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: Prof. Oscar Corcho

June, 2020

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

Abstract

Resumen

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Thesis Structure

1.2 Dissemination Results

Chapter 2

State of the Art

2.1 Knowledge Graph Construction

2.2 RDF: Resource Description Framework

2.2.1 SPARQL

2.3 KGC Annotations: Mapping Rules and Metadata

2.3.1 R2RML: W3C Recommendation

2.3.2 RML: Exteding R2RML for Heterogeneous Data

2.3.3 Transformation Functions in Mappings

2.3.4 Metadata for data on the web

2.4 KGC from Heterogeneous Data Sources

2.5 Evaluation of KGC approaches

2.5.1 Test-Cases

2.5.2 Benchmarks

2.6 Summary

Chapter 3

Objectives and Contributions

The main goal of this work is the use of declarative mapping rules and their knowledge encoded for the construction and evaluation of knowledge graphs from heterogeneous data sources. This chapter presents the objectives and contributions to the state of the art of our work. Additionally, we detail the assumptions considered when we started this work, hypothesis and restrictions that delimit and describe the scope of this thesis.

3.1 Objectives

In the context of this thesis we identify two different goals. First, we provide a definition and a set of properties over the new concept of *mapping translation* and we describe two approaches that take into account this concept to improve the construction and access of knowledge graphs from heterogeneous data sources. The second goal of the thesis consists in defining the main steps and resources for an objective and proper evaluation of these kind of approaches.

In order to achieve the first objective of this thesis, the open research problems that have to be solved are:

- The construction of knowledge graphs (virtual or materialized) from heterogeneous data sources using declarative mapping rules is still an open issue in the state of the art. Based on the W3C recommendation R2RML (Das *et al.*, 2012), and with the aim of providing support to other format beyond relational databases, many new declarative mapping languages are 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). One of the motivations to create such declarative mapping specifications is to solve specific heterogeneity issues of the input data, hence, they start to lose their generalization and the benefit of the declarative approach of the rules. To deal with these problems, we propose the possibility of the translation among these different languages, covering at least those common characteristics that are shared across languages, introducing one of the main ideas about a new generation of knowledge graph construction approaches.

- The construction of virtualized knowledge graphs from raw data (e.g., CSV, JSON or XML) has been tackled as an engineering process delegating the management of these data sources to databases such as Apache Drill[1] and Spark-SQL[2]. These systems allow the generation of a simple relational database layer over the input data sources. However, in order to tackle the advantage of proposed SPARQL-to-SQL optimization techniques (Calvanese *et al.*, 2017; Priyatna *et al.*, 2014), a well-formed RDB is required, including its corresponding constraints. Focused on the specific case of tabular data, we propose a framework that exploits and translates mapping rules and metadata annotations to deal with the typical heterogeneity issues for querying these datasets under an OBDA environment. Its main aim is to improve query evaluation performance, as well as query completeness.

- Multiple types of query interfaces have been proposed to query heterogeneous data on the web. One of the most recent and relevant incorporation is GraphQL (Facebook, Inc., 2018). This technology aims to solve problems such as under/over fetching (Bryant, 2017; Mukhiya *et al.*, 2019; Vogel *et al.*, 2017) of other common used methods for publishing data on the web (e.g., subject-guide HTTP APIs). This specification has been included by multiple organizations in their systems as an integrating query layer over multiple data sources, hence, create a bridge between GraphQL and knowledge graph technologies is relevant. We propose a framework that translates declarative mapping rules to programmed GraphQL data wrappers (known as GraphQL resolvers) helping in their creation and ensuring that their model uses common and shared vocabularies.

[1] `https://drill.apache.org/`
[2] `https://spark.apache.org/sql/`

The previous methodological goals have two associated technological goals. The first one is the implementation of a virtual knowledge graph creation system over tabular data (Morph-CSV). The second one is implemented by Morph-GraphQL, that translates R2RML mapping rules to programmed GraphQL resolvers for accessing legacy relational database instances. The second goal is focused on proposing evaluations of knowledge graph construction systems to understand what are their main current limits, and it has the following open research problems:

- There is currently no way to obtain objective information about the conformance of knowledge graph construction engines with specifications of declarative mapping languages that go beyond relational databases. Based on the main extension of R2RML, RML (Dimou *et al.*, 2014), we proposed a set of representative test cases covering other types of data formats such as JSON, CSV and XML.

- There is no analysis of what the parameters are that affect the process of constructing a knowledge graph from heterogeneous data sources. Previous evaluations have only took into account data size as a relevant parameter (Lefrançois *et al.*, 2017; Şimşek *et al.*, 2019). We identify the set of parameters, most of them extracted from the mapping rules, that can also affect to the behaviour of these engines.

- Finally, the current knowledge graph construction benchmarks (Bizer & Schultz, 2009; Lanti *et al.*, 2015) that mainly focused on relational databases, are not enough to test the capabilities of the new generation of engines that are able to integrate or run SPARQL queries over data in several formats. A representative benchmark to test the capabilities of these approaches with objective and useful information is defined.

3.2 Contributions to the State of the Art

In this work, we aim to provide solutions to the identify open research problems. Chapter 4 describes the mapping translation concept and its property, while Chapter 5 provides the solutions to enhance virtual knowledge graph access over heterogeneous data exploiting the aforementioned concept. Chapter 6 describe the previous steps to be performed before define a representative benchmark (Chapter 7) for KGC engines. The contributions for solving the first research problem are:

C1.1. The concept of *Mapping Translation* and its main properties.

C1.2. The formalization and application of constraints extracted from annotations (mappings and metadata) over tabular data to enhance OBDA approaches.

C1.3. The exploitation of standard declarative mapping rules to automatize the generated of programmed GraphQL servers.

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 test the conformance of KG construction engines in RML.

C2.2. Identification and experimental evaluation of the parameters that affect the behavior of the KGC engines.

C2.3. Complete and representative benchmark for evaluating KGC engines based on open data from the transport domain.

3.3 Assumptions

The work described in this document is based on a set of assumptions

A1 Mapping rules and annotations are declarative and following W3C standards (or extend them).

A2 The target schema (ontology) for integrating the source data is available.

A3 Mapping rules and metadata annotations are available.

A4 Data are represented in different formats but not in RDF.

3.4 Hypothesis

H1 It is possible to translate declarative mappings

H2 Virtualized Knowledge Graphs are improved in terms of completeness and performance over raw data exploiting declarative mapping rules and metadata annotations.

H3 Not only size data is relevant in the evaluation of KG construction engines

H4 A benchmark blablabla is able to stress and provide a full overview of the state of different engines

3.5 Restrictions

R1 Data are located in the same physical place.

R2 We restrict declarative mapping rules following RML+FnO (De Meester *et al.*, 2017) specification and metadata annotations in CSVW for Morph-CSV and R2RML for Morph-GraphQL.

R3 The source data generated at scale for evaluating KG construction systems do not need cleaning or preparation functions.

Chapter 4

New Generation of Knowledge Graph Construction systems

Database technologies play a vital role in the development of information systems for all sorts of organisations. So far, relational databases (RDB) are still the dominating type of structure and technology used for data management inside organisations, 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 organisations to develop their own data models, which were strongly aligned with their activities. This led to a large heterogeneity across organisations, and even across different departments inside the same organisation. Such heterogeneity was especially evident in the case of organisational changes, merges, etc. Similarly, data warehouses were also used in order to align and materialise data from different sources, normally from the same organisation, 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 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 organisations started using Web technologies to provide access to their data (via Web

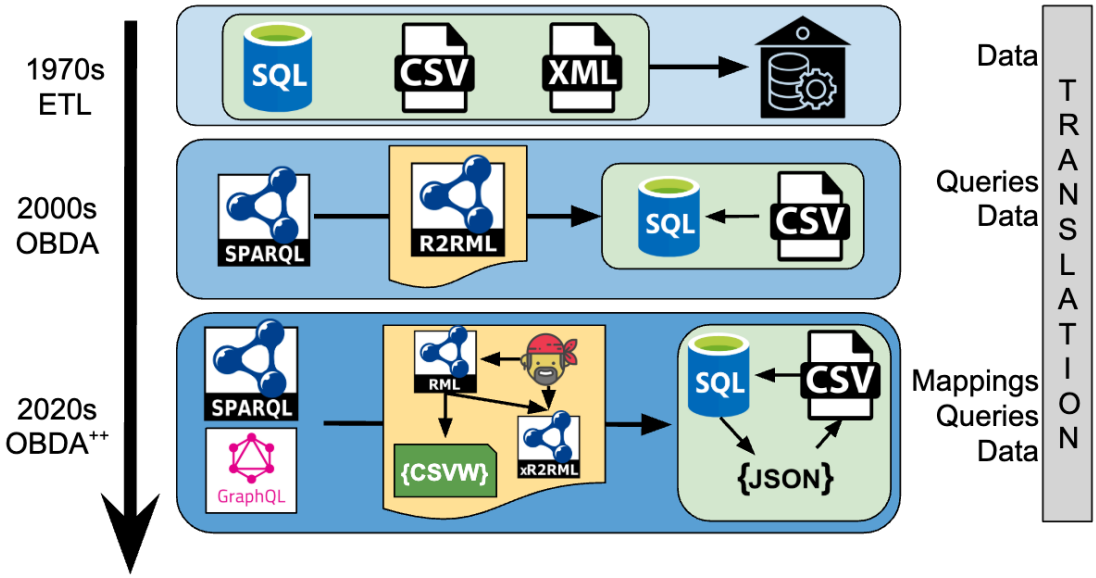


Figure 4.1: Timeline of data integration techniques. During the 1970s ETL approaches started with data translation techniques, current generation of OBDA incorporated techniques for query translation and next generation of OBDA systems which mapping translation approaches are to be applied.

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). From now on we will refer to both of them, in a general manner, as OBDA.

In OBDA, ontologies are used as a global view over heterogeneous data sources. It is quite common to use a mediator-based approach (Wiederhold, 1992), where mediators and wrappers are used as intermediaries to overcome the differences between the local schemas and the global view. In this setting, mappings are commonly used to describe such relationships in a

declarative manner. These mappings may be normally exploited in two directions: for **data translation** (Dimou *et al.*, 2014), so that the original data is transformed and materialised according to the global view (in a similar way as with the use of the aforementioned ETL workflows); and for **query translation** (Priyatna *et al.*, 2014; Rodriguez-Muro & Rezk, 2015), where queries written according to the global schema are transformed into the query language supported by the original data sources and evaluated in the original data management systems, with the results being transformed back according to the global view.

Many different types of OBDA mapping languages have been proposed over the last decades, with a large variety of syntaxes and formats especially in the early ones. Since the standardisation of languages like RDF and OWL, several languages were proposed focused on the transformation from relational databases into RDF (e.g. D2R, R2O). 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.*, 2012). The Direct Mapping approach specifies simple transformations that require no intervention from users. R2RML allows specifying transformation rules, such as how URIs should be generated, which columns to be used for the transformation, etc.

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[3] (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[4], Tarql[5] 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

[3] `https://www.w3.org/ns/csvw`
[4] `https://helio.linkeddata.es/`
[5] `https://github.com/tarql/tarql`

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 RMLC-Iterator (for statistical CSV files) (Chaves-Fraga *et al.*, 2018) or YARRRML (Heyvaert *et al.*, 2018). Lastly, many of existing mapping languages **lack formalisation**, making it difficult to apply, for example, query translation techniques. Therefore, the current situation of an OBDA 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 materialised 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 OBDA.

In this paper, we lay out our vision that the next generation of OBDA 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 OBDA so far, the OBDA 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 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.

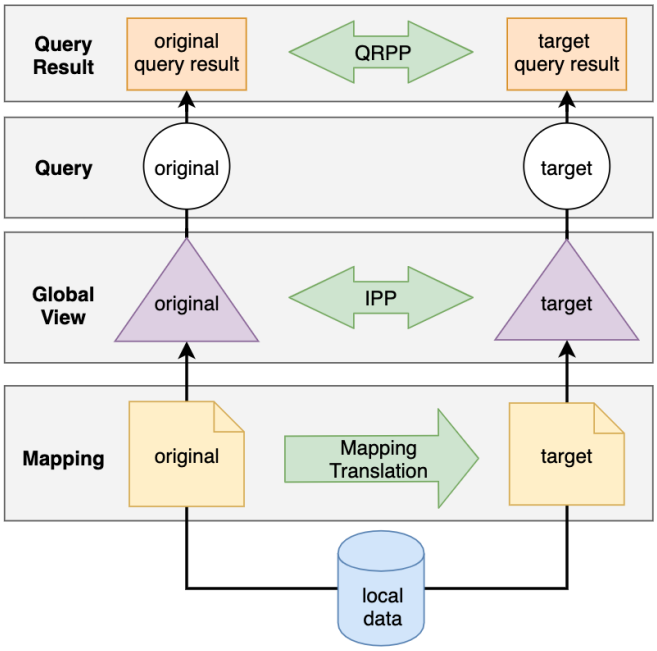


Figure 4.2: Mapping Translator Properties. The results (triangles) may satisfy the IPP property after the application of the source and target mappings over the same data. In the same way, query results (rectangles) may satisfy the QRPP property when equivalent queries are evaluated over the source and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

4.2 Use Cases

In this section we identify a set of scenarios and challenges in the creation and use of OBDA 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 summarised in Table 5.3.

Use Case	Engine	Translation
Maintenance	yarrrml-parser	YARRRML-to-RML
Maintenance	rmlc-statistic	RMLCIterator-to-R2RML
Declarative to Programmed	morph-graphQL	R2RML-to-GraphQLResolver
Access	morph-CSV	RML(+FnO)-to-R2RML
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 OBDA mappings is usually difficult, since mapping languages have been created so that they can be consumed by the corresponding OBDA 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 OBDA approach is used to provide access to multidimensional data sources, such as the ones commonly used to publish statistical data. We describe two 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[6]. It is designed with the objective to reduce the size and verbosity of RML. There is no specific engine or parser to exploit YARRRML

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

mappings in an OBDA setting. Instead, the tool Matey[7] is in charge of translating YARRRML mappings into RML, so that any RML-compliant OBDA engine can be used to exploit them.

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 OBDA 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, RMLC-Iterator (Chaves-Fraga *et al.*, 2018) is proposed as a simplified mapping language specifically designed for this type of data, including two new properties in the R2RML specification: a property to define the array of access columns and a corresponding dictionary if the value of a header needs to be changed. With this approach, the mapping size is drastically reduced. Additionally, a tool is provided to translate RMLC-Iterator mappings into R2RML, hence allowing the use of any R2RML-compliant OBDA engine.

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

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

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[8], joinmonster[9]).

From this basic description of the GraphQL framework, the analogy with the OBDA architecture is clear. Typically, the following tasks need to be done to setup a GraphQL server:

1. A domain expert will analyse the underlying datasets, propose a unified GraphQL schema and describe how the source data sources will need to be mapped into it. Note that there is no standard mechanism to represent these mappings.

2. A software developer will then implement those mappings as GraphQL resolvers. Generating GraphQL resolvers is difficult even for a standard-sized dataset which typically contains more than a handful tables and hundreds of properties. This situation is even worse if the underlying dataset evolves, considering that the corresponding resolvers have to be updated as well.

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 formalised mapping language, and apply a mapping translation technique to generate automatically the corresponding GraphQL schemas 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.

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 (materialisation) 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 (virtualisation) 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

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

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[10] 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. The main reason for using two approaches in this case is that individually they are not able to resolve all the CSV challenges identified to enable a efficient query-translation process over the data. When we started this work we noticed that there are enough proposals for dealing with the heterogeneity of the CSV files (CSVW together with RML+FnO) and the SPARQL-to-SQL techniques and optimisation have been studied and implemented for a decade. Finally, we imposed one requirement: our approach should use as much as possible existing resources and not introduce yet another mapping language nor require a new SPARQL-to-SQL implementation. The result is a solution that, considering mappings and annotations that deal with the heterogeneity of the CSV files, builds an enriched RDB instance that represents the original data and translates the RML+FnO input mapping into the corresponding R2RML mapping so that it can take advantage of existing SPARQL-to-SQL optimisations.

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

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

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 Virtual Knowledge Graph 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 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

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

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.

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

Listing 4.1: Data source mapping

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

using an iterator that has been incorporated into RMLC[11], an RDF Mapping Language for heterogeneous CSV files.

4.3.1 From CSV to OBDA

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)

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.

