For a tabular dataset, there is no native query language for querying this format, which generates an important difference with other common formats for exposing raw data on the web such as JSON and XML as they include methods to query them (JSONPath, XPath). This is the main issue that needs to be solved in order to query tabular datasets in a virtual OBDA context and has a direct impact on the rest of the assumptions.

- S typically includes a set of domain and integrity constraints. In the case of querying a tabular dataset $D_{tabular}$, S is defined using column names extracted from $D_{tabular}$ and it does not include any constraint types (neither domain nor integrity constraints). This has a negative impact not only in terms of query execution time but also over query result completeness as there will be queries that cannot be executed due to the lack of explicit domain constraints.

- D is an RDB instance or a NoSQL database instance, that includes an RDB wrapper able to provide a relational view over S and D. In the context of a tabular dataset $D_{tabular}$, D=$R_{wrapper}(D_{tabular})$ where $R_{wrapper}$ is a relational database wrapper that satisfies S.

6.1.3.2 From a virtual tabular dataset to an OBDA instance

Based on the previous OBDA assumptions, we define the concepts and functions to address the problem of querying a tabular dataset in OBDA.

Definition 6.1. A virtual tabular dataset is defined as a tuple VTD=$\langle D_{tabular}, O, M, MD\rangle$ where $D_{tabular}$ is a tabular dataset that is composed of a set of data sources, defined as $\mathcal{D}_{tabular}$ = $\{s_1,\ldots,s_n\}$ and where each s_i is a tabular relation defined over the domains of the attributes $Att(s_i) = \{A_{i1},\ldots,A_{im}\}$[99], where m is the number of attributes of s_i. O is an ontology, and M is a set of global as view mappings between O and $schema(D_{tabular})$[100]. MD is a set of metadata

[99] A relation is defined as the subset of the Cartesian product of the domains of the attributes.

[100] The set of the attributes of each tabular relation in $D_{tabular}$, i.e., $schema(D_{tabular}) = \{Att(s_i),\ldots,Att(s_n)\}$

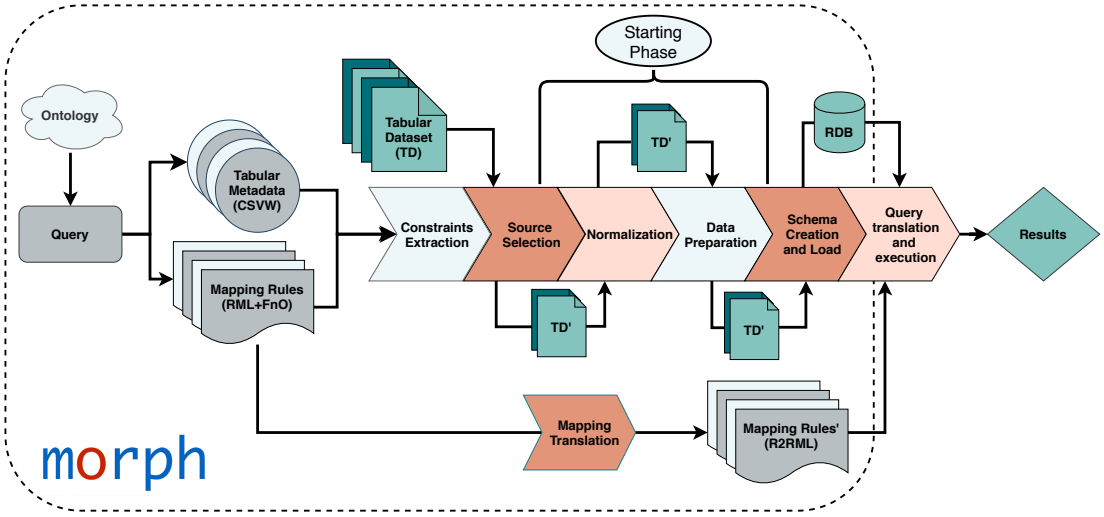


Figure 6.3: The Morph-CSV Framework. Morph-CSV extends the starting phase of a typical OBDA system including a set of steps for dealing with the identified tabular data querying challenges. The framework first, extracts the constraints from mappings and tabular metadata and then, implements them in a set of operators that are run before executing the query translation and query execution phases, which can be delegated to any SPARQL-to-SQL engine. The mapping rules are translated accordingly to the modified tabular dataset to allow its access by the underlying OBDA engine.

tabular (domain) annotations, where for each s_i there exists a set $\{(A_{i1}, Type(A_{i1})), \ldots, (A_{im}, Type(A_{im}))\}$ in MD.

Example 1. The virtual tabular dataset of the GTFS of Madrid's metro system can be defined as $VGTFS^{metro}_{madrid}$ where the dataset is composed by of 10 different tabular sources in CSV format $GTFS_{tabular}$, $LinkedGTFS$[101] is the ontology, the mappings $RML+FnO_{GTFS}$, following the RML+FnO (De Meester *et al.*, 2017) specification, define the relation between the input sources and the ontology and, finally, the metadata $CSVW_{GTFS}$ is defined according to the W3C recommendation, CSVW (Tennison *et al.*, 2015), specifying a set of constraints extracted from the GTFS reference data model[102].

Given a VTD, we define the function $\theta(VTD) = PI$ where PI is an OBDA instance $PI = \langle P, D \rangle$ where D=$R_{wrapper}(D_{tabular})$ and $P = \langle O, S, M \rangle$ is an OBDA definition where S does not contain any type of constraint. We extend the function $\theta(VTD)$ with the aim of enhancing the virtual OBDA baseline approach over tabular data. We define $\theta^{++}(VTD)$=PI as a function

[101] `https://lov.linkeddata.es/dataset/lov/vocabs/gtfs`
[102] `https://developers.google.com/transit/gtfs/reference`

Step	Constraint/Improvement	Rule/Annotation	Function	Challenge
Extraction	Reduce search space	SSG from Query	select_annotations	Selection
		Mapping Rules	select_sources	
Data Normalization	2NF	csvw:separator	split	Normalization
	3NF	TriplesMap with same source	cut	
Data Preparation	Standarization	csvw:null, csvw:default csvw:format, etc.	sub	Heterogeneity
		fnml:functionValue	create	
	Duplicates	-	duplicates	
Schema Creation and Load	Primary Key	csvw:primaryKey	primaryKey	Lightweight Schema
	Foreign Key	csvw:foreignKey	foreignKey	
	DataType	csvw:datatype	datatype	
	Index	selectivity on mapping join conditions	index	

Table 6.2: Summary of constraints, corresponding functions and OBDA annotations applied by Morph-CSV

that extracts a set of constraints from M and MD and then applies them over $D_{tabular}$ to obtain PI. More in detail, the function can be expressed as $\theta^{++}(VTD)$=$\gamma(D_{tabular}, O, M, \psi(M, MD))$ where the function $\psi(M, MD) = C$ extracts a set of constraints from OBDA annotations for tabular data. Then, $\gamma(D_{tabular}, O, M, C)$ applies the constraints C over $D_{tabular}$ to create a relational database schema $S^{'}$ and its corresponding instance $D^{'}$. In summary, the final output is an OBDA instance $PI^{'} = \langle P^{'}, D^{'} \rangle$, where $D^{'}$ is a relational database instance that is compliant with the main assumptions of the OBDA framework and $P^{'} = \langle O, S^{'}, M^{'} \rangle$ where $S^{'}$ contains a set of domain and integrity constraints and $M^{'}$ are the mapping rules that define the relations between O and $S^{'}$. Following the proposed workflow in Figure 6.2b, the user first defines the query based on the concepts defined in the ontology, and then, during the starting phase, the $\theta^{++}(VTD)$ is performed. During the execution of the function, first, the constraints from mappings and annotations ($\psi(M, MD)$) are extracted, and then the OBDA instance $PI^{'}$ is generated where the constraints are applied to efficiently create the schema $S^{'}$ and the relational database instance $D^{'}$. Mapping rules are also translated, from M to M' to be aligned with the new created schema. *Example 2.* The process of applying the function $\theta^{++}(VGTFS^{metro}_{madrid})$ generates the OBDA instance $PIGTFS^{metro}_{madrid}$. The features of this output are a relational database schema $GTFS_{schema}$, a relational database instance $GTFS_{SQL}$ compliant with the defined schema, and a set of mapping rules following the R2RML W3C recommendation, $R2RML_{GTFS}$, that represent the relations between $GTFS_{schema}$ and the $LinkedGTFS$ ontology.

Constraints are conjunctive rules specified for tabular data that restrict the valid data in one or more tables. C is a set of constraints, where each constraint c is a logical statement that expresses the condition that needs to be satisfied by the data in order to be valid. Each constraint is applied through a function.

Example 3. CSVW allows expressing a primary key constraint for a table. The function $\psi(M, MD) = C$ generates the corresponding constraints in the form of a function $primaryKey(t, a)$ that applies this constraint to a source t and a set of columns a, and generates a primary key in the output schema.

Given an OBDA instance PI=$\langle \mathcal{P}, \mathcal{D} \rangle$, we define the function $eval(Q, PI)$, that retrieves a SPARQL answer set that is the result of the translation of Q from SPARQL to SQL using the mapping rules M defined in P, and then evaluating the query directly over D.

6.1.3.3 Problem statement and solution

Based on the preliminaries and assumptions on the OBDA framework, we now define the problem that we address in this paper and Morph-CSV, our proposed solution.

Problem statement: Given a VTD, the problem of OBDA query translation over tabular data is defined as the problem of explicitly enforcing implicit constraints C extracted from mapping rules M and metadata MD on a tabular dataset $D_{tabular}$, such that:

- The number of results obtained in the evaluation of the SPARQL query Q over the function $eval(Q, \theta^{++}(VTD))$ is equal or greater than the number of results in the evaluation of the same query Q over the function $eval(Q, \theta(VTD))$, i.e.,
  $\#answers(eval(Q, \theta^{++}(VTD))) \geq \#answers(eval(Q, \theta(VTD)))$.
- The total execution time of evaluating a SPARQL query Q over $eval(Q, \theta^{++}(VTD))$ is less than or equal than the total execution time of the same SPARQL query Q over the function $eval(Q, \theta(VTD))$, i.e.,
  $time(eval(Q, \theta^{++}(VTD))) \leq time(eval(Q, \theta(VTD)))$.

Proposed solution: We propose Morph-CSV, an alternative to the traditional OBDA workflow for query translation when the input is a tabular dataset (see Figure 6.2b). Morph-CSV relies on the function $eval(Q, \theta^{++}(VTD, \psi(M, MD)))$, to apply the tabular dataset constrains. Thus, Morph-CSV extends a typical OBDA workflow by including a set of steps for a maintainable extraction and efficient application of constraints. The workflow proposal is as follows:

- **Constraint Extraction**: the evaluation of the function $\psi(M, MD)$ produces as output the set of constraints C; it exploits the information defined in the annotations of M and MD, i.e., the set of metadata tabular annotations and mapping rules, respectively. At implementation level they are expressed as CSVW specifications and RML+FnO mapping rules.

- **Source Selection**: in this step the sources required to evaluate the SPARQL query Q are selected. The required data sources correspond to the set of sources in the result of unfolding (Poggi *et al.*, 2008) Q according to the mapping rules in M.

- **Normalization**: metadata and mapping rules are used to extract functional dependencies between the attributes of the data sources. The algorithm by Beeri et al. (Beeri *et al.*, 1978) is followed to transform tabular data sources into tabular relations that meet third normal form (3NF).

- **Data Preparation**: application of the transformation functions based on the extracted domain constraints and on a set of optimization techniques that adapt the ideas proposed in (Iglesias *et al.*, 2020; Jozashoori & Vidal, 2019; Jozashoori *et al.*, 2020) to a virtual OBDA environment.

- **Schema Creation and Load**: creation of the schema and loading the data into the database instance applying a set of rules for index creation.

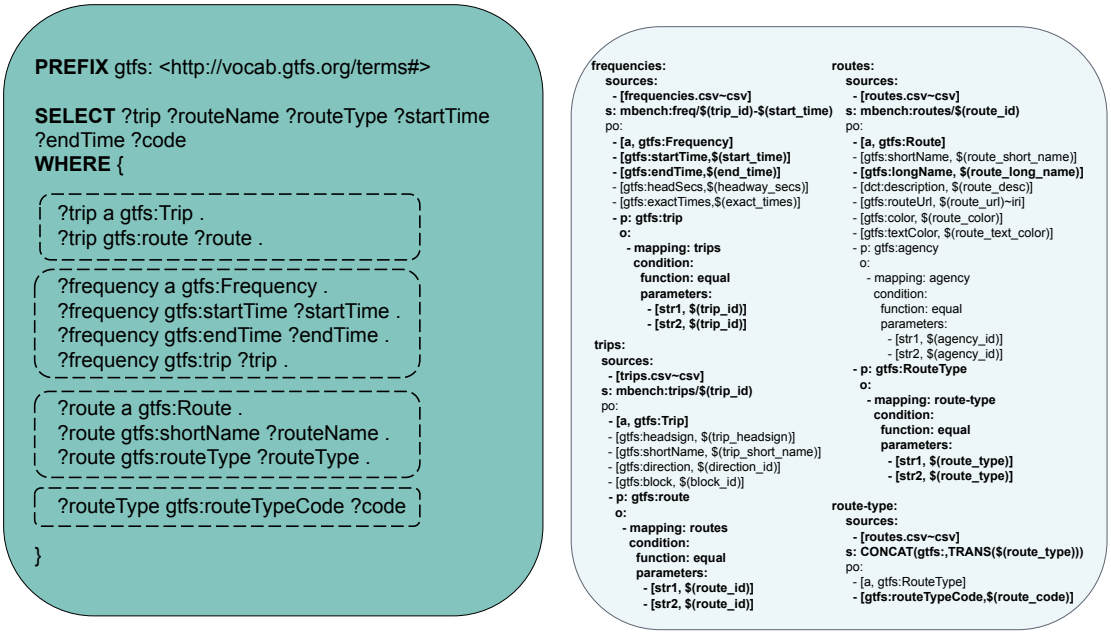
- **Query Translation and Execution**:the evaluation of the query Q is delegated to any OBDA SPARQL-to-SQL engine.

We show the workflow of Morph-CSV in Figure 6.3 with the inputs and outputs of each step.

6.1.3.4 Steps performed in the Morph-CSV framework

We describe in detail the steps proposed in Morph-CSV together with an example extracted from the benchmark for virtual knowledge graph access, Madrid-GTFS-Bench, using the query shown in Figure 6.4a, the GTFS feed from the Madrid metro as source data, and the corresponding RML+FnO mapping rules and CSVW annotations[103].

[103]Resources at: `https://github.com/oeg-upm/gtfs-bench`



(a) Input SPARQL query.

(b) Mapping rules selection.

Figure 6.4: Selection of Mapping Rules. Based on the SPARQL query relevant rules are selected (in bold), the rest are discarded.

Constraint Extraction

The first step performed by Morph-CSV is the extraction of the constraints that are applied to improve query execution and completeness. Morph-CSV benefits from having declarative and standard approaches to generalize this step: CSVW (Tennison *et al.*, 2015) for the metadata; and RML+FnO (De Meester *et al.*, 2017) for mapping rules and specific transformation functions. Thus, maintainability, understandability and readability of this process are improved in comparison with ad-hoc pre-processing approaches.

Most of the constraints such as PK-FK relations, datatypes or NULL values are explicitly declared in the metadata of the sources. However, there are a set of implicit constraints such as the conditions for the normalization of sources and the creation of indexes, that require complex rules to extract them and that are explained in detail in the corresponding steps. The summary of the constraints, associated functions, and properties used from OBDA annotations to extract them, are shown in Table 6.2.

Source selection

The second step is to select the relevant sources to answer the input query. The baseline approach delegates this step to the RDBMS: it loads all the sources of the dataset in the RDB instance because it does not have information about which sources are going to be queried. This has a negative impact in the total execution time of a query. Taking the input mapping rules, Morph-CSV performs query unfolding, and pushes down source selection by executing the function $select(Q,M)$, divided into two main steps. First, Morph-CSV performs an operation to select only the relevant annotations for answering the input query, $select\_annotations(Q,M)$. It first creates the set of star shaped groups $SSG_1 \ldots SSG_n$ of the query (Vidal *et al.*, 2010) (triple patterns with the same subject)[104]. Then, for each SSG_i and `rr:TriplesMap` TM_j defined in M, the engine selects the TM_j where the predicates in SSG_i are contained in the set of `rr:PredicateObjectMap` (POMs) defined in TM_j. Finally, for each selected `rr:TriplesMap` TM_j, Morph-CSV only selects the POMs according to the predicates defined in the SSG_i, hence, removing from each TM_j irrelevant rules for the input query. Using these mapping rules $M^{'}$, only relevant metadata annotations are also selected, $MD^{'}$. The obtained mapping rules in this step, $M^{'}$ and annotations $MD^{'}$, substitute the original ones in VTD. An example of this step is shown in Figure 6.4, where the input query asks for trips, their route type, routes names and corresponding time frequencies. Morph-CSV first creates the SSGs, 3 in this case, and using the predicates of each SSG, the `rr:TriplesMap` are selected from the general GTFS mapping document, discarding the rest of the rules. Then, it only selects the necessary POMs for evaluating the query such as `gtfs:startTime`, `gtfs:shortName` and `gtfs:routeType` (Figure 6.4b).

Second, Morph-CSV runs $select\_sources(M)$, where it projects, from the input $D_{tabular}$, the sources and columns that are referenced in M, hence, relevant sources for the input query. The output of this function generates a set of new tabular sources $s_i \ldots s_n$ that substitute the original $D_{tabular}$ in VTD. Following the previous example, Figure 6.5 shows the selection of the relevant columns of source *routes.csv*, where Morph-CSV has the original source as input (Figure 6.5a), and discards the unnecessary columns of the source based on the mapping rules, obtaining as output the source with the relevant columns for evaluating the input query (Figure 6.5b). Note that in this step, unnecessary sources from the input GTFS feed such as *agency.csv* and *stops.csv* are also discarded.

[104] As usual in these approaches, we assume bounded predicates in the triple patterns

route_id	agency_id	route_short_name	route_long_name	route_type	route_code	route_url	route_color
4_1	CRTM	1	Chamartín-Valdecarros	1	401	http://crtm/metro/4_1	2DBEF0
4_2	CRTM	2	Las Rosas - C. Caminos	1	401	http://crtm/metro/4_2	ED1C24
4_3	CRTM	3	Villaverde Alto-Moncloa	1	401	http://crtm/metro/4_3	FFD000
4_4	CRTM	4	Chamartín-Argüelles	1	401	http://crtm/metro/4_4	B65518
5_C1	CRTM	C1	P.Pío-AeropuertoT4	2	109	http://crtm/train/5_1	4FB0E5
5_C2	CRTM	C2	Guadalajara-Chamartín	2	109	http://crtm/train/5_2	008B45
5_C3	CRTM	C3	Aranjuez-Escorial	2	109	http://crtm/train/5_3	9F2E86
5_C4	CRTM	C4	Parla-Colmenar Viejo	2	109	http://crtm/train/5_4	005AA3

(a) Original routes.csv input source.

route_id	route_long_name	route_type	route_code
4_1	Chamartín-Valdecarros	1	401
4_2	Las Rosas - C. Caminos	1	401
4_3	Villaverde Alto-Moncloa	1	401
4_4	Chamartín-Argüelles	1	401
5_C1	P.Pío-AeropuertoT4	2	109
5_C2	Guadalajara-Chamartín	2	109
5_C3	Aranjuez-Escorial	2	109
5_C4	Parla-Colmenar Viejo	2	109

(b) Output of routes.csv source.

Figure 6.5: Source selection step by Morph-CSV. Based on the selection of the rules, only route_id and trip_id columns are selected, discarding the rest fields.

Normalization

There are two functions for performing data normalization. The first one is the treatment of multi-values in a column. In this case, Morph-CSV performs the function $split(A_{ij}, sep)$ where A_{ij} is the multi-valued column of source s_j and sep is the character defined in the CSVW metadata using the `csvw:separator` property. The output is a modified VTD with a new source s_t containing the separated values in one column with a common identifier ID_{ij} in another column and an $s_j^{'}$ source where the values of A_{ij} are substituted by the identifier defined in s_t, ID_{ij}. Additionally, this function modifies the mapping document M with a new `rr:TriplesMap` TM_t

generated for the new source s_t and a `rr:joinCondition` between the `rr:TriplesMap` of s_j, TM_j and TM_t. The application of this function is known as the normalization step for second normal form (2NF) (Codd, 1979).

The second function is the treatment of multiple entities in the same source. Morph-CSV takes the mapping rules and executes the function $cut(\mathcal{M}, \mathcal{D}_{tabular})$. This function analyzes the mapping rules $\mathcal{M}$, and performs a 3NF (Codd, 1979) normalization step over $D_{tabular}$ when there are two sets of mapping rules (TM_j and TM_i) that have the same source, and the intersection of their columns in the rules only contains the join condition references. Following a similar approach as in 2NF, the output is a modified VTD with a set of new sources $s_i \ldots s_n$, each one with the corresponding columns of each entity. For example, in Figure 6.6 we show the 3NF normalization of the *routes.csv* file, that generates an auxiliary source for the `rr:TriplesMap` with the *gtfs:RouteType* entity data (Figure 6.6), removing that information from *routes.csv*. In several data integration approaches, normalization steps are not taken into account in order to improve query execution (reducing the number of joins among sources). However, in the case of RDF, where each entity of a class has a unique URI (subject), joins cannot be reduced (see input mapping in Figure 6.4b). This means that taking into account normalization steps in an OBDA context not only helps to improve query completeness, but also helps to improve performance. Additionally, normalization is also essential for allowing Morph-CSV to efficiently run data preparation steps, as we show in the next step.

route_id	route_long_name	route_type
4_1	Chamartín-Valdecarros	1
4_2	Las Rosas - C. Caminos	1
4_3	Villaverde Alto-Moncloa	1
4_4	Chamartín-Argüelles	1
5_C1	P.Pío-AeropuertoT4	2
5_C2	Guadalajara-Chamartín	2
5_C3	Aranjuez-Escorial	2
5_C4	Parla-Colmenar Viejo	2

(a) Routes.csv after 3NF normalization step.

route_type	route_code
1	401
1	401
1	401
1	401
2	109
2	109
2	109
2	109

(b) Route_type.csv file generated with Morph-CSV.

Figure 6.6: Normalization step by Morph-CSV. 3NF Normalization step over the *routes.csv* file generating other file with the data for `gtfs:RouteType` class.

Data preparation

In this step, Morph-CSV addresses the challenge of *Heterogeneity* and executes three different functions: $duplicates$, *sub* and $create$. First, Morph-CSV removes all duplicates in the raw data, not only the original ones, but also other duplicates that can appear during the normalization step (see Figure 6.6b). It applies the ideas described in (Jozashoori & Vidal, 2019), performing $duplicates(s_j)$ where s_j is a source in $D_{tabular}$. As it has already been demonstrated in (Iglesias *et al.*, 2020; Jozashoori & Vidal, 2019; Jozashoori *et al.*, 2020), this step not only has a high impact on the behavior of these engines, but in this case, it also reduces the number of operations performed by Morph-CSV *sub* and $create$, as they are defined as deterministic functions. The first one is defined as $sub(exp(A_{ij}), val)$ where $exp(A_{ij})$ is a boolean function over column A_{ij} of source s_j that when true, the value of A_{ij} is substituted by val. There are multiple substitution functions that Morph-CSV executes such as default values, null values and date formats. This function is one of the most important for enhancing the completeness of the query (e.g., enforcing the default values of a column). The second function creates a new column in a specific source s_j. It is defined as $create(c(A_{nj}, \ldots, A_{mj}))$, where $c(A_{nj}, \ldots, A_{mj})$ is the application of a set of transformation functions over the columns $A_{nj}, \ldots, A_{mj}$ in source s_j. This function is used to push down the application of ad-hoc transformation functions, usually defined inside the mapping rules (De Meester *et al.*, 2017; Junior *et al.*, 2016a), thus, avoiding the incorporation of them inside the SQL translated query. In Figure 6.7 we show the *route_type.csv* file after the execution of this step. First, Morph-CSV removes the duplicates of the file obtaining as output a file with only two rows. Then, it executes the transformation function defined in the mapping rules and creates a new column in the file, generating the desired value for the subject of the class according to the LinkedGTFS ontology, "Subway". Additionally, the engine substitutes the definition of the transformation functions in the mapping rules by a reference to the created column. In this manner, Morph-CSV efficiently performs the *sub* and $create$ functions directly over the raw data and together with the normalization step. Thus, the number of joins in the input query is reduced.

Schema creation and load

The final step before translating and executing the query is the creation of an SQL schema applying the rest of the identified constraints, and loading the selected tabular data sources. Besides the typical integrity constraints that can be extracted from CSVW annotations (PK/FK),

route_type	route_code	route_type_fn
1	401	Subway
2	109	Train

```
route-type:
   sources:
     - [routes_types.csv~csv]
   s: gtfs:$(route_type_fn)
    po:
     - [gtfs:routeTypeCode,$(route_code)]
```

Figure 6.7: Data preparation of *route-types.csv* file.

Morph-CSV implements a rule for creating indexes in the RDB instance in order to optimize the execution of query joins. In tabular datasets, it is common that the join conditions defined in the mapping rules are based on columns that are not part of PK-FK relations; thus, they are not indexed and OBDA optimizations do not have the desired effect. To address this problem, Morph-CSV gets the `rr:child` and `rr:parent` references of the mapping rules and calculates their selectivity on the fly. Then, taking this selectivity into account Morph-CSV decides to create, or not, an index over these columns. Additionally, the mapping document is translated so that it is aligned with the RDB schema that has been created. Figure 6.8 shows the RDB schema generated by Morph-CSV for the input query in Figure 6.4a, with the applied domain and integrity constraints.

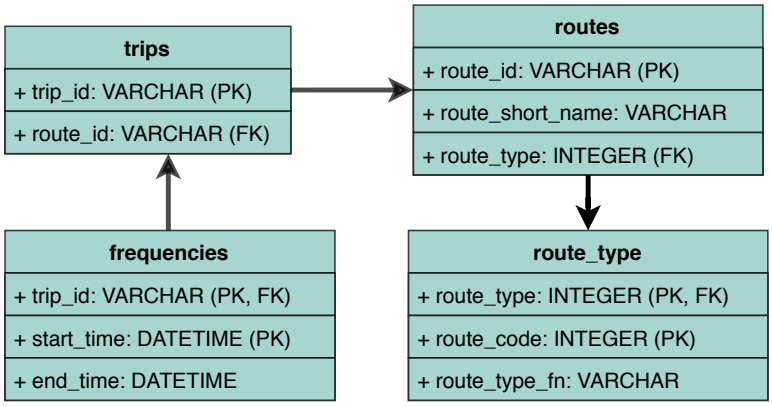


Figure 6.8: Generated schema. The schema generated by Morph-CSV extracting domain and integrity constraints from the annotations and based on the identified sources selected from the input query.

There are two main points that make the contributions of Morph-CSV relevant: (i) it incorporates the steps to the standard OBDA workflow without modifying the rest of the steps,

hence, it can also benefit from optimizations in other steps of the workflow like query rewriting (reasoning) (Mora *et al.*, 2014) or query translation (SPARQL-to-SQL) (Priyatna *et al.*, 2014), and (ii) the reliance of the approach on declarative and standard annotations for OBDA allows the generalization of the proposed steps, usually solved in an ad-hoc manner, not only automatizing the process but also improving its maintainability, understandability and readability.

6.1.4 Experimental Evaluation

This section reports on the results of the empirical evaluation conducted to test the effect of respecting constraints, on the fly, during OBDA query translation over tabular data. The hypothesis we want to validate in our work are:

- *H1)* The application of a set of domain and integrity constraints over tabular data sources to create an RDB instance that ensures the effectiveness of SPARQL-to-SQL optimizations proposed in the state of the art.
- *H2)* Extending a common OBDA workflow with a set of additional steps to deal with the challenges for querying tabular data, does not impact negatively on the total query execution time.
- *H3)* The exploitation of declarative and standard annotations in the process of querying tabular data in an OBDA environment, guarantees the independence of the solution and its application over different domains.

Aligned with the defined hypothesis, our aim is to answer the following research questions: **RQ1)** What is the effect of combining different types of constraints over a tabular dataset? **RQ2)** What is the impact of the constraints when the tabular dataset size increases? **RQ3)** What is the effect of different kinds of SPARQL query shapes in the extraction and application of constraints?. To answer these questions, we have performed three evaluations in different domains: e-commerce, transportation, and biology. Our first evaluation is in the e-commerce domain, in which we used the Berlin SPARQL Benchmark (BSBM) (Bizer & Schultz, 2009). Our second evaluation is in the transportation domain in which we used the GTFS-Madrid-Bench (Chaves-Fraga *et al.*, 2020a). This benchmark focuses on measuring the performance of ontology based data access for heterogeneous data sources, based on the publicly-released public transportation data in GTFS format. One of the resources provided by GTFS-Madrid-Bench is a tabular dataset together with its corresponding mappings and annotations together

with a set of representative SPARQL queries. Finally, our third evaluation is in the domain of biological data, in which we extend one of our previous proposals (Iglesias-Molina *et al.*, 2019) for the generation of an OBDA layer over Bio2RDF tabular datasets. Appendix D presents the features of the queries together with the constraints and number of sources used by Morph-CSV. In all of the evaluations the common configurations are:

Engines. The baselines of our study are two open source SPARQL-to-SQL OBDA engines: Ontop[105,106] v3.0.1 and Morph-RDB v3.9.15[107]. We select these two engines as they are open source engines (others such as Ultrawrap (Sequeda & Miranker, 2013) are not openly available) and also the ones that incorporate the set of most relevant optimizations in the SPARQL-to-SQL query translation process (Priyatna *et al.*, 2014; Rodriguez-Muro & Rezk, 2015). To evaluate the baseline approach, we manually generate the relational database schemes of each benchmark without any kind of constraints, and measure the load and query execution times. In order to measure the impact of the additional steps proposed by Morph-CSV[108,109], we integrate our solution on top of the two OBDA engines in two different configuration: Morph-CSV^{-} that does not include the source selection step, hence, it loads and applies all the constraints over the input data source each time a query has to be answered, and Morph-CSV that implements the full proposed workflow[110]. To ensure the reproducibility of the experiments, we also provide all of the resources in a docker image.

Metrics. We measure the loading time of each query and the total query execution time (including the steps proposed by Morph-CSV or baseline when appropriate), and the number of answers obtained (see Appendix E). Additionally, we detail the times of each proposed step of our workflow in the execution of each query using Morph-CSV in both configurations (see Appendix F) following the recommendations proposed in the GTFS-Madrid-Bench (Chaves-Fraga *et al.*, 2020a). Each query was executed 5 times with a timeout of 1 hour in cold mode, that means that the corresponding database is generated each time a query is going to be evaluated in order to ensure up to date number of answers. Regarding the completeness of the queries, both BSBM benchmark and GTFS-Madrid-Bench provide an RDF materialized version of the input sources that has been loaded in a triplestore (Virtuoso in the case) and used

[105] `https://github.com/ontop/ontop`

[106] We modified the default configuration of Ontop extending the maximum used memory from 512Mg to 8Gb

[107] `https://github.com/oeg-upm/morph-rdb`

[108] `https://doi.org/10.5281/zenodo.3731941`

[109] `https://github.com/oeg-upm/morph-csv`

[110] We name the combined engines as follows: a) Morph-CSV: Morph-CSV+Morph-RDB, and Morph-CSV+Ontop; b) Morph-CSV^{-}: Morph-CSV^{-}+Morph-RDB, and Morph-CSV^{-}+Ontop

as gold standard. To analyze the completeness of each query, we compare the cardinality of the result set of each configuration against the gold standard assuming its correctness. In the case of the Bio2RDF use case, we cannot compare our results with any gold standard as the last dump version of the project (Dumontier *et al.*, 2014) is not comparable with the current status of the input sources, as we declare in one of our previous works (Iglesias-Molina *et al.*, 2019). The experiments were run in an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60GHz 20 cores, 64GB memory and with the O.S. Ubuntu 16.04LTS.

