



Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

DOCTORAL THESIS

Methods and Techniques for the Generation and Efficient Exploitation of RDB2RDF Mappings

Author:

Freddy PRIYATNA

Supervisor:

Oscar CORCHO

*A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy*

in the

Escuela Técnica Superior de Ingenieros Informáticos
Departamento de Inteligencia Artificial

December 2015

Tribunal nombrado por el Sr. Rector Magfco. de la Universidad Politécnica de Madrid,
el día 24 de Noviembre de 2015

Presidente: Dra. Asunción Gómez Pérez

Vocal: Dr. Johan Montagnat

Vocal: Dra. Arantza Illarramendi

Vocal: Dr. Juan Sequeda

Secretario: Dra. Ernestina Menasalvas Ruiz

Suplente: Dr. Mariano Fernández López

Suplente: Dr. Boris Marcelo Villazon-Terrazas

Realizado el acto de defensa y lectura de la Tesis el día 18 de Diciembre de 2015 en la
Facultad de Informática

Calificación: _____

EL PRESIDENTE

VOCAL 1

VOCAL 2

VOCAL 3

EL SECRETARIO

Declaration of Authorship

I, Freddy PRIYATNA, declare that this thesis titled, 'Methods and Techniques for the Generation and Efficient Exploitation of RDB2RDF Mappings' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

“A wise man will hear, and will increase learning; and a man of understanding shall attain unto wise counsels:”

Proverbs 1:5, King James Version (KJV)

UNIVERSIDAD POLITÉCNICA DE MADRID

Abstract

Escuela Técnica Superior de Ingenieros Informáticos
Departamento de Inteligencia Artificial

Doctor of Philosophy

**Methods and Techniques for the Generation and Efficient Exploitation of
RDB2RDF Mappings**

by Freddy PRIYATNA

Abstract. RDB to RDF Mapping Language (R2RML) is a W3C recommendation that allows specifying rules for transforming relational databases into RDF. This RDF data can be materialized and stored in a triple store, so that SPARQL queries can be evaluated by the triple store. However, there are several cases where materialization is not adequate or possible, for example, if the underlying relational database is updated frequently. In those cases, RDF data is better kept virtual, and hence SPARQL queries over it have to be translated into SQL queries to the underlying relational database system considering that the translation process has to take into account the specified R2RML mappings.

The first part of this thesis focuses on query translation. We discuss the formalization of the translation from SPARQL to SQL queries that takes into account R2RML mappings. Furthermore, we propose several optimization techniques so that the translation procedure generates SQL queries that can be evaluated more efficiently over the underlying databases. We evaluate our approach using a synthetic benchmark and several real cases, and show positive results that we obtained.

Direct Mapping (DM) is another W3C recommendation for the generation of RDF data from relational databases. While R2RML allows users to specify their own transformation rules, DM establishes fixed transformation rules. Although both recommendations were published at the same time, September 2012, there has not been any study regarding the relationship between them.

The second part of this thesis focuses on the study of the relationship between R2RML and DM. We divide this study into two directions: from R2RML to DM, and from DM to R2RML. From R2RML to DM, we study a fragment of R2RML having the same expressive power than DM. From DM to R2RML, we represent DM transformation rules as R2RML mappings, and also add the implicit semantics encoded in databases, such as subclass, 1-N and N-N relationships.

This thesis shows that by formalizing and optimizing R2RML-based SPARQL to SQL query translation, it is possible to use R2RML engines in real cases as the resulting SQL is efficient enough to be evaluated by the underlying relational databases. In addition to that, this thesis facilitates the understanding of bidirectional relationship between the two W3C recommendations, something that had not been studied before.