[11] `https://github.com/RMLCio`

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 RMLC

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

4.3.1.2 Approach 2 - RMLC iterator

In this approach, we aim to reduce the size of a R2RML mapping for statistics data by incorporating an iterator variable into RMLC. This mapping language has been equipped with several features to be able to deal with the heterogeneity of tabular data. For example, the exploitation of implicit relations among CSV files, or the creation of an enriched database schema based on the information provided by the mappings and the datasets[12].

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, RMLC-Iterator reduces the size of the mapping while maintaining the semantics of the R2RML mapping. Tgdhis 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 4.5 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

[12] `https://github.com/RMLCio/RMLC-CSV`

Listing 4.6: Iterator variables in RMLC

```
<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;
    ];
];
```

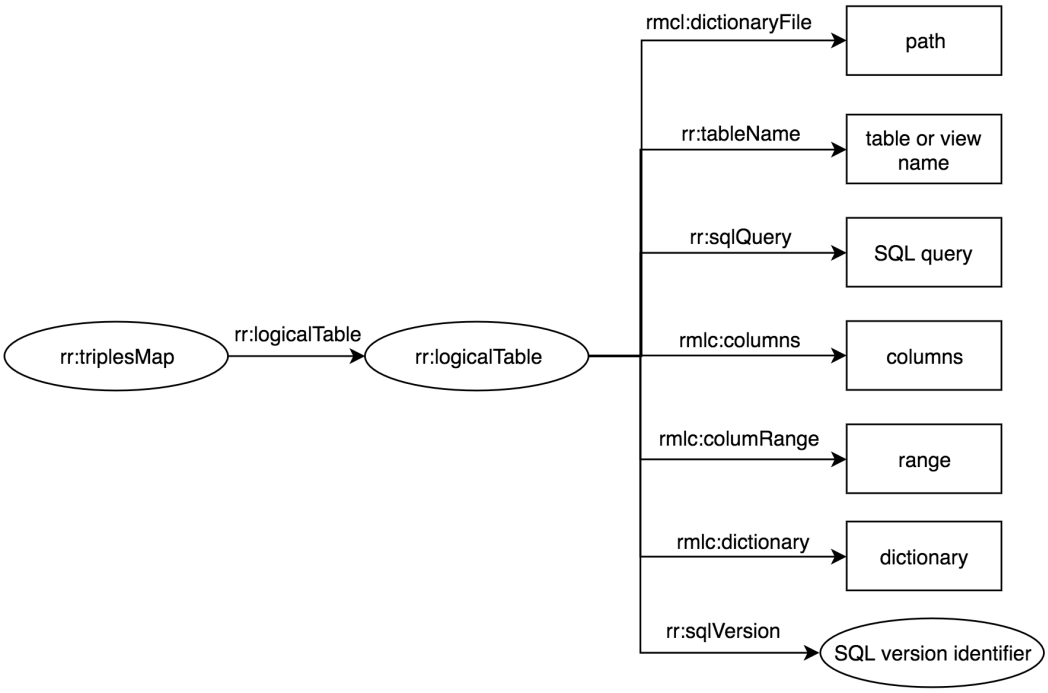


Figure 4.3: RMLC extension for statistics tabular data

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 RMLC 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 RMLC-Iterator 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 RMLC is aligned to R2RML, any R2RML available processor is able to deal with our approach. In our case, we integrate the RMLC-processor[13] with Morph-RDB[14] to query statistics data using SPARQL.

4.3.2 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 RMLC mappings with Morph-RDB to generate virtual SKGs and evaluate three SPARQL queries for each one. All the datasets, queries, results, code and mapping are available online[15].

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[16]. We use tabula-java[17] 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.

[13] `https://github.com/oeg-upm/rmlc-statistic`
[14] `https://github.com/oeg-upm/morph-rdb`
[15] `https://github.com/oeg-upm/rmlc-statistic`
[16] `http://www.sltda.lk/statistics`
[17] `https://github.com/tabulapdf/tabula-java`

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

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

- 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.

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 RMLC 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[18] 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[19]. 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.

[18] `http://ec.europa.eu/eurostat`

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

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

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

- 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.

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 RMLC mapping documents,one with `rmlc:columns` property and the other with `rmlc:range` property. Then we use the RMLC Processor to generate the naive version of R2RML.

The RMLC 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

Exploiting Mapping Rules to Enhance Virtual Knowledge Graph Access

5.1 Morph-CSV: Virtual Knowledge Graph Access over Tabular Data

Guided by Open Data principles, governments and private organizations are regularly publishing wide amounts of public data in open data portals. For example, almost a million of datasets are available in the European Open Data Portal (EODP)[20], and many of them are available in tabular formats (e.g., CSV, Excel), as observed in Table 7.1. Both the simplicity of a tabular representation and the variety of tools to manage a table (e.g., Excel, Calc) have influenced in the popularity of tabular formats to represent open data.

Although 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 that is 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.

Following Linked Data (Bizer *et al.*, 2011) and FAIR initiatives (Wilkinson *et al.*, 2016) [21], data providers are encouraged to make data available in an RDF-based representation following the 5-star linked data principles[22]. The Ontology-Based Data Access (OBDA) (Poggi *et al.*, 2008) paradigm facilitates the transformation of heterogeneous data into RDF. An OBDA corresponds to a data integration system (DIS) (Lenzerini, 2002) over heterogeneous data sources. A DIS unified schema is defined in terms of ontologies, while mapping rules establish correspondence between the unified schema concepts and the DIS data sources. An OBDA can be materialized or virtual. In a materialized OBDA, the integration of the DIS data sources is physically represented in RDF (Poggi *et al.*, 2008). Contrary, in a virtual OBDA, data integration is performed on the fly during query processing; DIS mapping rules are used to rewrite SPARQL queries into queries against the DIS data sources (Calvanese *et al.*, 2017; Priyatna *et al.*, 2014). Features like functions in mappings (De Meester *et al.*, 2017; Junior *et al.*, 2016) and metadata (Tennison *et al.*, 2015), (i.e., annotations) are usually used in materialized OBDAs to overcome the aforementioned challenges of tabular data.

[20] `https://www.europeandataportal.eu`
[21] `https://www.go-fair.org/fair-principles/`
[22] `https://5stardata.info/en/`

Data Portal	1st Format	2nd Format	3rd Format
Spain	**CSV (50%)**	**XLS (35%)**	JSON (33%)
Norway	**CSV (77%)**	GEOJSON (17%)	JSON (14%)
Italy	**CSV (76%)**	JSON (35%)	XML (25%)
Croatia	**XLS (63%)**	**CSV (40%)**	HTML (33%)

Table 5.1: Most commonly used formats and percentage over the total number of datasets to expose data in mature EU open data portals in October 2019. Each dataset may be shared in different formats.

Traditional virtual OBDA approaches, usually, rely on loading the tabular data into SQL-based systems[23,24](e.g., MySQL, Apache Drill, Spark SQL, Presto) to perform the 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 OBDA process over tabular data directly impacts over 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. As a consequence, query translation optimization techniques that normally exploit indexes (e.g., (Priyatna *et al.*, 2014; Rodriguez-Muro & Rezk, 2015)) do not produce the expected results.

OBDA 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 an OBDA tabular dataset. For example, we can standardize a column indicating its format, define integrity constraints or declare datatypes. The majority of OBDA query translation engines (Endris *et al.*, 2019; Priyatna *et al.*, 2014) do not include this information. Those engines that have partially included the constrains (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 a workflow that includes the exploitation of these tabular annotations during a virtual OBDA process will ensure correct and optimized SPARQL-to-SQL translations.

[23] `https://github.com/oeg-upm/morph-rdb/wiki/Usage#csv-files`
[24] `https://github.com/ontop/ontop/wiki/MappingDesignTips#database-tips`

Problem and Proposed Solution: We address the limitations of current OBDA query translation techniques over tabular data. Our goals are (i) define a framework that includes the application of a set of constraints over tabular data and (ii) define a set of operators that apply each type of constraint in order to improve query completeness and performance. We propose a set of new steps to be aligned with the current OBDA workflow. Further, we implement Morph-CSV, and evaluate its behavior 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 OBDA 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 OBDA 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.

3. Evaluation of Morph-CSV over two open source engines: Morph-RDB (Priyatna *et al.*, 2014) and Ontop (Calvanese *et al.*, 2017); two benchmarks (BSBM (Bizer & Schultz, 2009) and GTFS-Madrid-Bench[25]), and a real-world testbed from the Bio2RDF project (Belleau *et al.*, 2008) are used in the study.

5.1.1 Motivating Example

Consider the de-facto standard for publishing open data in the transport domain, GTFS[26]. This model provides information such as *schedules*, *stops* and *routes* using 15 different inter-related CSV files called GTFS feed. Each feed usually specifies the information 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 travelled using various types of transportation. The GTFS feeds from the metro and the buses of the city of Madrid have

[25]https://github.com/oeg-upm/gtfs-bench
[26]`https://developers.google.com/transit/gtfs/reference/`

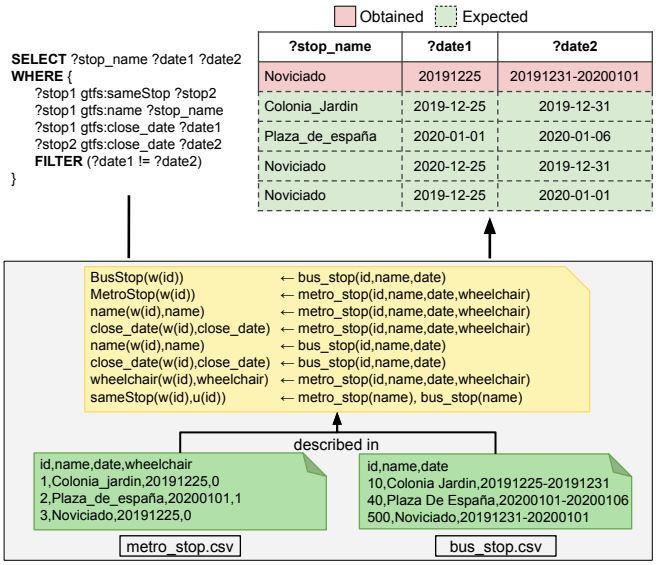


Figure 5.1: Motivating Example. SPARQL query evaluation over two tabular data files in the transport domain through a common OBDA approach. It loads the files as single tables in an SQL-based system and uses the mapping rules for query translation. The number of results differs with respect to the expected 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).

several stops and stations in common; they are created by different transport authorities, and the names of their stops are defined in different manners. Figure 5.1 depicts a SPARQL query asking for bus and metro stops with the same name, and information related to their closing dates during holidays. Since GTFS uses temporal identifiers for its resources, links have to be established joining stop names. However, 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). Following the approach commonly employed by typical OBDA 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 performing an improved join among the stop names of the bus and metro, through the normalization of multi-valued date columns.

The manual and ad-hoc preparation of a tabular dataset for a virtual OBDA process is usually the most time-consuming and less reproducible task. Exploiting available standard

OBDA annotations allows its generalization and automatization, as well as ensuring the effectiveness of query completeness and performance of SPARQL-to-SQL techniques, complying with OBDA assumptions.

5.1.2 Ontology Based Data Access over Tabular Data

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

5.1.2.1 Querying Challenges under virtual OBDA

There are specific challenges on querying tabular datasets using an OBDA 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:

- **Selection (S):** Existing frameworks load all of the files that are specified as sources in the OBDA 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, ODBA performance.

- **Normalization (N):** 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 (H):** 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 may have not been included in the dataset, query completeness can be affected.

- **Lightweight Schema (LS):** 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 indexes and datatypes. The existence of this type of information is assumed (Xiao *et al.*, 2018a) in an OBDA approach for performing optimizations in query evaluation techniques. Therefore, the lack of this information affects the performance of OBDA 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, as we reflect in Section **??**, 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 OBDA 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 OBDA workflow.

5.1.2.2 OBDA annotations for Tabular Data

R2RML (Das *et al.*, 2012) is a W3C Recommendation for describing transformation rules from RDB to RDF and a widely used mapping language in virtual OBDA approaches. RML (Dimou *et al.*, 2014) extends R2RML; it provides support to a variety of data formats, e.g., XML, CSV, and JSON. Both languages provide basic transformation functions to concatenate strings, which are especially useful for generating URIs from columns/fields of the dataset. Recently, RML has been integrated with the Function Ontology (FnO) (De Meester *et al.*, 2016) to support other types of transformations. Additionally, for tabular data, CSVW metadata (Tennison *et al.*, 2015) is a W3C Recommendation to describe tabular datasets. Although there are other proposals in the state of the art to deal with some of the aforementioned challenges (Debruyne & O'Sullivan, 2016; Junior *et al.*, 2016), Morph-CSV relies on these two proposals because they cover the identified challenges. Additionally, this election is supported by the fact that

Challenges	Relevant Properties
Describe the corresponding concept (LS)	rr:class
Describe the corresponding property (LS)	rr:predicateMap
Add header to a CSV file (H)	csvw:rowTitles
Column datatype (LS)	csvw:datatype
Constraining values (H)	csvw:minimum, csvw:maximum
Specify the format of a column (H)	csvw:format
Specify a join (H)	rr:refObjectMap, csvw:foreignKeys
Transform value (H)	fnml:functionValue
Support for multiple values in one cell (N)	csvw:separator
Primary key (N)	csvw:primaryKey
Default for missing values (H)	csvw:default
Specify NULL values (H)	csvw:null
Specify NOT NULL constraint (LS)	csvw:required
Specify columns to be transformed (H)	rr:reference, rr:template

Table 5.2: Properties of CSVW and RML+FnO that can be used to address the challenges of dealing with tabular data in a virtual OBDA approach

CSVW is a recommendation from the W3C and RML+FnO (in addition of being a extended version of a W3C recommendation) has been previously applied in other projects (De Meester *et al.*, 2017; Mami *et al.*, 2019b) and is widely used by several materialization engines, e.g., RMLMapper[27], CARML[28] and RocketRML (Şimşek *et al.*, 2019). Finally, relevant benefits of these annotations is that both of them are defined in a declarative manner. Thus, the maintainability, the readability, and the understanding of the virtual OBDA approach is improved and independent from any specific programming language.

We now describe relevant properties of RML+FnO and CSVW, summarized in Table 5.2, which are useful to deal with the challenges identified:

- **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 addi-

[27] https://github.com/RMLio/rmlmapper-java
[28] https://github.com/carml/carml/

tion, 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 `rr:joinCondition` can be seen as an indication that the parent column used over this rule could be a foreign key. 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, what 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).

5.1.3 The Morph-CSV Framework

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. 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.

5.1.3.1 OBDA assumptions

Analyzing the definition of OBDA of (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 not a 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 ways 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 over the rest of assumptions, that have been solved in a naive manner.

- 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 type of constraint (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.

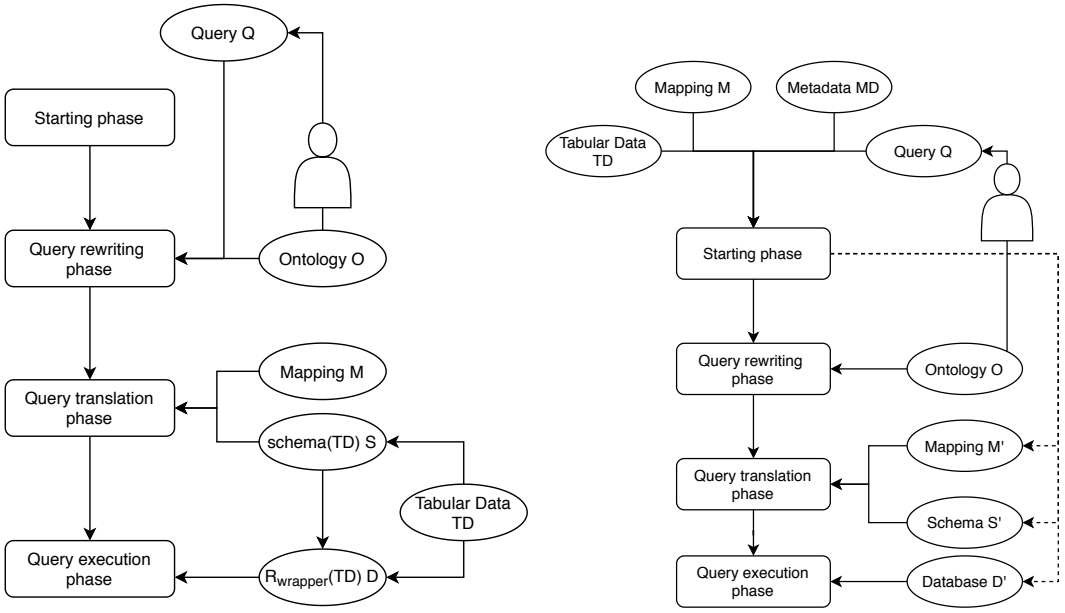
- D is an RDB instance or is a NoSQL database instance, equipped with an RDB wrapper that is able to provide a relational view over S and D. In the context of a tabular dataset $D_{tabular}$, D=$R_{wrapper}(D_{tabular})$ where $R_{wrapper}$ is a relational database wrapper that satisfies S.