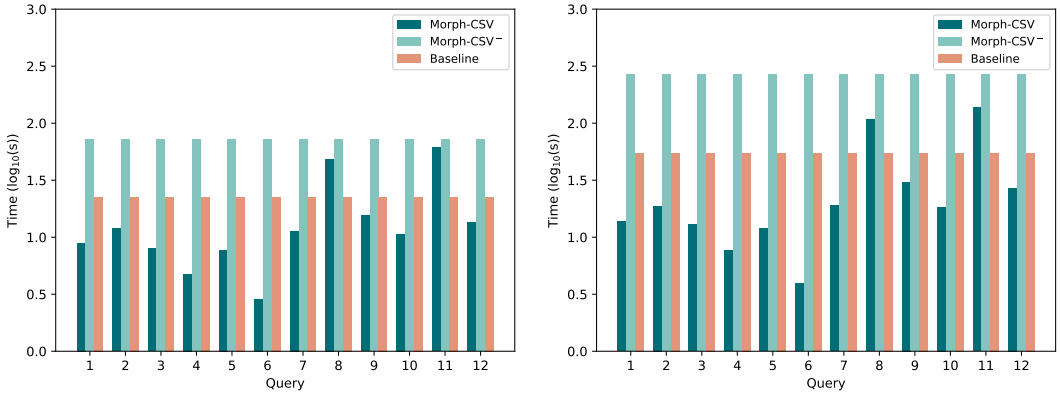
6.1.4.1 BSBM

The Berlin SPARQL Benchmark (Bizer & Schultz, 2009) is one the most popular benchmarks in the Semantic Web field that not only tests the performance of RDF triple stores, but also tests approaches that perform SPARQL-to-SQL query translations providing an RDB instance. It is the chosen benchmark to test the capabilities of many state-of-the-art OBDA engines (Calvanese *et al.*, 2017; Mami *et al.*, 2019b; Priyatna *et al.*, 2014).

Datasets, annotations and queries. In order to test our proposal we decided to adapt BSBM, extracting the tabular data sources in CSV format from the SQL generated instances. Additionally, we create the corresponding mapping rules in RML and the metadata following the CSVW specification. We measure the loading time of the two proposals (baseline and Morph-CSV) for each query in the benchmark. Since the focus of Morph-CSV is not the improvement of the support of SPARQL features in the query translation process, we only select the queries of the benchmark that include the supported features by each engine. This means that Morph-RDB will be evaluated over the queries Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10 and Q12 and Ontop will be evaluated over Q1, Q3, Q4, Q5 and Q10, both of them using the corresponding R2RML mapping document. For the baseline approach we manually create the RDB schema without constraints.

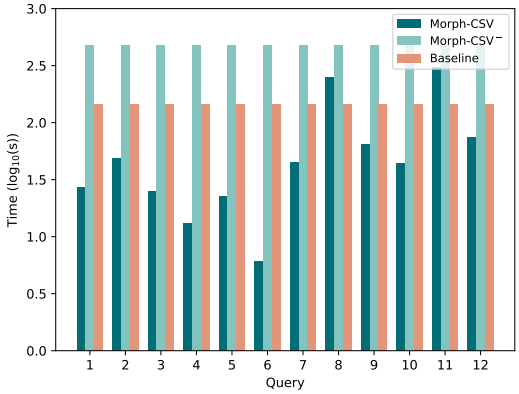
BSBM Results

Loading Time. The results of the load time for each query and dataset size are shown in Figure 6.9. The main difference between baseline and Morph-CSV$^-$ in comparison with Morph-CSV is that while the loading time for the first two methods is constant for each size, Morph-CSV loading time depends on several input parameters such as the query and the number and type of constraints. In the case of Morph-CSV, it could be understandable that the application

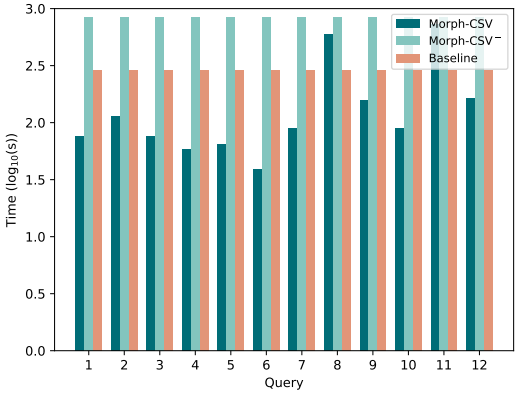


(a) Loading time for BSBM 45K.

(b) Loading time for BSBM 90K.



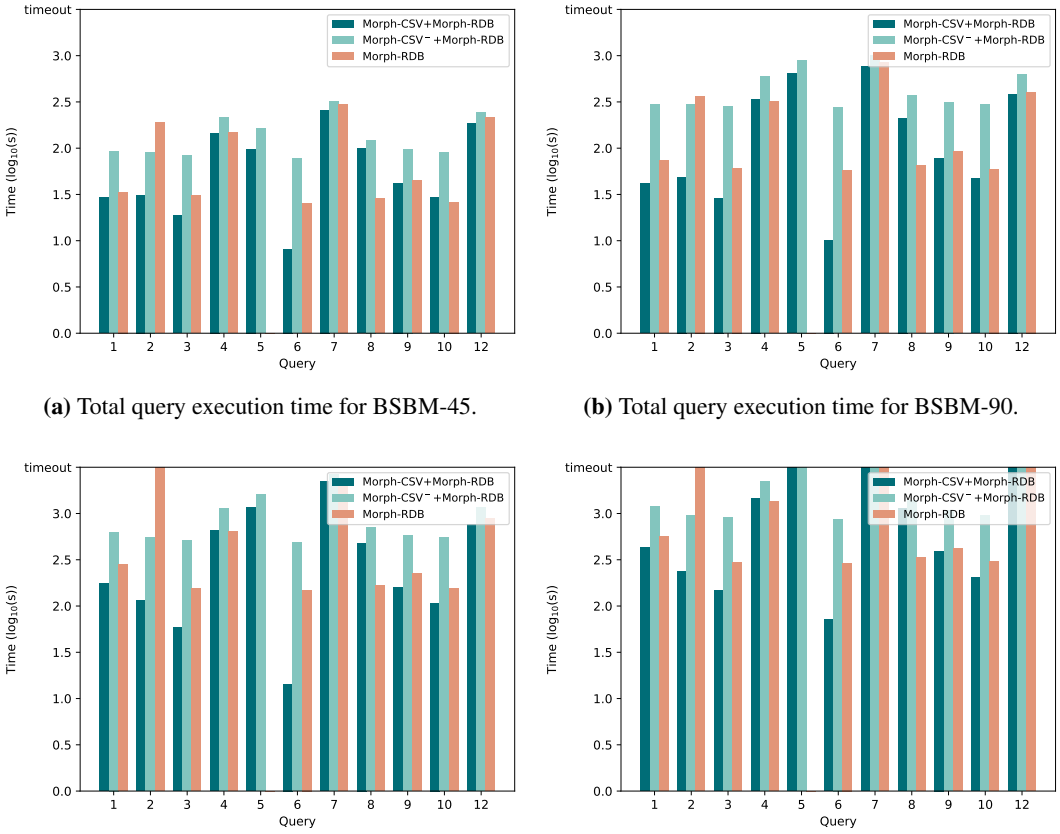
(c) Loading time for BSBM 180K.



(d) Loading time for BSBM 360K.

Figure 6.9: Loading Time of Tabular Datasets in BSBM. Loading time in seconds of the tabular datasets from the BSBM benchmark with number of products 45K, 90K, 180K and 360K. The baseline approach (red columns) and Morph-CSV$^-$ (light green) are constant for each dataset and query, while Morph-CSV (dark green) depends on the query and number of constraints to be applied over the selected sources.

of a set of constraints over the raw data in order to improve query performance and completeness, would have a negative impact in the loading time. This happens in queries Q8 and Q11, where the number of sources and the application of the constraints (mainly integrity constraints), impact negatively on the loading time of the data in the RDB instance in comparison with the baseline approach. However, in the rest of the queries, the Morph-CSV steps focus on the selection of constraints, sources and columns, and on exploiting the information in query and mapping rules, improving the loading time for each query in comparison with the baseline



(a) Total query execution time for BSBM-45.

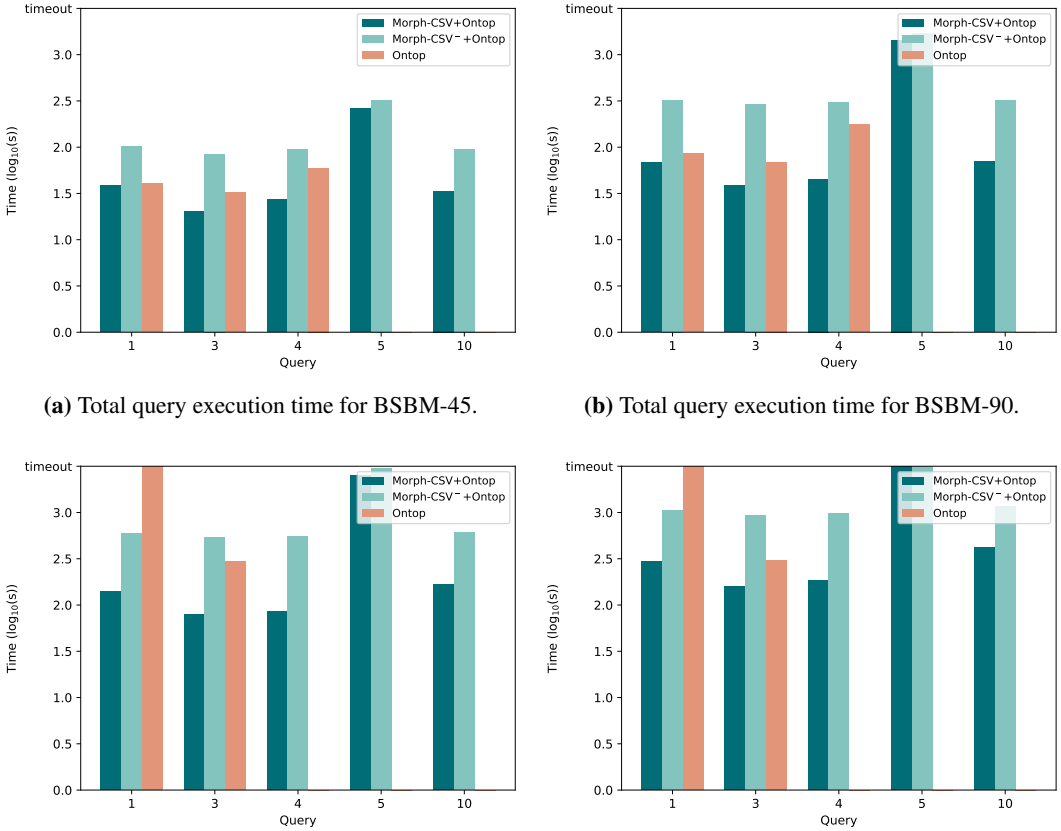
(b) Total query execution time for BSBM-90.

(c) Total query execution time for BSBM-180.

(d) Total query execution time for BSBM-360.

Figure 6.10: Query execution Time of Tabular Datasets in BSBM with Morph-RDB. Execution time in seconds of the tabular datasets from the BSBM benchmark with scale values 45K, 90K, 180K and 360K. The baseline Morph-RDB approach (red columns) is compared with the combination of Morph-CSV (dark green) and Morph-CSV^{-} (light green) together with Morph-RDB.

loading time. This means that, although the engine is including a set of additional steps during the starting phase of an OBDA system, the application of these steps only over the data that is required to answer the query, has a positive impact in the total query execution time. Additionally, we can observe that Morph-CSV is able to process, apply the different constraints, and generate the corresponding instance of the RDB for any query. In the case of Morph-CSV^{-}, applying all the constraints defined for the whole dataset each time a query has to be answered, has a negative impact in the loading time, obtaining the worst results in the loading phase.



(a) Total query execution time for BSBM-45.

(b) Total query execution time for BSBM-90.

(c) Total query execution time for BSBM-180.

(d) Total query execution time for BSBM-360.

Figure 6.11: Query execution Time of Tabular Datasets in BSBM with Ontop. Execution time in seconds of the tabular datasets from the BSBM benchmark with scale values 45K, 90K, 180K and 360K. The baseline Ontop approach (red columns) is compared with the combination of Morph-CSV (dark green) and Morph-CSV$^-$ (light green) together with Ontop.

Evaluation Time with Morph-RDB. The query execution time using Morph-RDB as the back-end OBDA engine is shown in Figure 6.10. The first remarkable observation can be seen in query Q5. Although this query contains features supported by Morph-RDB, the engine reports an error when evaluating the query over the database generated by the baseline approach, because it is not able to evaluate the arithmetic expressions in the FILTER clauses. On the contrary, the datatype of each column in the database generated by Morph-CSV (and also Morph-CSV$^-$) is properly defined, making it possible for Morph-RDB to evaluate the query without any problem and obtaining the expected results. Another remarkable difference is in query Q2, which contains a large number of joins, Morph-RDB reports a timeout error

for 180K and 360K with the database generated by the baseline approach. However, it is still able to evaluate this query in reasonable time over the databases generated by Morph-CSV and Morph-CSV$^-$. The effect of the application of integrity constraints in the generation of the RDB instance can also be seen in most of the queries (i.e., Q1, Q2, Q3, Q6, Q9, Q10) reducing considerably the query execution time in the database generated by Morph-CSV in comparison with the baseline approach. There are cases (i.e., Q4, Q7, Q12) where the amount of data to retrieve is large, minimizing the effect of the optimizations. Finally, there are cases where optimizations over the indexes cannot be applied (e.g. querying all the properties of a class). We observe this behavior in Q8, in which the difference between the Morph-CSV+Morph-RDB and Morph-RDB approaches is minimal and this behavior is consistent in all size of datasets. In general, Morph-CSV$^-$ obtains worse results than Morph-CSV+Morph-RDB and Morph-RDB alone. The results are understandable as this configuration has to invest time in preparing the full RDB instance for each query, executing many unnecessary steps in comparison with Morph-CSV. However, in some cases the evaluation time is better than the one obtained over the Morph-RDB configuration, where clearly the creation of indexes and integrity constraints play a key role in the performance of the query execution (see Q2).

Evaluation Time with Ontop. The query execution time using Ontop as the back-end OBDA engine is shown in Figure 6.11. Like Morph-RDB, Ontop needs the Morph-CSV generated databases to be able to evaluate Q5 due to the arithmetic expressions of its FILTER operators. Additionally, it also fails in Q10 because it cannot process a FILTER with a date value. In the rest of the queries (Q1, Q3, Q4) we can see that the query evaluation time in Ontop with Morph-CSV is lower than the query evaluation time over the baseline database. Note that in larger databases (180K and 360K), Q1 and Q4 can only be evaluated over the databases generated by Morph-CSV. The Morph-CSV$^-$ configuration is also able to answer the queries just as the Morph-CSV standard configuration, but in comparison with this configuration, the performance is being affected due the inclusion of the additional and unnecessary steps.

As mentioned in the Ontop repository page[111], integrity constraints are essential for the correct behavior of the engine. Although it is out of the scope of this paper, we observe in our experiments that the main reason why Ontop is only able to answer half of the queries in this benchmark, is related to some issues about maintaining the desirable properties (Corcho *et al.*, 2019) when translating R2RML mapping rules to its own mappings, called OBDA. The

[111] `https://github.com/ontop/ontop/wiki/MappingDesignTips`

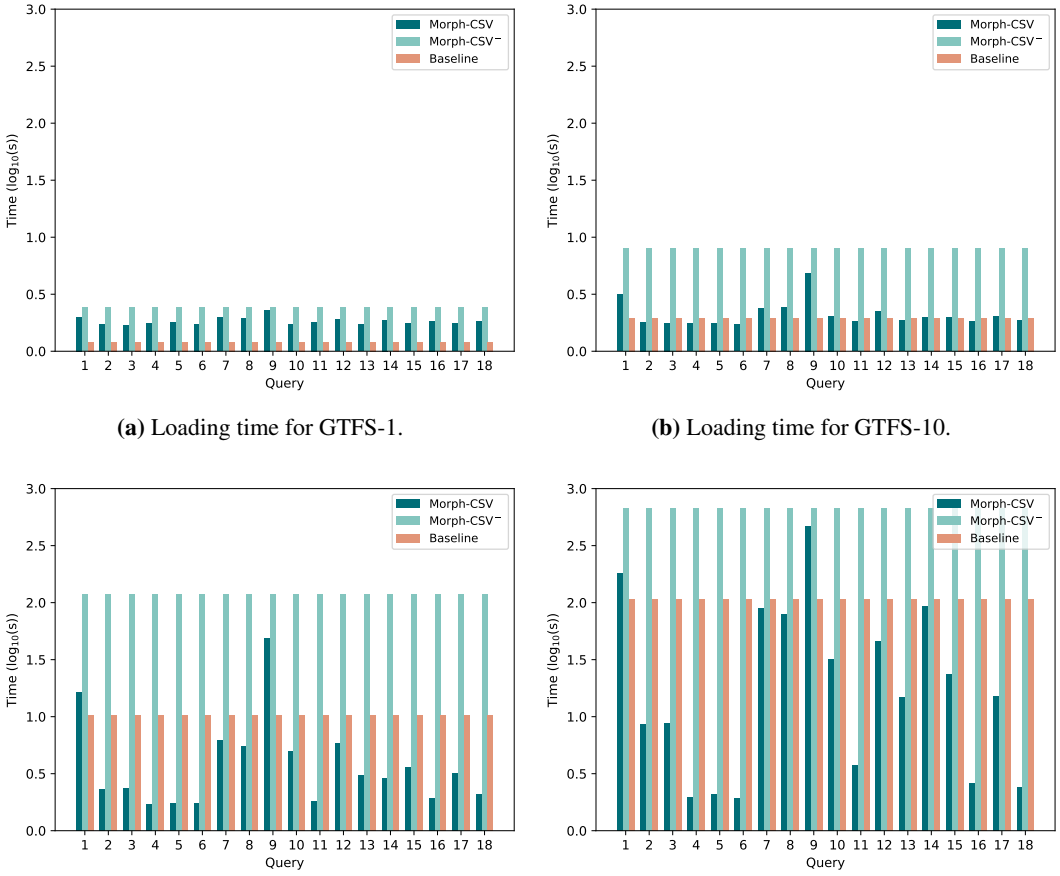
engine also fails to evaluate queries with OPTIONAL clauses when there are NULL values in the answers, as they acknowledged, it is possible that this support has not been implemented in the engine (Xiao *et al.*, 2018b).

Query completeness. In Table E.3 we show the query completeness obtained with the BSBM benchmark. It is important to remark that our intention to use this benchmark is for testing performance capabilities of our proposals, the input sources are extracted from the BSBM relational model, which is a well formed and normalized RDB instance. However, there are still some cases where we identify the need of applying constraints over the relational database, which are Q5 in the evaluation over Morph-RDB and Q5 and Q10 over Ontop. In these cases, the baseline configurations of the engines are not able to answer those queries, not because they do not support a feature of the SPARQL query or cannot do it on time, but because they cannot perform the correct comparison among different datatypes in the relational database instance. We demonstrate with the application of Morph-CSV that queries can be answered and the correct number of results can be obtained. Additionally, thanks to the application of indexes and integrity constraints there are some queries such as Q1 and Q2 that can be answered by Morph-CSV configuration but not by the baseline, which means that thanks to these steps we are ensuring the effectiveness of the optimizations provided by Ontop and Morph-RDB in the SPARQL-to-SQL translation process.

6.1.4.2 GTFS-Madrid-Bench

The GTFS-Madrid Benchmark (Chaves-Fraga *et al.*, 2020a) consists of an ontology, an initial dataset of the metro system of Madrid following the GTFS model, a set of mappings in several specifications, a set of queries according to the ontology that cover relevant features of the SPARQL query language, and a data scaler based on a state of the art proposal (Lanti *et al.*, 2015).

Datasets, annotations and queries. We select the tabular sources of this benchmark (i.e., the CSV files) and we scale up the original data in several instances (scale factors 10, 100 and 1000). Each generated dataset is denoted as GTFS-*S* where S is the scale factor. The resources of the benchmark already include the necessary mapping rules and tabular metadata. Like our previous evaluation with BSBM benchmark, we only select the queries with features that are supported by each engine: Morph-RDB will be evaluated using queries Q1, Q2, Q4, Q6, Q7,



(a) Loading time for GTFS-1.

(b) Loading time for GTFS-10.

(c) Loading time for GTFS-100.

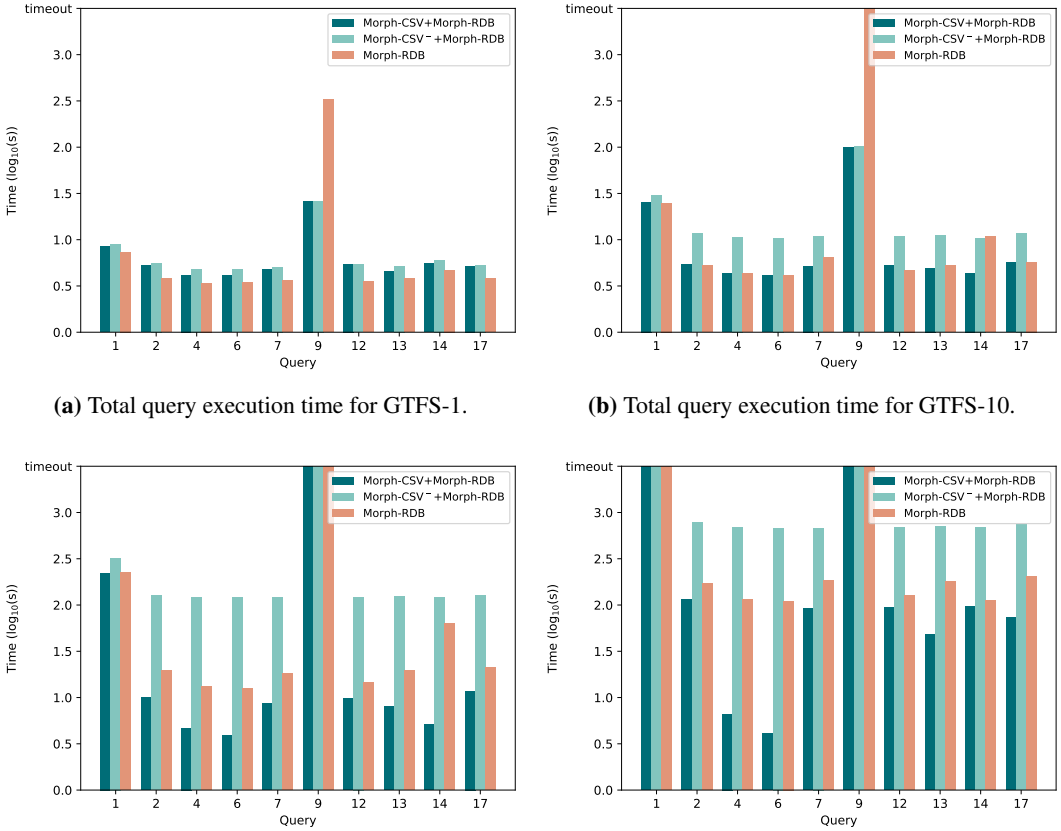
(d) Loading time for GTFS-1000.

Figure 6.12: Loading Time of Tabular Datasets in GTFS. Loading time in seconds of the tabular datasets from the Madrid-GTFS-Bench with scale values 1, 10, 100 and 1000. The baseline approach (red columns) and Morph-CSV^{-} (light green) are constant for each dataset and query, while Morph-CSV (dark green) depends on the query and number of constraints to be applied over the selected sources.

Q9, Q12, Q13, Q14, Q17 and Ontop will be evaluated using queries Q1, Q2, Q3, Q4, Q5, Q7, Q9, Q13, Q14, Q17. The description and features of each query are also available online[112].

Madrid-GTFS-Bench Results

[112] `https://github.com/oeg-upm/gtfs-bench/`



(a) Total query execution time for GTFS-1.

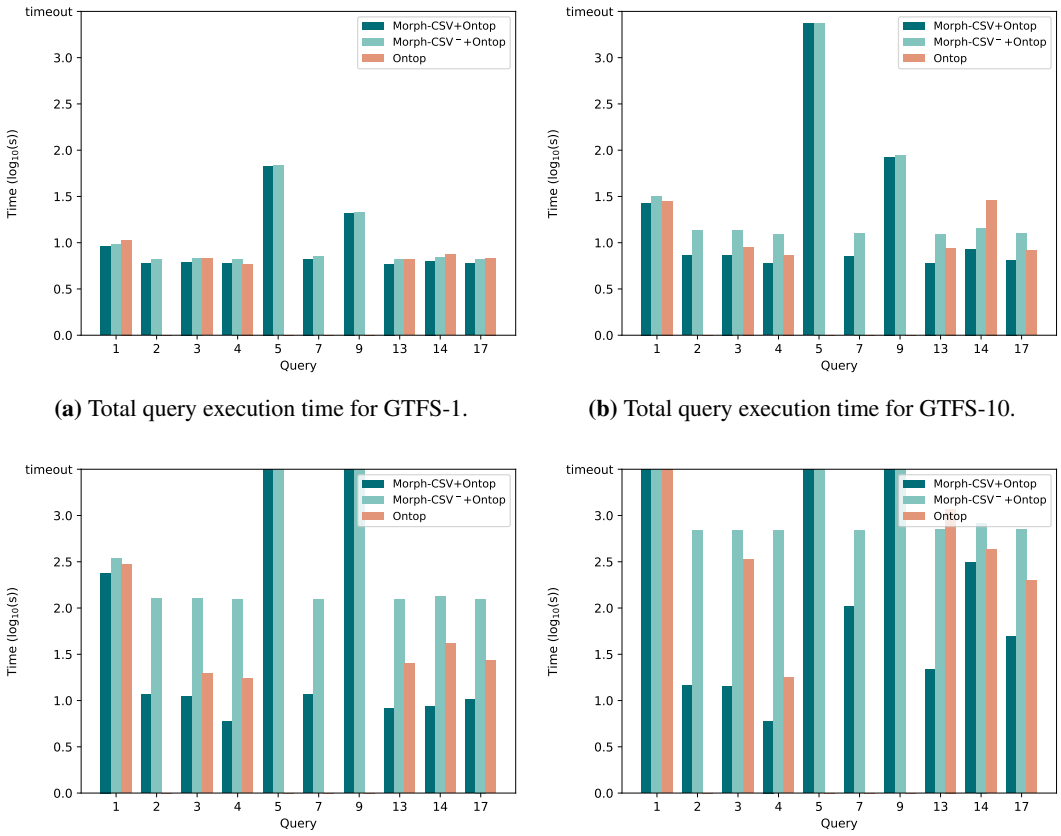
(b) Total query execution time for GTFS-10.

(c) Total query execution time for GTFS-100.

(d) Total query execution time for GTFS-1000.

Figure 6.13: Query execution Time of Tabular Datasets in GTFS with Morph-RDB. Execution time in seconds of the tabular datasets from the Madrid-GTFS-Bench with scale values 1, 10, 100 and 1000. The baseline Morph-RDB approach (red columns) is compared with the combination of Morph-CSV (dark green) and Morph-CSV^{-} (light green) together with Morph-RDB.

Loading Time. The loading time of the GTFS-Madrid-Bench queries is shown in Figure 6.12. For GTFS-1 the baseline approach clearly has better performance than Morph-CSV. However, when the size of the datasets increases, the positive effects of applying constraints become more apparent. For most of the queries, the loading time needed by Morph-CSV is lower in comparison to the loading time in the baseline approach. Additionally, similarly to BSBM, there are a set of queries where the application of integrity constraints has a negative impact on the loading time (queries Q1 and Q9). The impact of the application of all of the constraints for answering each query, presented by the configuration Morph-CSV^{-}, clearly impacts over the performance in the loading time.



(a) Total query execution time for GTFS-1.

(b) Total query execution time for GTFS-10.

(c) Total query execution time for GTFS-100.

(d) Total query execution time for GTFS-1000.

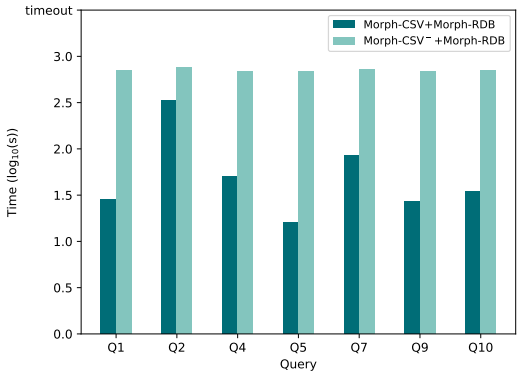
Figure 6.14: Query execution Time of Tabular Datasets in GTFS with Ontop. Execution time in seconds of the tabular datasets from the Madrid-GTFS-Bench with scale values 1, 10, 100 and 1000. The baseline Ontop approach (red columns) is compared with the combination of Morph-CSV (dark green) and Morph-CSV$^-$ (light green) together with Ontop.

Evaluation Time with Morph-RDB. The query execution time with Morph-RDB as the back-end OBDA engine is shown in Figure 6.13. Analyzing the results, we generally observe that the incorporation of Morph-CSV in the workflow of OBDA enhances query performance. With respect to the results of each query, we can observe that on the one hand the behavior of the engine over simple queries (Q1, Q2, Q7, Q12 and Q17) is similar. This is understandable as the selected data sources needed to answer the query do not include the application of several constraints (e.g. there are no joins in the query). On the other hand, in the case of complex queries such as Q4, Q6, Q9, Q13 and Q14, where several tabular sources are needed to answer the queries, the application of constraints has a better impact in comparison to the the baseline

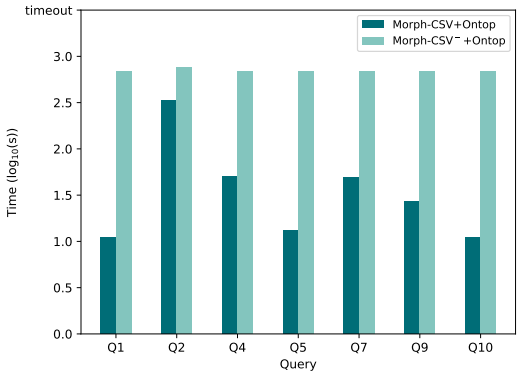
approach. Similar behavior is shown over Morph-CSV$^-$, where the complexity of the GTFS data model, with many sources, columns and relations among them, has a clear impact on the total execution time of each query, obtaining worse performance than the baseline in most of the cases. However, for example, in the case of query Q9, Morph-RDB is not able to evaluate the query over the 10th scale database generated by the baseline approach, while in the case of the database generated by Morph-CSV and Morph-CSV$^-$, the query can be answered in reasonable time. In general, due the complexity of GTFS model, we can observe that for small datasets (GTFS-1), the cost of applying the proposed steps of Morph-CSV impacts total execution time. However, when the size of the dataset increases, the baseline approach is impacted due to the fact that it has to load all of the input data sources in the RDB before executing the query, low performance is reported for GTFS-100 and GTFS-1000, including timeout in some queries of the latter. Thanks to the application of the constraints and to the source selection step, for Morph-CSV together with Morph-RDB, the return of the results of the queries has a high performance most of the time. In the cases where Morph-CSV reports a timeout (e.g., Q1 in GTFS-1000); it is because the extremely high number of obtained results cannot be handle by Morph-RDB.

Evaluation Time with Ontop. The experimental evaluation of the query execution in Ontop as the back-end OBDA engine is shown in Figure 6.14. This engine is more strict with datatypes in the RDB in comparison with Morph-RDB, and it is why Q2, Q5, Q7 and Q9 produce a failure in the execution over the databases generated by the baseline approach. All these queries have a FILTER clause on a specific datatype (e.g., date, integer, etc) and Ontop proceeds to check the domain constraints before executing the queries. Morph-CSV solves this problem by exploiting the annotations from the metadata and defines the correct datatypes of each column before evaluating the query. For the queries that can be answered by both approaches (Q1, Q3, Q4, Q13, Q14, Q17), the absence of integrity constraints has a negative impact in Ontop, resulting in lower execution time over the databases generated by Morph-CSV. However, similar to the evaluation over Morph-RDB, the complexity of the GTFS data model with a larque quantity of domain and integrity constraints to be applied over the whole dataset, makes that the behavior observed over Morph-CSV$^-$ is being impacted, hence, obtaining worse results that Morph-CSV configuration and the baseline in most of the cases. Finally, in the case where Ontop is not able to evaluate the query under the defined threshold, we report it as a timeout.