Resumen. RDB to RDF Mapping Language (R2RML) es una recomendación del W3C que permite especificar reglas para transformar bases de datos relacionales a RDF. Estos datos en RDF se pueden materializar y almacenar en un sistema gestor de tripletas RDF (normalmente conocidos con el nombre triple store), en el cual se pueden evaluar consultas SPARQL. Sin embargo, hay casos en los cuales la materialización no es adecuada o posible, por ejemplo, cuando la base de datos se actualiza frecuentemente. En estos casos, lo mejor es considerar los datos en RDF como datos virtuales, de tal manera que las consultas SPARQL anteriormente mencionadas se traduzcan a consultas SQL que se pueden evaluar sobre los sistemas gestores de bases de datos relacionales (SGBD) originales. Para esta traducción se tienen en cuenta los mapeos R2RML.

La primera parte de esta tesis se centra en la traducción de consultas. Se propone una formalización de la traducción de SPARQL a SQL utilizando mapeos R2RML. Además se proponen varias técnicas de optimización para generar consultas SQL que son más eficientes cuando son evaluadas en sistemas gestores de bases de datos relacionales. Este enfoque se evalúa mediante un benchmark sintético y varios casos reales.

Otra recomendación relacionada con R2RML es la conocida como Direct Mapping (DM), que establece reglas fijas para la transformación de datos relacionales a RDF. A pesar de que ambas recomendaciones se publicaron al mismo tiempo, en septiembre de 2012, todavía no se ha realizado un estudio formal sobre la relación entre ellas.

Por tanto, la segunda parte de esta tesis se centra en el estudio de la relación entre R2RML y DM. Se divide este estudio en dos partes: de R2RML a DM, y de DM a R2RML. En el primer caso, se estudia un fragmento de R2RML que tiene la misma expresividad que DM. En el segundo caso, se representan las reglas de DM como mapeos R2RML, y también se añade la semántica implícita (relaciones de subclase, 1-N y M-N) que se puede encontrar codificada en la base de datos.

Esta tesis muestra que es posible usar R2RML en casos reales, sin necesidad de realizar materializaciones de los datos, puesto que las consultas SQL generadas son suficientemente eficientes cuando son evaluadas en el sistema gestor de base de datos relacional. Asimismo, esta tesis profundiza en el entendimiento de la relación existente entre las dos recomendaciones del W3C, algo que no había sido estudiado con anterioridad.

Acknowledgements

”Five minutes”. That was her reply when I asked Asun whether she had some time to talk regarding my intention to join the Ontology Engineering Group (OEG). It turns out that in addition to the initial five minutes, she has given many more years of the support and resources I needed to do my research. She originally suggested I do my master thesis with Oscar, who at that time, had some research topics related to data integration.

I believe that every academic supervisor has to play multiple roles, such as a teacher, an advisor, a cheerleader and a match-maker. During my initial years, Oscar was my teacher, pointing out to me the necessary literature and open research topics. Then, once I found my topic, he became my advisor, guiding me along the right path, although intentionally, he also let me walk through some wrong paths so that I might learn what not to do. There has been many ups and downs during the period of doing my research, and that’s exactly when Oscar played his cheerleader role, encouraging me and telling me what he believed about the quality of my work. So I could say that this thesis is actually not my work alone, but also Oscar’s work, and also other researchers’, because Oscar also played a match-maker role, connecting me with other researchers, who had similar interests, so that we could work together.

Juan, Boris, Carlos, Luciano, Marcelo, Nandana, Miguel EG, Raul GC, Raul AC, Sergio, Gueton, Edna, Maria Esther...thank you for all the collaboration we did together, I learned so many things by working with you.

Thank you to all of the current and ex members of this hot (figuratively and literally speaking) group, OEG. I really enjoyed the chess games with Victor... coffee breaks with Carlos, Julia, Olga, Alex, Maria, Nandana, Miguel, Pablo, Andres... the interesting data integration discussions with Mora, Jean-Paul, Boris... the services of Raul, Ana, JARG... and the sharing of the phd study burden with Garijo, Idafen, Vila, Andres, Mariano, Filip.

To Raul, Jorge, Victor, Alex... for your time helping me with my presentations, I am so lucky to very helpful coworkers like you.

To my previous academic supervisors, Yohanes Stefanus, Luis M Pereira, João Leite, Susana M Hernández... thank you for providing me with all the necessary building blocks for my research career.