5.1.3.2 From Virtual Tabular Dataset to 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 5.1. 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 $\mathfrak{D}_{tabular}$ = $\{s_1,\ldots,s_n\}$ and where each s_i is a tabular relation defined over the domains of the attributes $Att(s_i) = \{A_{i1},\ldots,A_{im}\}$[29], where m is the number of attributes of s_i. O is an ontology, and M is a

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



(a) Baseline approach **(b)** Enhanced virtual OBDA workflow.

Figure 5.2: Virtual OBDA for tabular data approaches. 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.

set of global as view mappings between O and $schema(D_{tabular})$[30]. MD is a set of metadata 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.

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 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

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

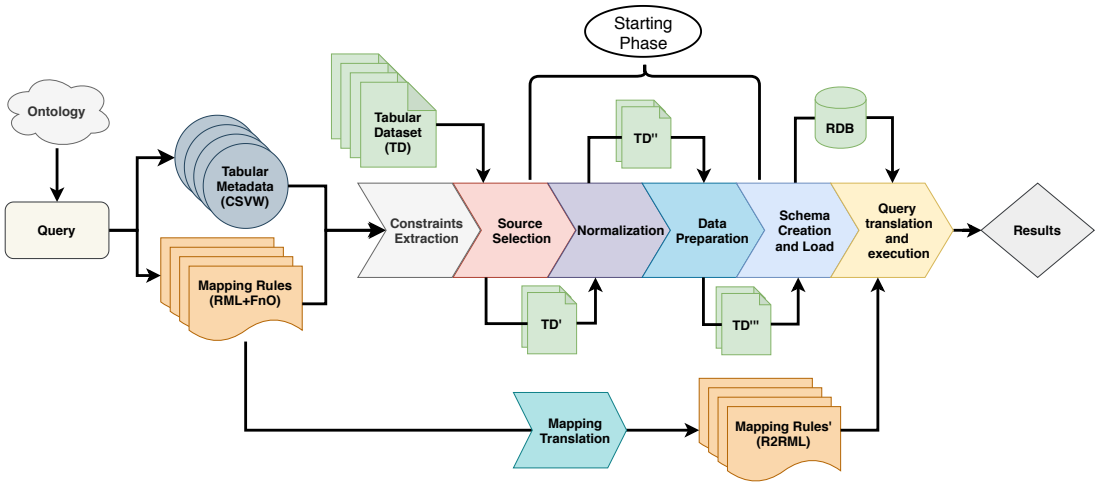


Figure 5.3: The Morph-CSV Framework. Morph-CSV extends the starting phase of a typical OBDA system including a set of steps for dealing with the identified tabular data querying challenges. The framework first, extracts the constraints from mappings and tabular metadata and then, implements them in a set of operators that are ran before executing the query translation and query execution phases. The mapping rules are translated accordingly to the modified tabular dataset to allow its access by the underlying OBDA engine.

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 (see Figure 5.2b).

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 1. 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.

5.1.3.3 Problem statement and solution

Based on the preliminaries and assumptions made over 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 decreased compared to the evaluation 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 5.2b and Appendix **??**). 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 utilized 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 3 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 (Jozashoori & Vidal, 2019) 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 5.3 with the inputs and outputs of each step.

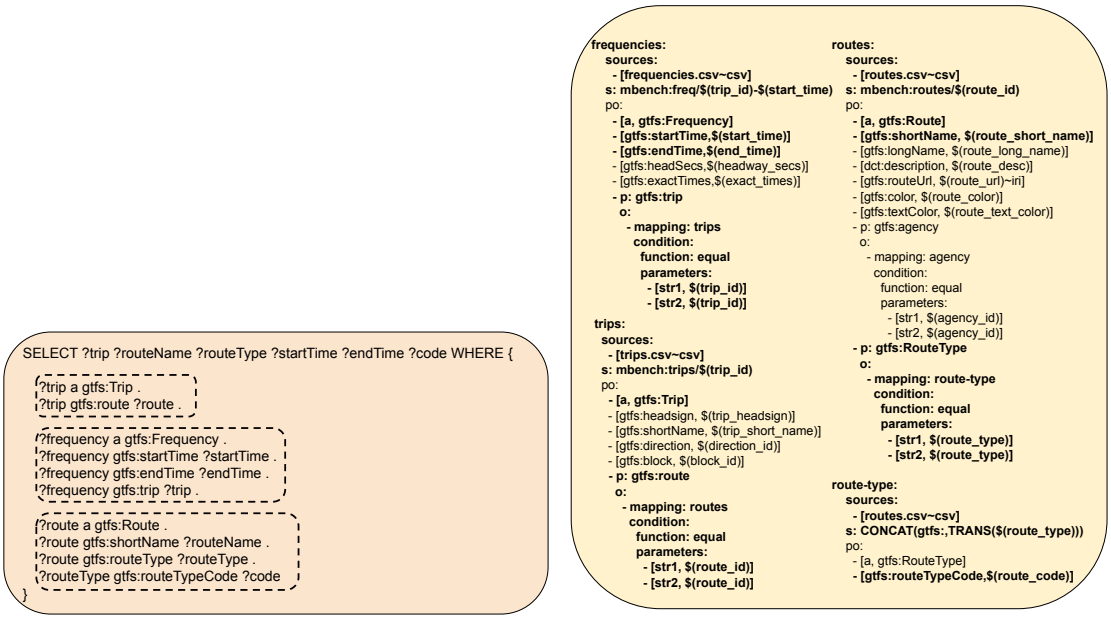
Step	Constraint/Improvement	Rule/Annotation	Function	Challenge
Extraction	Reduce search space	SSG from Query	select_annotations	Selection
		Mapping Rules	select_sources	
Data Normalization	2NF	csvw:separator	split	Normalization
	3NF	TriplesMap with same source	cut	
Data Preparation	Standarization	csvw:null, 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 5.3: Summary of constraints, corresponding functions and OBDA annotations applied by Morph-CSV

5.1.3.4 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 5.4a, the GTFS feed from the the Madrid metro as source data, and the corresponding RML+FnO mapping rules and CSVW annotations[31].

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



(a) Input SPARQL query. **(b)** Mapping rules selection.

Figure 5.4: 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 5.3.

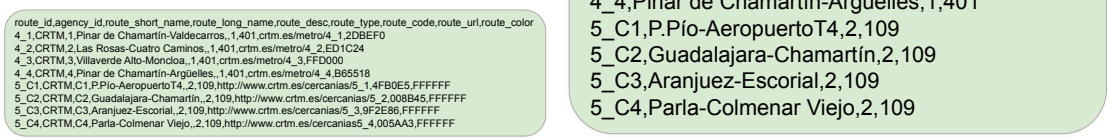
Source Selection The second step is to select the relevant sources to answer the input query. The baseline approach delegates this step to the RDBMS: it loads all the sources of the dataset

in the RDB instance because it does not have information about what 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)[32]. Then, for each SSG_i and `rr:TriplesMap` TM_j defined in M, the engine selects TM_j when 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 $M^{'}$ and annotations $MD^{'}$ by this step substitute the original ones in VTD. An example of this step is shown in Figure 5.4, 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 5.4b).

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 $D_{tabular}$ of VTD. Following the previous example, Figure 5.5 shows the selection of the relevant columns of source *routes.csv*, where Morph-CSV has the original source as input (Figure 5.5a), 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 5.5b). Note that in this step, unnecessary sources from the input GTFS feed such as *agency.csv* and *stops.csv* are also discarded.

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

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



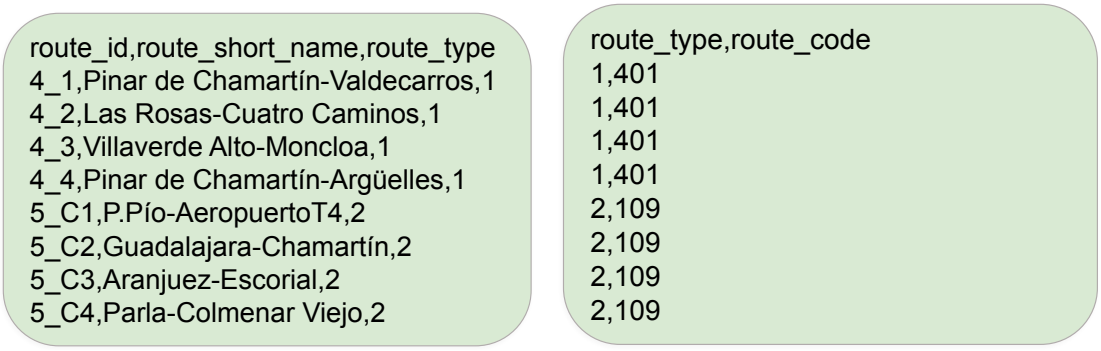
(a) Original routes.csv input source.

(b) Output of routes.csv source.

Figure 5.5: Source Selection. Based on the selection of the rules, only route_id and trip_id columns are selected, discarding the rest fields.

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, and a updated 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 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 5.6 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 5.6), removing that information for the *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 of Figure 5.4b). 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.



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

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

Figure 5.6: Normalization. 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 5.6b). 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 (Jozashoori & Vidal, 2019), 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. 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.*, 2016), thus, avoiding the incorporation of them inside the SQL translated query. In Figure 5.7 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.

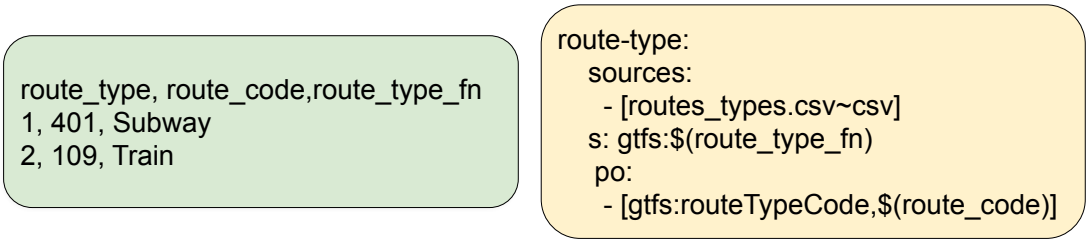


Figure 5.7: Data preparation of *route-types.csv* file.

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), 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. Figure 5.8 shows the RDB schema generated by Morph-CSV for the input query in Figure 5.4a, with the applied domain and integrity constraints.

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 generalizing the proposed steps, usually solved in an ad-hoc manner, not only automatizing the process but also improving its maintainability, understandability and readability.

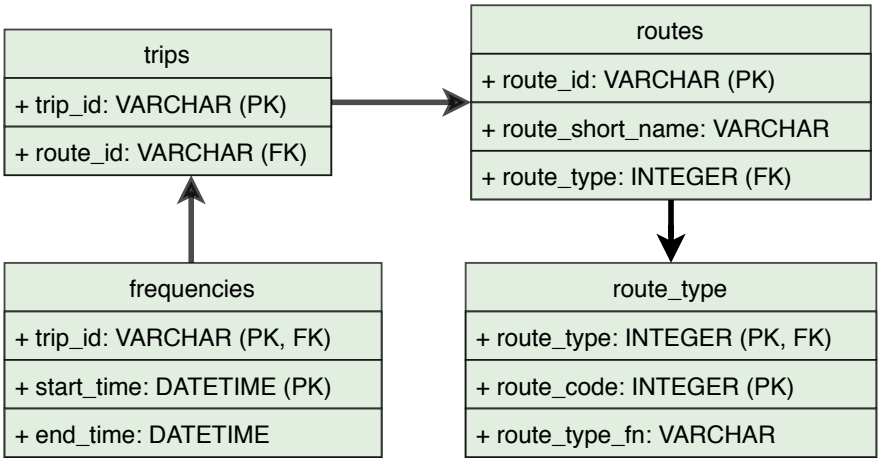


Figure 5.8: Generated schema. The schema generated by Morph-CSV extracting domain and integrity constraints from the annotations and based on the identified sources selected from the input query.

5.1.4 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. 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 levels of data heterogeneity int 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. GTFS-Madrid-Bench 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. 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 **??** 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 OBDA engines: Ontop[33,34] v3.0.1 and Morph-RDB v3.9.15[35]. To evaluate the baseline approach, we manually generate the relational database schemas 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[36,37], we integrate our solution on top of the two OBDA engines. To ensure the reproducibility of the experiments, we also provide all of the resources in a docker image. In order to test the number of answers, we use the gold standards provided by both benchmarks in RDF, loaded in a Virtuoso triple store.

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 it corresponds), and the number of answers obtained (see Appendix **??**). Each query was executed 5 times with a timeout of 2 hours 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. 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.

5.1.4.1 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 a 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 Morph-CSV is not the improvement of the support of SPARQL operators in the query translation process, we only select the queries of the benchmark that include supported operators and operations by each engine. This means

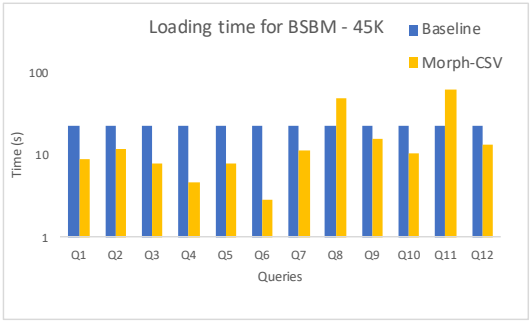
[33] `https://github.com/ontop/ontop`

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

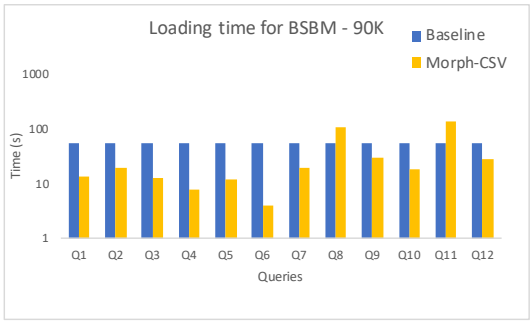
[35] `https://github.com/oeg-upm/morph-rdb`

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

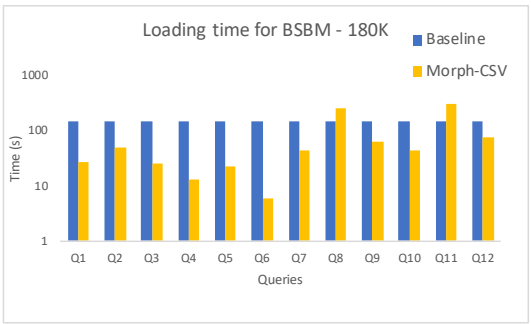
[37] `https://github.com/oeg-upm/morph-csv`



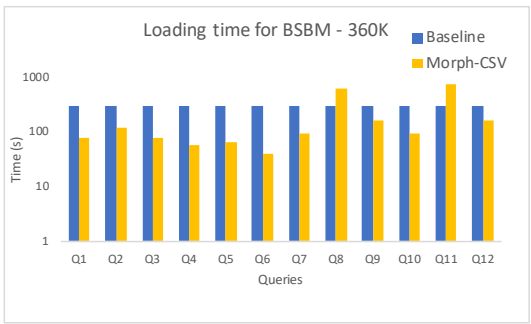
(a) Loading time for BSBM 45K.



(b) Loading time for BSBM 90K.



(c) Loading time for BSBM 180K.



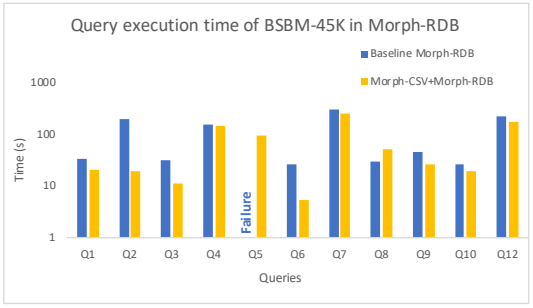
(d) Loading time for BSBM 360K.

Figure 5.9: 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 (blue columns) is constant for each dataset and query, while Morph-CSV (orange columns) depends on the query and number of constraints to be applied over the selected sources.