Query completeness. In the same manner as BSBM benchmark, the focus of the GTFS-Madrid-Bench is on testing the performance and scalability issues of virtual OBDA and OBDI engines. The input dataset is also well formed and normalized. The completeness results of the evaluation are shown in Table E.1, where as we describe before, Morph-RDB has a mechanism to infer the datatypes of the database using the *rr:dataType* annotation from R2RML, which allows the engine to answer the queries of this benchmark without the need of applying datatype constraints over the RDB instance. However, Ontop does not include such a mechanism and it needs the declaration of the correct datatypes over the RDB instance, which has a negative impact in the execution of many queries of the benchmark, that cannot be answered using the baseline database but they retrieve the correct results including Morph-CSV (or Morph-CSV^{-}) in the pipeline.



(a) Query execution Time of Tabular Datasets in Bio2RDF with Morph-RDB. Execution time in seconds of the tabular datasets from Bio2RDF of Morph-CSV and Morph-CSV^{-} using Morph-RDB as back-end engine. The baseline is not reported as the loading over the RDB instance reports an error.



(b) Query execution Time of Tabular Datasets in Bio2RDF with Ontop. Execution time in seconds of the tabular datasets from Bio2RDF of Morph-CSV and Morph-CSV^{-} using Ontop as back-end engine. The baseline is not reported as the loading over the RDB instance reports an error.

Figure 6.15: Query execution Time of Tabular Datasets in Bio2RDF

6.1.4.3 Use Case: The Bio2RDF project

Bio2RDF is one of the most popular projects that integrates and publishes biomedical datasets as Linked Data (Belleau *et al.*, 2008). Its community has actively contributed to the generation of those datasets using ad-hoc programming scripts, such as PHP. In our previous work (Iglesias-Molina *et al.*, 2019) we proposed an alternative way of generating the datasets

using a set of declarative mapping rules to improve the maintainability, readability and understanding of the procedure. In comparison with the other benchmarks where the focus of the evaluation was the improvement of the query evaluation time, this real use case contains multiple heterogeneity challenges that, for example, enforce the application of ad-hoc transformation functions (i.e., mappings in the form of RML+FnO). Thus, with this use case we want to demonstrate the benefits of exploiting declarative annotations (metadata and mappings) over the raw data in order to improve query completeness and the need of incorporating the proposed steps for executing queries over real world data sources.

Dataset, annotations, and queries. Tabular datasets in CSV or Excel formats cover over 35% of the total datasets in the Bio2RDF project (Iglesias-Molina *et al.*, 2019). In order to test the capabilities of Morph-CSV, we select a subset of the tabular datasets guaranteeing that they cover all of the identified challenges. Additionally, as far as we are aware, there is no standard benchmark over the Bio2RDF project; we also propose a set of SPARQL queries in order to exploit the selected data. Their main features are shown in Appendix D).

Bio2RDF Results The results obtained for query evaluation in Bio2RDF are shown in Figure 6.15b with Morph-RDB as back-end engine and in Figure 6.15b with Ontop. The detailed results obtained by Morph-CSV and Morph-CSV^{-} are shown in Table F.3 and the completeness in Table E.2. Analyzing the obtained results, we can observe that there are no results for the baseline approach, this means it was not possible to create an RDB schema and load the input data manually. The main reasons are the heterogeneity problems of a real use case that do not exist in the previous evaluations. GTFS and BSBM have well formed and standard source data models. Problems such as the absence of column names, multiple formats of same datatype in different files (numbers, dates) and the use of delimiters inside the column data, make it impossible to generate the baseline approach without a manual and ad-hoc preprocessing step. However, exploiting declarative annotations, Morph-CSV is able to apply the proposed workflow to this dataset, and successfully answer the proposed queries with both back-end OBDA engines. Similar to the previous benchmarks, loading the complete dataset for answering the input query (Morph-CSV^{-} configuration) has an negative impact on the total execution time. We can observe that for the proposed queries, most of the total evaluation time of each query is spent in the loading process, as the total execution time in Morph-CSV^{-} is pretty similar for all the queries. Contrary, query execution is benefited by this previous step obtaining the results in reasonable time for all of the queries.

6.1.4.4 Discussion of Experimental Results

We have run an experimental evaluation to analyze what are the effects on the use of declarative annotations to extract and apply constraints to enhance virtual OBDA approaches. We have tested our approach over three different cases: (i) a well known benchmark (BSBM) from the e-commerce domain; (ii) a benchmark focused on a virtual OBDA approach for the transport domain; and (iii) a real use case from the biological domain. We describe the main conclusions and findings based on the results obtained:

- **Query complexity:** Clear benefits are obtained from being able to analyze and take advantage of the information provided by the input query, before translating and running it. It allows to only select sources and constraints that are going to be useful for answering the query, avoiding carrying out additional and unnecessary functions over the raw data. Together with the mapping rules, the queries are essential to make relevant decisions during the on-the-fly physical design of the RDB instance (e.g., integrity constraints). Approaches such as the Morph-CSV^{-} configuration can be valuable when the freshness of the results is not a main requirement, for example to perform a materialization process, which will ensure high quality RDF files where the domain constraints have been applied.

- **Data size:** The total query evaluation time is being impact from how the engine manages the input dataset and the application of constraints. The delegation of these operations to the RDBMS system after loading the full dataset may not be efficient enough. Morph-CSV pushes down the source selection and the application of domain constraints over the raw data. Although it incorporates a set of additional steps in comparison with the baseline, the benefits in the query execution time by the SPARQL-to-SQL engine are already demonstrated, enhancing the total execution time of the queries in most of the cases.

- **Declarative annotations:** The use of declarative and standard mapping rules and metadata makes it possible the generalization of the proposal, avoiding ad-hoc and manual steps. It also incorporates a set of important benefits for the process such as the improvement of its maintainability, readability, and understandability.

- **Querying raw data in OBDA:** Most of the data shared on the web is currently raw data in well known formats such as CSV, JSON, and XML. Semantic Web and more specifically, OBDA technologies, play a key role in starting to see the web as an integrated database that can be queried. With this approach, we demonstrate that querying tabular data is: i) neither a trivial nor an easy task that can be delegated to naïve querying approaches and ii) optimizations and improvements can still be proposed taking advantage and exploiting current annotation proposals to not only enhance performance but also completeness.

6.1.5 Conclusions

We have presented an extension of the common OBDA specification to address the problem of query translation over tabular data. We describe and evaluate Morph-CSV, a framework that exploits the information of mapping rules and metadata OBDA annotations to extract and apply a set of relevant constraints. It pushes down the application of these elements directly over the raw data in order to improve query evaluation and query completeness. One of the main contributions of this proposal is that it can be used together with any OBDA framework. From the set of experiments that we have performed with two existing state-of-the-art OBDA engines (Morph-RDB and Ontop), we can see that the use of those engines inside the Morph-CSV framework brings several positive impacts: more queries can be answered and less time is needed to answer most queries.

The definition, application and optimization of new functions and constraints to address other challenges for querying tabular data is one of the main lines for future work (Iglesias-Molina *et al.*, 2019). We also want to study the performance of the proposed workflow over OBDA distributed query systems such as the ones proposed in (Endris *et al.*, 2019; Mami *et al.*, 2019b). More in detail, we want to analyze if the outcomes of the proposed steps by Morph-CSV can help in distributed environments where physical design of knowledge graphs are being proposed (Guerrero, 2020; Rohde & Vidal, 2020) to enhance query performance (i.e., deciding which input sources have to be transformed to RDF and which ones have to be maintained in their original format). Additionally, one of the possible future work lines is the comparison of the proposed approached, that exploits semantic web technologies and annotations, against non-semantic web solutions that provide support to deal with the identified challenges for querying tabular data (e.g., Apache Drill, Presto, Spark, etc.). The results obtained can also be useful to machine learning approaches that identify when the application of

the integrity constraints is needed or not, as we observe that there are special cases where it can have a negative impact. We will also study the challenges for querying other data formats (e.g., XML, JSON) in an OBDA context and extend our approach to incorporate them. We also want to remark the importance of having standard and shared methods and vocabularies to publish metadata of raw data on the web, available for tabular data but not for tree data formats such as XML and JSON. Finally, we will adapt this proposal for a materialization process and study its effects comparing it with previous proposals. We also want to study how the materialization process can be improved when historical versions of the same RDF-based knowledge graph are needed, for example, analyzing which input sources have been changed or not, to decide which parts of that knowledge graph have to be generated again.

6.2 GraphQL Server Generation from Declarative Mappings

Introduced in 2000, Representational State Transfer (REST) has become the most common manner to provide web services in the last decade. Those web services that conform to the REST principles, known as RESTful web services, use HTTP/S and its operations to make requests to the underlying server, such as GET to retrieve objects, POST to add objects, PUT to modify objects and DELETE to remove objects, among others.

Over the years, the complexity of modern software concept has evolved since the inception of REST. For example, typical mobile applications have to take into account aspects that receive little attention in traditional applications, such as the size of data being exchanged/transmitted and the number of API calls being made. These aspects are relevant to the problem known as *over-fetching* and *under-fetching* (Bryant, 2017; Mukhiya *et al.*, 2019; Vogel *et al.*, 2017). Over-fetching refers to the situation in which a REST endpoint returns more data than what is required by the developer (Bryant, 2017; Mukhiya *et al.*, 2019; Vogel *et al.*, 2017). For example, a developer may need some information about the name of a user, so she hits the corresponding endpoint (`/user`). However, the endpoint may return information that is not needed by the client, such as birth date and address. The opposite also raises a problem, which is having the REST endpoint provide less data than required. Such a case is called under-fetching (Bryant, 2017; Mukhiya *et al.*, 2019; Vogel *et al.*, 2017). It refers to the situation in which a single REST endpoint does not provide sufficient information requested by the client. For example, in order to obtain the names of all friends of a particular user, typically two endpoints may be needed: the first is the endpoint that returns the identifiers of all the friends

(`/friends`), and the second is the one that returns the details of each of the friends based on the identifier (`/user`).

In order to ameliorate the aforementioned problems, Facebook proposed the GraphQL query language (Facebook, Inc., 2018), initially being used internally by the company in 2012. GraphQL was released for public use in 2015 and since then has been adopted by companies from various sectors such as technology (GitHub), entertainment (Netflix), finance (PayPal), travel (KLM), among others.

Two main components of a GraphQL server are **schema** and **resolvers**. The GraphQL schema specifies the type of an object together with the fields that can be queried. GraphQL resolvers are data extraction functions responsible for accessing the underlying datasets. These functions are written by software engineers using a specific programming language and, are then used by GraphQL engines, which translate GraphQL queries to the corresponding underlying query language of the sources (e.g., SQL). Multiple GraphQL engines support major programming languages (e.g. JavaScript, Python, Java, Golang, Ruby). In addition to the aforementioned frameworks, query planning tools have been developed in order to translate GraphQL queries into other query languages (e.g. dataloader[113], joinmonster[114]).

Generating a GraphQL server requires expertise from both domain experts and software developers. Typically, the following tasks need to be done:

1. A domain expert would analyse the underlying datasets, propose a unified view schema as a GraphQL schema and how the source datasets would need to be mapped into the GraphQL schema. Note that there is no standard mechanism to represent these mappings. Domain experts may use a spreadsheet, which is not necessarily easy to understand by another domain expert. In the absence of a standard representation, different ways to represent mappings are possible. Some of such spreadsheets represent the relation among source and target concepts in a naïve manner using. Others use Excel files with pages, such that each page represents a concept. Others add ids instead of property names. It becomes even more messy when there is an operation involved (e.g. source has the name in two properties/columns, “first-name” and “last-name” while the target has one single attribute to represent both, “name”).

[113] `https://github.com/facebook/dataloader`
[114] `https://join-monster.readthedocs.io/en/latest/`

2. A software developer would then implement those mappings as GraphQL resolvers, a process that takes significant resources. Given that the complexity of any given source code grows faster than the size of the source code, generating GraphQL resolvers would become more difficult even for a standard-sized dataset which typically contains more than a handful tables and hundreds of properties. This situation might worsen if the underlying dataset evolves, considering that the corresponding resolvers have to be updated as well. GraphQL resolvers may not be easily understood by other developers who were not involved in the initial version, thus bringing the possibility of introducing errors.

In this section, we propose the exploitation of declarative mapping languages to specify the rules that relate the source datasets and the GraphQL schema. Declarative mapping languages, such as the W3C R2RML (Das *et al.*, 2012a) and its extensions, have been used to generate knowledge graphs from existing datasets. The use of declarative mappings is based on the idea that a standard mapping language (or such extensions) would facilitate a better understanding of the relationships between the underlying data source and the exposed GraphQL schema. Furthermore, they also allow for better maintainability as those mappings are independent from any programming language. Our main contribution is an approach that translates declarative mappings to GraphQL resolvers. We focus on the feasibility of the approach and leave the soundness, completeness, and complexity analysis for future work.

6.2.1 The Morph-GraphQL framework

The Morph-GraphQL framework (Figure 6.16) proposes the exploitation of the information encoded in declarative mapping rules following a well-known specification (e.g. R2RML and RML) to generate GraphQL servers. A domain expert can create these types of mappings without the need for programming skills. Despite that the creation of mappings might not be easy for domain experts to be created from scratch, there are several tools with easy to use graphical interface that already developed by researchers in the semantic Web community such as RMLEditor (Heyvaert *et al.*, 2016) or KARMA (Knoblock & Szekely, 2015). The generated GraphQL servers benefit from the wide range of tools available for GraphQL in order to access data stored in various formats (i.e. RDB, CSV, JSON). The approach consists of the following steps: 1) the generation of the definition of a query to be evaluated by the underlying dataset (e.g. ListEpisodes), 2) the generation of the types in the GraphQL schema and 3) the generation of GraphQL resolvers.

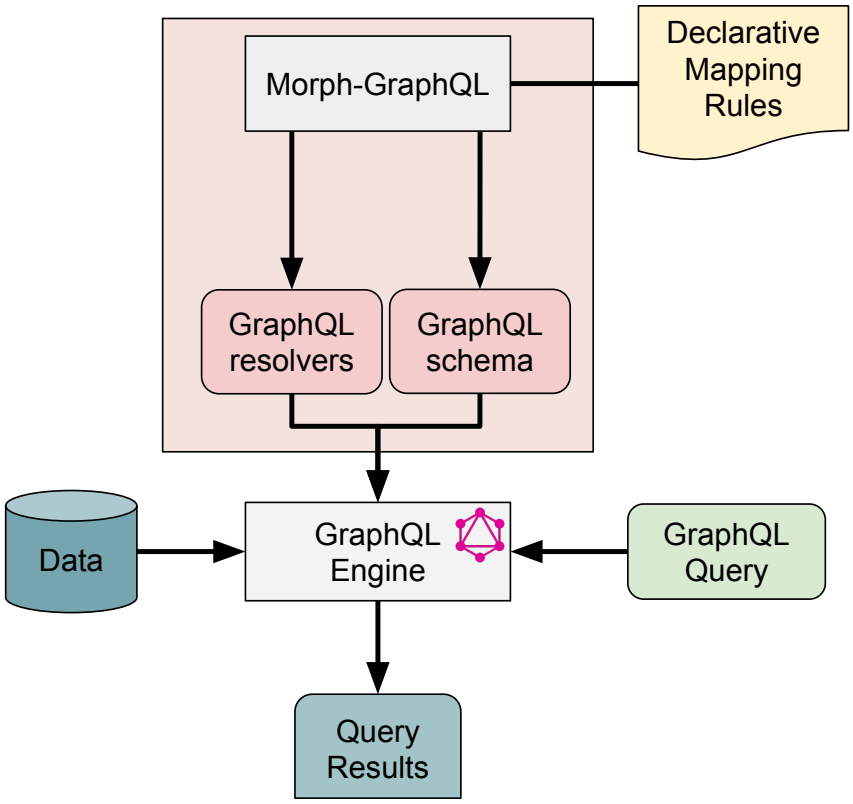


Figure 6.16: morph-GraphQL workflow. morph-GraphQL receives declarative mappings and generates GraphQL servers. These servers, as it is defined in the specification, contain their two main components (i.e. schema and resolvers) that can be used by any GraphQL engine to evaluate queries over the data source.

Auxiliary Functions. We present here a set of auxiliary functions that will be used in the functions that generate resolvers.

- $getConstant(TermMap)$ takes the constant `prefix:attr` in the constant-value term map where $TermMap =$ `rr:constant "prefix:attr"` and retrieves its specific value $attr$.
- $getReference(TermMap)$ retrieves the reference `ref` in the reference-value term map where $TermMap =$ `rr:column "ref"` or $TermMap =$ `rml:reference "ref"`.
- $getTemplate(TermMap)$ retrieves the template `template` in the template-value term map where $TermMap =$ `rr:template "template"`. In this case, the function retrieves the concatenation between the strings and the references that are part of the tem-

plate, hence, the implementation of this functions depend of the underlying query system used for retrieving the data. For example, given an SQL database and the term map `rr:template "ex.com/episode/{eid}"` as the inputs, this function returns `"ex.com/episode/" || eid` or `CONCAT("ex.com/episode/{eid}", eid)`.

- *getDataType*(*ObjectMap*) that given an `rr:ObjectMap` that contains a `rr:dataType "xsd:type"` returns the corresponding GraphQL type. For example, *getDataType*(*rr* : *dataType*"*xsd* : *string*") returns `String`.

- *getTypeFromClass*(*SubjectMap*) that given an `rr:SubjectMap` returns the type value based on the `rr:class` property. For example, *getDataType*(*rr* : *class*"*foaf* : *Person*") returns `Person`.

6.2.1.1 Generating Queries

We present a set of translation functions that convert a triples map into the corresponding query to be used in GraphQL resolvers. This set of functions is adapted from the work presented in (Chebotko *et al.*, 2009), originally proposed to translate SPARQL queries into SQL queries without the presence of any mappings.

- $\alpha(TriplesMap)$ returns a set of logical sources associated with the triples map $TriplesMap$, which is the logical source associated to the triples map $TriplesMap$ and additionally all the referenced source if $TriplesMap$ contains Referenced Object Maps.

- $\beta(TermMap)$ that given a term map $TermMap$ returns the corresponding query expression, that is:

  - $getConstant(TermMap)$ if $TermMap$ is a constant-value map
  - $getReference(TermMap)$ if $TermMap$ is a reference-value map
  - $getTemplate(TermMap)$ if $TermMap$ is a template-value map.

- $alias(TermMap)$ generates a unique alias to be used in the query generation.

- $genPR(TriplesMap)$ generates a query expression which projects the relevant query expressions of a triples map $TriplesMap$ (i.e., β of Subject Map and all Object Maps) together with their aliases.

- *genCond*($TriplesMap$) generates a query expression which is evaluated to true if they match the arguments passed in the resolver functions and additionally the join conditions if $TriplesMap$ contains Referenced Object Maps.

- Finally, $trans(TM)$ builds the valid query statement using the results of the previous functions. For example, in the case of an SQL database, $trans(TM)$ = "SELECT *genPR*(TM) FROM $\alpha(TM)$ WHERE *genCond*(TM)" translates a triples map into the corresponding SQL query.

6.2.1.2 Generating Schema

The generation of the Schema for GraphQL is divided into two steps. The first one is focused on the generation of the possible entry points of the server (i.e. the query root) and the second one generates the Types defined for the server. Exploiting the mapping rules as inputs, Morph-GraphQL automatically generates both components.

6.2.1.3 Generating Query Root

First, Morph-GraphQL generates the entry points of the server. Algorithm 1 shows how this step is executed. Taking as input a mapping document, it iterates over the TriplesMap and extracts the information needed for defining the entry points: the type value extracted from the class property of the subject map and the set of attributes together with the types extracted from the predicate-object maps. Morph-GraphQL is able to generate automatically a set of basic entry points (e.g. ListEpisodes, ListCharacter) that can be filtered by each attribute.

6.2.1.4 Generating Types

Algorithm 2 generates a GraphQL type from a Triples Map. It generates a GraphQL type *typeClass*, where *typeClass* is the class specified in the Subject Map of the Triples Map. The fields of the *typeClass* are all the mapped predicates in the Predicate Object Maps of the Triples Map. The datatypes of the fields are the results of function *getDataType*, which returns the corresponding GraphQL type from the datatype specified in the Object Maps of the Triples Map. This function is called for each TriplesMap defined in the mapping document.

Algorithm 1 GenerateQueryRoot(Mapping)

```
queryRoot.init()
for all TriplesMap ← Mapping do
    typeClass = getTypeFromClass(TriplesMap.getSubjectMap())
    poms = TriplesMap.getPredicateObjectMaps()
    for all pom ← poms do
        datatype = getDataType(pom.getObjectMap())
        attribute = getConstant(pom.getPredicateMap())
        attributes.add(attribute, datatype)
    end for
    query = createListQuery(typeClass, attributes)
    queryRoot.add(query)
end for
return queryRoot
```

Algorithm 2 GenerateType(TriplesMap)

```
type.init()
typeClass = getTypeFromClass(TriplesMap.getSubjectMap())
type.add(typeClass)
poms = TriplesMap.getPredicateObjectMaps()
for all pom ← poms do
    datatype = getDataType(pom.getObjectMap())
    attribute = getConstant(pom.getPredicateMap())
    type.add(createAtttribute(attribute, datatype))
end for
return type
```

6.2.1.5 Generating Resolvers

Algorithm 3 generates a GraphQL resolver from a TriplesMap. Based on the entry points defined in the schema, for each TripleMap, Morph-GraphQL generates the `list`*typeClass* resolver. First, it defines the function for querying the Type *typeClass* with the attributes defined in the mapping. Then, the algorithm use the function (*trans* function) to translate the query to the underlying query language adapting the approach defined in (Chebotko *et al.*, 2009). These two steps use the auxiliary functions $getRereference()$ and $getTemplate()$ in order to obtain the correct references of the data source columns/keys from the mapping rules. Finally, defines the manner how the engines has to executes the query on the underlying database engine and generates the corresponding instances by calling the constructor of Type *typeClass*.

Algorithm 3 GenerateResvolver(TriplesMap)

```
1: resolver.init()
2: typeClass = getTypeFromClass(TriplesMap.getSubjectMap())
3: poms = TriplesMap.getPredicateObjectMaps()
4: for all pom ← poms do
5:     attribute = getConstant(pom.getPredicateMap())
6:     attributes.add(attribute)
7: end for
8: resolver.add(defineListQueryFunction(typeClass, attributes))
9: resolver.add(translateQuery(trans(TriplesMap))
10: resolver.add(execute(query, rdb))
11: resolver.add(constructResults(attributes, queryResults))
12: return Resolver
```

6.2.2 Experimental Evaluation

In this section, we present the experimental evaluation of Morph-GraphQL. Our aim is to answer the following questions:

- **RQ1:** Can Morph-GraphQL generate a GraphQL server from declarative mappings that is able to answer the set of queries provided by a GraphQL benchmark?
- **RQ2:** Is there any significant difference between for the queries that can be answered by the generated GraphQL server in terms of response time between GraphQL queries

and their equivalent SPARQL queries posed over the RDF dataset generated by the same declarative mappings?

We have implemented our framework as an open-source project **Morph-GraphQL**[115,116]. In our previous work (Priyatna *et al.*, 2019) we described the full example of Star Wars generating a GraphQL server based on an R2RML mapping using Morph-GraphQL. Currently, the Morph-GraphQL framework is able to: translate R2RML mappings into a Javascript-based GraphQL server for accessing tabular datasets (CSV files or Relational Databases)(Priyatna *et al.*, 2019). We use the JoinMonster library[117] to generate efficient SQL queries when joins are needed.

6.2.2.1 Linköping GraphQL Benchmark

Currently, the only GraphQL benchmark available is the Linköping GraphQL Benchmark (LinGB)[118]. This benchmark focuses on exposing read-only GraphQL APIs over a legacy relational database. At the time of writing, the LinGBM benchmark v1.0 sets its context in the domain of e-commerce. It consists of a dataset generator and a set of query templates (a query with placeholder variables to be instantiated). Additionally, guidelines are provided on the mapping between the relational database schema and the GraphQL schema (e.g. Table Offer is mapped to GraphQL type Offer).

Dataset Generator. The dataset generator[119] is based on the Berlin SPARQL Benchmark (BSBM) (Bizer & Schultz, 2009). The dataset contains ten tables (e.g. Vendor, Offer, Producer, Product, and Person, Review) with different join cardinalities (e.g. 1-1, 1-N, M-N).

Choke-points and Queries. The benchmark includes a list of *choke-points*, which are challenges that have been identified for answering GraphQL queries. This is done following the design methodology for benchmark development[120] introduced by the Linked Data Benchmark Council[121]. Five classes of choke-points that are proposed are:

1. Choke Points Related to Attribute Retrieval. (1 check-point)

[115] `https://github.com/oeg-upm/morph-graphql`
[116] `https://doi.org/10.5281/zenodo.3584339`
[117] `https://join-monster.readthedocs.io`
[118] `https://github.com/LiUGraphQL/LinGBM/wiki`
[119] `https://github.com/LiUGraphQL/LinGBM/wiki/Datasets`
[120] `http://ldbcouncil.org/blog/choke-point-based-benchmark-design`
[121] `http://ldbcouncil.org/`

2. Choke Points Related to Relationship Traversal. (5 choke-points)

3. Choke Points Related to Ordering and Paging. (3 choke-points)

4. Choke Points Related to Searching and Filtering. (5 choke-points)

5. Choke Points Related to Aggregation. (2 choke-points)

These choke-points are covered in the 16 hand-crafted query templates provided by the benchmark. The summarise of the queries, the relation with the proposed choke-points and the support of Morph-GraphQL is show in Table 6.3.

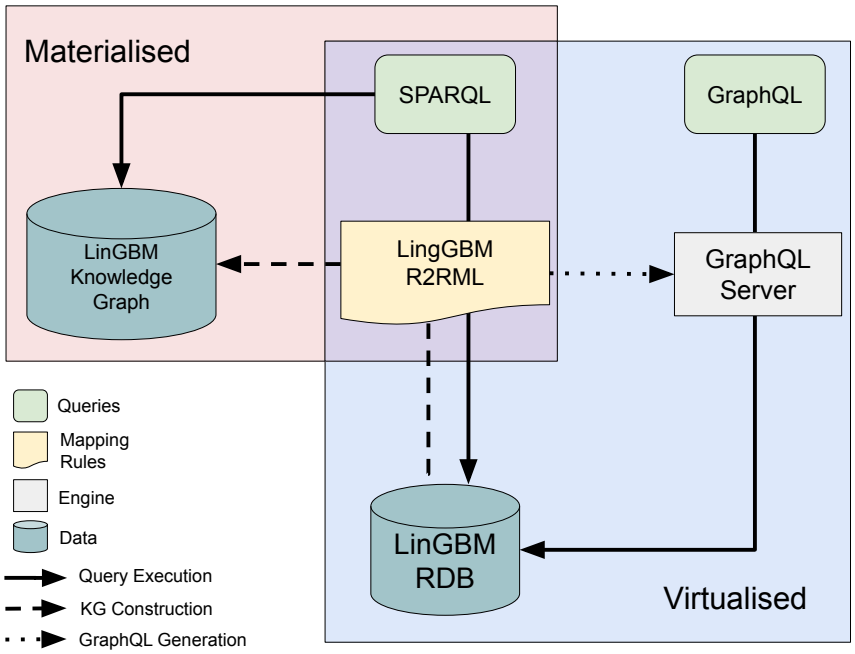


Figure 6.17: Evaluation Workflow. We evaluated Morph-GraphQL by comparing its performance over the supported LinGBM queries againts two equivalent Semantic Web approaches. First one is the materialisation of the LinGBM RDB to RDF using R2RML mappings and the second one is the translation from SPARQL-to-SQL using also the the same mapping rules.

6.2.2.2 Evaluation Setup

We used the LinGBM Data Generator to generate the various sizes of datasets[122] (1K, 2K, 4K, 8K, 16K, 32K, 64K and 128K) and loaded them in the relational database. We created

[122]The size is defined in terms of number of products in the database (1K means 1 thousand products)

Query	Choke Points	Morph-GraphQL Support
Q1	1.1, 2.1, 2.2	Yes
Q2	2.1	Yes
Q3	2.2, 2.3	Yes
Q4	2.2, 2.3, 2.5	Yes
Q5	2.1, 2.2, 2.3, 2.4	Yes
Q6	2.2, 2.5	Yes
Q7	2.2, 2.5, 3.2	No
Q8	3.1, 3.3	No
Q9	2.1, 2.2, 3.1, 3.3	No
Q10	1.1, 4.1	No
Q11	2.5, 4.4	No
Q12	2.5, 4.3	No
Q13	2.1, 4.2, 4.3	No
Q14	2.1, 4.2, 4.3, 4.5	No
Q15	5.2	No
Q16	5.1	No

Table 6.3: Supported Queries in Morph-GraphQL

declarative mappings following the mapping guidelines provided in the benchmark. For each query template, we generated 20 queries with different instances generated randomly, as it is the default settings of the benchmark. To measure the performance, we run each query instance 5 times in cold mode, and we calculate the average for each query. We also generated the equivalent SPARQL queries that are to be evaluated over a knowledge graph. The knowledge graph is generated by Morph-RDB (Priyatna *et al.*, 2014) from the datasets using the same declarative mappings. All the resources used in the evaluation is available online on our GitHub repository[123]. Figure 6.17 illustrates the evaluation workflow explained above. The figure shows a materialised and virtualised knowledge graphs. The materialised knowledge graph is the data transformed to RDF and stored into virtuoso "LinGBM Knowledge Graph" (a triple store to store knowledge graph data). The virtualised view does not transform the original datasets to another format (the data is stored in a RDB). It provides access to the underlying datasets via GraphQL. Once a GraphQL query is received, the GraphQL server uses GraphQL

[123] `https://github.com/oeg-upm/morph-graphql/`

engine to translate the query into SQL, which would query the Database "LinGBM RDB", get the results, and then the results of the SQL query will be changed into the requested format according to the GraphQL query. We measure the total query execution time for:

- GraphQL queries that are translated into SQL queries and evaluated in a relational database instance database. We use Morph-GraphQL v1.0.0 to evaluate these queries.
- SPARQL queries that are evaluated over the materialised knowledge graph that is, queries are evaluated using a triple store. We use a Virtuoso v7.2.5.1 instance in this case.
- SPARQL queries that are evaluated over the virtual knowledge graph that is, queries are translated into SQL queries and evaluated by Morph-RDB v3.9.15 over the relational database instance.

The experiments were run in an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60GHz 20 cores, 100G memory with Ubuntu 16.04LTS.

6.2.2.3 Results and Discussion

In terms of choke-points, Morph-GraphQL supported queries that belong solely to two classes: *attribute retrieval* and *relationship traversal*. Queries which belong to the classes *ordering-and-paging* and *searching-and-filtering* are not supported by Morph-GraphQL. Nonetheless, this can be addressed in a future version of Morph-GraphQL. The last class of chock points addresses *aggregations*, which has not been addressed yet in the GraphQL specification.

We show the results for the different dataset sizes in Table 6.4. We see that for smaller dataset sizes (1K,2K, and 4K), Morph-GraphQL outperforms the others for the majority of the queries. The main reason of these results for smaller dataset sizes is because of the overhead (for translating queries from SPARQL to SQL and optimising the resulting SQL) in Morph-RDB has a negative impact in the total performance of the query execution. Morph-GraphQL needs less time translating the query because it is relatively more simple to translate GraphQL to SQL compared to SPARQL to SQL. In most of the cases for these dataset sizes, Virtuoso needs more time than the other two systems due to the absence of indexes in RDF which has a negative impact depending on the features of the query (Endris *et al.*, 2019). For bigger datasets, SPARQL to SQL optimisations (Priyatna *et al.*, 2014) implemented in Morph-RDB pays off the translation time and gives better impact over the query execution process, outperforming Morph-GraphQL, which hints that the optimisations in the query translation from

Engine/Queries	Q1	Q2	Q3	Q4	Q5	Q6	Geometric Mean
LinkGBM 1K							
Morph-GraphQL	0.103	0.146	0.005	0.118	0.237	0.079	0.075
Morph-RDB	0.168	0.081	0.201	0.221	0.296	0.089	0.159
Virtuoso	0.182	0.017	0.091	0.079	1.204	0.131	0.124
LinkGBM 2K							
Morph-GraphQL	0.183	0.200	0.006	0.171	0.397	0.071	0.100
Morph-RDB	0.167	0.085	0.203	0.224	0.308	0.088	0.161
Virtuoso	0.096	0.025	0.057	0.068	1.291	0.073	0.098
LinkGBM 4K							
Morph-GraphQL	0.314	0.316	0.005	0.311	0.799	0.096	0.151
Morph-RDB	0.171	0.083	0.199	0.223	0.318	0.089	0.161
Virtuoso	0.109	0.035	0.059	0.078	12.293	0.101	0.167
LinkGBM 8K							
Morph-GraphQL	0.597	0.644	0.004	0.625	1.363	0.096	0.228
Morph-RDB	0.178	0.080	0.196	0.225	0.323	0.090	0.162
Virtuoso	0.096	0.070	0.064	0.069	2.142	0.097	0.135
LinkGBM 16K							
Morph-GraphQL	1.121	1.408	0.005	1.293	2.776	0.104	0.376
Morph-RDB	0.167	0.083	0.205	0.222	0.322	0.089	0.162
Virtuoso	0.100	0.122	0.057	0.073	1.412	0.090	0.137
LinkGBM 32K							
Morph-GraphQL	2.635	2.884	0.005	2.543	6.086	0.130	0.644
Morph-RDB	0.173	0.085	0.199	0.220	0.323	0.089	0.163
Virtuoso	0.108	0.274	0.069	0.085	1.591	0.122	0.179
LinkGBM 64K							
Morph-GraphQL	5.157	5.940	0.005	5.114	11.065	0.147	1.050
Morph-RDB	0.177	0.085	0.199	0.224	0.325	0.090	0.164
Virtuoso	0.116	0.417	0.057	0.091	1.666	0.102	0.187
LinkGBM 128K							
Morph-GraphQL	8.806	9.552	0.005	8.437	22.453	0.152	1.526
Morph-RDB	0.172	0.084	0.199	0.224	0.324	0.090	0.163
Virtuoso	0.120	0.381	0.058	0.090	1.613	0.115	0.188

Table 6.4: Query evaluation performance (time in seconds) over multiple sizes of the LinGBM (the number indicates the scale factor used). Execution time is a lower-is-better metric.

GraphQL to SQL can still be improved in order to query big datasets (32K, 64K, 128K). When we consider the whole set of queries and calculate the geometric mean of the results, we notice that Morph-GraphQL outperforms the others in some of the dataset sizes because of its fast execution time for the queries Q3 and Q6. Analysing the queries individually, we can observe that for all the engines, Q5 is the most costly one due to the number of nested queries that it consists.

6.2.3 Conclusions

We have presented an approach to generate GraphQL resolvers from R2RML mappings together with its corresponding implementation, Morph-GraphQL. Note that we do not aim to replace the traditional approach of generating GraphQL schema/resolvers manually, but we position this approach as a supplementary approach. This is to say that this approach allows domain experts to use the generated schema and resolvers as the initial proof of concept that can be used to query datasets without the need for software engineers to develop a full-fledged GraphQL server. Software engineers may also benefit from our approach as they may also use Morph-GraphQL to generate the initial version of a GraphQL server instead of building it from scratch.

Chapter 7

Optimizations for Scaling-up Knowledge Graph Materialization Techniques

In this Chapter we introduce two approaches to optimize the construction of materialized knowledge graphs. Diverse approaches have been proposed to define the process of integrating heterogeneous datasets into knowledge graphs (Calvanese *et al.*, 2017; Chaves-Fraga *et al.*, 2019; Chebotko *et al.*, 2009; Priyatna *et al.*, 2014). Mapping languages (e.g., R2RML (Das *et al.*, 2012a) and RML (Dimou *et al.*, 2014)) and engines (e.g., RMLMapper[124] and RocketRML (Şimşek *et al.*, 2019)) represent valuable contributions for performing this transformation process. Albeit highly used, existing approaches lack efficient data management techniques demanded to create knowledge graphs from large and heterogeneous datasets with duplicates. In the same manner as the approaches proposed in Chapter 6, the two contributions described exploit the information from mapping rules and ideas presented in Chapter 4 to scale-up the construction of KG.

Section 7.1 presents an efficient set of data structures and their corresponding physical operator to scaling up the construction of materialized KGs where the mappings are following the RML specification. We empirically demonstrate the efficiency of the proposal and compare the obtained results againts state of the art RML engines such as RocketRML y RMLMapper. Section 7.2 describes FunMap, a set of optimization techniques for efficiently pre-processing

[124] `https://github.com/RMLio/rmlmapper-java`

functional mappings (mappings such as RML+FnO, that include transformation functions) and generation of function-free RML mappings that can be used by any RML parser.

7.1 Efficient RML parsing for KG Materialization at Scale

Knowledge graphs have gained momentum as data structures to integrate–as factual statements–data and knowledge present in heterogeneous data sources. DBpedia and Wikidata are exemplary encyclopedic knowledge graphs frequently accessed by scientific and industrial communities; e.g., only Wikidata receives billions of visits per month[125]. Similarly, knowledge graphs are receiving significant attention in science and industrial developments (Auer *et al.*, 2018; Noy *et al.*, 2019). In fact, according to Google, knowledge graph is a trend term[126] and the Google Scholar indexes more than 3,5M entries of scientific publications with the term knowledge graph. Although results demonstrate the success in the adoption of Semantic Web technologies, put in perspective the need of providing efficient and mature technologies for constructing and maintaining knowledge graphs.

Motivating Example: Creating a knowledge graph from biomedical data sources is an exemplary scenario of being overwhelmed by the volume and heterogeneity of data. In Figure 7.1, we see a normal process of transforming two real-world data sources into an RDF knowledge graph using an available RML interpreter. In this example, the aim is to integrate data related to the biological concept RBP_RNA_PhysicalInteraction[127] from different sources into RDF. Accordingly, the subject of `TripleMap1` represents the mentioned concept. Considering the fact that related data is residing in two different sources, a *Join Condition* is applied in the mapping rules to create the required triples. It should be noted that even though only four different attributes of both data sources are utilized, the data volumes are considerably large; about 1 Gigabyte in total. In this example, to transform the raw data into RDF, two widely accepted RML-compliant interpreters[128], i.e., RMLMapper[129] and RocketRML (Şimşek *et al.*, 2019) are executed. However, none of the mentioned engines accomplish the task. In the case of RocketRML, the process stops early due to failure of the memory capacity. While applying

[125] `https://stats.wikimedia.org/`

[126] `https://trends.google.com/trends/explore?q=knowledge%20graph`

[127] Protein(RBP)-RNA binding interactions are shown to play important roles in diseases. Although there is a lack of enough experimental data, various computational methods are filling this gap by predicting physical interactions between RBP and target RNAs.

[128] `https://github.com/RMLio/rml-implementation-report`

[129] `https://github.com/RMLio/rmlmapper-java`

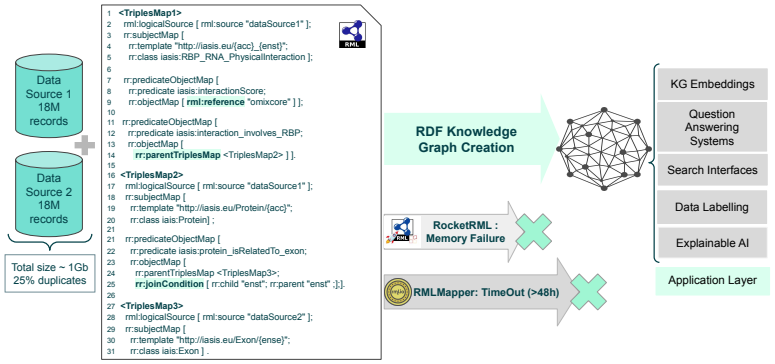


Figure 7.1: Motivating example. Available RML engines, implementing naive join strategy, fail to create a KG from two biomedical datasets with a total size of 1GB and 25% duplicates.

RMLMapper, the transformation process times out after 48 hours. The inability of available engines to perform in this scenario demonstrates the need to develop engines able to perform complex transformations in an efficient manner.

Our Resource: We address the problem of efficient knowledge graph creation, and propose a resource named SDM-RDFizer which is able to transform data from myriad data sources into an RDF knowledge graph. SDM-RDFizer implements a set of unique physical operators and data structures that speed up the execution of the mapping rules that specify a knowledge graph creation process. The current version of SDM-RDFizer is customized for RML, a mapping language extensively used for the creation of knowledge graphs in diverse domains (Dimou *et al.*, 2014). SDM-RDFizer is publicly available as a resource in a Github[130] and in Zenodo[131]. SDM-RDFizer is used in more than eight international projects. Moreover, experimental results reveal the contribution that SDM-RDFizer makes to the repertoire of efficient technologies for knowledge graph management.

7.1.1 The SDM-RDFizer: An RML Engine

This section describes the SDM-RDFizer in terms of its architecture, the physical operators that make up the execution engine of RML triples maps, and the main properties of these operators.

[130] `https://github.com/SDM-TIB/SDM-RDFizer`
[131] `https://doi.org/10.5281/zenodo.3872103`

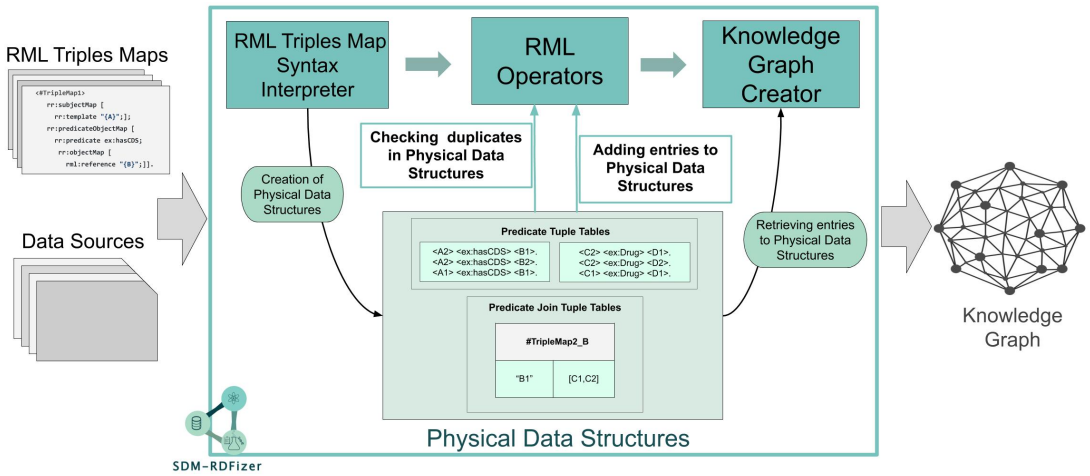


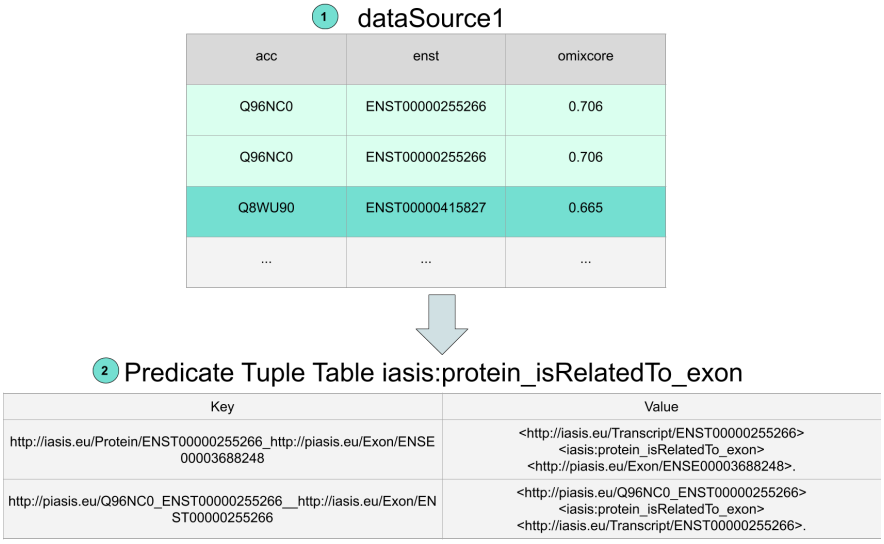
Figure 7.2: The architecture of the SDM-RDFizer.

7.1.1.1 The SDM-RDFizer Architecture

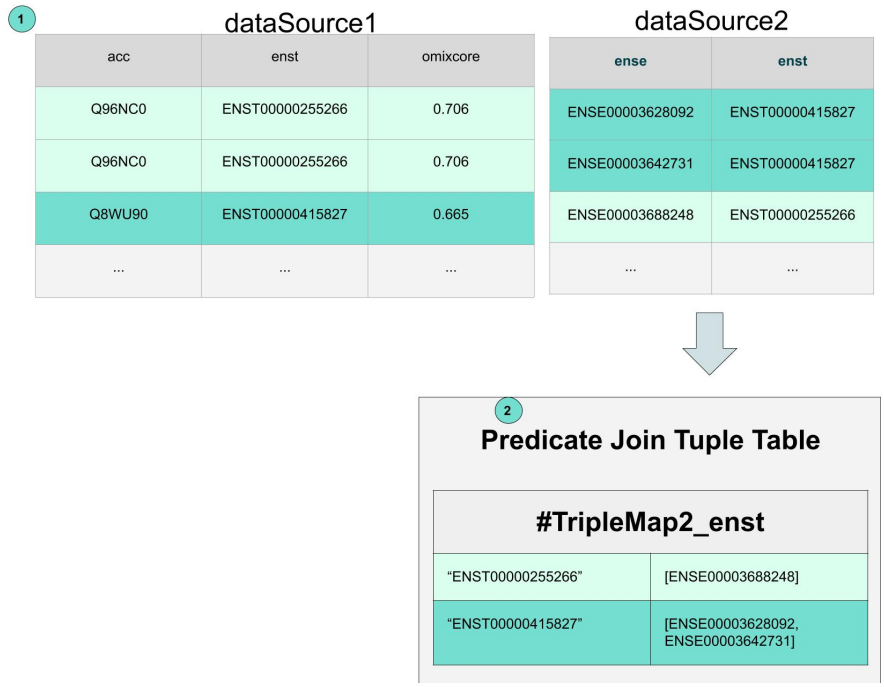
A data integration system *DI* provides an abstract representation $\langle O, S, M \rangle$ to specify the mapping rules in a set M that define the integration of a set S of data sources into instances of a unified schema or ontology O. SDM-RDFizer receives as input a data integration system *DI* and produces as output instances of the concepts in O that result from the execution of the mapping rules in M over the data sources in S. The current version of SDM-RDFizer is customized for interpreting data integration systems where mapping rules are specified in RML, and the output corresponds to an RDF knowledge graph with the ontology O. However, the SDM-RDFizer architecture can be easily implemented for other RDF based mapping languages (e.g., R2RML (Das *et al.*, 2012a)). The execution of the RML triples maps requires the interpretation of triples maps, the creation of physical data structures to store the results of the execution of the RML rules, and the generation of the knowledge graph from the results stored in the data structures. Figure 7.2 depicts SDM-RDFizer in terms of four main components that implement these steps.

RML Triples Map Syntax Interpreter: translates the RML triples maps into the physical data structures that SDM-RDFizer uses to execute the RML triples maps and generate the RDF triples.

RML Operators: execute the interpreted triples maps over the respective data sources to generate RDF triples. During the execution of these operators, the physical data structures are accessed to check if an RDF triple has already been created. If so, the generation of a



(a) Predicate Tuple Table



(b) Predicate Join Tuple Table

Figure 7.3: The Physical Data Structures. The two physical data structures used by SDM-RDFizer are illustrated. (a) A Predicate Tuple Table with three entries. (b) A Predicate Join Tuple Table with two entries.

duplicated RDF triple is avoided, otherwise, the triple is stored in the physical data structure. SDM-RDFizer has three operators; they are explained in more detail in subsubsection 7.1.1.3.

- *Simple Object Map*: is the most basic of the operators to evaluate a *simple predicate object map* statement in an RML triples map. The values of the *object values* are collected from an attribute in the triples map source or are a constant. In the motivating example, this operator generates RDF triples according to the predicate object map in lines 7-9 in Figure 7.1.

- *Object Reference Map*: this operator *references a second triples map*. The object of the first triples map is the subject of the second triples map. The main condition for this operator to work is that both triples maps have the same data source. An application of this operator in the motivating example can be seen in lines 11-14.

- *Object Join Map*: this operator executes a *join condition* between two RML triples maps with different data sources. In the motivating example, this operator is utilized to execute the predicate object map in lines 21-25 in Figure 7.1.

Physical Data Structures: store results generated so far and avoid the generation of duplicates during the execution of RML triples maps. They are of two types:

- i) Predicate Tuple Table (PTT): to store per each of predicate p in at least one triple map, the RDF triples generated for p so far.

- ii) Predicate Join Tuple Table (PJTT): to store the values of the subjects generated by a triples map that are associated with the values that meet a join condition in the triples map.

These structures are explained in more detail in subsubsection 7.1.1.2.

Knowledge Graph Creator: collects RDF triples stored in PTTs and adds them to the output knowledge graph. The knowledge graph creation is performed incrementally, i.e., as soon as a new RDF triple is added into a PTT, the RDF triple is also included in the knowledge graph. To avoid the same RDF triple to be added more than once, the knowledge graph creator maintains per PTT t, the timestamp of the last RDF triple that was selected from t.

7.1.1.2 Physical Data Structures

The SDM-RDFizer utilizes two physical data structures as a means to optimize the creation of knowledge graphs. These data structures help with the removal of duplicates and to avoid unnecessary operations, like uploading the parent triples map's data source of a join multiple times. In the following subsection, the physical data structures used by SDM-RDFizer are described.

Predicate Tuple Table (PTT) For each predicate p defined in an object triples map, a PTT is created to store the RDF triples generated so far. Physically, PTTs are implemented as hash tables where the hash key of an entry corresponds to an encoding of the subject and object of a generated RDF triple, and the value of the entry corresponds to the RDF triple. The main use of this table is to avoid the duplicate generation of an RDF triple. If a generated RDF triple is present within PTT, that means that the triple has been previously generated, and it needs to be discarded. But if the generated RDF triple is not present within PTT, then it is new and must be added to PTT and to the knowledge graph. As it can be seen in the figure 7.3a, the data source is transformed into RDF triples which checked in the corresponding PTT. As RDF triples of a predicate p can be generated from the execution of different triples maps, PTTs bring great savings not only in sources with high-duplicated rates but also when data sources that generate RDF triples of p also overlap.

Predicate Join Tuple Table (PJTT) A PJTT stores the values generated during the execution of a join condition between two RML triple maps, e.g., lines 21-25 in Figure 7.1, the predicate object map is defined in terms of a join of triples map `TriplesMap1` (child map) to `TriplesMap2` (parent map). For each RML triples map M_i that is referred as a parent triples map in a join condition B, a predicate join tuple table M_i_B is created, e.g., `TriplesMap2_enst` in our running example. Physically, predicate join tuples are hash tables. The hash key of an entry corresponds to the encoding of each of the values of the attributes in the condition B (e.g., `enst`), while the value of the entry is a set with all the subject values generated by M_i (e.g., values of the subject of `TriplesMap2`) that are associated with the values of the attributes in B represented in the entry hash key. Additionally, a PJTT enables direct access to the subjects associated with the join condition B, allowing thus for the join implementation as an index join. In the example shown in Figure 7.3b, "enst" is the join condition between the triples maps. The data is organized as the values of the join conditions with its respective value in dataSource2. For example, we have the value "ENST00000415827" and its associated values

”ENSE00003628092” and ”ENSE00003642731”. In PJTT, ”ENST00000415827” is the key in the hash table and ”ENSE00003628092” and ”ENSE00003642731” are the values. Finally, to identify an entry in PJTT, a key is generated from the identifier of the parent triples map and the join condition.

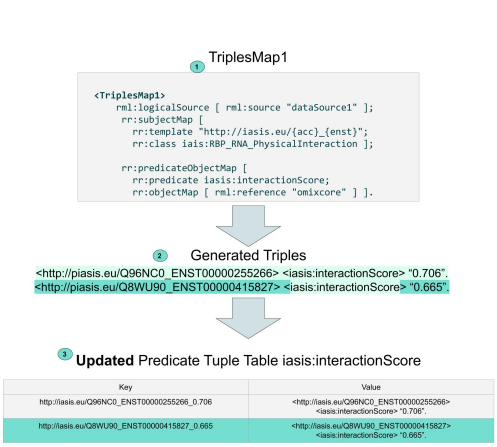
7.1.1.3 RML Operators and Algorithms

SDM-RDFizer implements three different operators for the creation of knowledge graphs. Depending on the type of the triples map, the SDM-RDFizer executes the respective operator. If the triples map has a join condition, then an **Object Join Map** operator is used. If the triples map have a reference to another triples map but do not have a join condition, then the **Object Reference Map** operator is used. Finally, if the triples map does not have a join condition or a reference to another triples map, then the **Simple Object Map** is used. We now describe the operators in more detail.

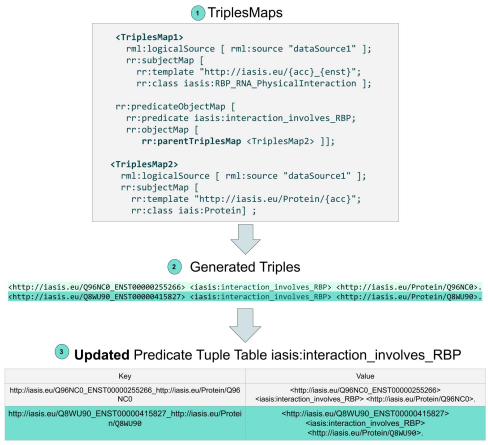
Simple Object Map (SOM) It is the most basic operator that SDM-RDFizer can execute and enables the generation of an RDF triple by executing a simple predicate object map statement. As it is illustrated in Figure 7.4a, given a triples map and its respective data source, SDM-RDFizer generates RDF triples following what is established in the map. Each generated RDF triple is checked against the corresponding predicate tuple table (**PTT**). If the generated RDF triple already exists in PTT, then it is discarded. In the opposite case, the RDF triple is added both to PTT and to the knowledge graph. This operation is depicted in Figure 7.4a where two RDF triples are generated. ”¡http://iasis.eu/Q8WU90_ENST00000415827¿ ¡iasis:interactionScore¿ “0.665”.” is not in PTT, then, it is added both to the table and to the knowledge graph.

Object Reference Map (ORM) It seeks to expand what is established in Simple Object Map. By using the subject of a triples map as the object of another triples map. The main condition for this operator to work is that both triples maps have the same data source. Afterwards, the same process as in Simple Object Map is applied on the generated RDF triples, i.e., the triples are checked against PTT to determine if the triples are required for the knowledge graph creation. An example of this operation is in Figure 7.4b. In the figure, there are two triples maps, where the *¡TripleMap2¿* acts as the parent triples map. Two RDF triples are generated but only the new one is included in the PTT.

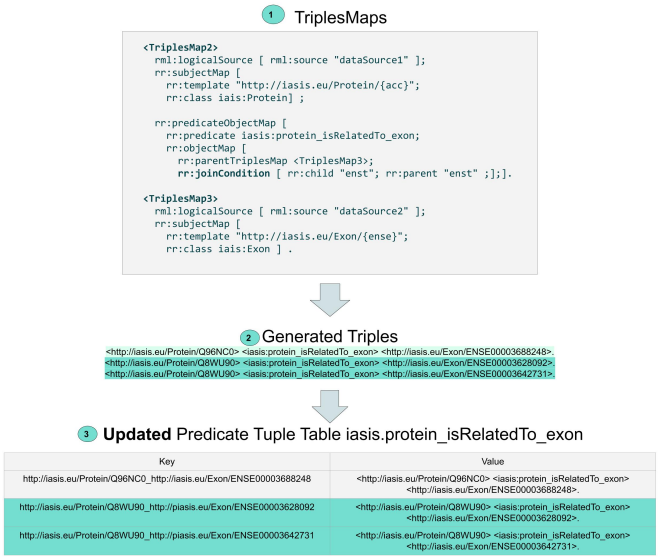
Object Join Map (OJM) It seeks to expand what is established in Object Reference Map, but the main difference is that triples maps have different data sources. By using the corresponding



(a) Simple Object Map



(b) Object Reference Map



(c) Object Join Map

Figure 7.4: SDM-RDFizer implements three physical RML operators that rely on PTTs to avoid the generation of duplicates. Object Join Maps resort to PJTTs to provide a direct access to the inner tables (i.e., the parent triples maps) of a join between two triples maps; also, PJTTs avoid the traversal of a parent triples map in case it is referenced by more than one triples map.

PJTT, SDM-RDFizer implements an index join where the outer table of the join corresponds to the values in the child map, and the inner table to the PJTT. Thus, to validate the satisfaction of a join condition B, the value of B is checked in PJTT and if an entry e exists with that hash key, all the subjects in e are used to generate the resulting RDF triples. Finally, similar to the last two operations, the generated RDF triples are checked against the corresponding PTT to validate duplication and decide if they are going to be included in the knowledge graph. A way to better understand this operation is to view Figure 7.4c. In the figure, the join condition is the column "enst" in both data sources. A PJTT table is created from the data associated with the join condition as shown in Figure 7.3b. Three RDF triples are generated and only two are not duplicates (i.e., they are not in the PTT). This operation is similar to Object Reference Map, since the object of the triples is the subject of the parent triples map.

7.1.1.4 Properties

We present the main properties of the RML operators implemented by SDM-RDFizer. Per operator $\mathtt{o}$, we seek to compare the number of operations done by SDM-RDFizer versus the ones done by a naïve implementation of $\mathtt{o}$; we named these expressions $\phi_{\mathtt{o}}(.)$ and $\widehat{\phi}_{\mathtt{o}}(.)$, respectively. Without lost of generality, we just focus on main-memory operations per operator, i.e., comparisons and insertions in main-memory data structures. Consider a predicate p, a multiset N_p, and set S_p; N_p includes all the RDF triples of p while S_p is the corresponding set of N_p. Consider $|N_p|$ and $|S_p|$ as the cardinality of N_p and S_p, respectively. In presence of a high-duplicate rate of RDF triples of p, $|S_p|$ is much smaller than $|N_p|$ (i.e., $|S_p| \ll |N_p|$).

- **Simple Object Map (SOM):** Let M be an RML triples map with an object triples map that defines p, $\phi_{\mathtt{o}}(M)$ and $\widehat{\phi}_{\mathtt{o}}(M)$ are defined as follows. The naïve implementation of the simple object map operator o in M generates all the duplicates and then, it needs to execute a duplicate elimination process to add the RDF triples to the knowledge graph. Suppose a merge sort algorithm is conducted to eliminate duplicates (Bitton & DeWitt, 1983)[132], then the following number of operations are required:

$$\widehat{\phi}_{\mathtt{o}}(M) = |N_p| + |S_p| + \Theta(N_p log(N_p))$$

[132] $\Theta(.)$ corresponds to the asymptotic notation

Contrary, the SDM-RDFizer algorithm of a simple object map resorts to a PTT of p and never generates duplicates. As a result, the number of operations is defined as follows:

$$\phi_{\circ}(M) = |N_p| + 2|S_p|$$

- **Object Reference Map (ORM):** This operator requires to define p, a reference of M to a parent triple map M_i expressed over the same data source s of M. That is, the operator corresponds to a self-join over s with a natural join condition on the attribute(s) that corresponds to the subject of M. As in a natural join, the join condition is not required. Assume $\Theta(1)$ is the cost of accessing the value of the subject of M_i when M is executed, then the number of operations is the same as executing a simple object map, i.e.,

$$\widehat{\phi_{\circ}}(M) = |N_p| + |S_p| + \Theta(N_p log(N_p))$$

$$\phi_{\circ}(M) = |N_p| + 2|S_p|$$

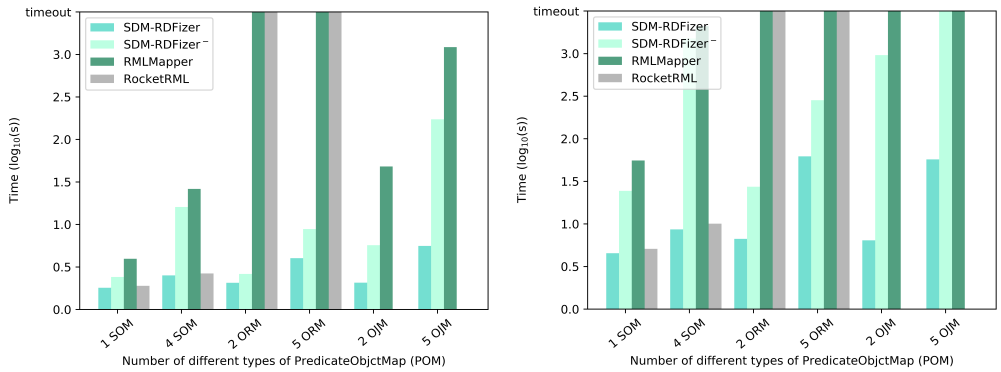
- **Object Join Map (OJM):** An Object Join Map executes a join between the data source of a child triple map M and the data source of a parent triple map M_i on a join condition B. In this case, $|N_p|$ represents the number of RDF triples resulting of evaluating the join and $|S_p|$ the number of duplicate-free RDF triples in N_p. Further, assume $|N_{parent}|$ and $|N_{child}|$ are the number of rows in the parent and child maps, respectively, to check to validate the join condition. If the naïve approach follows a nested loop join (Steinbrunn *et al.*, 1997), then

$$\widehat{\phi_{\circ}}(M) = |N_{parent}| \times |N_{child}| + |N_p| + |S_p| + \Theta(N_p log(N_p))$$

Contrary, SDM-RDFizer relies on the PJTT M_i_B (of size N_{parent}[133]) and the PTT of p to implement an index join that produces duplicate-free RDF triples. Thus, both physical data structures enable an efficient implementation of OJM. As a result, the number of operations is as follows:

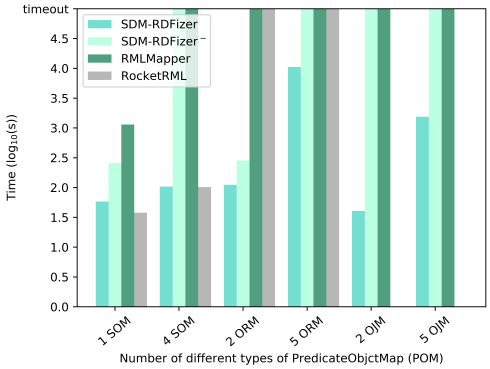
$$\widehat{\phi_{\circ}}(M) = 2|N_{parent}| + |N_{child}| + |N_p| + 2|S_p|$$

[133] We assume that a PJTT creation costs N_{parent} main-memory operations.



(a) 10k records

(b) 100k records



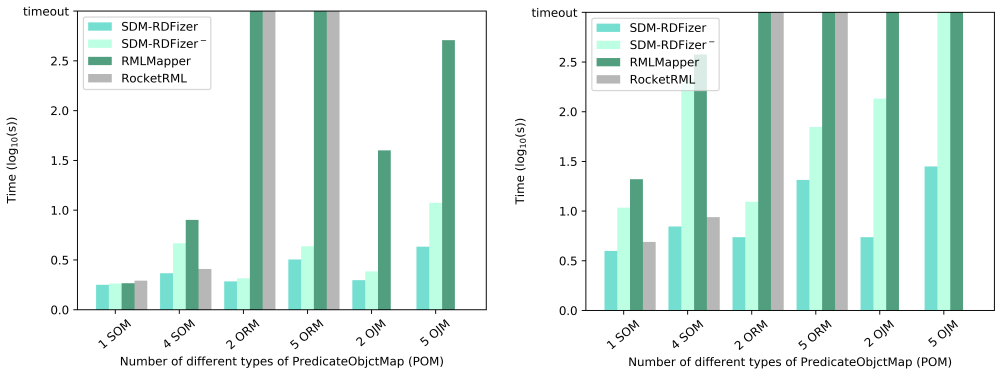
(c) 1M records

Figure 7.5: Total execution time of experiments on datasets with 25% duplicates. SOM means simple object map, ORM object reference map and OJM object join map. RocketRML generates incorrect results running OJM mappings.