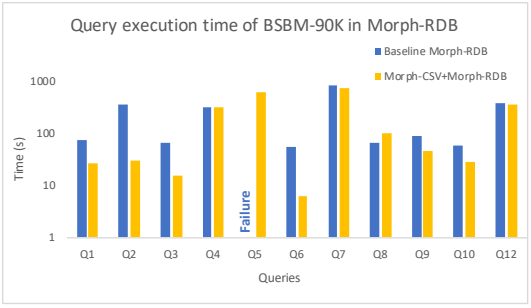
that Morph-RDB will be evaluated over the queries Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10 and Q12 and Ontop will be evaluated over Q1, Q3, Q4, Q5 and Q10, both of them using the corresponding R2RML mapping document. For the baseline approach we manually create the RDB schema without any constraints.

BSBM Results

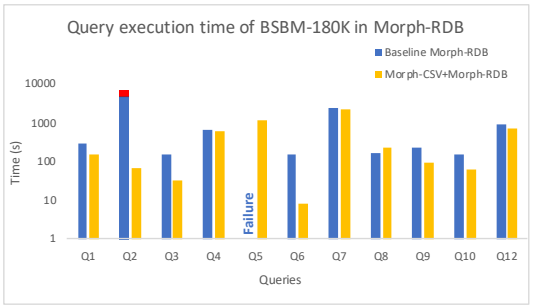
Loading Time. The results of the load time for each query and dataset size are shown in Figure 5.9. The main difference between the two methods is that while the loading time for the baseline approach is constant for each size, Morph-CSV loading time depends on several input parameters such as the query and the number and type of constraints. It could be understandable that the application of a set of constraints over the raw data in order to improve



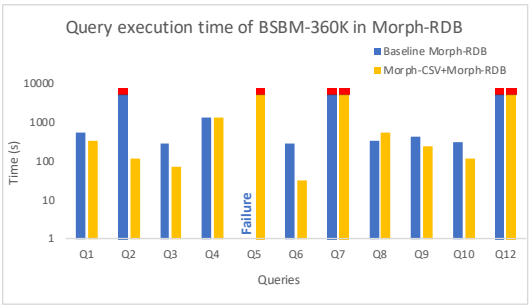
(a) Total query execution time for BSBM-45.



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



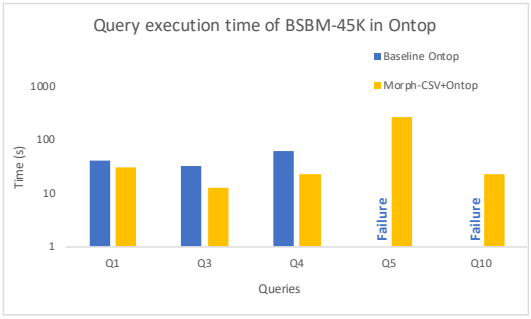
(c) Total query execution time for BSBM-180.



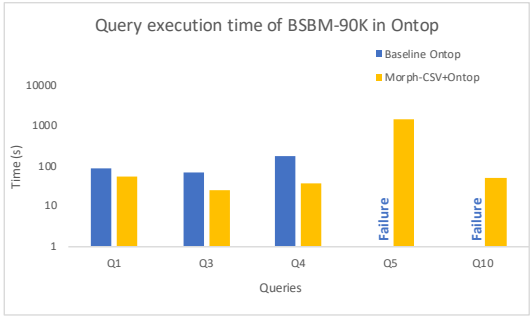
(d) Total query execution time for BSBM-360.

Figure 5.10: 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 (blue columns) is compared with the combination of Morph-CSV together with Morph-RDB (orange columns). Red marks on the top of the columns mean a timeout (72000 seconds) in query execution.

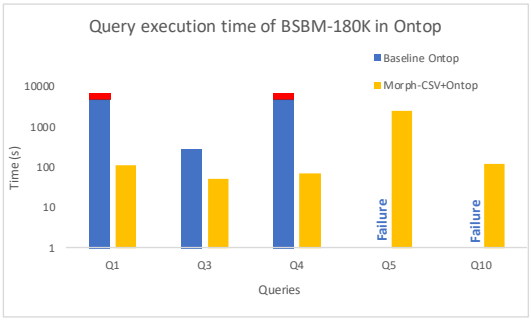
query performance and completeness would have a negative impact in the loading time of our proposal. This happens in queries Q8 and Q11, where the number of sources and the application of the constraints (mainly integrity constraints) impact negatively on the loading time of the data in the RDB instance in comparison with the baseline approach. However, in the rest of the queries, the Morph-CSV step is focused on the selection of constraints, sources and columns, and exploiting the information in query and mapping rules improves the loading time for each query in comparison with the baseline 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



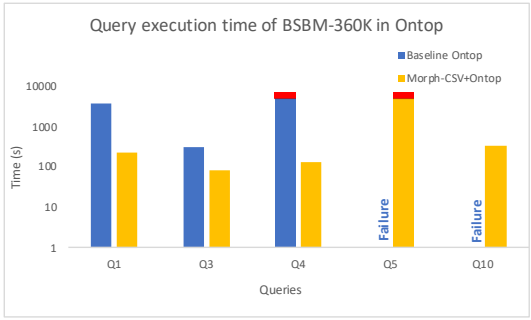
(a) Total query execution time for BSBM-45.



(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 5.11: 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 (blue columns) is compared with the combination of Morph-CSV together with Ontop (orange columns). Red marks on the top of the columns mean a timeout (72000 seconds) in query execution.

for any query.

Evaluation Time with Morph-RDB. The query execution time using Morph-RDB as the back-end OBDA engine is shown in Figure 5.10. The first remarkable observation can be seen in query Q5. Although this query contains operators 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 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. 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. asking for all the properties of a class). We observe this behavior in Q8, although the difference between the two approaches is not very relevant and is maintained across the datasets.

Evaluation Time with Ontop. The query execution time using Ontop as the back-end OBDA engine is shown in Figure 5.11. 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.

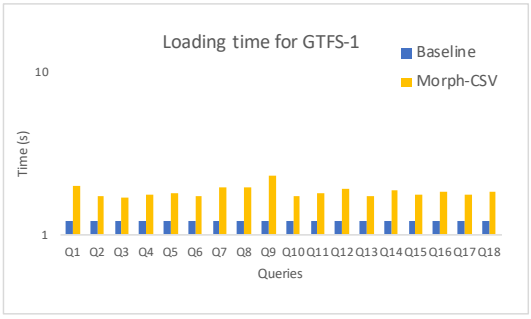
As mentioned in the Ontop repository page[38], integrity constraints are essential for the correct behavior of the engine. Although it is out of the scope of this paper, we observe in our experiments that the main reason why Ontop is only able to answer half of the queries in this benchmark, is related to some issues about maintaining the desirable properties Corcho *et al.* (2019) when translating R2RML mapping rules to its own mappings, called OBDA. The 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).

5.1.4.2 GTFS-Madrid-Bench

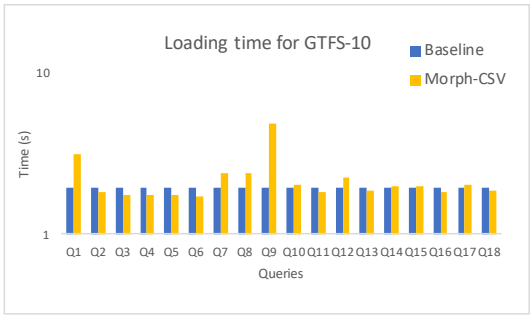
The GTFS-Madrid Benchmark[39] consists of an ontology, an initial dataset of the metro system of Madrid following the GTFS model, a set of mappings in several specifications, a set of

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

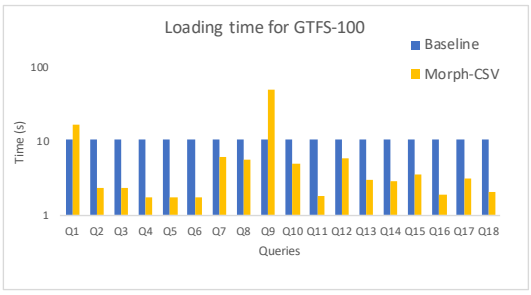
[39] Paper under review. Resources available at: `https://github.com/oeg-upm/gtfs-bench`



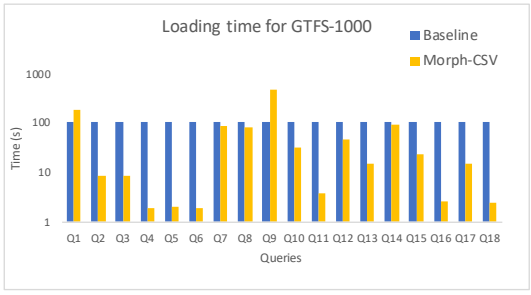
(a) Loading time for GTFS-1.



(b) Loading time for GTFS-10.



(c) Loading time for GTFS-100.



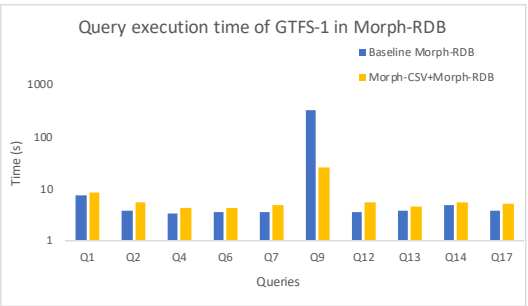
(d) Loading time for GTFS-1000.

Figure 5.12: 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 (blue columns) is constant for each dataset and query while Morph-CSV (orange columns) depends on the query and number of constraints to be applied over the selected sources.

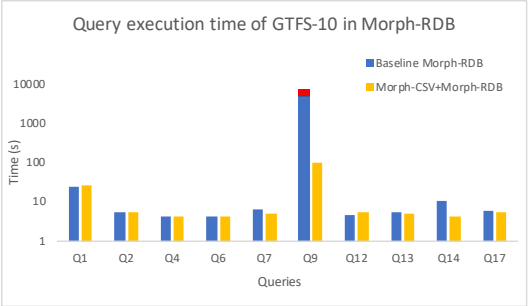
queries according to the ontology that cover relevant features of the SPARQL query language, and a data scaler based on a state of the art proposal Lanti *et al.* (2015).

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 by 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 operators that are supported by each engine: Morph-RDB will be evaluated using queries Q1, Q2, Q4, Q6, Q7, 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[40].

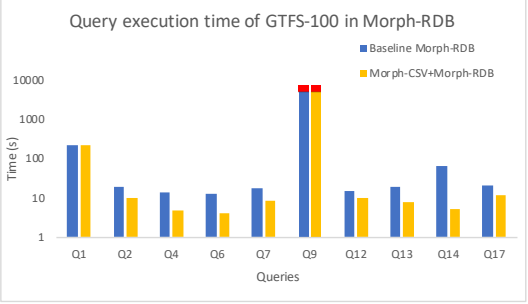
[40] `https://github.com/oeg-upm/gtfs-bench/tree/master/queries`



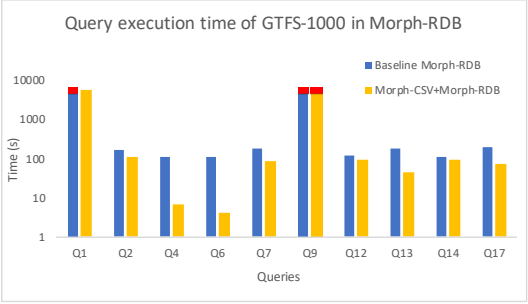
(a) Total query execution time for GTFS-1.



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



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



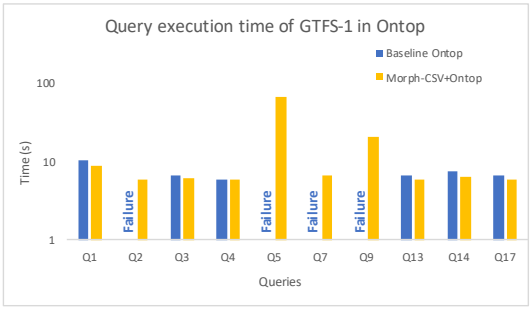
(d) Total query execution time for GTFS-1000.

Figure 5.13: 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 (blue columns) is compared with the combination of Morph-CSV together with Morph-RDB (orange columns). Red marks on the top of the columns mean a timeout (72000 seconds) in query execution.

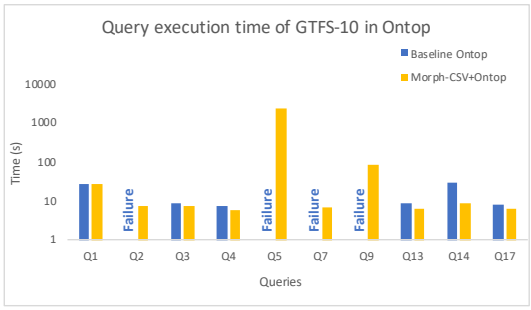
Madrid-GTFS-Bench Results

Loading Time. The loading time of the GTFS-Madrid-Bench queries is shown in Figure 5.12. 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).

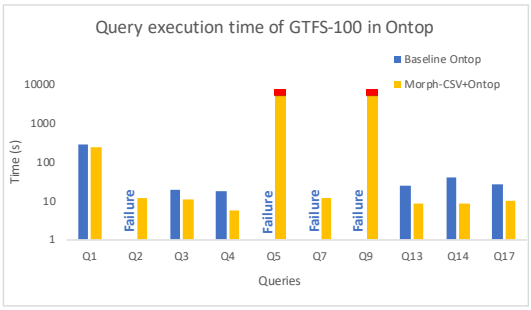
Evaluation Time with Morph-RDB. The query execution time with Morph-RDB as the back-end OBDA engine is shown in Figure 5.13. Analyzing the results, we generally observe that



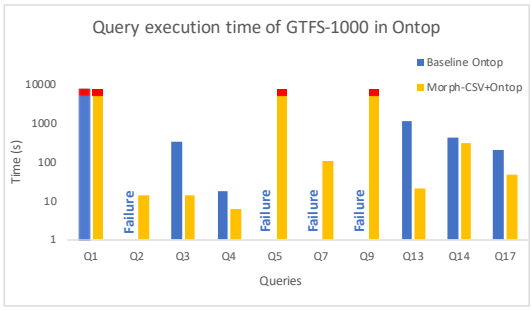
(a) Total query execution time for GTFS-1.



(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 5.14: 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 (blue columns) is compared with the combination of Morph-CSV together with Ontop (orange columns). Red marks on the top of the columns mean a timeout (72000 seconds) in query execution.

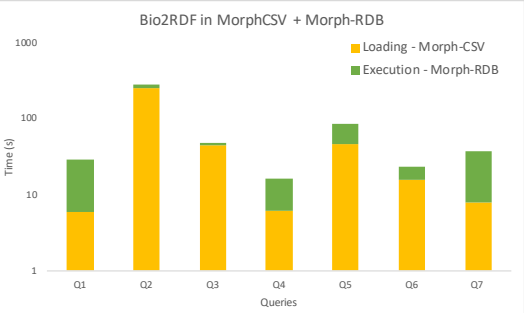
the incorporation of Morph-CSV in the workflow of OBDA enhances query performance. With respect to the results of each query, we can observe that on the one hand the behavior of the engine over simple queries (Q1, Q2, Q7, Q12 and Q17) is similar. This is understandable as the selected data sources needed 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. 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, the query can be answered in reasonable time. If we analyze the results obtained, 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.

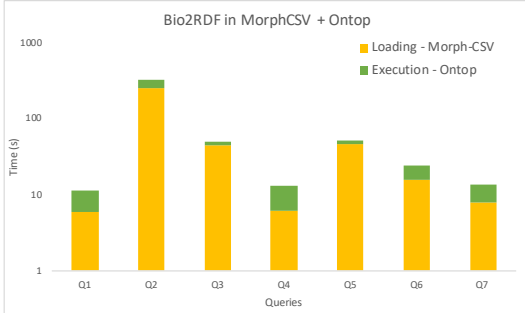
Evaluation Time with Ontop. The experimental evaluation of the query execution in Ontop as the back-end OBDA engine is shown in Figure 5.14. This engine is more strict with datatypes in the RDB in comparison with Morph-RDB, and it is why Q2, Q5, Q7 and Q9 produce a failure in the execution over the databases 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. Finally, in the case where Ontop is not able to evaluate the query under the defined threshold, we report it as a time-out.

5.1.4.3 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



(a) Query execution Time of Tabular Datasets in Bio2RDF with Morph-RDB. Execution time in seconds of the tabular datasets from Bio2RDF. The loading time applying a set of constraints (yellow) and query execution with Morph-RDB (green).