7.1.2 SDM-RDFizer as a Resource

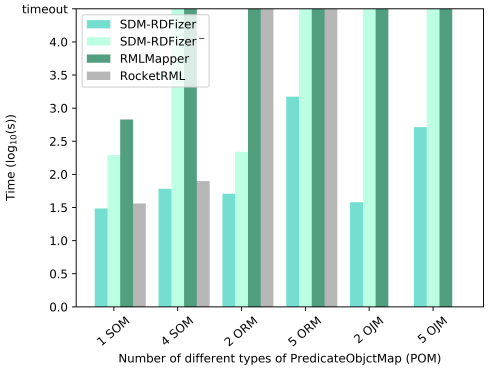
Novelty: SDM-RDFizer introduces a novel set of operators to execute mapping rules in a data integration system; they allow for an efficient creation of knowledge graphs from heterogeneous data sources. Although the current version of SDM-RDFizer is customized for RML, the set of operators can be easily extended for other mapping rule languages and data models to represent knowledge graphs. Results of the experimental studies comparing the performance of SDM-RDFizer illustrate the novelty of the proposed work with the state of the art. We hope that these results encourage the community to advance existing approaches to scale up to the avalanche of available data that is expected in the next years.

Availability: SDM-RDFizer is released publicly by the Scientific Data Management (SDM)



(a) 10k records

(b) 100k records



(c) 1M records

Figure 7.6: Total execution time of experiments on datasets with 75% duplicates. SOM means simple object map, ORM object reference map and OJM object join map. RocketRML generates incorrect results running OJM mappings.

group at TIB, Hannover[134]. TIB is one of the largest libraries for science and technology in the world, and following its policy of engaging open access to scientific artifacts, will keep available SDM-RDFizer as a tool for supporting the creation of knowledge graphs. The SDM-RDFizer is open source, written in Python 3, and available under the Apache License 2.0 license in an open Github repository[135]; it is regularly updated with new features. Additionally, following open science good practices, we register the tool at the Zenodo platform, which takes the Github repository and gives a general DOI[136] to the engine and also a DOI for each release

[134] `https://www.tib.eu/en/research-development/scientific-data-management/`
[135] `https://github.com/SDM-TIB/SDM-RDFizer`
[136] `https://doi.org/10.5281/zenodo.3872103`

of the code[137]. Thus, users and practitioners can use and cite a specific version of the engine, ensuring reproducibility and traceability of any experimental evaluation.

Utility: A docker image of SDM-RDFizer is available at DockerHub[138] and the Github repository of the resource, provides a detailed explanation of how to create and run the Docker container. The use case presented in the motivating example is utilized to facilitate the understanding. Furthermore, the activity of commits in the Github repository evidence the attention paid to the creation of new versions, as well as to the resolution of the issues identified by the users of the tool.

Predicted Impact: The number of visits of knowledge graphs like DBpedia and Wikidata, and the current developments in scientific (e.g., (Auer *et al.*, 2018)) and industrial areas (e.g., (Noy *et al.*, 2019)) evidence the need of providing efficient tools for knowledge graph management at scale. Results of experimental evaluations of SDM-RDFizer illustrate the benefits of grounding solutions for the problem of knowledge graph creation in the well-established areas of data integration systems and query processing. Thus, we ambition that they will be the starting point of future developments, e.g., for the optimization and distribution of mapping rule executions, as well as for semantically enriching data integration systems whose execution enable the explainability of the whole process of knowledge graph creation.

Adoption and Reusability: Several projects from different domains already use SDM-RDFizer. iASiS[139] and BigMedilytics - lung cancer pilot [140] are exemplary of EU H2020 projects. The iASiS RDF knowledge graph comprises more than 1.2B RDF triples collected from more than 40 heterogeneous sources using more than 1300 RML triple maps, while 800 RML triple maps are used to create from 25 data sources, a lung cancer knowledge graph with 500M RDF triples. SDM-RDFizer has also created the *Knowledge4COVID-19* knowledge graph during the EUvsVirus Hackathon[141]; it comprises 28M RDF triples describing 63527 COVID-19 articles and related COVID-19 concepts (e.g., drug-drug interactions and molecular disfunctions). SDM-RDFizer is also used in EU H2020, EIT-Digital, and Spanish national projects where the Ontology Engineering Group (Technical University of Madrid) participates. These projects, mainly focused on the transportation and smart cities domain, include: SPRINT[142], SNAP [143]

[137] SDM-RDFizer v3.2:`https://doi.org/10.5281/zenodo.3872104`
[138] `https://hub.docker.com/repository/docker/sdmtib/sdmrdfizer`
[139] `http://project-iasis.eu/`
[140] `https://www.bigmedilytics.eu/`
[141] `https://blogs.tib.eu/wp/tib/2020/05/06/how-do-knowledge-graphs-contribute-to-understanding-covid-19-related`
[142] `http://sprint-transport.eu/`
[143] `https://snap-project.eu/`

and Open Cities [144]. Similar as the *Knowledge4COVID-19* knowledge graph, SDM-RDFizer also created the Knowledge Graph of the Drugs4Covid project [145] where NLP annotations and metadata from more than 60,000 COVID-19 articles are integrated in almost 44M RDF triples.

7.1.3 Empirical Evaluation

We compare SDM-RDFizer with a baseline and existing RML interpreters. We aim to answer the following research questions:

- Q1) What is the impact of data duplication rate in the execution time of a knowledge graph creation approach?
- Q2) What is the impact of input data size in the total execution time of a knowledge graph creation process?
- Q3) What is the effect of the triples map types in the `PredicateObjectMap` of a RML mapping affect the existing engines?

All the resources used in this evaluation are publicly available[146]. The experimental configuration is as follows:

Datasets and Mappings. To the best of our knowledge, there is no testbeds to evaluate the performance of a materialized KG construction approaches from heterogeneous data sources. Consequently, following the real-world scenario that initially motivated this research, we create our testbed from the biomedical domain. From the coding point mutation dataset in COSMIC[147], we randomly select records to create six datasets with different sizes, i.e., 10K, 100K, and 1M number of rows. Accordingly, each two datasets with the same volume size differ each other in the number of duplicated values; including 25% or 75% of duplicates with each duplicated value to be repeated 20 times. In total, three mapping files are created with different types of `PredicateObjectMap`: Simple Object Map rules with reference to columns (SOM), Object Reference Map rules (ORM) and Object Join Map rules (OJM). Each type of rules also varies from 1 to 4 number of `PredicateObjectMap`.

[144] `https://ciudades-abiertas.es/`
[145] `https://drugs4covid.oeg-upm.net/`
[146] `https://github.com/SDM-TIB/SDM-RDFizer-Experiments`
[147] `https://cancer.sanger.ac.uk/cosmic` GRCh37, version90, released August 2019

Engines. The SDM-RDFizer v3.2 is tested in two different configurations: optimized version including the proposed operators (SDM-RDFizer) and the baseline with the naïve operators (SDM-RDFizer$^-$). Additionally, we also run the experiments over two well-known RML-compliant engines: RMLMapper v4.7[148] and RocketRML v1.7.0[149]. There is available a docker image per tested engine to facility reproducibility of the study.

Metrics. *Execution time:* Elapsed time spent by an engine to complete the creation of a knowledge graph; it is measured as the absolute wall-clock system time as reported by the `time` command of the Linux operating system. *Number of RDF triples* in the knowledge graph. Each experiment was executed five times and average is reported. The time out is set to 5 hours. The experiments were run in an Intel(R) Xeon(R) equipped with a CPU E5-2603 v3 @ 1.60GHz 20 cores, 64GB memory and with the O.S. Ubuntu 16.04LTS.

7.1.3.1 Discussion

In this section, we describe the outcomes of our experiments evaluating the performance of the selected engines (i.e., SDM-RDFizer, RMLMapper, and RocketRML) in different testbeds. Figure 7.5 and Figure 7.6 report on execution time for creating a knowledge graph from datasets with 25% and 75% of duplicates, respectively. It should be noted that since RocketRML does not support N-M join relations and generates incorrect outputs subsequently, we only provide the results of SOM and ORM mappings for this engine. For the rest of the experiments, we have verified that the generated outputs are the same for all the approaches in terms of cardinality and correctness.

The obtained results clearly reveal the benefits of applying the proposed operators during the process of creating a knowledge graph. As illustrated in Figure 7.5 and Figure 7.6, independent of the size of the input datasets and the percentage of existing duplicates, RMLMapper and RocketRML fail to generate RDF triples from mappings including 2-ORM and 5-ORM; they time out in five hours. Moreover, the execution time of RMLMapper and RocketRML increases as the size of data and number triples maps increase. Nonetheless, as it can be observed, SDM-RDFizer completes the RDF triples generation in all testbeds within a reasonable time period. Additionally, the performance of SDM-RDFizer$^-$ provides evidence of the quality of the SDM-RDFizer operators and their ability of speeding up a knowledge graph creation process.

[148] `https://github.com/RMLio/rmlmapper-java`
[149] `https://github.com/semantifyit/RocketRML/`

7.1.4 Conclusions

The observation that both industrial and scientific applications demand efficient solutions for knowledge graph creation motivated the need of making SDM-RDFizer available as a resource. SDM-RDFizer implements novel physical operators and data structures that speed up the generation of duplicate-free RDF triples even in presence of data sources with high-duplication rate. Empirical results indicate that SDM-RDFizer outperforms the state of the art by up to three orders of magnitude. Thus, SDM-RDFizer broaden the portfolio of technologies for knowledge graph management and provides the basis for the development of real-world knowledge graph applications. In the future, we plan to devise optimization techniques to plan the execution of the mapping rules, as well as to extend SDM-RDFizer to other mapping languages.

7.2 Efficient Processing of Functional Mappings for KG Materialization

A rich spectrum of mapping languages has been proposed to specify schema-ontology alignments across data sources implemented in a variety of semi-structured and structured formats; exemplar approaches include R2RML (Das *et al.*, 2012a), RML (Dimou *et al.*, 2014), and xR2RML (Michel *et al.*, 2015). Furthermore, function-based mapping languages (De Meester *et al.*, 2017; Debruyne & O'Sullivan, 2016; Junior *et al.*, 2016a; Vu *et al.*, 2019) are equipped with abstractions that enable interoperable and reusable specifications of data transformations by means of user-defined functions. Moreover, formalisms like RML+FnO (De Meester *et al.*, 2017) combine the Function ontology and RML, enabling declarative specification of the schema-ontology alignments and data transformations that define the process of KG construction. Albeit expressive, existing mapping languages lack efficient interpreters able to scale up to complex KG construction scenarios. The incoming data avalanche urges KG construction approaches capable of integrating large and diverse data, and efficiently transforming this data to comply with application-specific KG formats.

Problem and Objectives: We tackle the problem of scaled-up KG construction from functional mapping rules and study the impact of functions when applied to large data sources with a high data duplication rate. A KG construction process is defined as a data integration system (Lenzerini, 2002). Mappings among data sources and the system ontology are expressed using the RDF mapping language (RML) (De Meester *et al.*, 2017) and the Function Ontology

(FnO); they define how the ontology concepts are populated with data from the sources in the resulting KG. We aim at transforming complex data integration systems composed of large data sources and mappings with functions into equivalent ones that generates the same KG but in less time.

Our Proposed Approach: We present FunMap, an interpreter of RML+FnO, that converts a data integration system defined using RML+FnO into an equivalent one where RML mappings are function-free. FunMap resembles existing mapping translation proposals (e.g., (Ali & Wrembel, 2019; Corcho *et al.*, 2019; Junior *et al.*, 2016a)) and empowers a KG construction process with optimization techniques to reduce execution time. Transformations of data sources include the projection of the attributes used in the RML+FnO mappings. They are supported on well-known properties of the relational algebra, e.g., the pushing down of projections and selections into the data sources, and enable not only the reduction of the size of data sources but also the elimination of duplicates. Additionally, FunMap materializes functions –expressed in FnO– and represents the results as data sources of the generated data integration system; the translation of RML+FnO into RML mappings that integrate the materialization of functions is performed using joins between the generated RML mappings. The combination of data source and function transformations results in data integration systems where only the data required to execute the RML mappings are retained. The computation of the functions used in the original data integration system is performed once. As a result, the new data integration system's execution is sped up while the same knowledge graph is generated.

Contributions. i) FunMap, an interpreter of RML+FnO that resorts to syntax-based translation (Aho *et al.*, 1986) to push down projections and selections, and materialize functions. ii) Empirical evaluations of the performance of FunMap in real-world testbeds with data of various formats (CSV and Relational), sizes, and degrees of duplication that show reductions in KG construction time by up to a factor of 18.

7.2.1 A Real-World Example from the Biomedical Domain

Our work is motivated by the challenges revealed during genomic variant reconciliation while creating a biomedical knowledge graph. Although the vast majority of the single variations in the genome of a person causes no disease, benign variants can appear in sequenced genomic data repeatedly. In addition to the large heterogeneous volumes generated during genome sequencing and analysis, high-frequency of genomic variants impose data integration challenges

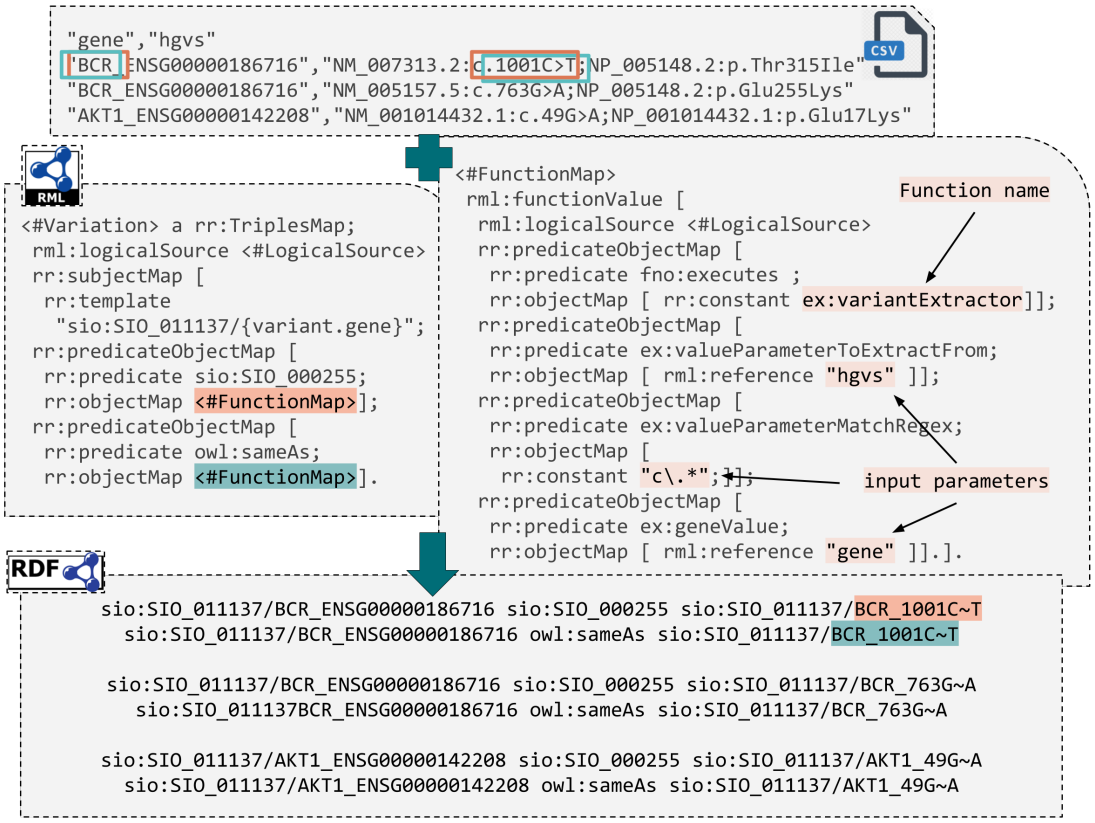


Figure 7.7: Motivating example. Knowledge graph construction using RML+FnO mapping rules for the biomedical domain. The input source in the top is transformed to RDF output (at the bottom) through the processing of the mapping (middle) where the transformation functions are defined. Repeated computations of a function negatively impacts on the performance of an RML engine.

while collecting genomic data from different sources. Additionally, genomic variants are expressed in diverse standard formats (den Dunnen *et al.*, 2016) and reported at DNA, RNA, or protein level. Moreover, this representation can be done according to any of the accepted terminologies and genomic reference versions. Unified representations for variants are required to semantically recognize and integrate equivalent variants residing in different data sources. Variant representations can result from a composition of several factors, such as gene name, genomic position, and residue alteration. Pre-processing functions (e.g., FnO functions) are needed to extract and compose values from different attributes from each data source and generate such a combined representation of variants. These functions are part of the data integration system's mapping rules that define the KG construction process.

Figure 7.7 depicts a mapping rule in RML+FnO where the `FunctionMap` class is utilized. Consider that according to the `LogicalSource` provided in this example, a `FunctionMap` is defined in the mapping rules to create a unified representation for a variant by extracting the values of "gene name" (e.g., BCR) from the attribute `gene` and "coding alteration" (e.g., c.1001C>T) from the attribute named `hgvs` and combine them (e.g., BCR_1001C~T). Current approaches evaluate `FunctionMap` for each `variant`, which can be expensive in presence of large data sources. Nevertheless, the large number of redundant values leaves room for the scalable transformations to execute functional mappings.

7.2.2 The FunMap Approach

Notation	**Explanation**
$DIS_G = \langle O, S, M \rangle$	Data Integration System which creates a KG G
O	Unified Ontology of $DIS_G = \langle O, S, M \rangle$
S	Finite set of Data Sources S_i of $DIS_G = \langle O, S, M \rangle$
M	Finite set of `TriplesMaps` T_i in $DIS_G = \langle O, S, M \rangle$
$RDFize(.)$	A function producing RDF triples from a data integration system
T_i' and T_k'	`TriplesMaps` resulting of applying MTRs
F_i	A Transformation Function in a `TriplesMap` in M
S'	Finite set of Data Sources S_i' resulting of applying DTRs
M'	Finite set of Mapping Rules M_i' resulting of applying MTRs
S_i^{output}	Data source resulting of applying DTR1, with attributes o_i' and a_i' representing the materialization of a transformation function F_i
$S_i^{project}$	Data source resulting of applying DTR2

Table 7.1: Summary of the notation used for defining FunMap

FunMap is an interpreter of data integration systems $DIS_G = \langle O, S, M \rangle$, where O stands for a unified ontology, and S and M represent sets of sources and mapping rules, respectively (Lenzerini, 2002). The evaluation of DIS_G (a.k.a. $RDFize(DIS_G)$) results into a knowledge graph G that integrates data from S according to the mapping rules in M; entities and properties in G are described in terms of O. A complex data integration system DIS_G consists of large data sources with high-duplicated data and mapping rules including functions for both schema-ontology alignments and data transformations. FunMap converts DIS_G into an equiv-

alent data integration system that creates the same knowledge graph but in less time. Table 7.1 summarizes the notation utilized in the FunMap approach.

Problem Statement: Given a data integration system DIS_G=$\langle O, S, M \rangle$, the problem of scaled-up knowledge graph creation from functional mappings requires the generation of a data integration system DIS'_G=$\langle O, S', M' \rangle$:

- The knowledge graphs resulting of the evaluations of both data integration systems are the same, i.e., $RDFize(DIS'_G = \langle O, S', M' \rangle)$=$RDFize(DIS_G = \langle O, S, M \rangle)$ where $RDFize(.)$ is a function producing RDF triples utilizing the input data integration system.
- The execution time of $RDFize(DIS'_G = \langle O, S', M' \rangle)$ is *less than* the execution time of $RDFize(DIS_G = \langle O, S, M \rangle)$.

Solution: FunMap implements a heuristic-based approach; it relies on the assumption that eliminating duplicates, maintaining in the data sources only the attributes mentioned in the mappings, and materializing the functions in the mappings, reduces the execution time of knowledge graph creation process. FunMap receives a data integration system DIS_G=$\langle O, S, M \rangle$ where the mappings M are expressed in RML+FnO. FunMap interprets the mappings in M and converts DIS into the data integration system DIS'_G in which the mappings M' are function free and duplicates in the data sources S' are reduced. Figure 7.8 depicts the FunMap approach; it performs a syntax-based translation of the mappings in M and ensures that each redundant function is evaluated exactly once on the same data values. FunMap transforms S to S' by means of data transformation rules (DTR1 and DTR2). For each F_i over a given S_i, DTR1 creates a temporal source S'_i that includes the attributes from S_i that correspond to the input of F_i; it also generates a source S_i^{output} that contains the attributes in S'_i and attributes representing the output of F_i. For each `FunctionMap` defined over a source S_i, DTR2 creates a source $S_i^{project}$ that includes all attributes of S_i used in the `FunctionMap`. Additionally, FunMap converts mapping rules that include functions by using mapping transformation rules (MTRs); a `FunctionMap` is transformed into `FunctionMaps` without functions while connected by `joinConditions`; initially, S' and S are equal, as well as M' and M. Properties 7.2.2.2, 7.2.2.2, and 7.2.2.2 state the pre- and post-conditions of DTRs and MTRs.

7.2.2.1 Transformation Rules in FunMap

The FunMap syntax-based translation component parses `FunctionMaps` exactly once, i.e., `FunctionMaps` repeated in various mappings are not evaluated more than once. Given `FunctionMaps`,

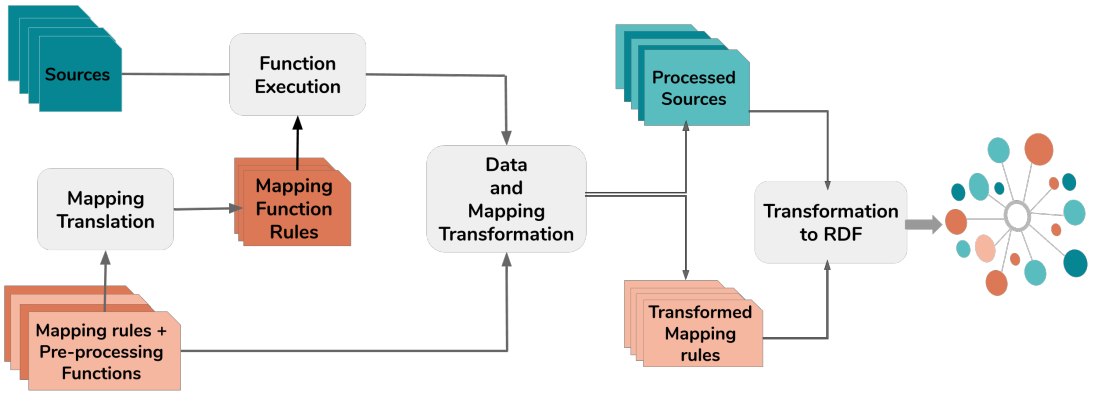


Figure 7.8: The FunMap approach

original data sources, and mappings, FunMap executes transformation rules on data sources and mappings, accordingly. Meanwhile, given the transformed data sources, FunMap detects that a `FunctionMap` has been computed for a given value and avoids repeating this computation. As an outcome, FunMap provides: a) a new set of data sources S' consisting of the original ones in conjunction with transformed data sources, and b) a set M' of transformed function-free mappings. FunMap is loyal to the formats of data sources and mappings. Thus, any RDF mapping language is compatible with the process implemented in FunMap, as far as the language enables the definition of joins between mapping rules. Next, we present the transformation rules.

Data Source Transformation Rules (DTRs): Considering the fact that a `TriplesMap` may only be used some attributes of a dataset, FunMap relies on the properties of the relational algebra and performs DTRs to project only the attributes mentioned in the `TriplesMap`. DTRs are followed by transformation rules (MTRs) that update mappings defined over the transformed data sources.

DTR1: Projection of Functional Attributes: For each transformation function F_i over a given source S_i in the set of data sources S, FunMap projects all attributes a'_i in S_i that are input attributes of F_i, into a temporal data source S'_i followed by duplicate removal. Subsequently, it evaluates F_i over S'_i and stores the results into the attribute o_i. Lastly, it creates a new data source S_i^{output} with the attributes a'_i and o_i; S_i^{output} is added to S'.

DTR2: Projection of Non-Functional Attributes: FunMap provides an additional DTR to further optimize the knowledge graph creation process. Exploiting transformation rules that are proposed in (Jozashoori & Vidal, 2019), FunMap projects all attributes in S_i that are needed

by TriplesMap including those that are received by FunctionMap as input into a new data source $S_i^{project}$ which is added to S'. To better conceive DTRs, consider the original mappings in Figure 7.9 (left-side) and corresponding data source source1.csv that can be seen in Figure 7.10. As shown in Figure 7.9, FunctionMap1 receives Mutation genome position as input. According to DTR1, FunMap projects Mutation genome position from source1 into a new data source named output1.csv which is shown in Figure 7.11.c. The rows number 2 and 4 have the same value for attribute Mutation genome posit-ion which leads FunMap to remove the duplicated value from output1.csv. Afterwards, FunctionMap1 is evaluated given output1.csv as input and the output values are inserted as a new at-

Figure 7.9: Example of DTR and Object-based MTR. On the left, an exemplary mapping including two TriplesMaps and a FunctionMap provided by the original data integration system. On the right side, the mappings are transformed by FunMap including two new TriplesMaps and one new TriplesMap.

ID	Gene name	GRCh	Mutation genome position	Mutation CDS	Primary site	GENOMIC_MUTATION_ID	...
1	DGCR6L	37	22:20302597-20302597	c.514-250C>A	Liver	COSV50619134	...
2	HMCN1	37	1:186072702-186072702	c.10672C>T	lung	COSV54901969	...
3	SLC5A10_ET0 000039564	37	17:18874996-18874996	c.597+465T>A	Skin	COSV58755801	...
4	HMCN1_ET000 00367492	37	1:186072702-186072702	c.10672C>T	Skin	COSV54901969	...
5	COL21A1_ET0 000037081	37	6:56246049-56246049	c.-39+12720G>A	Prostate	COSV63690608	...
6	AKT3	37	1:243692781-243692781	c.1251+16031C>G	Pancreas	COSV55606438	...
7	WDFY4_ET000 00413659	37	10:50044166-50044166	c.*1825+3412G>A	Oesophagus	COSV55433638	...
...	...		...	...	...	...	...

Figure 7.10: Original input data source used by FunMap in KGC workflow. The data source includes many attributes among which only a few are required by the transformation function or function-free mappings in the process of knowledge graph creation.

ID	Mutation genome position	GENOMIC_MUTAT ION_ID
1	22:20302597-2030 2597	COSV50619134
3	17:18874996-1887 4996	COSV58755801
4	1:186072702-1860 72702	COSV54901969
5	6:56246049-56246 049	COSV63690608
6	1:243692781-2436 92781	COSV55606438
7	10:50044166-5004 4166	COSV55433638
...	...	...

(a) Projected1

ID	Mutation genome position	Gene name
1	22:20302597-203025 97	DGCR6L
2	1:186072702-186072 702	HMCN1
3	17:18874996-188749 96	SLC5A10_ET0000 039564
4	1:186072702-186072 702	HMCN1_ET000003 67492
5	6:56246049-5624604 9	COL21A1_ET0000 037081
6	1:243692781-243692 781	AKT3
7	10:50044166-500441 66	WDFY4_ET000004 13659
...	...	...

(b) Projected2

ID	Mutation genome position	functionOutput
1	22:20302597-2030 2597	22:20302597:203 02597
3	17:18874996-1887 4996	17:18874996:188 74996
4	1:186072702-1860 72702	1:186072702:186 072702
5	6:56246049-56246 049	6:56246049:5624 6049
6	1:243692781-2436 92781	1:243692781:243 692781
7	10:50044166-5004 4166	10:50044166:500 44166
...	...	...

(c) Output1

Figure 7.11: Transformed sources generated by FunMap. The DTR2 generates a new source by projecting attributes for each TripleMap (Figures a and b) while DTR1 projects input and output attributes of each FunctionMap into a new source (Figure c). Both remove the generated duplicates.

tribute named `functionOutput` into the `output1.csv` data source. Moreover, attributes `GENOMIC_MUTATION_ID` and `Primary site` from `source1.csv` that are in `TriplesMap1` are projected into the new data source that is shown in Figure 7.11.a and duplicated values are removed. Similarly, `Projected2.csv` is created based on the attributes of `TriplesMap2`. **Mapping Transformation Rules (MTRs)** Mappings are transformed to create the same knowledge

Figure 7.12: Example of Subject-based MTR. An example of mappings including a TriplesMaps and FunctionMap are illustrated on the left and their transformed version including three TriplesMap are shown on the right side.

graph utilizing the transformed data sources. MTRs are defined considering the role of a transformation function F_i in each `TriplesMap` T_i. I) F_i as an `ObjectMap`: We refer to the MTRs that are required in this case as `Object-based`. First of all, for each F_i, a new `TriplesMap` T_i' is created; it refers to the data source generated as the outcome of F_i, i.e., S_i^{output}. Accordingly, the `SubjectMap` of T_i' refers to the output attributes o_i in S_i^{output}. Afterwards, in `TriplesMap` T_i where F_i is presented as an `ObjectMap`, F_i is replaced by a `joinCondition` which joins T_i and T_i' over attributes a_i', i.e., the input attributes of F_i. Moreover, the `logicalSource` of T_i is changed to $S_i^{project}$, i.e., the corresponding projected data source provided as an outcome of DTR2. II) F_i as a `SubjectMap`: Contrary to the `Object-based`, in this set of MTR - we refer to as `Subject-based`- for each `predicateObjectMap` that follows a F_i of the type `SubjectMap`, a new `TriplesMap` T_i' refers to the data source $S_i^{project}$ which is generated as an outcome of DTR2 by projecting the attribute a_i' from S_i that are referenced as `objectMap` in the original `predicateObjectMap`. The `subjectMap` of T_k' –the transformed T_i – refers to the o_i and its `logicalSource` is S_i^{output}. Note that `subjectMap` of T_i' is by definition a `TermMap`, which means that its value can be any RDF term according to the RML specification.

Each `objectMap` in T_i that is a `FunctionMap` is replaced by a `joinCondition` between T_i and corresponding T_i' over input attributes a_i' of F_i. In both cases, the transformed T_i– denoted as T_k'– and T_i' are added to M' and T_i is removed from M'.
Figures 7.9 and 7.12 illustrate two examples of rewritten mappings based on DTRs and MTRs. In the left side of both figures, the original mappings are presented while the transformed mappings are depicted on the right side. In the transformed mappings in Figure 7.9, `TriplesMap3` is created for `FunctionMap1`; it refers to the attribute `functionOutput` in the projected data source `output1.csv`– shown in Figure 7.11.c. Then, `FunctionMap1` is replaced in both `TriplesMap1` and `TriplesMap2` by a join condition over the attribute `Mutation genome position` which is the input attribute of `FunctionMap` in the original mapping file as it is highlighted by the same **color**. Accordingly, data sources -highlighted- of `TriplesMaps` are also transformed to refer to the projected data sources. Consider Figure 7.12 where `FunctionMap` is a `subjectMap`. In both `predicateObjectMa-ps` of `TriplesMap1`, `FunctionMap1` is replaced by a `joinCondition` over the attribute `Mutation genome position` that is the input of `FunctionMap1`. To better clarify the performed transformation, consider the first `predicateObjectMap` in `TripleMap1` in the original mappings; the `predicate` is **represents** and the `ObjectMap` refers to the attribute **Mutation**. After the transformation, the first `predicateObjectMap` has the same `predicate` **represents** and through the `joinCondition` refers to the same attribute **Mutation** in `projected1.csv`.

7.2.2.2 Lossless Transformation Rules

We validated the correctness of the transformations that are performed by FunMap, by proving that the RDF triples produced by DIS_G' are identical to the ones generated by DIS_G.

Consider:

- The Ontology O is defined as a triple, $O = (C, P, Axioms)$ where C and P represent the classes and properties of O respectively. The $Axioms$ stands for a set of statements expressing the characteristics of the properties of O.

- The data sources of DIS_G are defined as a set of $S_j^{A_j}$ where S_j stands for a data source and A_j represents attributes of S_j that are utilized by M.

- The M describing the classes C and properties P in O in terms of sources in S comprises a set of mapping rule r_i that is defined as:

$$r_i : c_j(X, \overline{X}) : -S_1(\overline{X_1}), S_2(\overline{X_2}), \ldots, S_m(\overline{X_m})$$

  Where c_j is a class in C, X is a variable, and $\overline{X}$ is a set of pairs , and $X_{i,j}$ is a variable. The predicate $S_z(\overline{X_z})$ represents a source S_z in S and $\overline{X_z}$ is a set of pairs $(a_{i,z}, X_{i,z})$ where $X_{i,z}$ is a variable and $att_{i,z}$ is an attribute of S_z.

For each mapping rule in M with sources $S_z(\overline{X_z})$ and a set of utilized attributes as $\prod_{att} S_z$, DTR1 and DTR2 add new sources $S_y(\overline{X_y})$ and $S_w(\overline{X_w})$ in the way that $\prod_{att} S_y + \prod_{att} S_w$ equals $\prod_{att} S_z$. Accordingly, for each mapping rule in M:

- If a mapping includes `FunctionMap` in `ObjectMap`:
  $(att_{A,z}, X_{sub,z}) + (att_{f(B),z}, X_{obj,z})$ =? $(att_{A,w}, X_{sub,w}) + (att_{B,w}, X_{join,w}) + (att_{B,y}, X_{join,y}) + (att_{f(B),y}, X_{obj,y})$

Pre- and post-conditions of Data Source Transformation Rules (DTRs) and Mapping Transformation Rules (MTRs) are stated in the following properties: *Property 1.*(Lossless Function) Given data integration systems DIS_G=$\langle O, S, M\rangle$ and DIS'_G=$\langle O, S', M\rangle$ such that DIS'_G is the result of applying one DTR1 transformation to DIS_G. Then, there are data sources S_i and S_i^{output} in S and S', respectively, and the following statements hold:

- $S' - S = \{S_i^{output}\}$, there is a mapping T_i in M with a function F_i, and *Attrs* contains the attributes a'_i of F_i in S_i and the output attributes o_i of F_i.
- S_i^{output} comprises the attributes *Attrs* and $\pi_{a'_i}(S_i^{output})$=$\pi_{a'_i}(S_i)$.
- For each tuple $t_{i,j}$ in S_i^{output}, the values of the attributes o_i in $t_{i,j}$ correspond to the result of F_i over the values of a'_i in $t_{i,j}$, i.e., $t_{i,j}.o_i$=$F_i(t_{i,j}.a'_i)$.

Property 2.(Lossless Projection) Given data integration systems DIS_G=$\langle O, S, M\rangle$ and DIS'_G=$\langle O, S', M\rangle$ such that DIS'_G is the result of applying one DTR2 transformation to DIS_G. Then, there are data sources S_i and $S_i^{project}$ in S and S', respectively, and the following statements hold:

- $S' - S = \{S_i^{project}\}$, and there is a mapping T_i in M defined over the attributes *Attrs* from S_i, and $S_i^{project} = \pi_{Attrs}(S_i)$.

Property 3. (Lossless Schema-Ontology Alignments)[150] Given data integration systems DIS_G=$\langle O, S, M\rangle$ and DIS'_G=$\langle O, S, M'\rangle$ such that DIS'_G is the result of applying one MTR transformation to DIS_G. Then, there are `TriplesMaps` T_i in M, and T_i' and T_k' in M', and the following statements hold:

- $M - M' = \{T_i\}$ and $M' - M = \{T_i', T_k'\}$.
- There is a function F_i in T_i as the `ObjectMap` of a `PredicateMap` p, and there is a data source S_i^{output} in S which is the `LogicalSource` of T_i. The attributes of S_i^{output} are the union of a_i' and o_i, while a_i' and o_i are input and output attributes of F_i, respectively.
- T_i and T_k' are defined over the same `LogicalSource` $S_i^{project}$. S_i^{output} is the `LogicalSource` of T_i' and o_i is the `SubjectMap` of T_i'.
- T_i and T_k' only differ on the `ObjectMap` p. In T_i, `ObjectMap` of p is defined as F_i, while in T_k', a `joinCondition` to T_i' on a_i' defines the `ObjectMap` of p.

7.2.3 Experimental Evaluation

We evaluate FunMap[151] in comparison to current approaches that create a knowledge graph using the specified data sources and RML+FnO mappings. We aim to answer the following research questions: Q1) What is the impact of data duplication rate in the execution time of a knowledge graph creation approach?; Q2) What is the impact of different types of complexity over transformation functions during a knowledge graph creation process?; Q3) How does the repetition of a same function in different mappings affect the existing RML engines?; Q4) What is the impact of relational data sources in the knowledge graph creation process? All the resources used to perform this evaluation are available in our Github repository[152]. The experimental configuration is as follows:

Datasets and Mappings. To the best of our knowledge, there are no testbeds to evaluate the performance of a knowledge graph construction approach that applies functional mappings. Consequently, following the real-world scenario that initially motivated this research, we create our testbed from the biomedical domain. We generate a baseline dataset by randomly selecting 20,000 records from the coding point mutation dataset in COSMIC[153] database. We keep all

[150] Similarly, this property can be stated for the result of applying MTR over the subject position of a property in a mapping of a data integration system.

[151] `https://doi.org/10.5281/zenodo.3993657`

[152] `https://github.com/SDM-TIB/FunMap`

[153] `https://cancer.sanger.ac.uk/cosmic` GRCh37, version90, released August 2019

39 attributes of the original dataset in the baseline dataset, while only five to seven of them are utilized in mappings. In total, four different mapping files are generated consisting of one `FunctionMap` and four, six, eight, or ten `TriplesMaps` with a `predicateObjectMap` linked to the function. To additionally validate FunMap in case of large-sized data, we create another dataset following the same criteria, with 4,000,000 records and the size of about 1.3GB.

Engines. The baselines of our study are three different open source RML-complaint engines that are able to execute RML+FnO mappings and have been extensively utilized in multiple applications and tested by the community: SDM-RDFizer v3.0(Iglesias *et al.*, 2020), RMLMapper[154] v4.7, and RocketRML[155] v1.6. [156]. In order to evaluate the impact of transformation rules, we implement FunMap v1.0 on the top of the aforementioned engines with DTR2 optimization as an optional parameter. We refer to the approach which applies FunMap excluding DTR2 as FunMap^{-}[157]. We created a docker image per tested engine for reproducibility.

Metrics. *Execution time:* Elapsed time spent by an engine to complete the creation of a knowledge graph and also counts FunMap pre-processing; it is measured as the absolute wall-clock system time as reported by the `time` command of the Linux operating system. Each experiment was executed five times and average is reported. The experiments were executed on an Ubuntu 16.04 machine with Intel(R) Xeon(R) Platinum 8160, CPU 2.10GHz and 700Gb RAM.

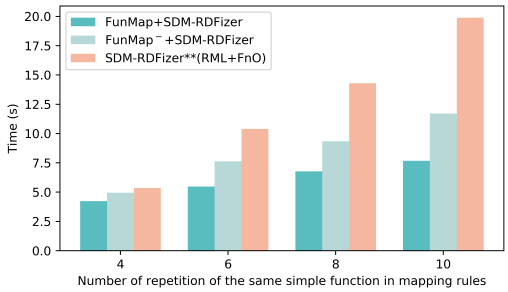
Experimental setups. Based on our research questions, we set up in overall 198 experiments as the combinations of the following scenarios. We create two datasets from our baseline with 25% and 75% duplicates which means in the 25% duplicate dataset, 25% and in the 75% duplicate dataset, 75% of the records are duplicated. Additionally, two functions with different levels of complexity are created. We describe the complexity level of the functions based on the number of required input attributes and operations to be performed. Accordingly, "simple" function is defined to receive one input attribute and perform one operation, while a "complex" function receives two input attributes and completes five operations. In total, we create eight mapping files including four, six, eight, and ten `TriplesMap` and one `FunctionMap` to be either "simple" or "complex". Additionally, six experiments using 75% duplicate datasets of

[154] `https://github.com/RMLio/rmlmapper-java`

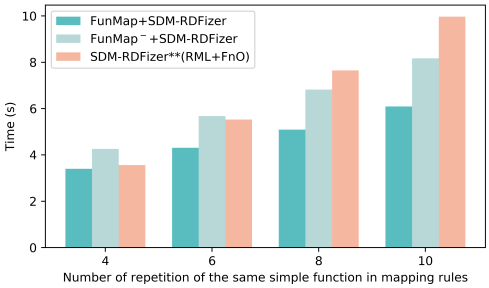
[155] `https://github.com/semantifyit/RocketRML/`

[156] We name them SDM-RDFizer**(RML+FnO), RMLMapper**(RML+FnO), and RocketRML**(RML+FnO).

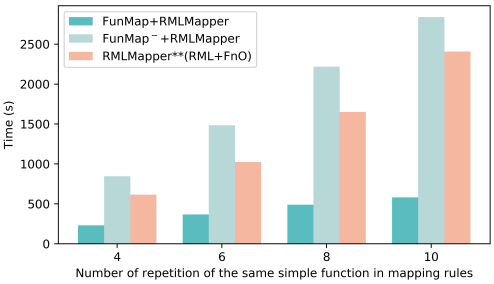
[157] We name these combined engines as follows: a) FunMap: FunMap+SDM-RDFizer, FunMap+RMLMapper, and FunMap+RocketRML; b) FunMap^{-}: FunMap^{-}+SDM-RDFizer, FunMap^{-}+RMLMapper, and FunMap^{-}+RocketRML.



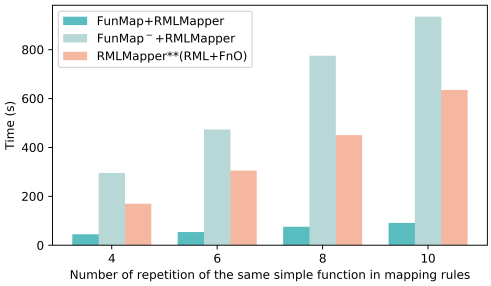
(a) SDM-RDFizer - 25% of duplicates



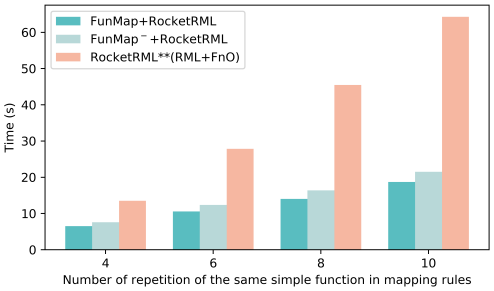
(b) SDM-RDFizer - 75% of duplicates



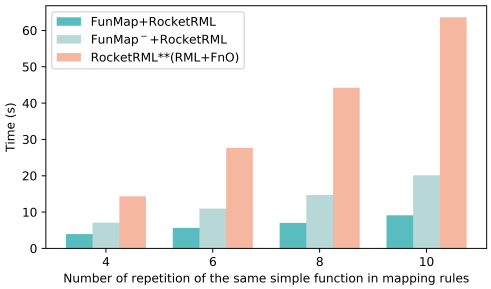
(c) RMLMapper - 25% of duplicates



(d) RMLMapper - 75% of duplicates



(e) RocketRML - 25% of duplicates



(f) RocketRML - 75% of duplicates

Figure 7.13: Total execution time of experiments with simple functions 25-75% of duplicates. SDM-RDFizer, RMLMapper and RocketRML executing simple functions in RML+FnO mappings and with FunMap and FunMap^{-}.

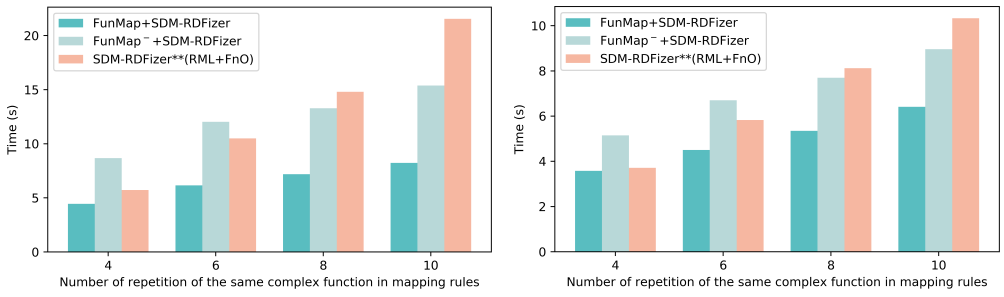
20,000 and 4,000,000 records and a mapping file including ten complex functions are set up in order to be run over a relational database (RDB) implemented in MySQL 8.0 [158].

[158]https://www.mysql.com/

7.2.3.1 Discussion of Observed Results

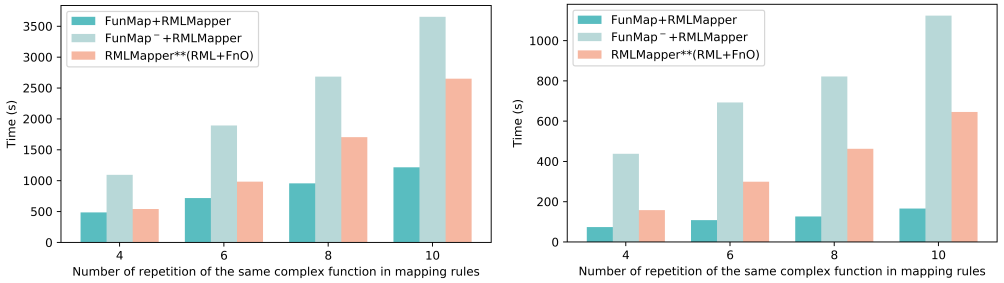
In this section, we describe the outcomes of our experimental evaluation. Figure 7.13 reports on the execution time of the different testbeds in which the functions are considered to be "simple" whereas Figure 7.14 shows the experiments involving "complex" functions. Both figures represent the total execution time for constructing the knowledge graph applying selected engines (i.e., SDM-RDFizer, RMLMapper, and RocketRML) in three different configurations: a) the current version of the engine that is able to directly interpret RML+FnO mappings in the engine (e.g., RMLMapper**(RML+FnO)); b) FunMap$^-$ in conjunction with the engine (e.g., FunMap$^-$+RMLMapper); and c) FunMap together with the engine (e.g., FunMap+RMLMapper). In the case of all the configuration of RocketRML, we only provide the results for the execution of simple functions because the engine does not execute joins with multiple conditions[159] correctly, hence, the proposed optimizations cannot be applied. For the rest of the experiments, we have verified that the results are the same for all the approaches in terms of cardinality and correctness. The results obtained by the application of SDM-RDFizer with the repetition of simple functions (Figures 7.13a and 7.13b) reflect an improvement of the execution time when FunMap is applied in the process. With the growth of number of duplicates and repeated functions, the difference between the performance of SDM-RDFizer**(RML+FnO) and FunMap+SDM-RDFizer increases. Using this engine, FunMap$^-$ shows the same behavior as FunMap, however, in the case of having a large number of duplicates and a few repeated functions FunMap$^-$ does not improve the performance of SDM-RDFizer**(RML+FnO). In the case of using RMLMapper (Figures 7.13c and 7.13d), we observe that the results obtained together with FunMap$^-$ (i.e., DTR1 optimization) do not show better performance than RMLMapper**(RML+FnO). DTR1 which only focuses on transforming functions, delegates the removal of the duplicates to the engine which is not accomplished efficiently by RMLMapper. However, in FunMap+RMLMapper, that includes DTR1 and DTR2 optimizations, duplicates are removed before the execution of the RML mappings and leads to obtain the results that clearly show improvements with respect to the baseline. In the same manner as the SDM-RDFizer, the number of repetitions of the functions affects the execution time of the RMLMapper**(RML+FnO), while FunMap maintains similar execution times. Finally, RocketRML (Figures 7.13e and 7.13f) seems not to be affected by the number of duplicates over the input data, obtaining similar execution times for 25% and

[159] Check an example in the zip file of the supplementary material.



(a) SDM-RDFizer - 25% of duplicates

(b) SDM-RDFizer - 75% of duplicates



(c) RMLMapper - 25% of duplicates

(d) RMLMapper - 75% of duplicates

Figure 7.14: Total execution time for complex functions 25-75% of duplicates. SDM-RDFizer and RMLMapper executing complex functions in RML+FnO mappings and with FunMap and FunMap^{-}.

75% rate for RocketRML**(RML+FnO). However, the number of repetitions over functions impacts the performance of RocketRML**(RML+FnO), increasing the total execution time. The incorporation of DTR1 (i.e., FunMap^{-}+RocketRML) and DTR2 (i.e., FunMap + RocketRML) enhances the performance and scalability during the construction of the knowledge graph, obtaining a similar behavior as the other two tested engines.

The effect of function complexity over SDM-RDFizer can be observed in Figures 7.14a and 7.14b. Whenever the number of repetitions is low (4-6), the join with multiple conditions affects FunMap^{-} + SDM-RDFizer, obtaining worse results than SDM-RDFizer**(RML+FnO). However, if repetitions increase (8-10), DTR1 empowers SDM-RDFizer**(RML+FnO) due the reduction of repeated operations during the evaluation of the mappings. Conversely, FunMap+SDM-RDFizer exhibits better results than SDM-RDFizer**(RML+FnO) in all the testbeds. Finally, the behavior of RMLMapper – when it has to execute complex transformation functions (Figures 7.14c and 7.14d)– is affected in terms of execution time for the configuration FunMap^{-}+RMLMapper in comparison to the case of simple functions. As similar as SDM-

RDFizer, the join with several conditions is impacting the performance. However, together with data transformation optimizations, FunMap+RMLMapper outperforms the baseline.

The experimental results on RDBs show even more significant improvement in the performance of both RMLMapper and SDM-RDFizer in the presence of FunMap. In FunMap+RMLMapper, applying `joins` in the SQL queries that define the `logicalSources` instead of using `joinConditions` reduces execution time by up to a factor of 18. These results evidence that `joinConditions` are not efficiently implemented by RMLMapper, and explain why FunMap+RMLMapper is showing less improvement compared to FunMap+SDM-RDFizer in Figure 7.14. Moreover, FunMap+SDM-RDFizer successfully performs on the large-sized relational dataset of 1.3GB in 5,670.67 seconds, while SDM-RDFizer**(RML+FnO) cannot create the KG and times out after 10,000 seconds.

In overall, we observe that the configurations that interpret RML+FnO mappings directly are affected by the repetition of the functions and the degree of data duplicates, i.e., execution time monotonically increases with number of functions and data duplication degree. In contrast, the incorporation of FunMap to the engines shows less fluctuated behavior when the data duplication rate increases. Additionally, the studied engines handle the repetition of the functions during the construction of the knowledge graph thanks to the pushing down of the execution of the functions directly over the dataset. In summary, the observed results indicate that the FunMap heuristics improve the performance of data integration systems and generate solutions to the problem of scaled-up knowledge graph construction. The effectiveness of the proposed transformations has been empirically demonstrated on various RML+FnO and RML-compliant engines. However, we observe that there are cases where the application of DTR1 alone is not enough (i.e., FunMap^{-}), being required the applications of all the transformations (i.e., DTRs and MTRs) to provide an effective solution.

7.2.4 Conclusions

We addressed the problem of scaled-up KG construction in complex data integration systems, i.e., systems with large data sources, high data duplication rate, and functional mappings. We presented a heuristic-based approach for efficiently evaluating data integration systems with data sources in diverse formats (e.g., CSV or relational). The proposed heuristics are implemented in FunMap, an interpreter of RML+FnO, that converts data integration systems in RML+FnO into equivalent data integration systems specified in RML. Besides shaping an

RML-engine independent interpreter of RML+FnO, FunMap generates data integration systems that enhance RML-compliant engines whenever transformation functions are repeatedly used, and data sources are large and have highly-duplicated data. Empirical evaluations of the combination of FunMap with RML-compliant engines suggest that the execution time of RML+FnO can be reduced by up to a factor of 18. Thus, FunMap widens the repertory of tools to scale up knowledge graphs to the enormous increase of incoming data and ease the development of real-world KG applications. As the main limitation, FunMap can only be applied with an RML-compliant engine which supports either `joinCondition` or RDB on the backend. We plan to devise cost-based optimization approaches that, together with the proposed heuristics, allow for the generation of the best solution for a complex data integration system in RML+FnO.

Chapter 8

Conclusions and Future work

This thesis presents several contributions to the state of the art to address research objectives in the are of knowledge graph construction using declarative mapping languages. The contributions and identified future lines of work are summarized below.

8.1 Achievements

Constructing knowledge graphs from heterogeneous data sources is a complex data integration problem. Open research problems addressed in this thesis are: (i) the generation and interoperability of different mapping rules specifications to facilitate to users the KGC process, (ii) the creation of representative evaluation methods to provide an overview of the state of the art on the KGC engines and to understand their current limitations, (iii) as well as optimizations techniques to scale up the construction of virtual but also materialized KGs.

The first objective of the this thesis is focused on define **representative features of a new knowledge graph construction generation systems**. This is done in Chapter 4, where the *mapping translation* concept is defined, adding a new layer into a KGC workflow. As we demonstrate with several use cases, exploiting the benefits of making interoperable different mapping languages specifications can enhance several steps of this process. The specific use case shown in this thesis is on the statistics domain, where we propose a set of new properties over the R2RML specification to improve the maintainability of the creation of the mapping rules in this domain. The ideas around this concept are also used over the different optimizations shown in Chapters 6 and 7.

The **exploitation of mapping rules to enhance the construction of virtual and materialized knowledge graphs** techniques is one of the main contributions of this thesis. To the best of our knowledge, the mapping driven optimizations techniques proposed in this work are the first ones that put the focus and exploit information from the semantic annotations. The heuristic based approaches proposed by Morph-CSV (Section 6.1) and FunMap (Section 7.2) empirically demonstrate over several benchmark and use cases the importance of declarative annotations in a KGC process to efficiently deal with the heterogeneity of input data sources in the current web of data. Additionally, Morph-GraphQL (Section 6.2) emphasizes the necessity of semantic web technologies, and more specific, the mapping rules, for avoiding data silos where non-semantic web approaches (e.g., GraphQL, API Rest, etc) are used to expose data on the web. Finally, SDM-RDFizer (Section 7.1) reveals the importance of well design physical data structures and their corresponding operators to scale-up the construction of knowledge graphs. Summarizing, we have identified the limitations of the proposals of the state of the art together with their open problems and we tackle them from a research perspective, highlighting that engineering solutions are not enough to solve complex data integration problem for constructing knowledge graphs.

To accomplish the second objective of this thesis, described as **representative evaluation systems for knowledge graph construction engines from heterogeneous data sources**, we present three different contributions. First, we analyze and extend the test cases presenting for RDB2RDF engines to coverage heterogeneous data sources, using RML as mapping language. In this manner, we can provide an overview of the compliance of the engines over this mapping language, which help user and practitioners to select an specific engine for their use cases. Second, we select and analyze the parameters that can impact in the performance and completeness of KGC engines. Our ambition is that the reported results of this contribution, inspire the community to define general testbeds that facilitate the understanding of the state of the art and the development of novel tools for constructing knowledge graphs at large scale. Following this ambition, we define the GTFS-Madrid-Bench, a benchmark for (virtual) KGC engines over the transport domain. Integrating the parameters defined in our previous work and defining a set of representative SPARQL queries, we propose the first benchmark that contributes to evaluate in a representative manner virtual KGC engines from one or multiple data sources and formats. We empirically test our approach over a set of heterogeneous KGC engines and identify multiple and promising future research work lines in this topic. Although the first and second contributions have been tested over materialized KGC engines and the third

one over virtual KGC engines, notice that the contributions of this thesis are agnostic to the type of process to be performed, and can be used to test the capabilities of both approaches.

8.2 Future Work

In this section we describe those research problems that were not tackled during this thesis, due to time-permitting issues, or that were appearing as continuation of the proposed solutions in this work.

Aligned with the vision of the future generation of knowledge graph construction systems, we think that the mapping translation concept needs to be explored further, and this would allow a new range of KGC approaches that may be part of a new generation. In our opinion, the KGC community should see this variety of mapping languages not only as challenges (e.g., interoperability) but also, and mainly, as an opportunity for further research and development in this area, to address the need to cover more types of data sources while taking advantage of all the work that has been done in advanced aspects like query translation. Providing mapping translator services across mapping languages would bring further benefits and increase the availability of ontology-based data for its exploitation by search engines and query answering systems at Web scale. Additionally, the definition of a conceptual model describing the concepts of different mapping languages using the same vocabulary can be one of the first points to provide such translation services across different specifications. Finally, the analysis of the role of the users in the process of constructing knowledge graphs will be essential to develop robust and useful solutions in complex data integration environments.

One of the main future lines we have identify during this thesis, extending the contributions on the enhancement of KGC systems, is to define methodologies and techniques for an optimal physical design of knowledge graphs. The main idea is to be able to decide which parts of a KG have to be materialized or virtualized analyzing the features of the typical inputs of a KGC process (data, constraints, mapping rules, ontology). We believe that these methodologies will help to start to see the web as an integrated database that can be queried using Semantic Web technologies. Together with the application of the optimizations techniques proposed in this thesis over distributed environments, such as the ones proposed in (Endris *et al.*, 2019; Mami *et al.*, 2019b), leverage the use of declarative KGC techniques to its next steps providing the basis for developing real-world knowledge graph applications.

For the evaluation systems, we need to extend the current proposals in order to coverage other be more flexibly to evaluate a KGC workflow, taking into account all the parameters that can have an impact in their behavior. Some examples of these possible future lines are: the inclusion of mapping rules with transformation functions, the adaption of mapping rules construction in a data integration system to isolated parameters from this input in the evaluation, or the improvement in creation of datasets at scale, exploiting the information from mapping rules or graph constraints (e.g, SHACL shapes). Finally, it is important to create evaluation systems that include a ground truth in order to test not only the performance and scalabitily of the engines but also other important features such as correctness and completeness.

Finally, the use of declarative and standard mapping rules and metadata descriptions makes possible the generalization of KGC engines and optimizations, avoiding ad-hoc and manual steps. It also incorporates a set of important benefits for these processes such as the improvement of its maintainability, readability, and understandability. We believe that this kind of solutions should be promoted in academic, industry and public organizations as good practices for data management on the web to for example, avoid to have data cemeteries such as the current open data portals. Our vision is that, analyzing the role of the users in complex data integration environments on the web, will help to understand how to promote and develop robust and useful semantic web solutions for constructing knowledge graphs at scale in distributing scenarios.

ANNEX A

GTFS-Madrid-Bench Completeness

Dataset	Source	Tool	queries																
			q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-1	SQL	Ontario	58540	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
		Morph-RDB	58540	765	-	13	84	1	2	-	151439	1	-	6	734	2234	26	-	855	-
		Ontop	58540	765	734	-	0	-	0	-	151439	-	-	-	734	2234	26	0	855	-
	MongoDB	Morph-xR2RML	0	0	0	0	0	0	0	0	0	0	0	0	0	2364	0	0	855	0
	CSV	Morph-RDB	58540	765	-	13	-	1	-	-	-	1	-	6	734	-	-	-	855	-
		Morph-CSV	58540	765	-	13	-	1	2	-	151439	1	-	6	734	-	-	-	855	128
		Ontario	0	-	734	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	XML	Ontario	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	JSON	Ontario	58540	-	734	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Best	Ontario	0	-	734	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Worst	Ontario	0	-	734	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	RDF	Virtuoso	58540	765	734	13	28	1	2	4728	151439	1	130	6	734	2364	26	34	855	64

Table A.1: Completeness of benchmark queries in experiment configurations with GTFS-1 dataset. Minus means that the processor is not able to execute the query (i.e. generates an error) or it does not evaluate the query within the timeout duration.

Dataset	Source	Tool	queries																	
			q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-5	SQL	Ontario	176830	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
		Morph-RDB	176830	3161	-	65	350	1	62	-	-	1	-	65	1325	11170	4949	-	4275	-
		Ontop	176830	3161	2104	-	0	-	0	-	0	-	-	-	1325	11170	4828	0	4275	-
	MongoDB	Morph-xR2RML	0	0	0	0	0	0	0	0	0	0	643	0	0	6593	0	0	3357	0
	CSV	Morph-RDB	176830	3161	-	65	-	1	-	-	-	1	-	-	1325	-	-	-	4275	-
		Morph-CSV	176830	6310	-	65	-	1	0	-	-	1	-	65	1325	-	-	-	4275	0
		Ontario	0	-	2104	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	XML	Ontario	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	JSON	Ontario	0	-	2104	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Best	Ontario	0	-	2104	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Worst	Ontario	0	-	2104	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	RDF	Virtuoso	176830	3161	2104	65	350	1	62	23640	359113	1	650	65	1325	11170	4949	2080	4275	624

Table A.2: Completeness of benchmark queries in experiment configurations with GTFS-5 dataset. Minus means that the processor is not able to execute the query (i.e. generates an error) or it does not evaluate the query within the timeout duration.

Dataset	Source	Tool	queries																	
			q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-10	SQL	Ontario	353660	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
		Morph-RDB	353660	6312	-	130	700	1	67	-	-	1	-	130	2650	22340	8641	-	8550	-
		Ontop	353660	6312	4207	-	0	-	0	-	0	-	-	-	2650	22340	8521	0	8550	-
	MongoDB	Morph-xR2RML	0	0	0	0	0	0	0	0	0	0	1292	0	0	11348	0	0	5508	0
	CSV	Morph-RDB	353660	6312	-	130	-	1	-	-	-	1	-	-	2650	-	-	-	8550	-
		Morph-CSV	353660	12620	-	130	-	1	0	-	0	1	-	130	2650	-	-	-	8550	0
		Ontario	0	-	4207	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	XML	Ontario	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	JSON	Ontario	0	-	4207	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Best	Ontario	0	-	4207	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Worst	Ontario	0	-	4207	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	RDF	Virtuoso	353660	6312	4207	130	350	1	67	47280	718317	1	1300	130	2650	22340	8641	1820	8550	1300

Table A.3: Completeness of benchmark queries in experiment configurations with GTFS-10 dataset. Minus means that the processor is not able to execute the query (i.e. generates an error) or it does not evaluate the query within the timeout duration.

Dataset	Source	Tool	queries																	
			q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-50	SQL	Ontario	-	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
		Morph-RDB	1768300	31550	-	650	3500	1	59	-	-	1	-	650	13250	111700	21958	-	42750	-
		Ontop	1768300	31550	21034	-	0	-	0	-	0	-	-	-	13250	111700	17537	0	42750	-
	MogoDB	Morph-xR2RML	0	0	0	0	0	0	0	0	0	0	5058	0	0	-	0	0	-	0
	CSV	Morph-RDB	1768300	31550	-	650	-	1	-	-	-	1	-	-	1325	-	-	-	-	-
		Morph-CSV	1768300	63100	-	650	-	1	0	-	0	1	-	650	13250	-	-	-	42750	0
		Ontario	0	-	21034	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	XML	Ontario	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	JSON	Ontario	0	-	21034	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Best	Ontario	0	-	21034	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Worst	Ontario	0	-	21034	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	RDF	Virtuoso	1768300	31550	21034	650	1750	1	59	236400	3591503	1	6500	650	13250	111700	21958	2730	42750	6500

Table A.4: Completeness of benchmark queries in experiment configurations with GTFS-50 dataset. Minus means that the processor is not able to execute the query (i.e. generates an error) or it does not evaluate the query within the timeout duration.

Dataset	Source	Tools	queries																	
			q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-100	SQL	Ontario	-	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
		Morph-RDB	3536600	63100	-	1300	7000	1	67	-	-	1	-	1300	26500	223400	35502	-	85500	-
		Ontop	3536600	63100	42067	-	0	-	0	-	0	-	-	-	26500	223400	31080	0	85500	-
	MongoDB	Morph-xR2RML	0	0	0	0	0	0	0	0	0	0	10336	0	0	-	0	0	-	0
	CSV	Morph-RDB	-	63100	-	1300	-	1	-	-	-	1	-	-	-	-	-	-	-	-
		Morph-CSV	3536600	126200	-	1300	-	1	0	-	-	1	-	1300	26500	-	-	-	85500	0
		Ontario	0	-	42067	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	XML	Ontario	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	JSON	Ontario	0	-	42067	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Best	Ontario	0	-	42067	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Worst	Ontario	0	-	42067	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	RDF	Virtuoso	3536600	63100	42067	1300	3500	1	67	472800	7183874	1	13000	1300	26500	223400	35502	1690	85500	13000

Table A.5: Completeness of benchmark queries in experiment configurations with GTFS-100 dataset. Minus means that the processor is not able to execute the query (i.e. generates an error) or it does not evaluate the query within the timeout duration.