(b) Query execution Time of Tabular Datasets in Bio2RDF with Ontop. Execution time in seconds of the tabular datasets from Bio2RDF. The loading time applying a set of constraints (yellow) and query execution with Ontop (green).

Figure 5.15: Query execution Time of Tabular Datasets in Bio2RDF

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 **??**).

Bio2RDF Results The results obtained for query evaluation in Bio2RDF are shown in Figure 5.15b with Morph-RDB as back-end engine and in Figure 5.15b with Ontop. First, 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. 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 pre-processing 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. More in detail, we observe that due to the size of the input datasets and the number of constraints to be applied, most of the total evaluation time of each query is spent in the loading process. Contrary, query execution is benefited by this previous step obtaining the results in reasonable time for all of the queries.

5.2 Morph-GraphQL: 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[41], joinmonster[42]).

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").

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 paper, 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.*, 2012) and its extensions, have been used to generate

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

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 in this paper 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.

5.2.1 The Morph-GraphQL framework

The Morph-GraphQL framework (Figure 5.16) 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.

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-

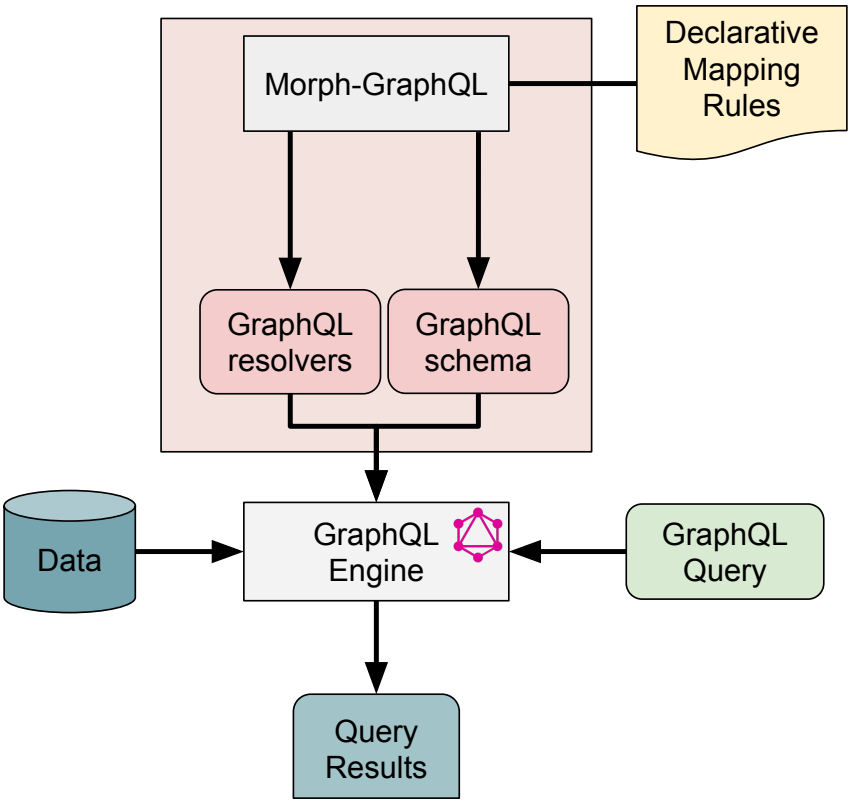


Figure 5.16: morph-GraphQL workflow. morph-GraphQL receives declarative mappings and generates GraphQL servers. These servers, as it is defined in the specification, contain their two main components (i.e. schema and resolvers) that can be used by any GraphQL engine to evaluate queries over the data source.

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`.

5.2.1.1 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. For example, given Listing **??** as the input, these functions generate the SQL query shown in variable *sql* in Listing **??**.

- $\alpha(TriplesMap)$ 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.

5.2.1.2 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.

5.2.1.3 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.

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
```

5.2.1.4 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 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
```

5.2.2 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 functions defined in section **??** (*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*.

5.2.3 Morph-GraphQL

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?

Algorithm 3 GenerateResvolver(TriplesMap)

```
resolver.init()
typeClass = getTypeFromClass(TriplesMap.getSubjectMap())
poms = TriplesMap.getPredicateObjectMaps()
for all pom ← poms do
    attribute = getConstant(pom.getPredicateMap())
    attributes.add(attribute)
end for
resolver.add(defineListQueryFunction(typeClass, attributes))
resolver.add(translateQuery(trans(TriplesMap))
resolver.add(execute(query, rdb))
resolver.add(constructResults(attributes, queryResults))
return Resolver
```

- **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**[43,44]. 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[45] to generate efficient SQL queries when joins are needed.

5.2.3.1 Linköping GraphQL Benchmark

Currently, the only GraphQL benchmark available is the Linköping GraphQL Benchmark (LinGBM), proposed by Hartig et al. (Hartig *et al.*, 2019). 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).

[43] `https://github.com/oeg-upm/morph-graphql`
[44] `https://doi.org/10.5281/zenodo.3584339`
[45] `https://join-monster.readthedocs.io`

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[46] 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[47] introduced by the Linked Data Benchmark Council[48]. Five classes of choke-points that are proposed are:

1. Choke Points Related to Attribute Retrieval. (1 check-point)
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 5.4.

5.2.3.2 Evaluation Setup

We used the LinGBM Data Generator to generate the various sizes of datasets[49] (1K, 2K, 4K, 8K, 16K, 32K, 64K and 128K) and loaded them in the relational database. We created 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

[46] `https://github.com/LiUGraphQL/LinGBM/wiki/Datasets`

[47] `http://ldbcouncil.org/blog/choke-point-based-benchmark-design`

[48] `http://ldbcouncil.org/`

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

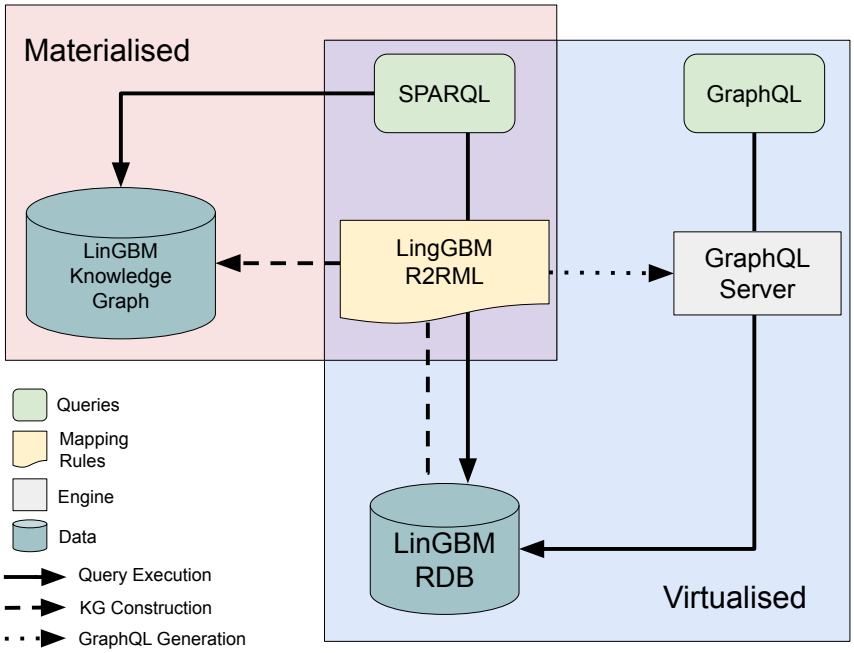


Figure 5.17: Evaluation Workflow. We evaluated Morph-GraphQL by comparing its performance over the supported LinGBM queries againts two equivalent Semantic Web approaches. First one is the materialisation of the LinGBM RDB to RDF using R2RML mappings and the second one is the translation from SPARQL-to-SQL using also the the same mapping rules.

declarative mappings. All the resources used in the evaluation is available online on our GitHub repository[50]. Figure 5.17 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 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.

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

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 5.4: Supported Queries in Morph-GraphQL

- 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.

5.2.3.3 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.

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 5.5: 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.

We show the results for the different dataset sizes in Table 5.5. 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 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.

Chapter 6

Evaluation in Knowledge Graph Construction

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

Knowledge graphs are often generated based on rules that apply semantic annotations to certain data. For example, the DBpedia knowledge graph is generated by applying classes and predicates of the DBpedia ontology to Wikipedia (Lehmann *et al.*, 2015). Software tools execute these rules and generate corresponding RDF triples and quads (Cyganiak *et al.*, 2014), which materialize knowledge graphs. In the past, custom scripts prevailed, but lately rule-driven tools emerged. Such tools distinguish the rules that define how RDF terms and triples are generated from the tool that executes them. R2RML (Das *et al.*, 2012) is the W3C recommended language to define such rules for generating knowledge graphs from data in relational databases (RDBs). An R2RML processor is a system that, given a set of R2RML rules and a relational database, generates an output RDF dataset. Examples of R2RML processors are, e.g., Ultrawrap (Sequeda & Miranker, 2013), Morph-RDB (Priyatna *et al.*, 2014), Ontop (Calvanese *et al.*, 2017), and XSPARQL (Bischof *et al.*, 2012). A subset of them was included in the RDB2RDF Implementation Report (Das *et al.*, 2012) to determine their conformance to the R2RML specification [51], i.e., the correct knowledge graph is generated for a set of rules and certain relational database.

Extensions and adaptations were applied to R2RML to account for other types of data sources, given that R2RML is focused on relational databases only, such as RML (Dimou *et al.*,

[51] Some of those available in the report are no longer actively maintained and used

2014), XSPARQL (Bischof *et al.*, 2012), xR2RML (Michel *et al.*, 2015), KR2RML (Slepicka *et al.*, 2015), and D2RML (Chortaras & Stamou, 2018a). RML provides an extension of R2RML to support heterogeneous data sources, including different formats, e.g., CSV, XML, JSON, and access interfaces, e.g., files and Web APIs. Similarly, RML processors emerged that execute RML rules, such as the RMLMapper[52], CARML[53], GeoTriples[54], and Ontario[55]. 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. Consequently, 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 work, (i) 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, (ii) we tested the conformance of the RMLMapper 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?

The test cases results shows that the RMLMapper passes all test cases regarding CSV, XML, and JSON format, and most test cases for RDBs, but fails the test cases for automatic datatyping of literals. CARML passes most test cases regarding CSV, XML, and JSON format, except of the test cases that deal, for example, with multiple RDF terms generation. Users can now determine how conformant the different processors are to the RML specification and use this conformance to determine the most suitable processor for their use cases.

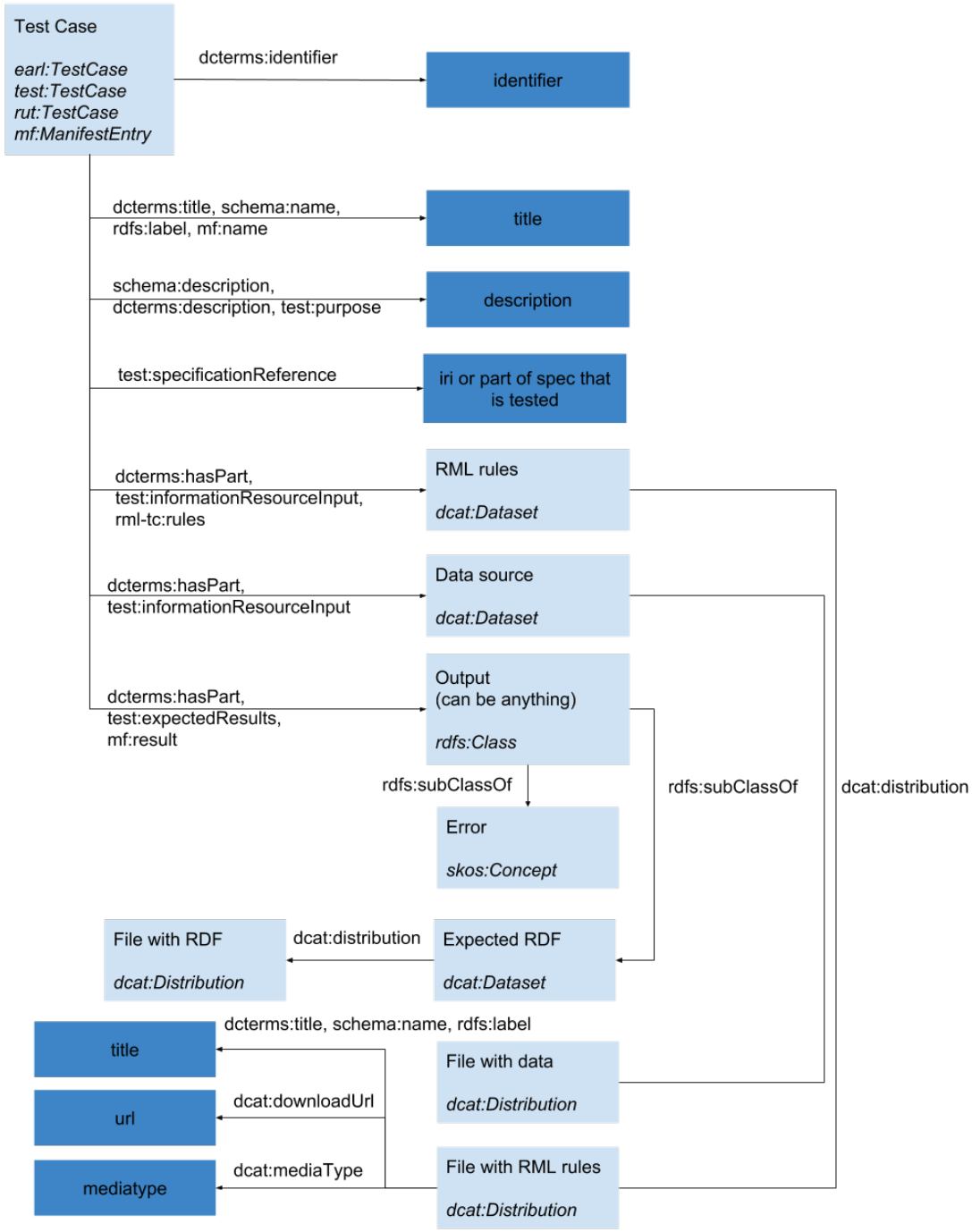


Figure 6.1: Data model of the RML test cases

6.1.1 RML Test Cases

In this section, 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[56], PostgreSQL[57], and SQL Server[58]) 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`. In Section **??**, we describe the data model that is used to represent the test cases. In Section **??**, we elaborate on the different files making up a test case. In Section **??**, we discuss the differences between the R2RML and RML test cases.

6.1.1.1 Data model

We describe the test cases semantically to increase their reusability and sharability. To this end, we created a semantic data model[59], with as main entity the test case (see Figure 6.1). For 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 Evaluation and Report Language (EARL) 1.0 Schema[60], the Test case manifest vocabulary[61], the Test Metadata vocabulary[62], and the Data Catalog Vocabulary[63]. 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

[52] RMLMapper, `https://github.com/RMLio/rmlmapper-java`
[53] CARML, `https://github.com/carml/carml`
[54] GeoTriples, `https://github.com/LinkedEOData/GeoTriples`
[55] Ontario,`https://github.com/WDAqua/Ontario`
[56] `https://www.mysql.com/`
[57] `https://www.postgresql.org/`
[58] `https://www.microsoft.com/en-us/sql-server/`
[59] `http://rml.io/test-cases/#datamodel`
[60] `https://www.w3.org/TR/EARL10/`, with prefix `earl`
[61] `http://www.w3.org/2001/sw/DataAccess/tests/test-manifest#`, with prefix `mf`
[62] `https://www.w3.org/2006/03/test-description#`, with prefix `test`
[63] `https://www.w3.org/TR/vocab-dcat/`, with prefix `dcat`

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`[64]. 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.

6.1.1.2 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 organised 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 behaviours of the processors. Certain test cases assess the behaviour 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 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).

6.1.1.3 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

[64] `http://rml.io/ns/test-cases`, with prefix `rml-tc`

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[65]. 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.

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.

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

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.

6.1.2 Conformance of RML Parsers

In this section, we describe the executing of the test cases and their results for two RML processors: the RMLMapper and CARML. 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[66]. After this, we present the results of the RMLMapper and CARML.

6.1.2.1 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 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

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

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. The code is available online[67] together with a Web page showing the results of all tested tools[68]. 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". **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 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.

6.1.2.2 Results

We perform the test-cases over two processors: RMLMapper and CARML. In Table 6.1 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[69]. 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 6.2 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

[67] `https://github.com/RMLio/rml-implementation-report`
[68] `http://rml.io/implementation-report`
[69] `https://www.w3.org/TR/r2rml/#dfn-natural-rdf-literal`

RMLMapper	CSV	XML	JSON	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 6.1: Results of the RMLMapper

CARML	CSV	XML	JSON	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 6.2: RML test-cases results for CARML

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.

Finally, we can declare that testing a RML processor with the defined cases and analysing 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.

6.2 Parameters that Affect a Knowledge Graph Construction

Following the FAIR principles (Wilkinson *et al.*, 2016) and Open data initiatives, the size of publicly available data has grown exponentially in the last decade, expecting a faster growth rate in the following years as a result of the advances in the technologies for data generation and ingestion. In order to extract values for existing datasets, several data integration approaches have been proposed in the literature (Halevy, 2018). The Semantic Web community has also proposed various approaches that enable the integration of data presented in diverse formats into a knowledge graph. Knowledge graphs comprise data and the knowledge that describe the main characteristics of the integrated data following a graph-based data model, e.g. RDF

(Vidal *et al.*, 2019). With the aim of transforming structured data in tabular or nested formats like CSV, relational, JSON, and XML, into RDF knowledge graphs, diverse mapping languages have been proposed. Exemplary mapping languages include RDF Mapping Language (RML) (Dimou *et al.*, 2014), R2RDF (Sequeda *et al.*, 2014), xR2RML (Michel *et al.*, 2015), and R2RML (Das *et al.*, 2012), as well as tools like KARMA (Gupta *et al.*, 2012), SPARQL-Generate (Lefrançois *et al.*, 2017), and DIG (Knoblock & Szekely, 2015). Despite these developments, the absence of testbeds has prevented the community from conducting fair evaluations of the existing tools for knowledge graph creation. 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.

Our goals: We study the process of knowledge graph creation and analyze various variables and configurations that can impact on the performance of RDFizers – tools for transforming (semi)-structured data following mapping rules specified in the RDF Mapping Language (RML). The relevant parameters studied in this paper include selectivity of the joins between mapping rules, types of relations, and percentage of duplicates. We also present diverse examples that evidence the heterogeneous behaviour that each RDFizer may exhibit whenever small changes are conducted to the variables and the configurations of a testbed.

Our Approach: We devise a set of parameters involved in a knowledge graph construction process and we empirically show how they can impact on the behaviour of two existing RDFizers: RMLMapper[70] and SDM-RDFizer[71]; these engines are compliant with the RML specification according to a set of defined test-cases[72]. 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[73].

Contributions: 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

[70] `https://github.com/RMLio/rmlmapper-java`
[71] `https://github.com/SDM-TIB/SDM-RDFizer`
[72] `http://rml.io/implementation-report/`
[73] `https://github.com/SDM-TIB/KGC-Param-Eval`

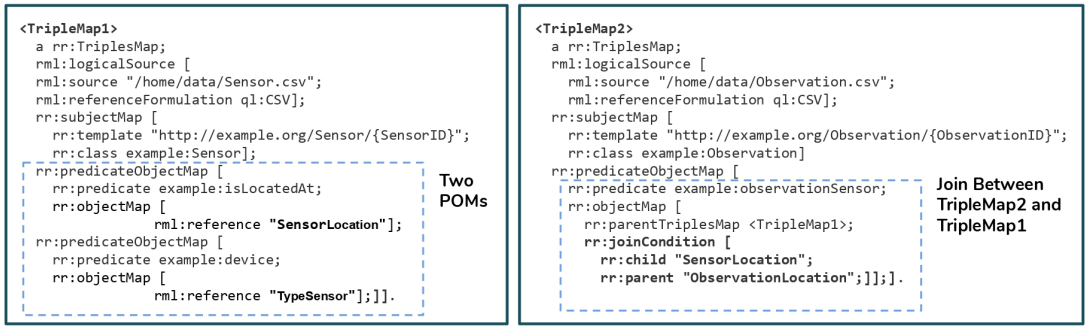


Figure 6.2: Motivating Example. RML triple maps to transform two CSV files into RDF. TripleMap1 is composed of two predicate-object, i.e., Two POM. TripleMap2 has a join to TripleMap1; Observation.csv (outer relation) is joined to Sensor.csv (inner relation) and the result, SensorID, is used as an object value.

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

6.2.1 Motivation

We motivate our work by analysing different scenarios where the performance of RMLMapper and SDM-RDFizer may be affected by changing the configuration of the testbeds utilised 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.

RML expresses mappings to transform sources represented in (semi)-structured format, e.g. CSV or XML, into RDF. Each mapping rule in RML, named RML triple map, is represented in RDF and consists of the following parts (Dimou *et al.*, 2014):

- A *Logical Source* that refers to a data source from where data is collected.
- A *Subject Map* that defines the subject of the generated RDF triples.

- *Predicate-Object Maps* (POM) that expresses the predicate and the object the RDF triple to be generated; a triple map can comprise several POMs.
- A *Referencing Object Map*, that indicates the reference or join condition to another triple map; the subject URL is the referenced triple map corresponds to the result of the evaluation of the join.

Figure 6.2 illustrates two RML triple maps. `TripleMap1` is composed of two predicate-object maps, i.e. it is a Two POM mapping rule. `TripleMap2` has a referencing object map that joins the records of file Observation.csv with the records of the file Sensor.csv on the attributes `SensorLocation` and `ObservationLocation`. The result of executing the join between the two RML triple maps is the identifier of the sensor that collected the observation; this value is used as the object value of the predicate `observationSensor`.

6.2.1.1 Impact of Number of Predicates and Objects in Mapping Rules

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 RDFizers exhibit a similar behaviour while the number of predicate-object maps varies from two to five POMs, as shown in Table 6.3. However, when more complex mapping rules with more POMs are considered, the behaviour 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.

6.2.1.2 Impact of Join Selectivity

We consider 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; Figure 6.2 depicts an example of a join between two RML triple maps. The join selectivity varies from **High Selectivity**, **Medium Selectivity**, and **Low Selectivity**, and Table 6.4 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 behaviour is not equally affected by the selectivity of the join condition. As can be seen in Table 6.4, 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

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 6.3: Impact of Number of Predicate-Object Maps. Various predicate object maps (POM) specified in the mapping rules. The behaviour of the two RDFizers 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.

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.

The uncorrelated behaviour of studied RDFizers shows clearly the need to considering diverse variables and configurations during the definition of testbeds, and thus, uncovering characteristics of these engines. In this paper, 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.

6.2.2 Relevant Parameters for Testbed Design

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.

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

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 6.5. These variables are described in detail in the next section.

6.2.2.1 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.

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 6.5: Variables and Configurations. Set of variables and configurations that impact on the behaviour of the tools for knowledge graph creation. Independent variables are divided into five groups and the impact on the observed variables is depicted.

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.

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 6.6: 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.

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 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 (as illustrated in Section **??**) and types of relation have also impact on the performance of an RDFizer. In Table 6.6, we illustrate how the relation type affects the total execution time of the studied RDFizers. In this case, the behaviour of the RMLMapper only occurs when the relation type is N-M. However, the SDM-RDFizer behaviour is impacted during the evaluation of 1-N and N-M joins. Additionally, during the join evaluation, there are many cases when 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 6.7: 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.

are generated, then the engines have to eliminate them. Table 6.7 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.

6.2.2.2 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 generation 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 com-

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 6.8: 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 behaviours in vertical partitioning.

pleteness 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 6.8 compares the behaviour of the RMLMapper and SDM-RDFizer 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.

6.2.2.3 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.

6.2.2.4 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.

6.2.2.5 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 RDFizer generates 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 RDFizers 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 RDFizers 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 behaviour 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.

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

Table 6.9: Datasets. Properties of Datasets used in the Empirical Evaluations.

6.2.3 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 creation 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 behaviour of existing knowledge graph creation tools? To answer these research questions, we set up the following experimental studies:

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 6.9 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 6.9, 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.

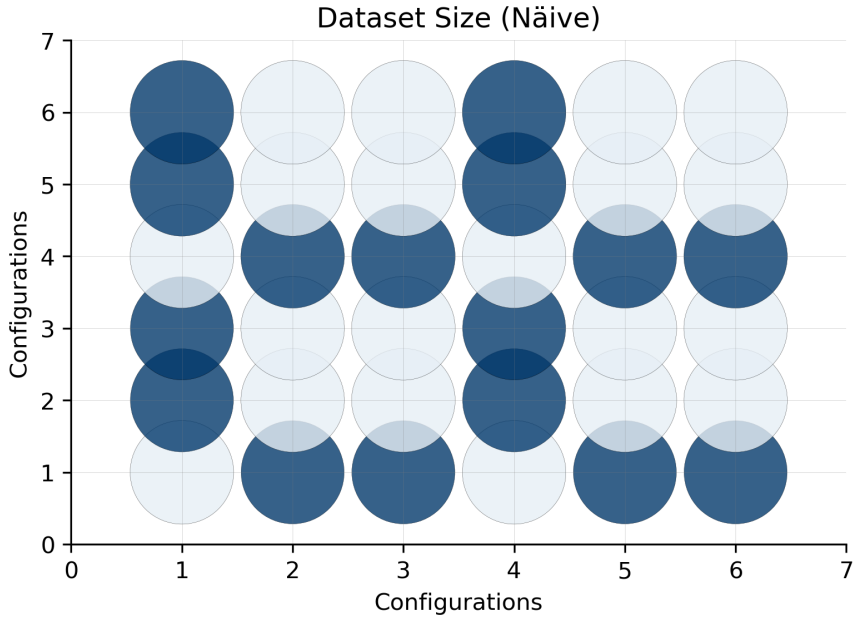


Figure 6.3: Comparison of Knowledge Graph Creation Tool on Different Dataset Sizes (Naïve). The first three configurations, i.e. 1, 2, and 3 in x-axis and y-axis, correspond to SDM-RDFizer on datasets 1K, 10K, and 50K, respectively. The last three configurations, i.e. 4, 5, and 6 on x-axis and y-axis, 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 behaviour.

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 behaviour 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 RDFizer and the delivery of the results.*Number of Results*: Number of triples generated by the RDFizer.

Implementations. The SDM-RDFizer and the testbeds are implemented in Python 3.6; the

SDM-RDFizer is publicly available[74]. 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[75]. 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 configurations indicates that the corresponding RDFizers 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 RDFizers, i.e. they exhibit an opposite behaviour; they are represented with red bubbles.

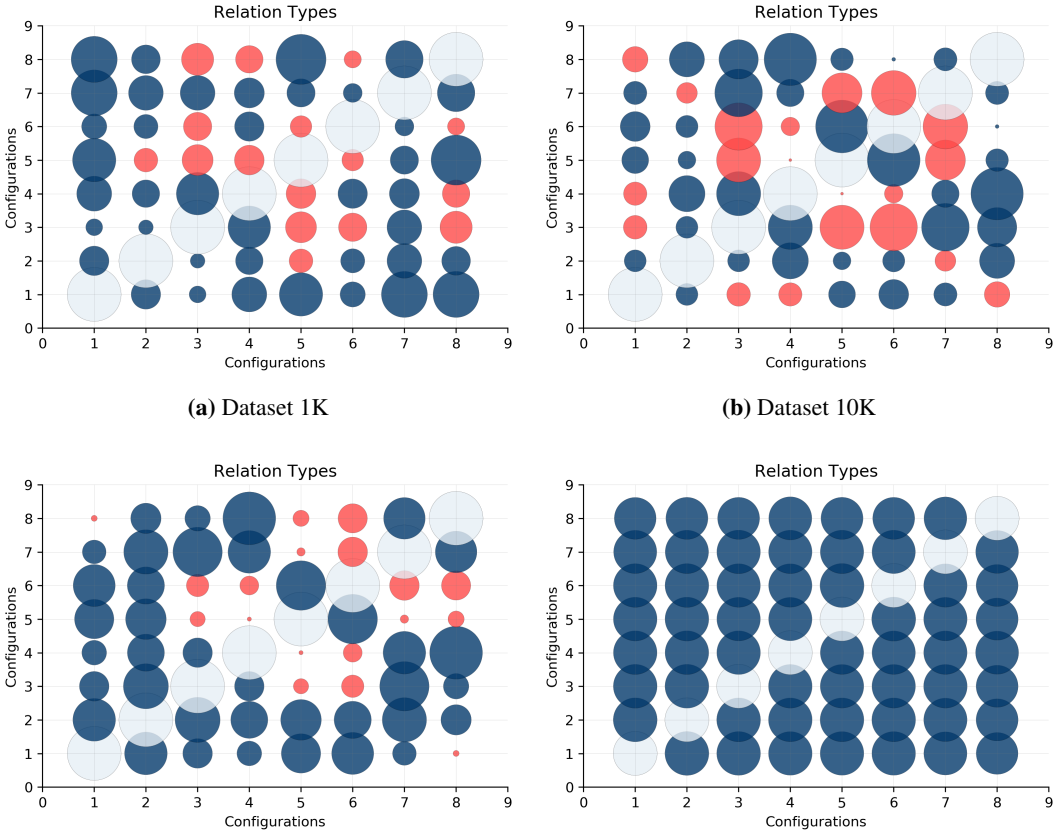
Discussion of the Observed Results

We observe that the behaviour 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 6.3 depicts the comparison of RDFizers 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 RDFizers 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 6.4 reports on the correlation of different configurations for various join relation types. We can observe in Figures 6.4a, 6.4b, and 6.4c several red bubbles, indicating a negative correlation in the behaviour of the compared configurations and RDFizers. Contrary, Figure 6.4d does not depict any red bubble, suggesting thus, that the two RDFizers in all the configurations exhibit the same behaviour. These results clearly illustrate the need of

[74] `https://github.com/SDM-TIB/SDM-RDFizer`
[75] `https://github.com/SDM-TIB/KGC-Param-Eval`



(a) Dataset 1K

(b) Dataset 10K

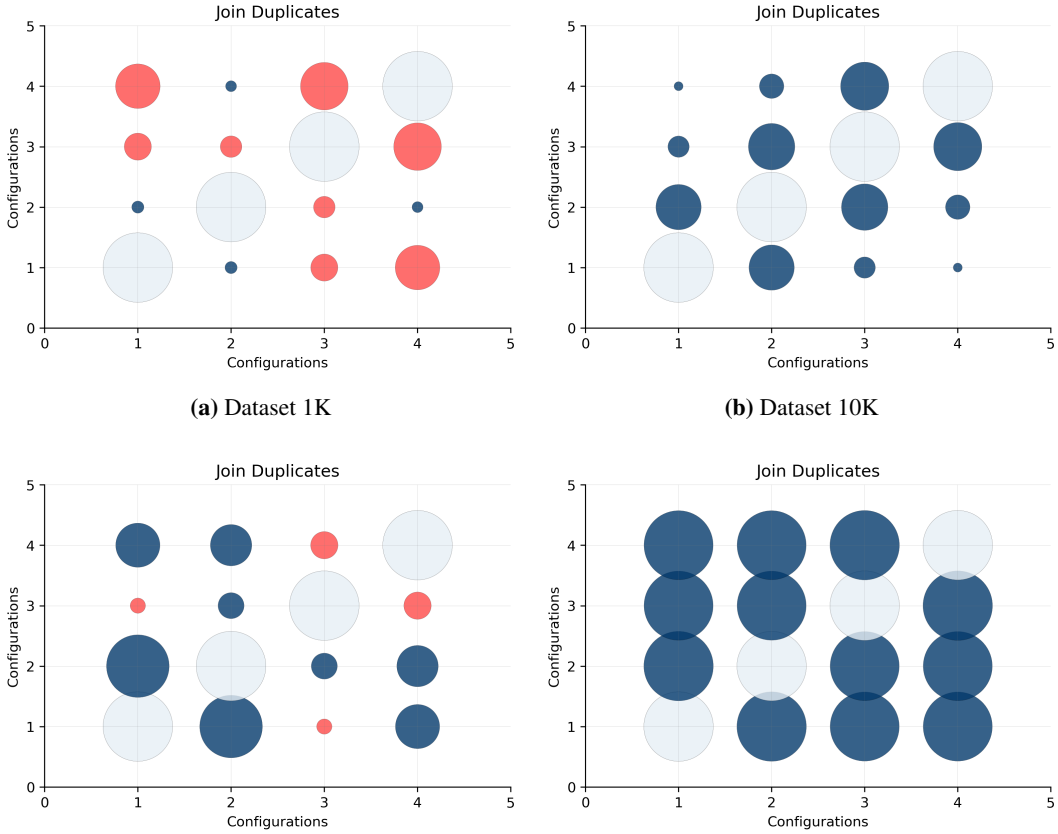
(c) Dataset 50K

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

Figure 6.4: Comparison of Knowledge Graph Creation 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 behaviour of the engines.

considering different configurations and parameters in order to avoid drawing wrong conclusions about the main characteristics of existing tools.