Dataset	Source	Tool	queries																
			q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18
GTFS-500	SQL	Ontario	-	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
		Morph-RDB	-	315499	-	6500	35000	1	53	-	-	1	-	6500	132500	1117000	38749	-	427500	-
		Ontop	0	315499	210334	-	0	-	0	-	0	-	-	-	132500	1117000	34323	0	427500	-
	MongoDB	Morph-xR2RML	0	0	0	0	0	0	0	0	0	0	-	0	0	-	0	0	-	0
	CSV	Morph-RDB	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
		Morph-CSV	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		Ontario	0	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	XML	Ontario	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	JSON	Ontario	-	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Best	Ontario	0	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	Worst	Ontario	0	-	-	-	-	-	-	0	-	-	-	-	-	0	-	-	-	-
	RDF	Virtuoso	17683000	315499	210334	6500	17500	1	53	2364000	35919991	1	65000	6500	132500	1117000	38749	2340	427500	65000

Table A.6: Completeness of benchmark queries in experiment configurations with GTFS-500 dataset. Minus means that the processor is not able to execute the query (i.e. generates an error) or it does not evaluate the query within the timeout duration.

ANNEX B

GTFS-Madrid-Bench Queries

Listing B.1: Prefixes

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gtfs: <http://vocab.gtfs.org/terms#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX schema: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

Listing B.2: Query 1 - List all shapes with some of their data

```
SELECT * WHERE {
 ?shape a gtfs:Shape .
 ?shape geo:lat ?shape_pt_lat .
 ?shape geo:long ?shape_pt_lon .
 ?shape gtfs:pointSequence ?shape_pt_sequence .
}
```

Listing B.3: Query 2 - List all stops with some of their data including geographic coordinates, where the latitude is bigger than its mean

```
SELECT * WHERE {
 ?stop a gtfs:Stop .
 OPTIONAL { ?stop dct:description ?stopDescription . }
 OPTIONAL { ?stop gtfs:wheelchairAccessible ?wheel }
 ?stop geo:lat ?stopLat .
 ?stop geo:long ?stopLong .
```

```
 FILTER (?stopLat > %LAT%) .
}
```

Listing B.4: Query 3 - Find the accessibility information for the stations, if available

```
SELECT * WHERE {
 ?stop a gtfs:Stop .
 ?stop gtfs:locationType ?location .
 OPTIONAL { ?stop dct:description ?stopDescription . }
 OPTIONAL { ?stop geo:lat ?stopLat ; geo:long ?stopLong . }
 OPTIONAL { ?stop gtfs:wheelchairAccessible ?wheel . }
 FILTER (?location=<http://transport.es/resource/LocationType/2>)
}
```

Listing B.5: Query 4 - List all agencies and their routes with some of their data

```
SELECT * WHERE {
 ?route a gtfs:Route .
 OPTIONAL { ?route gtfs:shortName ?routeShortName . }
 OPTIONAL { ?route gtfs:longName ?routeLongName . }
 OPTIONAL { ?route dct:description ?routeDescription . }
 ?route gtfs:agency ?agency .
 ?agency a gtfs:Agency .
 ?agency foaf:page ?agencyPage .
 ?agency foaf:name ?agencyName .
 OPTIONAL { ?agency foaf:phone ?agencyPhone . }
}
```

Listing B.6: Query 5 - Services that have been added on a specific day

```
SELECT * WHERE {
 ?service a gtfs:Service .
 ?service gtfs:serviceRule ?serviceRule .
 ?serviceRule a gtfs:CalendarDateRule .
 ?serviceRule dct:date ?date .
 ?serviceRule gtfs:dateAddition "true"^^xsd:boolean .
 FILTER(?date > %DATE%)
}
```

Listing B.7: Query 6 - Check the number of routes of a particular agency

```
SELECT (count(?route) as ?nRoutes) WHERE {
 ?route a gtfs:Route .
```

```
 ?route gtfs:agency ?agency .
 FILTER (?agency=%AGENCY%)
}
```

Listing B.8: Query 7 - List all wheelchair accessible stops along a particular route, with some of their additional data

```
SELECT DISTINCT ?routeShortName ?routeDescription ?tripShortName
?stopDescription ?stopLat ?stopLong WHERE {
 ?route a gtfs:Route .
 OPTIONAL { ?route gtfs:shortName ?routeShortName . }
 OPTIONAL { ?route dct:description ?routeDescription . }
 ?trip a gtfs:Trip .
 OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
 ?trip gtfs:service ?service .
 ?trip gtfs:route ?route .
 ?stopTime a gtfs:StopTime .
 ?stopTime gtfs:trip ?trip .
 ?stopTime gtfs:stop ?stop .
 ?stop a gtfs:Stop .
 OPTIONAL { ?stop dct:description ?stopDescription . }
 OPTIONAL { ?stop geo:lat ?stopLat ; geo:long ?stopLong . }
 ?stop gtfs:wheelchairAccessible gtfsaccessible:1 .
 FILTER (?route=%ROUTE%)
}
```

Listing B.9: Query 8 - List the routes and their related trips, services, stops and stop times with some of their additional data, if available

```
SELECT * WHERE {
 ?route a gtfs:Route .
 OPTIONAL { ?route gtfs:shortName ?routeShortName . }
 OPTIONAL { ?route dct:description ?routeDescription . }
 ?trip a gtfs:Trip .
 OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
 ?trip gtfs:service ?service .
 ?trip gtfs:route ?route .
 ?stopTime a gtfs:StopTime .
 ?stopTime gtfs:trip ?trip .
 ?stopTime gtfs:stop ?stop .
 ?stop a gtfs:Stop .
 OPTIONAL {?stop dct:description ?stopDescription . }
```

```
 ?service a gtfs:Service .
 ?service gtfs:serviceRule ?serviceRule .
}
```

Listing B.10: Query 9 - Trips and associated shapes where lat is bigger than its average and some of their additional data

```
SELECT * WHERE {
 ?trip a gtfs:Trip .
 OPTIONAL { ?trip gtfs:shortName ?tripShortName . }
 ?trip gtfs:service ?service .
 ?trip gtfs:route ?route .
 ?trip gtfs:shape ?shape .
 ?shape a gtfs:Shape .
 ?shape geo:lat ?lat .
 FILTER (?lat > %LAT%)
}
```

Listing B.11: Query 10 - Number of trips that have a duration over 30 minutes

```
SELECT (count(distinct ?trip) as ?count) WHERE {
 ?trip a gtfs:Trip .
 ?stopTime a gtfs:StopTime .
 ?stopTime gtfs:trip ?trip .
 ?stopTime gtfs:departureTime ?departureTime .
 FILTER (?departureTime >= "00:30:00"^^xsd:duration)
}
```

Listing B.12: Query 11 - Trips that are available on a certain date and some of their additional data

```
SELECT * WHERE {
 ?service a gtfs:Service .
 ?service gtfs:serviceRule ?calendarRule .
 ?trip gtfs:service ?service .
 ?calendarRule a gtfs:CalendarRule .
 ?calendarRule schema:startDate ?startDate .
 ?calendarRule schema:endDate ?endDate .
 FILTER (?startDate <%DATE%) .
 FILTER (?endDate > %DATE%) .
 FILTER NOT EXISTS {
  ?service gtfs:serviceRule ?calendarDateRule .
  ?calendarDateRule a gtfs:CalendarDateRule .
```

```
  ?calendarDateRule dct:date %DATE% .
  ?calendarDateRule gtfs:dateAddition "false"^^xsd:boolean
 }
}
```

Listing B.13: Query 12 - Number of stops that are wheelchair-accessible grouped by route and some of their additional data

```
SELECT ?longName (count(?name) as ?count) WHERE {
 ?route a gtfs:Route .
 ?route gtfs:longName ?longName .
 ?trip a gtfs:Trip .
 ?trip gtfs:route ?route .
 ?stopTime a gtfs:StopTime .
 ?stopTime gtfs:trip ?trip .
 ?stopTime gtfs:stop ?stop .
 ?stop a gtfs:Stop .
 ?stop foaf:name ?name .
 ?stop gtfs:wheelchairAccessible gtfsaccessible:1 .
}
GROUP BY ?longName
```

Listing B.14: Query 13 - All the accesses of the stations

```
SELECT * WHERE {
 ?stop a gtfs:Stop .
 ?stop gtfs:parentStation ?parStation .
 OPTIONAL {?stop foaf:name ?accName} .
 ?stop gtfs:locationType gtfslocation:2 .
 ?parStation a gtfs:Stop .
 ?parStation foaf:name ?name
}
```

Listing B.15: Query 14 - All stops times and their related routes and stops order by their sequence

```
SELECT * WHERE {
 ?stopTime a gtfs:StopTime .
 ?stopTime gtfs:trip ?trip .
 ?stopTime gtfs:stop ?stop .
 ?stopTime gtfs:stopSequence ?sequence .
 ?stop a gtfs:Stop .
 ?trip a gtfs:Trip .
```

```
 ?trip gtfs:route ?route .
 OPTIONAL {?stop foaf:name ?stopName}
} ORDER BY ?sequence
```

Listing B.16: Query 15 - Everything that contains a specific string in the object placeholder (any property)

```
SELECT * WHERE {
 ?stop a gtfs:Stop .
 ?stop ?p ?str .
 FILTER regex (?str, %STRING%)
}
```

Listing B.17: Query 16 - For all the routes, all the calendar changes during a specific month

```
SELECT * WHERE {
 ?trip a gtfs:Trip .
 ?trip gtfs:service ?service .
 ?trip gtfs:route ?route .
 ?service a gtfs:Service .
 ?service gtfs:serviceRule ?serviceRule .
 ?serviceRule a gtfs:CalendarDateRule .
 ?serviceRule  dct:date ?servDate .
 ?serviceRule  gtfs:dateAddition "true"^^xsd:boolean .
 FILTER (?servDate >= %DATE1%) .
 FILTER (?servDate <= '%DATE2%) .
}
```

Listing B.18: Query 17 - Trips with their start and end time of the frequencies and associated routes

```
SELECT ?routeName ?routeType ?trip ?startTime ?endTime WHERE {
 ?trip a gtfs:Trip .
 ?trip gtfs:route ?route .
 ?frequency a gtfs:Frequency .
 ?frequency gtfs:startTime ?startTime .
 ?frequency gtfs:endTime ?endTime .
 ?frequency gtfs:trip ?trip .
 ?route a gtfs:Route .
 ?route gtfs:shortName ?routeName .
 ?route gtfs:routeType ?routeType .
}
```

Listing B.19: Query 18 - All routes that have trips on Sunday

```
SELECT * WHERE {
 ?service a gtfs:Service .
 ?service gtfs:serviceRule ?serviceRule .
 ?serviceRule a gtfs:CalendarRule .
 ?serviceRule gtfs:sunday "true"^^xsd:boolean .
 ?trip gtfs:service ?service .
 ?trip gtfs:route ?route  .
 { ?route gtfs:longName ?longName } UNION
 { ?route gtfs:shortName ?shortName }  .
}
```

ANNEX C

GTFS-Madrid-Bench Mappings

Listing C.1: Prefixes

```
prefixes:
  rr: http://www.w3.org/ns/r2rml#
  foaf: http://xmlns.com/foaf/0.1/
  xsd: http://www.w3.org/2001/XMLSchema#
  rdfs: http://www.w3.org/2000/01/rdf-schema#
  dc: http://purl.org/dc/elements/1.1/
  rev: http://purl.org/stuff/rev#
  gtfs: http://vocab.gtfs.org/terms#
  geo: http://www.w3.org/2003/01/geo/wgs84_pos#
  schema: http://schema.org/
  dct: http://purl.org/dc/terms/
  rml: http://semweb.mmlab.be/ns/rml#
  ql: http://semweb.mmlab.be/ns/ql#
  rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
  mad: http://transport.linkeddata.es/madrid/metro/
  gtfsres: http://transport.linkeddata.es/resource/
  wheelchair: gtfsres:WheelchairBoardingStatus/
  mad: http://transport.linkeddata.es/madrid/metro/
```

Listing C.2: Routes TripleMap

```
routes:
 sources:
  - [ROUTES.format]
 s: mad:routes/$(route_id)
 po:
```

```
  - [a, gtfs:Route]
  - [gtfs:shortName, $(route_short_name)]
  - [gtfs:longName, $(route_long_name)]
  - [dct:description, $(route_desc)]
  - [gtfs:routeType, gtfsres:RouteType/$(route_type)~iri]
  - [gtfs:routeUrl, $(route_url)~iri]
  - [gtfs:color, $(route_color)]
  - [gtfs:textColor, $(route_text_color)]
  - p: gtfs:agency
    o:
      - mapping: agency
        condition:
          function: equal
          parameters:
            - [str1, $(agency_id)]
            - [str2, $(agency_id)]
```

Listing C.3: Calendar_Date TripleMap

```
calendar_rules:
 sources:  - [CALENDAR.format]
 s: mad:calendar_rules/$(service_id)
 po:
  - [a, gtfs:CalendarRule]
  - [gtfs:monday, $(monday), xsd:boolean]
  - [gtfs:tuesday, $(tuesday), xsd:boolean]
  - [gtfs:wednesday, $(wednesday), xsd:boolean]
  - [gtfs:thursday, $(thursday), xsd:boolean]
  - [gtfs:friday, $(friday), xsd:boolean]
  - [gtfs:saturday, $(saturday), xsd:boolean]
  - [gtfs:sunday, $(sunday), xsd:boolean]
  - [schema:startDate, $(start_date), xsd:date]
  - [schema:endDate, $(end_date), xsd:date]
```

Listing C.4: Service_Calendar_Date TripleMap

```
services2:
 sources:
  - [CALENDAR_DATES.format]
 s: mad:services/$(service_id)
 po:
  - [a, gtfs:Service]
```

```
  - p: gtfs:serviceRule
    o:
      - mapping: calendar_date_rules
        condition:
          function: equal
          parameters:
            - [str1, $(service_id)]
            - [str2, $(service_id)]
```

Listing C.5: Agency TripleMap

```
agency:
 sources:
    - [AGENCY.format]
 s: mad:agency/$(agency_id)
 po:
   - [a, gtfs:Agency]
   - [foaf:page, $(agency_url)~iri]
   - [foaf:name,$(agency_name)]
   - [gtfs:timeZone,$(agency_timezone)]
   - [dct:language,$(agency_lang)]
   - [foaf:phone,$(agency_phone)]
   - [gtfs:fareUrl,$(agency_fare_url)~iri]
```

Listing C.6: Calendar_Date_Rules TripleMap

```
calendar_date_rules:
 sources:
  - [CALENDAR_DATES.format]
 s: mad:calendar_date_rule/$(service_id)-$(date)
 po:
  - [a, gtfs:CalendarDateRule]
  - [dct:date, $(date), xsd:date]
  - [gtfs:dateAddition, $(exception_type), xsd:boolean]
```

Listing C.7: Stop_Times TripleMap

```
stoptimes:
 sources:
  - [STOP_TIMES.format]
 s: mad:metro/stoptimes/$(trip_id)-$(stop_id)-$(arrival_time)
 po:
```

```
  - [a, gtfs:StopTime]
  - [gtfs:arrivalTime, $(arrival_time),xsd:duration]
  - [gtfs:departureTime, $(departure_time),xsd:duration]
  - [gtfs:stopSequence, $(stop_sequence),xsd:integer]
  - [gtfs:headsign, $(stop_headsign)]
  - [gtfs:pickupType, gtfsres:PickupType/$(pickup_type)~iri]
  - [gtfs:dropOffType, gtfsres:DropOffType/$(drop_off_type)~iri]
  - [gtfs:distanceTraveled, $(shape_dist_traveled)]
  - p: gtfs:trip
    o:
      - mapping: trips
        condition:
          function: equal
          parameters:
            - [str1, $(trip_id)]
            - [str2, $(trip_id)]
  - p: gtfs:stop
    o:
      - mapping: stops
        condition:
          function: equal
          parameters:
            - [str1, $(stop_id)]
            - [str2, $(stop_id)]
```

Listing C.8: Frequencies TripleMap

```
frequencies:
 sources:
  - [FREQUENCIES.format]
 s: mad:frequency/$(trip_id)-$(start_time)
 po:
  - [a, gtfs:Frequency]
  - [gtfs:startTime,$(start_time)]
  - [gtfs:endTime,$(end_time)]
  - [gtfs:headwaySeconds,$(headway_secs),xsd:integer]
  - [gtfs:exactTimes,$(exact_times),xsd:boolean]
  - p: gtfs:trip
    o:
      - mapping: trips
        condition:
          function: equal
```

```
          parameters:
            - [str1, $(trip_id)]
            - [str2, $(trip_id)]
```

Listing C.9: Trips TripleMap

```
trips:
 sources:
  - [TRIPS.format]
 s: mad:trips/$(trip_id)
 po:
  - [a, gtfs:Trip]
  - [gtfs:headsign, $(trip_headsign)]
  - [gtfs:shortName, $(trip_short_name)]
  - [gtfs:direction, $(direction_id)]
  - [gtfs:block, $(block_id)]
  - [gtfs:wheelchairAccessible,wheelchair:$(wheelchair_accessible)]
  - p: gtfs:service
    o:
      - mapping: services1
        condition:
          function: equal
          parameters:
            - [str1, $(service_id)]
            - [str2, $(service_id)]
      - mapping: services2
        condition:
          function: equal
          parameters:
            - [str1, $(service_id)]
            - [str2, $(service_id)]
  - p: gtfs:route
    o:
      - mapping: routes
        condition:
          function: equal
          parameters:
            - [str1, $(route_id)]
            - [str2, $(route_id)]
  - p: gtfs:shape
    o:
      - mapping: shapes
```

```
        condition:
          function: equal
          parameters:
            - [str1, $(shape_id)]
            - [str2, $(shape_id)]
```

Listing C.10: Feed_Info TripleMap

```
feed:
 sources:
  - [FEED_INFO.format]
 s: mad:feed/$(feed_publisher_name)
 po:
  - [a, gtfs:Feed]
  - [dct:publisher,$(feed_publisher_name)]
  - [foaf:page,$(feed_published_url)~iri]
  - [dct:language,$(feed_lang)]
  - [schema:startDate,$(feed_start_date), xsd:date]
  - [schema:endDate,$(feed_end_date), xsd:date]
  - [schema:version,$(feed_version)]
```

Listing C.11: Stops TripleMap

```
stops:
 sources:
  - [STOPS.format]
 s: mad:stops/$(stop_id)
 po:
  - [a,gtfs:Stop]
  - [gtfs:code,$(stop_code)]
  - [dct:identifier,$(stop_id)]
  - [foaf:name,$(stop_name)]
  - [dct:description,$(stop_desc)]
  - [geo:lat,$(stop_lat),xsd:double]
  - [geo:long,$(stop_lon),xsd:double]
  - [gtfs:zone,$(zone_id)]
  - [foaf:page,$(stop_url)~iri]
  - [gtfs:locationType, gtfsres:LocationType/$(location_type)~iri]
  - [gtfs:timeZone,$(stop_timezone)]
  - [gtfs:wheelchairAccessible,wheelchair:$(wheelchair_boarding)~iri]
  - p: gtfs:parentStation
    o:
```

```
      - mapping: stops
        condition:
          function: equal
          parameters:
            - [str1, $(parent_station)]
            - [str2, $(stop_id)]
```

Listing C.12: Shapes TripleMap

```
shapes:
 sources:
  - [SHAPES.format]
 s: mad:shape/$(shape_id)-$(shape_pt_sequence)
 po:
  - [a, gtfs:Shape]
  - [geo:lat,$(shape_pt_lat),xsd:double]
  - [geo:long,$(shape_pt_lon),xsd:double]
  - [gtfs:pointSequence,$(shape_pt_sequence)]
  - [gtfs:distanceTraveled,$(shape_dist_traveled)]
```

Listing C.13: Service_Calendar TripleMap

```
services1:
 sources:
  - [CALENDAR.format]
 s: mad:services/$(service_id)
 po:
  - [a, gtfs:Service]
  - p: gtfs:serviceRule
    o:
      - mapping: calendar_rules
        condition:
          function: equal
          parameters:
            - [str1, $(service_id)]
            - [str2, $(service_id)]
```

ANNEX D

Query complexity for Morph-CSV

Query	Query characteristics	Constraints		# Sources
		Integrity	Domain	
Madrid-GTFS-Bench				
Q1	4 TP	-	3 DataType, 4 Sub	1
Q2	5 TP, 2 OPT, 1 Filter	1 INDEX	3 DataType, 5 Sub	1
Q3	5 TP, 3 OPT, 1 Filter	1 INDEX	4 DataType, 5 Sub	1
Q4	9 TP, 1 Join, 4 OPT	2 PK, 1 FK	7 Sub	2
Q5	5 TP, 2 Join, 1 Filter	2 PK	2 DataType, 2 Sub	2
Q6	3 TP, 1 Join, 1 Filter	2 PK, 1 FK	-	2
Q7	15 TP, 5 Join, 5 OPT, 1 Filter	6 PK, 5 FK	3 DataType, 8 Sub	6
Q8	14 TP, 4 Join, 3 OPT	6 PK, 5 FK	3 DataType, 8 Sub	6
Q9	7 TP, 5 Join, 1 OPT, 1 Filter	5 PK, 3 FK	2 DataType, 3 Sub	5
Q10	4 TP, 1 Join, 1 Filter	2 PK, 1 FK	2 Sub	2
Q11	10 TP, 3 Join, 3 Filter (1 not exists)	3 PK, 2 FK	2 DataType, 2 Sub	3
Q12	10 TP, 3 Joins	4 PK, 3 FK	1 DataType, 4 Sub	4
Q13	6 TP, 1 Join, 1 OPT	1 PK, 1 FK	1 DataType, 3 Sub	1
Q14	8 TP, 3 Join, 1 OPT	4 PK, 3 FK	1 DataType, 3 Sub	3
Q15	3 TP, 1 Filter	1 PK, 1 FK	4 DataType, 11 Sub	1
Q16	8 TP, 3 Join, 2 Filter	4 PK, 2 FK	2 DataType, 2 Sub	3
Q17	9 TP, 2 Join	3 PK, 2 FK	1 DataType, 4 Sub	3
Q18	8 TP, 1Union, 3 Join	4 PK, 3 FK	1 DataType, 3 Sub	4
Bio2RDF				
Q1	4 TP	-	3 Sub	1
Q2	4 TP, 1 Join, 1 Filter	1 PK, 1 INDEX	7 Sub	2
Q3	4 TP, 1 Join	1 PK, 3 INDEX	5 Sub	3
Q4	4 TP, 1 Join	1 PK, 1 INDEX	7 Sub	2
Q5	5 TP, 1 Join	1 PK, 2 INDEX	6 Sub	2
Q6	4 TP	-	2 Sub	1
Q7	6 TP, 1 Join, 2 Filter	1 PK	1 DataType, 4 Sub, 1 Create	1
BSBM				
Q1	5 TP, 3 Join, 1 Filter	3 PK, 2FK	7 DataType, 1 Sub	3
Q2	15 TP, 3 Join, 3 OPT	4 PK, 3 FK	10 DataType, 12 Sub	4
Q3	7 TP, 3 Join, 2 Filter, 1 OPT	3 PK, 2FK	8 DataType, 3 Sub	3
Q4	12 TP, 1 Union, 6 Join, 2 Filter	3 PK, 2FK	2 DataType, 4 Sub	2
Q5	7 TP, 2 Join, 2 Filter	2 PK, 1FK	6 DataType, 3 Sub	2
Q6	2 TP, 1 Filter	-	1 Sub	1
Q7	14 TP, 5 Join, 1 Filter, 2 OPT	5 PK, 4 FK	11 DataType, 2 Sub	5
Q8	10 TP, 2 Join, 4 OPT	3 PK, 2 FK	8 DatatType, 8 Sub	3
Q9	DESCRIBE, 1 TP	-	-	1
Q10	7 TP, 3 Join, 2 Filter	3 PK, 3 FK	7 DatatType, 2 Sub	3
Q11	2 TP, 1 Union	11 PK, 11 FK	29 DataType, 53 Sub	11
Q12	CONSTRUCT, 9 TP, 2 Join	3 PK, 2 FK	6 DataType, 7 Sub	3

Table D.1: Query features of the evaluation of Morph-CSV. Domain constraints are described based on the function performed by Morph-CSV and reflect the number of the columns where that functions has been applied. Improvement functions (duplicates, source selection) are always applied.

ANNEX E

Completeness Morph-CSV

Engines/Queries	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q9	Q12	Q13	Q14	Q17	Total
GTFS-1													
Virtuoso	58540	765	765	13	28	1	2	151439	6	734	2364	855	156972
Morph-RDB	58540	765	-	13	-	1	2	151439	6	734	2364	855	156179
Morph-CSV & Morph-RDB	58540	765	-	13	-	1	2	151439	6	734	2364	855	156179
Ontop	58540	0	765	13	0	-	0	0	-	734	2364	855	4731
Morph-CSV & Ontop	58540	765	765	13	28	-	2	151439	-	734	2364	855	156965
GTFS-10													
Virtuoso	353660	6312	4207	130	350	1	67	718317	130	2650	23640	8550	764354
Morph-RDB	353660	6312	-	130	-	1	67	*timeout*	130	2650	23640	8550	41480
Morph-CSV & Morph-RDB	353660	6312	-	130	-	1	67	718317	130	2650	23640	8550	759797
Ontop	353660	0	4207	130	0	-	0	0	-	2650	23640	8550	39177
Morph-CSV & Ontop	353660	6312	4207	130	350	-	67	718317	-	2650	23640	8550	764223
GTFS-100													
Virtuoso	3536600	63100	42067	1300	3500	1	67	7183874	1300	26500	236400	85500	7643609
Morph-RDB	3536600	63100	-	1300	-	1	67	*timeout*	1300	26500	236400	85500	414168
Morph-CSV & Morph-RDB	3536600	63100	-	1300	-	1	67	*timeout*	1300	26500	236400	85500	414168
Ontop	3536600	0	42067	1300	0	-	0	0	-	26500	236400	85500	391767
Morph-CSV & Ontop	3536600	63100	42067	1300	*timeout*	-	67	*timeout*	-	26500	236400	85500	454934
GTFS-1000													
Virtuoso	35366000	1261368	420667	13000	35000	1	69	19077083	13000	420666	2364000	855000	24459854
Morph-RDB	*timeout*	1261368	-	13000	-	1	69	*timeout*	13000	420666	2364000	855000	4927104
Morph-CSV & Morph-RDB	35366000	1261368	-	13000	-	1	69	*timeout*	13000	420666	2364000	855000	4927104
Ontop	*timeout*	0	420667	13000	0	-	0	0	-	420666	2364000	855000	4073333
Morph-CSV & Ontop	*timeout*	1261368	420667	13000	*timeout*	-	69	*timeout*	-	420666	2364000	855000	5334770

Table E.1: Query completeness over multiple sizes of a GTFS dataset (the number indicates the scale factor: 1, 10, 100 and 100). The absence of a value means that the OBDA engine does not support the features of the SPARQL query.

Engines/Queries	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total
Morph-RDB	0	0	0	0	0	0	0	0
Morph-CSV + Morph-RDB	1000	1190181	10	102594	200	28224	¿10000	¿1422209
Ontop	0	0	0	0	0	0	0	0
Morph-CSV + Ontop	1000	1190181	10	102594	200	28224	13481	1335690

Table E.2: Query completeness over of Bio2RDF tabular dataset.

Engines/Queries	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q12	Total
45K												
Virtuoso	10	19672	10	10	5	3	580691	20	450000	10	900000	1950431
Morph-RDB	10	19672	10	10	0	3	580691	20	450000	10	900000	1950426
Morph-CSV & Morph-RDB	10	19672	10	10	5	3	580691	20	450000	10	900000	1950431
Ontop	10	-	10	10	0	-	-	-	-	0	-	30
Morph-CSV & Ontop	10	-	10	10	5	-	-	-	-	10	-	45
90K												
Virtuoso	10	38665	10	10	5	5	1161448	20	900000	10	1800000	3900183
Morph-RDB	10	38665	10	10	0	5	1161448	20	900000	10	1800000	3900178
Morph-CSV & Morph-RDB	10	38665	10	10	5	5	1161448	20	900000	10	1800000	3900183
Ontop	10	-	10	10	0	-	-	-	-	0	-	30
Morph-CSV & Ontop	10	-	10	10	5	-	-	-	-	10	-	45
180K												
Virtuoso	10	69434	10	10	5	9	2168792	20	1800000	10	3600000	7638300
Morph-RDB	10	*timeout*	10	10	0	9	2168792	20	1800000	10	3600000	7568861
Morph-CSV & Morph-RDB	10	69434	10	10	5	9	2168792	20	1800000	10	3600000	7638295
Ontop	*timeout*	-	10	*timeout*	0	-	-	-	-	0	-	10
Morph-CSV & Ontop	10	-	10	10	5	-	-	-	-	10	-	45
360K												
Virtuoso	10	137359	10	10	5	18	4337584	20	3600000	10	7200000	15275026
Morph-RDB	10	*timeout*	10	10	0	18	*timeout*	20	3600000	10	*timeout*	3600078
Morph-CSV & Morph-RDB	10	137359	10	10	*timeout*	18	*timeout*	20	3600000	10	*timeout*	3737437
Ontop	*timeout*	-	10	*timeout*	0	-	-	-	-	0	-	10
Morph-CSV & Ontop	10	-	10	10	*timeout*	-	-	-	-	10	-	40

Table E.3: Query completeness over multiple sizes of a BSBM dataset. The absence of a value means that the OBDA engine does not support the features of the SPARQL query.

ANNEX F

Detailed Loading Times for Morph-CSV

Step/Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Morph-CSV^{-}
GTFS-1																			
Selection	0.370	0.387	0.374	0.376	0.381	0.368	0.381	0.386	0.390	0.378	0.377	0.373	0.430	0.396	0.363	0.370	0.389	0.379	0.410
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	0.286	0.065	0.062	0.113	0.114	0.106	0.337	0.359	0.464	0.126	0.169	0.226	0.073	0.246	0.069	0.205	0.169	0.226	0.772
Creation & Load	0.345	0.075	0.074	0.063	0.059	0.051	0.176	0.182	0.381	0.090	0.084	0.130	0.064	0.141	0.055	0.087	0.091	0.101	0.612
M. Translation	0.506	0.521	0.532	0.500	0.525	0.524	0.536	0.545	0.532	0.535	0.523	0.509	0.540	0.581	0.498	0.530	0.543	0.519	0.635
Total	1.507	1.049	1.041	1.052	1.080	1.049	1.430	1.472	1.767	1.129	1.153	1.237	1.107	1.365	0.985	1.192	1.193	1.224	2.430
GTFS-10																			
Selection	0.998	1.005	1.033	1.059	1.010	1.012	1.023	1.041	1.004	1.030	1.021	0.994	1.006	1.019	1.009	1.019	1.013	1.028	1.0717
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	1.201	0.139	0.147	0.123	0.130	0.125	0.524	0.504	1.378	0.247	0.193	0.401	0.143	0.408	0.176	0.234	0.220	0.239	2.1550
Creation & Load	1.296	0.139	0.137	0.067	0.067	0.060	0.475	0.467	2.214	0.323	0.116	0.460	0.160	0.453	0.221	0.117	0.198	0.117	4.2042
M. Translation	0.509	0.536	0.525	0.531	0.524	0.507	0.522	0.530	0.522	0.516	0.536	0.538	0.503	0.577	0.513	0.522	0.536	0.542	0.6442
Total	4.004	1.820	1.842	1.780	1.732	1.704	2.545	2.542	5.119	2.116	1.866	2.393	1.811	2.458	1.920	1.892	1.967	1.926	8.0750
GTFS-100																			
Selection	7.181	7.249	7.257	7.294	7.195	7.254	7.209	7.305	7.566	7.581	7.333	7.274	7.314	7.242	7.328	7.373	7.241	7.276	8.156
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	11.434	0.789	0.941	0.259	0.316	0.252	1.946	1.955	11.446	1.201	0.280	1.858	0.690	1.899	1.108	0.441	0.666	0.459	16.411
Creation & Load	11.812	0.751	0.679	0.075	0.120	0.085	3.369	3.811	35.058	2.435	0.285	3.839	0.981	3.038	1.244	0.346	1.093	0.296	92.785
M. Translation	0.531	0.519	0.507	0.507	0.520	0.532	0.526	0.571	0.540	0.538	0.556	0.534	0.524	0.519	0.534	0.538	0.533	0.578	0.761
Total	30.959	9.308	9.384	8.135	8.151	8.123	13.050	13.642	54.609	11.755	8.454	13.504	9.509	12.698	10.213	8.698	9.533	8.611	118.113
GTFS-1000																			
Selection	76.815	73.390	71.395	71.686	71.770	72.521	72.749	73.408	78.764	73.982	72.248	73.084	71.511	73.874	73.003	71.692	72.449	71.849	72.920
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Data Preparation	140.784	8	6.826	0.657	0.737	0.441	19.167	18.768	126.915	12.364	1.318	17.717	6.718	18.356	10.506	1.505	4.200	1.552	184.215
Creation & Load	123.770	6.843	7.123	0.239	0.665	0.271	52.349	52.294	3121.927	66.614	1.620	69.820	9.169	48.674	13.434	2.028	10.732	1.905	420.123
M. Translation	0.546	0.521	0.528	0.541	0.533	0.532	0.541	0.550	0.557	0.524	0.535	0.532	0.511	0.532	0.557	0.551	0.528	0.541	0.607
Total	341.915	88.795	85.871	73.123	73.705	73.766	144.808	145.021	3328.163	153.485	75.722	161.153	87.909	141.437	97.500	75.776	87.909	75.847	677.865

Table F.1: Detailed results of Morph-CSV over GTFS-Madrid-Bench. As the input sources of this benchmark are extracted from a well-formed data model, the normalization step is not performed.