Join Duplicates: Figure 6.5 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 6.5a, and 6.5c include several red bubbles that indicate an opposite behaviour of the RDFizers. Contrary, Figures 6.5b, and 6.5d suggest that both engines behave



(a) Dataset 1K

(b) Dataset 10K

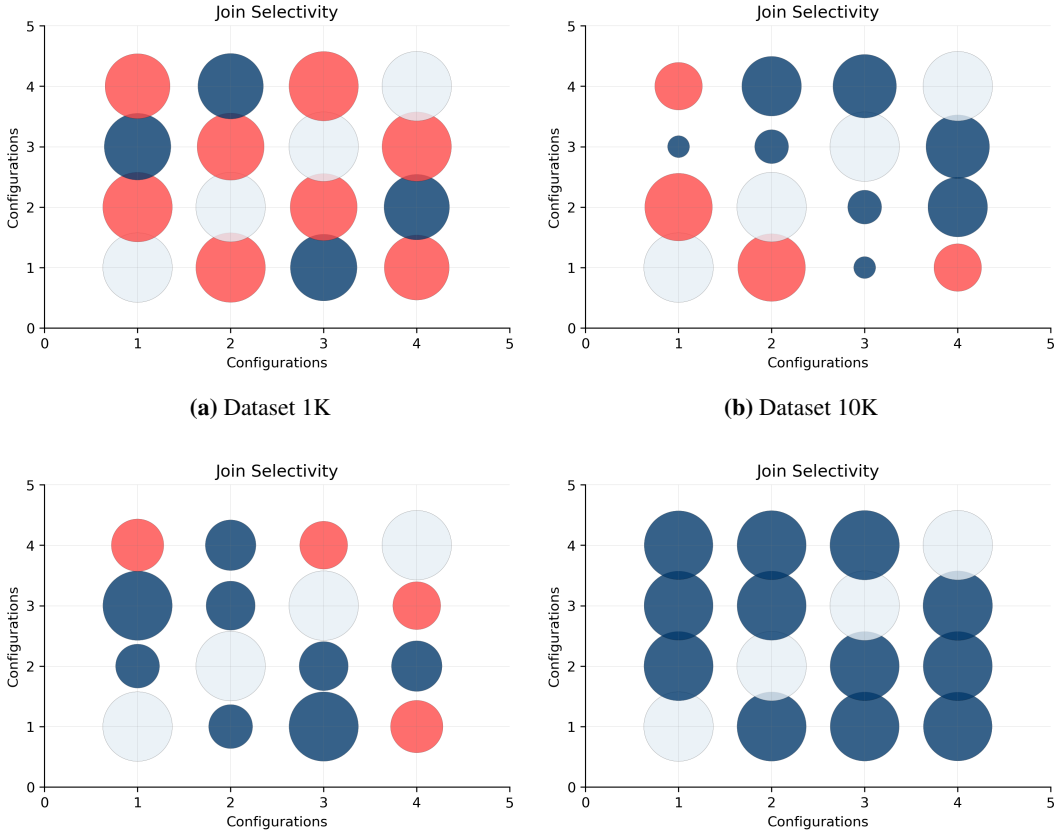
(c) Dataset 50K

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

Figure 6.5: Comparison of Knowledge Graph Creation 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.

similarly.

Join Selectivity: Figure 6.6 shows the correlation between different configurations for the selectivity of join conditions. Similarly, these testbeds reveal contradicting patterns in the behaviours of the studied RDFizers. On one hand, Figures 6.6a, 6.6b, and 6.6c are composed of several red bubbles and indicate that these engines perform differently whenever the selectivity



(a) Dataset 1K

(b) Dataset 10K

(c) Dataset 50K

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

Figure 6.6: 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.

of the join condition is changed. Surprisingly, when the size of these datasets are also taken into account in the testbed (Figure 6.6d), these patterns are hidden, and the results of the evaluation suggest that both RDFizers perform similarly whenever the selectivity of the join condition is changed.

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 that the characteristics of knowledge graph creation 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 creation tools to bear in mind them during benchmarking in order to draw clear conclusions about the performance of their tools.

6.2.3.1 Conclusions

In this paper, we performed an in-depth analysis of the variables and configurations that impact on the behaviour of two RDFizers. The observation that existing RDFizers exhibit heterogeneous behaviours 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 behaviour of the studied engines. Additionally, the lack of testbeds encouraged us to acquit the definition of variables and configurations that enable for the characterisation of the pitfalls of existing RDFizers 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 creation tools and mapping languages (e.g. SPARQL-Generate, TARQL, or R2RML). Furthermore, our set of variables and configurations can be utilised 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 RDFizers 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 creation of knowledge graphs at large scale. In the future, we plan to define testbeds and conduct a more detailed analysis of other RDFizers 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 creation tools with replicable and generalizable results.

Chapter 7

GTFS-Madrid-Bench: Evaluation of Virtual Knowledge Graph Access

Over the last few years, a growing number of datasets have been made available in various open data portals. For example, the European Data Portal[76] aggregates, at the time of writing, approximately 500K datasets from EU countries in a diversity of domains. In this context, RDF has been proposed as a standard format for data interchange on the Web, and RDF Schema and OWL ontologies have begun to appear so as to provide shared models in some domains. However, the amount of non-RDF data (e.g., CSV, JSON, XML) that is published in these open data portals continues to dominate the scene (see Table 7.1) and interoperability issues hinder their (re)use and consumption.

Data integration is not a new problem but it is exacerbated by the availability of such open data on the Web, that is, it was already identified and addressed several decades ago with an emphasis on data in relational databases. Different techniques and tools have been used to address this problem. In our work, we focus on those based on ontologies. In Ontology Based Data Access (OBDA) (Poggi *et al.*, 2008) data consumers issue queries over a dataset according to a common unified view (an ontology). The information needed to reformulate the queries is usually available in the form of declarative mappings. In Ontology Based Data Integration (OBDI) (Poggi *et al.*, 2008), these techniques are expanded to address heterogeneous datasets, whose data needs to be integrated to provide answers to these queries. In both ontology-based approaches, two different alternatives exist to enable data access: (1) those where data are materialised taking into account the mappings and the ontologies (for example, data is

[76] https://www.europeandataportal.eu/catalogue-statistics/CurrentState

transformed into RDF and loaded into a triple store, so that it can be queried using SPARQL), and (2) those where the transformation is done on the queries, which can then be evaluated on the original data sources. This last alternative is the one considered for our work because it removes the need for materialisation, something especially useful for very dynamic data sources (Corcho *et al.*, 2019). We refer to it as "virtualized knowledge graph access".

To facilitate data exploitation in this context, application developers need to understand the strengths and weaknesses of existing data integration tools. Additionally, tool developers may want to know if their engines cover the requirements of real use-case scenarios. In both cases the challenge is to develop a benchmark that covers the requirements of "virtualized knowledge graph access" and that is extensible and sustainable over time. 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.

Several benchmarks already exist in the state of the art of OBDA (Bizer & Schultz, 2009; Lanti *et al.*, 2015), as well as in SPARQL query federation (Hasnain *et al.*, 2017; Montoya *et al.*, 2012; Schmidt *et al.*, 2011). The OBDA 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 OBDA engines that access relational data stores. The NPD benchmark (Lanti *et al.*, 2015) specifically analyzes OBDA requirements related to datasets, query sets, mapping rules and query languages. In the area of federated SPARQL engines, existing benchmarks (Hasnain *et al.*, 2017; Montoya *et al.*, 2012; Schmidt *et al.*, 2011) are tailored to the context of SPARQL endpoint federation in an homogeneous format. As a result, none of these benchmarks address the requirement of virtualized access of multiple datasets available in heterogeneous formats. Additionally, OBDI 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 OBDI proposals in a systematic manner.

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

Data Portal	1st Format	2nd Format	3rd Format
Spain	CSV (50%)	XLS (35%)	JSON (33%)
Norway	CSV (77%)	GEOJSON (17%)	JSON (14%)
Italy	CSV (76%)	JSON (35%)	XML (25%)
Croatia	XLS (63%)	CSV (40%)	HTML (33%)
Luxembourg	ZIP (25%)	CSV (24%)	PDF (18%)
Ireland	JSON (49%)	CSV (39%)	TXT (22%)

Table 7.1: Most commonly used formats (and percentage over the total number of datasets) to publish data in mature EU open data portals[77]

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.

In this paper 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 OBDA 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 OBDI engines; and (iii) to evaluate the strengths and weaknesses of both, OBDA and OBDI engines. The general case of GTFS-Madrid-Bench is the comparison of the performance of OBDA and OBDI 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[78] 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[79].
- 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 OBDI workflow (Acosta *et al.*, 2011;

[77] Statistics obtained in January 2019 (note that one dataset can be made available in multiple formats)

[78] `https://developers.google.com/transit/gtfs/` 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.

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

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 OBDA and OBDI 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 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[80], Ontop[81], Ontario[82], Morph-CSV[83] and Morph-xR2RML[84].

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 OBDA and OBDI 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 OBDA/OBDI 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.

[80] `https://github.com/oeg-upm/morph-rdb`
[81] `https://github.com/ontop/ontop`
[82] `https://github.com/SDM-TIB/Ontario`
[83] `https://github.com/oeg-upm/morph-csv`
[84] `https://github.com/frmichel/morph-xr2rml`

7.1 Preliminaries

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 the paper.

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.

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 7.5, and M is the set of mappings $\{shapes, trips\}$ reported in Table 7.4.

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.

7.2 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 7.2). All the resources described in this section are available online[85].

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

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 7.2: Virtual Knowledge Graph Access Benchmark Requirements

7.2.1 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[86] 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 7.1 and a description of its classes is given in Table 7.3. 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`

[86] `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 7.3: LinkedGTFS classes and their descriptions

and `gtfs:FareRule` because the Madrid GTFS data does not contain information on these two entities. The `gtfs:RouteType` class is not considered because the data covers only the Metro system.

7.2.2 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 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

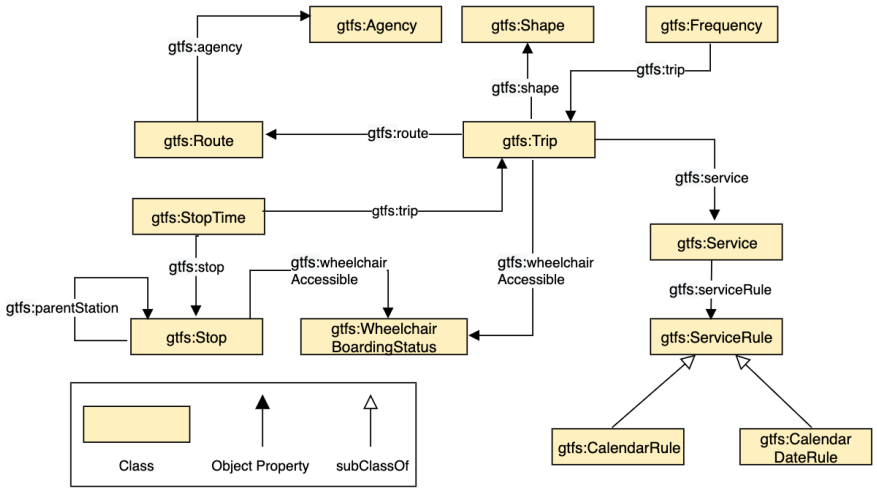


Figure 7.1: LinkedGTFS Ontology. Subset of the LinkedGTFS ontology used in the GTFS-Madrid-Bench for virtual knowledge graph access. There are eleven object property relations among the classes, and two subClassOf relations.

an example in Figure 7.4). We describe the detailed steps of the proposed data generation workflow together with examples following the definitions provided in Section **??**:

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[87], 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 $\text{GTFS}_{mad}^{CSV} = (\text{GTFS}_{mad}, \text{GTFS}, \text{CSV})$. GTFS_{mad} is the set of data sources of the subway network of Madrid that has been provided by its transport authority according to the schema GTFS as described in its specification[88]. 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

[87] `https://github.com/oeg-upm/morph-csv`
[88] `https://developers.google.com/transit/gtfs/`

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[89] and di-csv2xml[90], 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 OBDI 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 (joins) among sources in different formats is minimised or maximised. The formats of the sources in these datasets is also configurable during the generation data process, in order to allow users to analyse 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

[89] `https://csvkit.readthedocs.io/en/1.0.3/scripts/csvjson.html`
[90] `https://github.com/blue-yonder/di-csv2xml`

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 behaviour 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 7.2.
- 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 OBDI 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 7.3.

We want to be able to compare the results obtained by processors with the results obtained by the materialised graph in RDF. For this purpose, we take the output of VIG (e.g GTFS-CSV-5, GTFS-CSV-10) and we run a knowledge graph creation process using the SDM-RDFizer[91] engine, which generates the materialised 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[92], hence we assume that the generation is correct, and that it provides a set of techniques to optimise the generation of RDF at scale.

[91] `https://github.com/SDM-TIB/SDM-RDFizer`
[92] `http://rml.io/implementation-report/`

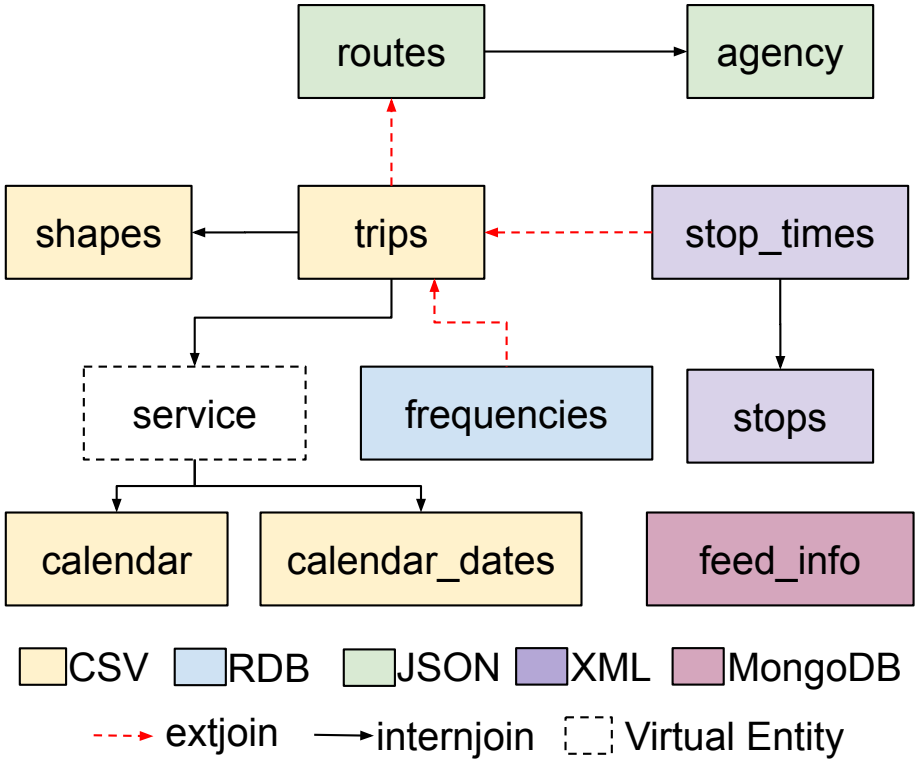


Figure 7.2: Example of Best Dataset. Best dataset distributes the formats over the data sources ensuring that at least there is one source per each format and the joins among different formats are minimised. *extjoin* means that there is a relation between sources in different formats and *internjoin* means that the joins are between sources in the same format.

7.2.3 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:

- One R2RML mapping document for accessing SQL datasets.
- One xR2RML mapping document for accessing MongoDB datasets.
- Seven RML mapping documents[93] for accessing CSV, JSON, XML, SQL, MongoDB,

[93] Provided in RML and YARRRML serializations

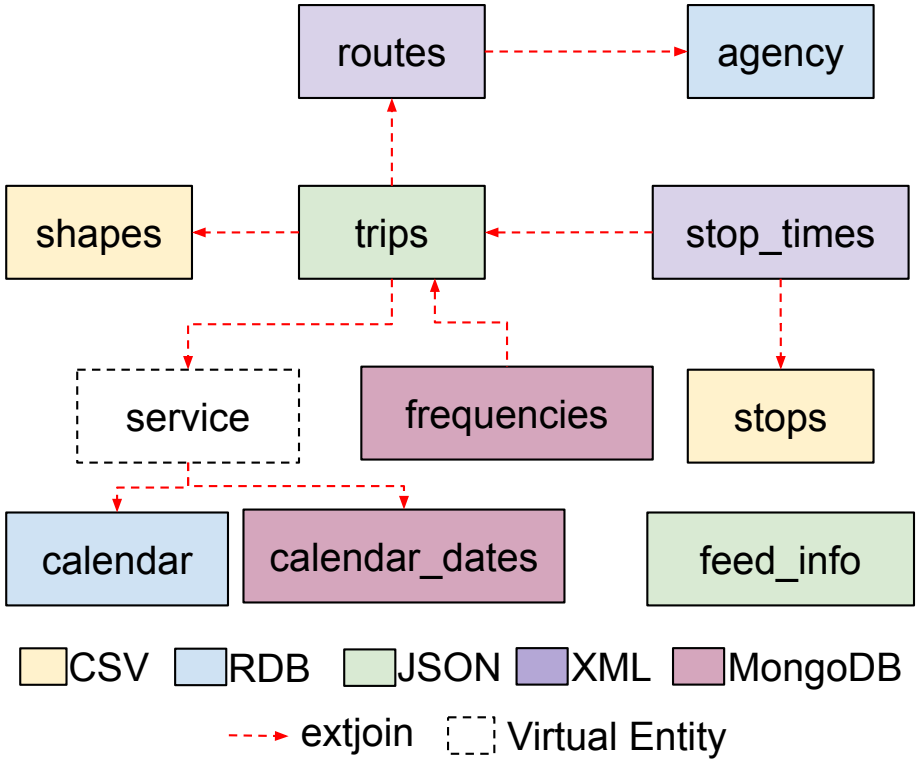


Figure 7.3: Example of Worst Dataset. Worst dataset distributes the formats over the data sources ensuring that at least there is one source per each format and the joins among different formats are maximised. *extjoin* means that there is a relation between sources in different formats.

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 this benchmark is shown in Table 7.4. 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

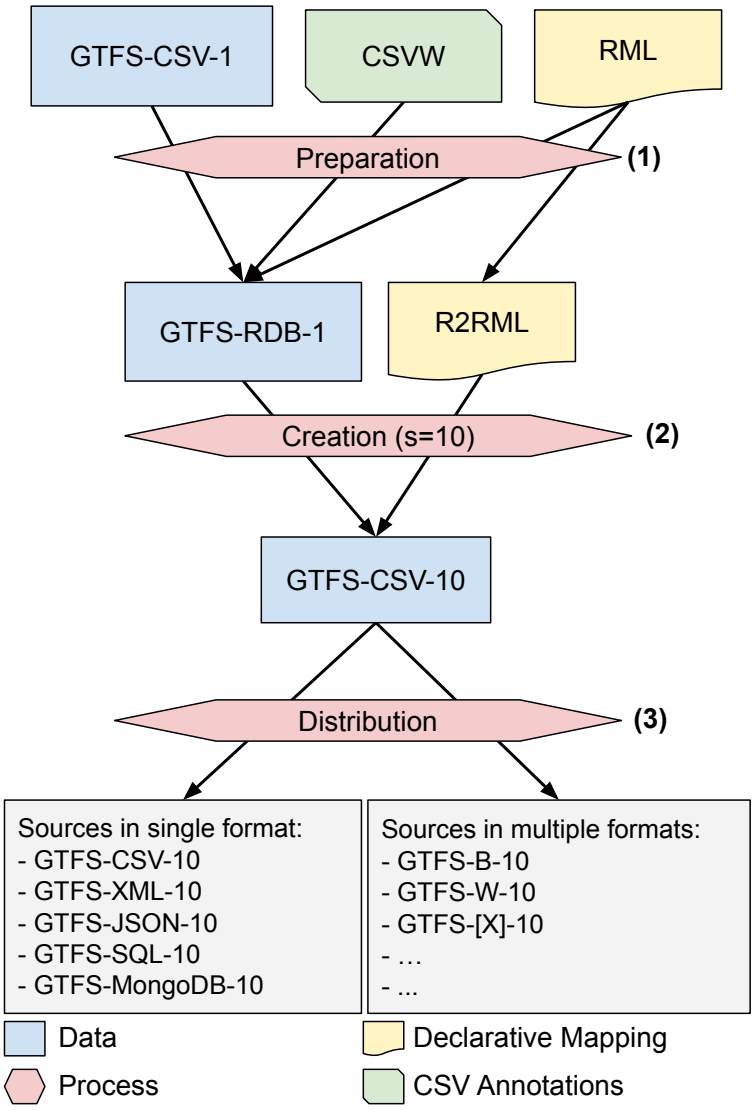


Figure 7.4: GTFS-Madrid-Bench Generation Workflow with scale value 10. From the original 10 CSV files of Madrid Metro GTFS we use (i) Morph-CSV to generate the corresponding RDB instance and an R2RML mapping that are the required inputs for (ii) scaling up the data using VIG and (iii) distributing the 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 7.4: Mapping features of GTFS. Each TriplesMap of the GTFS mapping file and its corresponding features: the related source, number of Classes, PredicateObjectMaps, Predicates, Objects and RefObjectMaps (joins).

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 OBDA/OBDI. All of the GTFS mappings are detailed in Appendix C using the YARRRML (Heyvaert *et al.*, 2018) serialisation.

7.2.4 Queries

Table 7.5 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 described in Section **??**.

7.2.5 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 behaviour (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 access 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 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

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 7.5: GTFS-Madrid-Bench 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 7.6: 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.

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 7.6.

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 maximised). 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 7.1, `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 (OBDA), 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 (OBDI), the engine has to create the query plan. The performance will be affected by the plan proposed by the OBDI 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 optimisation 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 optimise 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 7.4. 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 in Section **??**, 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.

7.2.6 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 [94] 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 engines called *rdfizers* such as RMLMapper [95], RocketRML [96] or SDM-RDFizer [97] 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

[94] `https://github.com/oeg-upm/gtfs-bench/`
[95] `https://github.com/RMLio/rmlmapper-java`
[96] `https://github.com/semantifyit/RocketRML/`
[97] `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.

7.2.7 Experimental Evaluation

In this section we describe the evaluation performed using our benchmark. We first describe the selected OBDA and OBDI 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[98].

7.2.7.1 Tools

We selected the most relevant open source OBDA and OBDI engines in the state of the art:

Ontario. Ontario (Endris *et al.*, 2019)[99] is an OBDI engine 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)[100] is an OBDA system that includes both materialisation and virtualisation techniques. Ontop translates R2RML mappings into its own

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

[99] `https://github.com/SDM-TIB/Ontario`

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

mapping language, called "OBDA mappings". These mappings, and a SPARQL query if available, are transformed into datalog rules, allowing semantic optimisation 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)[101] is an R2RML engine that also includes materialisation and virtualisation techniques. The formalisation of its query translation technique is based on the R2RML-based extension of SPARQL-to-SQL query translation algorithm proposed by Chebotko et al. (Chebotko *et al.*, 2009), originally designed to work with RDB-backed triples store. Similar to Ontop, several optimisation 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)[102] 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.*, 2020)[103] 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 OBDI 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.

7.2.7.2 Setup

In this section we describe how we use our benchmark to evaluate several processors/engines that have been described in Section **??**.

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 7.7. These experiment configurations have a fixed set of mappings with routes and agencies. The processor

[101] `https://github.com/oeg-upm/morph-rdb`
[102] `https://github.com/frmichel/morph-xr2rml`
[103] `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 7.7: 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.

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.

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 7.8: 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.

7.2.7.3 Results

In this section we report the results obtained through our experimental configurations. Table 7.8 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 (7.9,7.10,7.11,7.12,7.13) 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 OBDI 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.

7.2.7.4 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 7.9: 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 7.10: 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 7.11: 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 7.12: 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 7.13: 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 OBDA/OBDI 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 optimisation 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).

- OBDA/OBDI 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 optimisations, 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 OBDI engines. Although the adaptation of current techniques proposed by federated SPARQL engines to OBDI 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.ç

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.

Chapter 8

Conclusions

8.1 Achievements

8.2 Future Work

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 ?wheelchairAccesibl
```

```
        ?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 ?wheelchairAccessible .
}
        FILTER (?location=<http://transport.linkeddata.es/resource/LocationT
}
```

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/
```

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: http://transport.linkeddata.es/madrid/metro/calendar_date_rule/$(s
 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, $(block_id)]
  - [gtfs:wheelchairAccessible,gtfsres:WheelchairBoardingStatus/$(wheelchair
  - 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,gtfsres:WheelchairBoardingStatus/$(wheelchair
  - 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)]
```

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. 103

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. 116

ANGLES, R. & GUTIÉRREZ, C. (2016). Negation in SPARQL. In *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, 2016*. 121

ARENAS, M., BERTAILS, A., PRUD'HOMMEAUX, E. & SEQUEDA, J. (2013). A direct mapping of relational data to rdf, w3c recommendation 27 september 2012. 13

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. 13

BEERI, C., BERNSTEIN, P.A. & GOODMAN, N. (1978). A sophisticate's introduction to database normalization theory. In *VLDB*. 43

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. 34, 59

BISCHOF, S., DECKER, S., KRENNWALLNER, T., LOPES, N. & POLLERES, A. (2012). Mapping between RDF and XML with XSPARQL. *Journal on Data Semantics*. 75, 76

BIZER, C. & SCHULTZ, A. (2009). The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, **5**, 1–24. 7, 34, 50, 51, 70, 102, 123

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. 32

BOTOEVA, E., CALVANESE, D., COGREL, B., CORMAN, J. & XIAO, G. (2019). Ontology-based data access–beyond relational sources. *Intelligenza Artificiale*, **13**, 21–36. 40

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. 6, 62

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. 6, 21, 32, 34, 51, 75, 130

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. 14, 16, 17

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. 103, 113, 121, 122

CHAVES-FRAGA, D., RUCKHAUS, E., PRIYATNA, F., VIDAL, M.E. & CORCHO, O. (2020). Enhancing OBDA Query Translation over Tabular Data with Morph-CSV. 124

CHEBOTKO, A., LU, S. & FOTOUHI, F. (2009). Semantics preserving SPARQL-to-SQL translation. *Data & Knowledge Engineering*, **68**, 973–1000. 66, 68, 119, 124

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**. 13, 76

CHORTARAS, A. & STAMOU, G. (2018b). Mapping diverse data to rdf in practice. In *International Semantic Web Conference*, 441–457, Springer. 5

CODD, E.F. (1979). Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, **4**, 397–434. 47

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*. 20, 21

CORCHO, O., PRIYATNA, F. & CHAVES-FRAGA, D. (2019). Towards a New Generation of Ontology Based Data Access. *Semantic Web Journal*. 55, 102, 104, 116, 132

CYGANIAK, R., REYNOLDS, D. & TENNISON, J. (2012). The rdf data cube vocabulary, w3c recommendation 16 january 2014. *World Wide Web Consortium*. 20

CYGANIAK, R., WOOD, D. & LANTHALER, M. (2014). RDF 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium (W3C). 75

DAS, S., SUNDARA, S. & CYGANIAK, R. (2012). R2RML: RDB to RDF Mapping Language. W3C Recommendation, W3C, `http://www.w3.org/TR/r2rml/`. 5, 13, 37, 63, 75, 84

DE MEESTER, B., DIMOU, A., VERBORGH, R. & MANNENS, E. (2016). An ontology to semantically declare and describe functions. In *ISWC*, 46–49, Springer. 37, 123, 132

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. 9, 19, 32, 33, 37, 38, 45, 48

DEBRUYNE, C. & O'SULLIVAN, D. (2016). R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In *LDOW@ WWW*. 37, 132

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*. 5, 7, 13, 37, 75, 84, 85, 109

DOAN, A., HALEVY, A. & IVES, Z. (2012). *Principles of data integration*. Elsevier. 37

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. 123

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. 33, 74, 79, 102, 123, 131

FACEBOOK, INC. (2018). GraphQL. `https://facebook.github.io/graphql/June2018/`, accessed: 2018-12-07. 6, 17, 62

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. 17

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. 37

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. 84

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. 37

HALEVY, A.Y. (2018). Information integration. In *Encyclopedia of Database Systems, Second Edition*. 83

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. 14

HARTIG, O., CHENG, S. & LINDQVIST, L. (2019). Linköping GraphQL Benchmark (LinGBM) kernel description. 69

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. 102

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. 16, 64

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*. 14, 16, 81, 115

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. 111

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*. 50, 59, 60

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. 44, 48

JUNIOR, A.C., DEBRUYNE, C., BRENNAN, R. & O'SULLIVAN, D. (2016). 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. 32, 37, 48

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. 124

KNOBLOCK, C.A. & SZEKELY, P. (2015). Exploiting semantics for big data integration. *Ai Magazine*, **36**. 64, 84

LANTI, D., REZK, M., XIAO, G. & CALVANESE, D. (2015). The npd benchmark: Reality check for obda systems. OpenProceedings. org. 7, 56, 102, 103, 104, 106, 116, 123

LANTI, D., XIAO, G. & CALVANESE, D. (2017). VIG: Data scaling for OBDA benchmarks. *Semantic Web*, 1–21. 104, 106, 108, 110

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. 7, 13, 84

LEHMANN, J., ISELE, R., JAKOB, M., JENTZSCH, A., KONTOKOSTAS, D., MENDES, P.N., HELLMANN, S., MORSEY, M., VAN KLEEF, P., AUER, S. & BIZER, C. (2015). Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, **6**, 167–195. 75

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. 32

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. 102, 124, 131

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. 33, 38, 51

MCGUINNESS, D.L., VAN HARMELEN, F. *et al.* (2004). OWL web ontology language overview. *W3C recommendation*, **10**, 2004. 105

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. 5, 13, 18, 76, 84, 124

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. 102, 120, 122

MORA, J. & CORCHO, O. (2013). Towards a systematic benchmarking of ontology-based query rewriting systems. In *International Semantic Web Conference*, 376–391, Springer. 103, 116

MORA, J., ROSATI, R. & CORCHO, O. (2014). kyrie2: Query rewriting under extensional constraints in \mathcal {ELHIO}. In *International Semantic Web Conference*, 568–583, Springer. 49, 119

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. 6, 62

PÉREZ, J., ARENAS, M. & GUTIÉRREZ, C. (2009). Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, **34**, 16:1–16:45. 120

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. 12, 21, 32, 43, 101

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*. 6, 13, 21, 32, 33, 34, 49, 51, 70, 74, 75, 111, 124, 130

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. 18, 69, 122

RODRIGUEZ-MURO, M. & REZK, M. (2015). Efficient SPARQL-to-SQL with R2RML mappings. *Web Semantics*, **33**, 141–169. 13, 20, 33, 123

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. 102

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. 116

SEQUEDA, J.F. & MIRANKER, D.P. (2013). Ultrawrap: SPARQL execution on relational data. *Web Semantics: Science, Services and Agents on the WWW*. 75

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. 14

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. 84

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. 116

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*. 7, 38

SLEPICKA, J., YIN, C., SZEKELY, P.A. & KNOBLOCK, C.A. (2015). KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources. In *COLD*. 5, 13, 76

TENNISON, J., KELLOGG, G. & HERMAN, I. (2015). Model for tabular data and metadata on the web. W3C recommendation. *World Wide Web Consortium (W3C)*. 5, 19, 32, 33, 37, 45, 109

THE W3C SPARQL WORKING GROUP (2013). SPARQL 1.1 overview. W3C recommendation, W3C, http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/. 105, 106

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. 46, 119

VIDAL, M., ENDRIS, K.M., JAZASHOORI, S., SAKOR, A. & RIVAS, A. (2019). Transforming heterogeneous data into knowledge for personalized treatments - A use case. *Datenbank-Spektrum*, **19**, 95–106. 84

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. 6, 62

WIEDERHOLD, G. (1992). Mediators in the architecture of future information systems. *Computer*, **25**, 38–49. 12

WILKINSON, M.D., DUMONTIER, M., AALBERSBERG, I.J., APPLETON, G., AXTON, M., BAAK, A., BLOMBERG, N., BOITEN, J.W., DA SILVA SANTOS, L.B., BOURNE, P.E. *et al.* (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, **3**. 32, 83

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*. 34, 37, 39, 40

XIAO, G., KONTCHAKOV, R., COGREL, B., CALVANESE, D. & BOTOEVA, E. (2018b). Efficient handling of SPARQL OPTIONAL for OBDA (extended version). *CoRR*, **abs/1806.05918**. 55, 120, 130