Step/Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Morph-CSV$^-$
BSBM-45K													
Selection	0.004	0.004	0.003	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.007
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	3.291	5.699	3.424	3.507	3.425	2.318	20.750	28.657	13.397	7.433	43.732	8.443	43.452
Creation & Load	3.044	4.355	2.707	3.049	2.648	0.102	5.819	14.168	1.223	3.894	27.034	5.151	28.036
M. Translation	0.514	0.564	0.498	0.522	0.505	0.524	0.564	0.551	0.519	0.569	0.572	0.541	0.547
Total	6.853	10.622	6.633	7.082	6.581	2.949	27.137	43.380	15.142	11.900	71.342	14.139	72.043
BSBM-90K													
Selection	0.004	0.004	0.004	0.004	0.004	0.003	0.004	0.004	0.004	0.004	0.005	0.004	0.005
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	6.882	10.022	6.334	6.378	7.167	3.149	42.191	61.591	24.135	13.907	85.432	16.059	208.798
Creation & Load	6.118	8.667	5.529	5.711	6.067	0.168	12.668	30.614	2.227	8.003	56.776	10.638	58.119
M, Translation	0.540	0.525	0.516	0.509	0.527	0.509	0.546	0.551	0.512	0.519	0.569	0.546	0.574
Total	13.544	19.219	12.384	12.602	13.764	3.830	55.409	92.761	26.877	22.434	142.783	27.247	267.496
BSBM-180K													
Selection	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.003	0.004	0.005	0.004	0.005
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	12.675	19.946	11.978	12.459	11.969	5.255	83.486	122.173	47.833	29.420	185.650	34.682	339.254
Creation & Load	10.740	15.848	11.134	12.542	11.450	0.268	25.693	67.677	5.243	15.268	137.522	21.411	141.737
M. Translation	0.534	0.508	0.545	0.513	0.532	0.514	0.584	0.554	0.553	0.574	0.607	0.599	0.606
Total	23.953	36.307	23.661	25.518	23.955	6.041	109.767	190.408	53.634	45.266	323.784	56.695	481.602
BSBM-360K													
Selection	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.005	0.004	0.005
Normalization	-	-	-	-	-	-	-	-	-	-	-	-	-
Preparation	23.846	43.880	24.970	24.597	23.670	10.401	198.975	293.087	110.852	57.878	415.052	66.759	578.798
Creation & Load	26.804	44.031	24.667	30.709	23.089	0.435	55.623	136.090	10.037	32.036	262.529	44.716	260.139
M. Translation	0.545	0.571	0.536	0.533	0.540	0.494	0.580	0.583	0.503	0.563	0.632	0.540	0.578
Total	51.199	88.486	50.176	55.842	47.302	11.333	255.183	429.765	121.396	90.481	678.218	112.019	839.521

Table F.2: Detailed results of Morph-CSV over BSBM. As the input sources of this benchmark are extracted from a well-formed relational database, the normalization step is not performed.

Step/Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Morph-CSV$^-$
Selection	3.705	3.749	3.777	3.714	3.732	3.787	3.719	3.775	3.712	3.632	48.088
Normalization	-	-	0.194	-	-	0.212	-	0.301	-	-	38.577
Preparation	0.903	123.798	0.580	7.414	0.628	126.852	8.457	2.660	5.531	0.555	253.461
Creation & Load	0.318	131.912	0.113	34.569	0.994	147.659	32.203	0.901	8.968	0.790	265.301
M. Translation	0.541	0.542	0.546	0.534	0.548	0.560	0.535	0.539	0.543	0.552	0.693
Total	5.467	260.000	5.211	46.231	5.901	279.071	44.915	8.176	18.756	5.529	606.121

Table F.3: Detailed results of Morph-CSV over Bio2RDF.

Bibliography

ACOSTA, M., VIDAL, M.E., LAMPO, T., CASTILLO, J. & RUCKHAUS, E. (2011). Anapsid: an adaptive query processing engine for sparql endpoints. In *International Semantic Web Conference*, 18–34, Springer. 87

ACOSTA, M., VIDAL, M.E. & SURE-VETTER, Y. (2017). Diefficiency metrics: measuring the continuous efficiency of query processing approaches. In *International Semantic Web Conference*, 3–19, Springer. 100

AHO, A.V., SETHI, R. & ULLMAN, J.D. (1986). Compilers, principles, techniques. *Addison Wesley*, **7**. 186

ALI, S.M.F. & WREMBEL, R. (2019). Towards a Cost Model to Optimize User-Defined Functions in an ETL Workflow Based on User-Defined Performance Metrics. In *Advances in Databases and Information Systems, ADBIS*. 186

ANASTASIA, D. (2020). High-quality knowledge graphs generation: R2rml and rml comparison, rules validation and inconsistency resolution. *Applications and Practices in Ontology Design, Extraction, and Reasoning*, **49**, 55. xxi, 22

ANGLES, R. & GUTIÉRREZ, C. (2016). Negation in SPARQL. In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, 2016*. 105

ARENAS, M., BERTAILS, A., PRUD'HOMMEAUX, E. & SEQUEDA, J. (2012). A Direct Mapping of Relational Data to RDF. W3C Recommendation, W3C, `https://www.w3.org/TR/rdb-direct-mapping/`. 33

ARENAS, M., BERTAILS, A., PRUD'HOMMEAUX, E. & SEQUEDA, J. (2013). A direct mapping of relational data to rdf, w3c recommendation 27 september 2012. 3, 47

AUER, S., DIETZOLD, S., LEHMANN, J., HELLMANN, S. & AUMUELLER, D. (2009). Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th international conference on World wide web*, ACM Press. 3, 47

AUER, S., KOVTUN, V., PRINZ, M., KASPRZIK, A., STOCKER, M. & VIDAL, M. (2018). Towards a knowledge graph for science. In *Proceedings of WIMS*. 170, 182

BARRASA, J., CORCHO, Ó. & GÓMEZ-PÉREZ, A. (2004). R2o, an extensible and semantically based database-to-ontology mapping language. SWDB. 3, 47

BEERI, C., BERNSTEIN, P.A. & GOODMAN, N. (1978). A sophisticate's introduction to database normalization theory. In *VLDB*. 132

BELLEAU, F., NOLIN, M.A., TOURIGNY, N., RIGAULT, P. & MORISSETTE, J. (2008). Bio2rdf: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, **41**, 706–716. 121, 151

BEN-KIKI, O., EVANS, C. & INGERSON, B. (2001). Yaml ain't markup language (yaml™) version 1.1. 23

BERSHAD, B.N., LAZOWSKA, E.D. & LEVY, H.M. (1988). Presto: A system for object-oriented parallel programming. *Software: Practice and Experience*, **18**, 713–732. 4

BITTON, D. & DEWITT, D.J. (1983). Duplicate record elimination in large data files. *ACM Trans. Database Syst.*, **8**. 178

BIZER, C. & SCHULTZ, A. (2009). The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, **5**, 1–24. 4, 34, 39, 107, 121, 139, 141, 163

BIZER, C. & SEABORNE, A. (2004). D2rq-treating non-rdf databases as virtual rdf graphs. In *Proceedings of the 3rd international semantic web conference (ISWC2004)*, vol. 2004, Proceedings of ISWC2004. 3

BIZER, C., HEATH, T. & BERNERS-LEE, T. (2011). Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, 205–227, IGI Global. 40

BOTOEVA, E., CALVANESE, D., COGREL, B., CORMAN, J. & XIAO, G. (2019). Ontology-based data access–beyond relational sources. *Intelligenza Artificiale*, **13**, 21–36. 32, 128

BRICKLEY, D., GUHA, R.V. & LAYMAN, A. (1999). Resource description framework (rdf) schema specification. 1

BRICKLEY, D., GUHA, R.V. & MCBRIDE, B. (2014). Rdf schema 1.1. *W3C recommendation*, **25**, 2004–2014. 1

BRYANT, M. (2017). Graphql for archival metadata: An overview of the ehri graphql api. In *2017 IEEE International Conference on Big Data (Big Data)*, 2225–2230, IEEE. 155

CALÌ, A., CALVANESE, D., DE GIACOMO, G. & LENZERINI, M. (2002). Data integration under integrity constraints. In *International Conference on Advanced Information Systems Engineering*, 262–279, Springer. 26

CALVANESE, D., COGREL, B., KOMLA-EBRI, S., KONTCHAKOV, R., LANTI, D., REZK, M., RODRIGUEZ-MURO, M. & XIAO, G. (2017). Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, **8**, 471–487. 3, 26, 31, 38, 54, 114, 121, 141, 169

CHAVES-FRAGA, D., PRIYATNA, F., PEREZ-SANTANA, I. & CORCHO, O. (2018). Virtual statistics knowledge graph generation from CSV files. In *Emerging Topics in Semantic Technologies: ISWC 2018 Satellite Events*, vol. 36 of *Studies on the Semantic Web*, 235–244, IOS Press. 47, 50, 51

CHAVES-FRAGA, D., ENDRIS, K.M., IGLESIAS, E., CORCHO, Ó. & VIDAL, M. (2019). What are the Parameters that Affect the Construction of a Knowledge Graph? In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, Springer. 87, 99, 105, 106, 169

CHAVES-FRAGA, D., PRIYATNA, F., CIMMINO, A., TOLEDO, J., RUCKHAUS, E. & CORCHO, O. (2020a). GTFS-Madrid-Bench: A Benchmark for Virtual Knowledge Graph Access in the Transport Domain. *Journal of Web Semantics*, **65**. 121, 139, 140, 146

CHAVES-FRAGA, D., RUCKHAUS, E., PRIYATNA, F., VIDAL, M.E. & CORCHO, O. (2020b). Enhancing OBDA Query Translation over Tabular Data with Morph-CSV. 108

CHAWATHE, S., GARCIA-MOLINA, H., HAMMER, J., IRELAND, K., PAPAKONSTANTINOU, Y., ULLMAN, J. & WIDOM, J. (1994). The tsimmis project: Integration of heterogenous information sources. In *Information Processing Society of Japan (IPSJ 1994)*. 12

CHEBOTKO, A., LU, S. & FOTOUHI, F. (2009). Semantics preserving SPARQL-to-SQL translation. *Data & Knowledge Engineering*, **68**, 973–1000. 3, 31, 40, 103, 108, 119, 159, 162, 169

CHORTARAS, A. & STAMOU, G. (2018a). D2rml: Integrating heterogeneous data and web services into custom rdf graphs. *Proceedings of the LDOW. CEUR, ceur-ws. org*, **2073**. 3, 47

CHORTARAS, A. & STAMOU, G. (2018b). Mapping diverse data to rdf in practice. In *International Semantic Web Conference*, 441–457, Springer. 38

CODD, E.F. (1979). Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, **4**, 397–434. 136

CORCHO, O., SANTANA-PÉREZ, I., LAFUENTE, H., PORTOLÉS, D., CANO, C., PERIS, A. & SUBERO, J.M. (2017). Publishing linked statistical data: Aragón, a case study. In *HybridSemStats@ ISWC*. 54

CORCHO, O., PRIYATNA, F. & CHAVES-FRAGA, D. (2019). Towards a New Generation of Ontology Based Data Access. *Semantic Web Journal*. 88, 103, 116, 145, 186

CYGANIAK, R., REYNOLDS, D. & TENNISON, J. (2012). The rdf data cube vocabulary, w3c recommendation 16 january 2014. *World Wide Web Consortium*. 54

CYGANIAK, R., WOOD, D. & LANTHALER, M. (2014). RDF 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium (W3C). 13, 33

DAS, S., SUNDARA, S. & CYGANIAK, R. (2012a). R2RML: RDB to RDF Mapping Language. W3C Recommendation, W3C, `http://www.w3.org/TR/r2rml/`. xix, 3, 13, 18, 19, 20, 30, 33, 38, 47, 157, 169, 172, 185

DAS, S., SUNDARA, S. & CYGANIAK, R. (2012b). RDB2RDF Implementation Report. W3C Note, W3C, `https://www.w3.org/TR/rdb2rdf-implementations/`. 33

DE MEESTER, B., DIMOU, A., VERBORGH, R. & MANNENS, E. (2016). An ontology to semantically declare and describe functions. In *ISWC*, 46–49, Springer. 26, 107, 116

DE MEESTER, B., MAROY, W., DIMOU, A., VERBORGH, R. & MANNENS, E. (2017). Declarative data transformations for Linked Data generation: the case of DBpedia. In *ESWC*, 33–48, Springer. 26, 52, 120, 125, 129, 133, 137, 185

DE MEESTER, B., SEYMOENS, T., DIMOU, A. & VERBORGH, R. (2019). Implementation-independent function reuse. *Future Generation Computer Systems*. 26

DEBRUYNE, C. & O'SULLIVAN, D. (2016). R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In *LDOW@ WWW*. 27, 116, 125, 185

DEN DUNNEN, J.T., DALGLEISH, R., MAGLOTT, D.R., HART, R.K., GREENBLATT, M.S., MCGOWAN-JORDAN, J., ROUX, A.F., SMITH, T., ANTONARAKIS, S.E., TASCHNER, P.E. *et al.* (2016). HGVS recommendations for the description of sequence variants: 2016 update. *Human mutation*, **37**. 187

DIMOU, A., VANDER SANDE, M., COLPAERT, P., VERBORGH, R., MANNENS, E. & VAN DE WALLE, R. (2014). RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *LDOW*. 3, 13, 20, 30, 38, 47, 93, 169, 171, 185

DOAN, A., HALEVY, A. & IVES, Z. (2012). *Principles of data integration*. Elsevier. 125

DUMONTIER, M., CALLAHAN, A., CRUZ-TOLEDO, J., ANSELL, P., EMONET, V., BELLEAU, F. & DROIT, A. (2014). Bio2rdf release 3: a larger connected network of linked data for the life sciences. In *Proceedings of the 2014 International Conference on Posters & Demonstrations Track-Volume 1272*, 401–404. 141

ELLIOTT, B., CHENG, E., THOMAS-OGBUJI, C. & OZSOYOGLU, Z.M. (2009). A complete translation from sparql into efficient sql. In *Proceedings of the 2009 International Database Engineering & Applications Symposium*, 31–42. 3

ENDRIS, K.M., GALKIN, M., LYTRA, I., MAMI, M.N., VIDAL, M.E. & AUER, S. (2017). MULDER: querying the linked data web by bridging RDF molecule templates. In *International Conference on Database and Expert Systems Applications*, 3–18, Springer. 32, 107

ENDRIS, K.M., ROHDE, P.D., VIDAL, M.E. & AUER, S. (2019). Ontario: Federated Query Processing Against a Semantic Data Lake. In *International Conference on Database and Expert Systems Applications*, 379–395, Springer. 4, 32, 34, 67, 107, 115, 120, 154, 166, 205

FACEBOOK, INC. (2018). GraphQL. `https://facebook.github.io/graphql/June2018/`, accessed: 2018-12-07. 51, 156

FERNÁNDEZ, J.D., MARTÍNEZ-PRIETO, M.A., GUTIÉRREZ, C., POLLERES, A. & ARIAS, M. (2013). Binary rdf representation for publication and exchange (hdt). *Journal of Web Semantics*, **19**, 22–41. 15

FIELDING, R.T. & TAYLOR, R.N. (2000). *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Doctoral dissertation. 51

GALHARDAS, H., FLORESCU, D., SHASHA, D., SIMON, E. & SAITA, C. (2001). Declarative data cleaning: Language, model, and algorithms. 26

GARCÍA-GONZÁLEZ, H., BONEVA, I., STAWORKO, S., LABRA-GAYO, J.E. & CUEVA LOVELLE, J.M. (2020). Shexml: improving the usability of heterogeneous data mapping languages for first-time users. *PeerJ Computer Science*. 25, 50

GOLSHAN, B., HALEVY, A., MIHAILA, G. & TAN, W.C. (2017). Data integration: After the teenage years. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 101–106. 125

GRUSER, J.R., RASCHID, L., VIDAL, M.E. & BRIGHT, L. (1998). Wrapper generation for web accessible data sources. In *Proceedings. 3rd IFCIS International Conference on Cooperative Information Systems (Cat. No. 98EX122)*, 14–23, IEEE. 2

GUERRERO, J.A. (2020). Physical knowledge graph design for efficient federated query processing. 154

GUPTA, S., SZEKELY, P.A., KNOBLOCK, C.A., GOEL, A., TAHERIYAN, M. & MUSLEA, M. (2012). Karma: A system for mapping structured sources into the semantic web. In *The Semantic Web: ESWC 2012 Satellite Events - ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012. Revised Selected Papers*, 430–434. 26

HAESENDONCK, G., MAROY, W., HEYVAERT, P., VERBORGH, R. & DIMOU, A. (2019). Parallel rdf generation from heterogeneous big data. In *Proceedings of the International Workshop on Semantic Big Data*, 1–6. 31

HALEVY, A., RAJARAMAN, A. & ORDILLE, J. (2006). Data integration: The teenage years. In *Proceedings of the 32nd international conference on Very large data bases*, 9–16. 125

HALEVY, A.Y. (2001). Answering queries using views: A survey. *The VLDB Journal*, **10**, 270–294. 17, 18

HALEVY, A.Y. (2018). Information integration. In *Encyclopedia of Database Systems, Second Edition*. 2

HARRIS, S. & SEABORNE, A. (2013). SPARQL 1.1 Query Language. Recommendation, World Wide Web Consortium (W3C). 13, 16, 23, 33

HARTIG, O. (2017). Foundations of rdf* and sparql*:(an alternative approach to statement-level metadata in rdf). In *AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017.*, vol. 1912, Juan Reutter, Divesh Srivastava. 40, 48

HASNAIN, A., MEHMOOD, Q., E ZAINAB, S.S., SALEEM, M., WARREN, C., ZEHRA, D., DECKER, S. & REBHOLZ-SCHUHMANN, D. (2017). BioFed: federated query processing over life sciences linked open data. *Journal of biomedical semantics*, **8**, 13. 34

HAUSENBLAS, M. & NADEAU, J. (2013). Apache drill: interactive ad-hoc analysis at scale. *Big data*, **1**, 100–104. 4

HEYVAERT, P., DIMOU, A., HERREGODTS, A.L., VERBORGH, R., SCHUURMAN, D., MANNENS, E. & DE WALLE, R.V. (2016). RMLEditor: A graph-based mapping editor for linked data mappings. In *The Semantic Web. Latest Advances and New Domains*, 709–723, Springer International Publishing. 50, 157

HEYVAERT, P., DE MEESTER, B., DIMOU, A. & VERBORGH, R. (2018). Declarative Rules for Linked Data Generation at your Fingertips! In *Proceedings of the 15th ESWC: Posters and Demos*. 21, 23, 47, 50, 68, 99

HEYVAERT, P., CHAVES-FRAGA, D., PRIYATNA, F., CORCHO, O., MANNENS, E., VERBORGH, R. & DIMOU, A. (2019). Conformance Test Cases for the RDF Mapping Language (RML). In *Iberoamerican Knowledge Graphs and Semantic Web Conference*, 162–173, Springer. 96

HOGAN, A., BLOMQVIST, E., COCHEZ, M., D'AMATO, C., DE MELO, G., GUTIERREZ, C., GAYO, J.E.L., KIRRANE, S., NEUMAIER, S., POLLERES, A., NAVIGLI, R., NGOMO, A.N., RASHID, S.M., RULA, A., SCHMELZEISEN, L., SEQUEDA, J.F., STAAB, S. & ZIMMERMANN, A. (2020). Knowledge graphs. *CoRR*, **abs/2003.02320**. 2

HOVLAND, D., LANTI, D., REZK, M. & XIAO, G. (2016). Obda constraints for effective query answering. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, 269–286, Springer. 32

IGLESIAS, E., JOZASHOORI, S., CHAVES-FRAGA, D., COLLARANA, D. & VIDAL, M.E. (2020). Sdm-rdfizer: An rml interpreter for the efficient creation of rdf knowledge graphs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 3039–3046. 132, 137, 197

IGLESIAS-MOLINA, A., CHAVES-FRAGA, D., PRIYATNA, F. & CORCHO, O. (2019). Enhancing the Maintainability of the Bio2RDF Project Using Declarative Mappings. In *Proceedings of the 12th International Conference on Semantic Web Applications and Tools for Healthcare and Life Sciences*. 140, 141, 151, 152, 154

JOZASHOORI, S. & VIDAL, M.E. (2019). MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, 58–75, Springer. 132, 137, 190

JOZASHOORI, S., CHAVES-FRAGA, D., IGLESIAS, E., VIDAL, M.E. & CORCHO, O. (2020). Funmap: Efficient execution of functional mappings for knowledge graph creation. In *Proceedings of the 19th International Semantic Web Conference*. 132, 137

JUNIOR, A.C., DEBRUYNE, C., BRENNAN, R. & O'SULLIVAN, D. (2016a). FunUL: a method to incorporate functions into uplift mapping languages. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, 267–275, ACM. 27, 137, 185, 186

JUNIOR, A.C., DEBRUYNE, C. & O'SULLIVAN, D. (2016b). Incorporating functions in mappings to facilitate the uplift of CSV files into RDF. In *ISWC*, 55–59, Springer. 27

KHAN, Y., ZIMMERMANN, A., JHA, A., GADEPALLY, V., D'AQUIN, M. & SAHAY, R. (2019). One Size Does Not Fit All: Querying Web Polystores. *IEEE Access*, **7**, 9598–9617. 32, 108

KNOBLOCK, C.A. & SZEKELY, P. (2015). Exploiting semantics for big data integration. *Ai Magazine*, **36**. 157

LANTI, D., REZK, M., XIAO, G. & CALVANESE, D. (2015). The npd benchmark: Reality check for obda systems. OpenProceedings. org. 4, 34, 39, 87, 91, 100, 107, 146

LANTI, D., XIAO, G. & CALVANESE, D. (2017). VIG: Data scaling for OBDA benchmarks. *Semantic Web*, 1–21. 5, 34, 88, 91, 92, 94

LEFRANÇOIS, M., ZIMMERMANN, A. & BAKERALLY, N. (2017). A SPARQL extension for generating RDF from heterogeneous formats. In *The Semantic Web*, 35–50, Springer International Publishing. 3, 25, 31, 38, 39, 47

LENZERINI, M. (2002). Data Integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, 233–246. 2, 11, 185, 188

MAALI, F., ERICKSON, J. & ARCHER, P. (2018). Data catalog vocabulary (dcat). w3c recommendation, 2014. 27

MAHMUD, S.M.H., HOSSIN, M., JAHAN, H., NOORI, S. & HOSSAIN, M. (2018). Csv2rdf: Generating rdf data from csv file using semantic web technologies. *Journal of Theoretical and Applied Information Technology*, **96**. 31

MAMI, M.N., GRAUX, D., SCERRI, S., JABEEN, H. & AUER, S. (2019a). Querying Data Lakes using Spark and Presto. In *International World Wide Web Conference*, 3574–3578, ACM. 4, 32, 108, 115

MAMI, M.N., GRAUX, D., SCERRI, S., JABEEN, H., AUER, S. & LEHMANN, J. (2019b). Squerall: virtual ontology-based access to heterogeneous and large data sources. In *International Semantic Web Conference*, 229–245, Springer. 4, 120, 141, 154, 205

MCGUINNESS, D.L., VAN HARMELEN, F. *et al.* (2004). OWL web ontology language overview. *W3C recommendation*, **10**, 2004. 1, 88

MICHEL, F., DJIMENOU, L., FARON-ZUCKER, C. & MONTAGNAT, J. (2015). Translation of relational and non-relational databases into RDF with xR2RML. In *11th International Conference on Web Information Systems and Technologies (WEBIST'15)*, 443–454. 3, 13, 23, 32, 38, 47, 52, 108, 185

MONTOYA, G., VIDAL, M.E., CORCHO, O., RUCKHAUS, E. & BUIL-ARANDA, C. (2012). Benchmarking federated SPARQL query engines: Are existing testbeds enough? In *International Semantic Web Conference*, 313–324, Springer. 104, 106

MORA, J. & CORCHO, O. (2013). Towards a systematic benchmarking of ontology-based query rewriting systems. In *International Semantic Web Conference*, 376–391, Springer. 87, 100

MORA, J., ROSATI, R. & CORCHO, O. (2014). kyrie2: Query rewriting under extensional constraints in \mathcal {ELHIO}. In *International Semantic Web Conference*, 568–583, Springer. 103, 139

MUKHIYA, S.K., RABBI, F., PUN, V.K.I., RUTLE, A. & LAMO, Y. (2019). A graphql approach to healthcare information exchange with hl7 fhir. *Procedia Computer Science*, **160**, 338–345. 155

NOY, N.F., GAO, Y., JAIN, A., NARAYANAN, A., PATTERSON, A. & TAYLOR, J. (2019). Industry-scale knowledge graphs: lessons and challenges. *Commun. ACM*, **62**. 170, 182

PÉREZ, J., ARENAS, M. & GUTIÉRREZ, C. (2009). Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, **34**, 16:1–16:45. 1, 13, 15, 16, 104

POGGI, A., LEMBO, D., CALVANESE, D., DE GIACOMO, G., LENZERINI, M. & ROSATI, R. (2008). Linking data to ontologies. In *Journal on data semantics X*, 133–173, Springer. 2, 12, 46, 54, 132

PRIYATNA, F., CORCHO, O. & SEQUEDA, J. (2014). Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph. In *23rd International Conference on WWW*. 3, 26, 31, 38, 54, 95, 108, 114, 120, 121, 139, 140, 141, 165, 166, 169

PRIYATNA, F., CHAVES-FRAGA, D., ALOBAID, A. & CORCHO, O. (2019). morph-GraphQL: GraphQL servers generation from r2rml mappings (s). In *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering*, KSI Research Inc. and Knowledge Systems Institute Graduate School. 52, 106, 163

PRUD'HOMMEAUX, E., LABRA GAYO, J.E. & SOLBRIG, H. (2014). Shape expressions: an rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, 32–40. 23, 25, 50

RAHM, E. & DO, H.H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, **23**. 26

RAJARAMAN, A.L.A., ORDILLE, J. *et al.* (1996). Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*. 12

RAKHMAWATI, N.A., SALEEM, M., LALITHSENA, S. & DECKER, S. (2014). Qfed: Query set for federated sparql query benchmark. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services*, 207–211. 34

RAMAN, V. & HELLERSTEIN, J.M. (2001). Potter's wheel: An interactive data cleaning system. In *VLDB*, vol. 1. 26

RÖDER, M., KUCHELEV, D. & NGONGA NGOMO, A.C. (2020). Hobbit: A platform for benchmarking big linked data. *Data Science*, **3**, 15–35. 34

RODRIGUEZ-MURO, M. & REZK, M. (2015). Efficient SPARQL-to-SQL with R2RML mappings. *Web Semantics*, **33**, 141–169. 53, 107, 120, 140

ROHDE, P.D. & VIDAL, M.E. (2020). Optimizing federated queries based on the physical design of a data lake. *arXiv preprint arXiv:2002.08102*. 154

ROTH, M.T. & SCHWARZ, P.M. (1997). Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *VLDB*, vol. 97, 25–29. 12

SCHMIDT, M., HORNUNG, T., LAUSEN, G. & PINKEL, C. (2009). Spˆ 2bench: a sparql performance benchmark. In *2009 IEEE 25th International Conference on Data Engineering*, 222–233, IEEE. 34

SCHMIDT, M., GÖRLITZ, O., HAASE, P., LADWIG, G., SCHWARTE, A. & TRAN, T. (2011). Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference*, 585–600, Springer. 34

SCHWARTE, A., HAASE, P., HOSE, K., SCHENKEL, R. & SCHMIDT, M. (2011). Fedx: Optimization techniques for federated query processing on linked data. In *International semantic web conference*, 601–616, Springer. 100

SCROCCA, M., COMERIO, M., CARENINI, A. & CELINO, I. (2020). Turning transport data to comply with eu standards while enabling a multimodal transport knowledge graph. In *International Semantic Web Conference*, 411–429, Springer. 31

SEQUEDA, J.F. & MIRANKER, D.P. (2013). Ultrawrap: SPARQL execution on relational data. *Web Semantics*, **22**, 19–39. 3, 31, 140

SEQUEDA, J.F., ARENAS, M. & MIRANKER, D.P. (2012). On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st international conference on World Wide Web*, ACM Press. 48

SEQUEDA, J.F., ARENAS, M. & MIRANKER, D.P. (2014). Obda: query rewriting or materialization? in practice, both! In *International Semantic Web Conference*, 535–551, Springer. 32

SHARAF, M.A., CHRYSANTHIS, P.K., LABRINIDIS, A. & PRUHS, K. (2008). Algorithms and metrics for processing multiple heterogeneous continuous queries. *ACM Transactions on Database Systems (TODS)*, **33**, 5. 100

ŞIMŞEK, U., KÄRLE, E. & FENSEL, D. (2019). RocketRML-A NodeJS implementation of a use-case specific RML mapper. In *Proceeding of the First International Workshop on Knowledge Graph Building*. 4, 30, 38, 39, 169, 170

SLEPICKA, J., YIN, C., SZEKELY, P.A. & KNOBLOCK, C.A. (2015). KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources. In *COLD*. 3, 27, 38, 47

STEINBRUNN, M., MOERKOTTE, G. & KEMPER, A. (1997). Heuristic and randomized optimization for the join ordering problem. *VLDB J.*, **6**. 179

TENNISON, J., KELLOGG, G. & HERMAN, I. (2015). Model for tabular data and metadata on the web. W3C recommendation. *World Wide Web Consortium (W3C)*. xix, 28, 29, 38, 52, 93, 120, 125, 129, 133

THE W3C SPARQL WORKING GROUP (2013). SPARQL 1.1 overview. W3C recommendation, W3C, http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/. 88, 90

ULLMAN, J.D. (1997). Information integration using logical views. In *International Conference on Database Theory*, 19–40, Springer. 17

VASSILIADIS, P. (2009). A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining*, **5**, 1–27. 12

VIDAL, M., RUCKHAUS, E., LAMPO, T., MARTÍNEZ, A., SIERRA, J. & POLLERES, A. (2010). Efficiently joining group patterns in SPARQL queries. In *Extended Semantic Web Conference*, 228–242. 103, 134

VILLAZÓN-TERRAZAS, B. & HAUSENBLAS, M. (2012). R2RML and Direct Mapping Test Cases. W3C Note, W3C, `http://www.w3.org/TR/rdb2rdf-test-cases/`. 4, 33, 41

VOGEL, M., WEBER, S. & ZIRPINS, C. (2017). Experiences on migrating restful web services to graphql. In *International Conference on Service-Oriented Computing*, 283–295, Springer. 155

VU, B., PUJARA, J. & KNOBLOCK, C.A. (2019). D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF. In *Intern. Confer. on Knowledge Capture*. 185

WIEDERHOLD, G. (1992). Mediators in the architecture of future information systems. *Computer*, **25**, 38–49. 2, 12

XIAO, G., CALVANESE, D., KONTCHAKOV, R., LEMBO, D., POGGI, A., ROSATI, R. & ZAKHARYASCHEV, M. (2018a). Ontology-based data access: A survey. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*. 12, 121, 125, 127, 128

XIAO, G., KONTCHAKOV, R., COGREL, B., CALVANESE, D. & BOTOEVA, E. (2018b). Efficient handling of SPARQL OPTIONAL for OBDA (extended version). *CoRR*, **abs/1806.05918**. 16, 31, 104, 114, 146