My friends in Spain: Lupe, Dian, Adit, Junar, Icha, David, Jenni...for the lunches and dinners, knowing that I need the necessary nutrients and diversions. Part of this thesis

was written when I was living with Luis and family. I was looking for roommates and a house to live in, but instead, I found my second family and home. Luis, Haydee, Lian, Kaylee...thank you for providing me a very supporting environment, either for relaxing or working.

To my dear friend, Christin, that by being herself, setting the example for me that all things are possible, including of course, writing master and phd theses.

Thanks to my uncle, Leonardus, who introduced me to computer programming and give me his computer and all of his computer magazines collection when I was a teenager.

To my mother and my sister for always believing in me. You are the best in the world that I have

Above all, to my Lord Jesus for all his blessings in my life. He surrounds me with so many amazing people. All glories belong to Him. *Ad Maiorem Dei Gloriam.*

And for everyone else, this SPARQL query is for you:

```
CONSTRUCT { ex:fpriyatna ex:acknowledges ?x }  
WHERE {  
  { ?x ex:contributesDirectlyTo ex:thisThesis }  
  UNION  
  { ?x ex:contributesIndirectlyTo ex:thisThesis }  
}
```

Contents

Declaration of Authorship	ii
Abstract	v
Acknowledgements	viii
Contents	x
List of Figures	xiv
List of Tables	xv
1 Introduction	1
1.1 Thesis Structure	3
1.2 Dissemination Results	4
2 State of the Art	6
2.1 Ontology Based Data Access (OBDA)	7
2.2 Source Layer in OBDA: Relational Databases (RDB)	7
2.2.1 Relational Algebra and SQL	9
2.3 Target Layer in OBDA: Resource Description Framework (RDF)	10
2.3.1 SPARQL	12
2.3.1.1 Semantics of SPARQL	12
2.3.2 RDB-based Triple Stores	13
2.4 Mapping Layer in OBDA	16
2.4.1 OBDA Mapping Approaches	16
2.4.1.1 Local As View (LAV)	17
2.4.1.2 Global As View (GAV)	17
2.4.1.3 Global Local As View (GLAV)	18
2.4.2 Proprietary Mapping Languages	18
2.4.2.1 R_2O	18
2.4.2.2 D2RQ	20
2.4.3 W3C Recommendations on Mapping Languages	21
2.4.3.1 Direct Mapping	22
2.4.3.2 R2RML	23

2.4.4	Mapping Generation Approaches	26
2.4.5	Relationship Between Direct Mapping and R2RML	28
2.5	Query Answering in OBDA	29
2.6	Summary	31
3	Objectives and Contributions	32
3.1	Problem Statement and Objectives	32
3.2	Assumptions and Limitations	34
3.3	Contributions	35
4	Formalization and Optimizations of R2RML-based SPARQL to SQL Query Translation	36
4.1	An R2RML-based extension of Chebotko's approach	37
4.1.1	Illustrative Example	37
4.1.2	Function <i>trans</i> for translating SPARQL graph patterns	39
4.1.3	Function <i>trans</i> for translating triple patterns	39
4.1.4	Mapping <i>alpha</i> for triple patterns	42
4.1.5	Mapping <i>beta</i> for triple patterns	44
4.1.6	Function <i>genCondSQL</i> for triple patterns	45
4.1.6.1	Function <i>genCondSQL</i> _{sub} ^{tp}	45
4.1.6.2	Function <i>genCondSQL</i> _{pre} ^{tp}	46
4.1.6.3	Function <i>genCondSQL</i> _{obj} ^{tp}	48
4.1.6.4	Function <i>genCondSQL</i> _{tp} ^{tp}	50
4.1.7	Function <i>genPRSQL</i> for triple patterns	51
4.2	Optimisations	53
4.2.1	Self-join Elimination	53
4.2.1.1	Mapping <i>alpha</i> for STG patterns	54
4.2.1.2	Function <i>genCondSQL</i> ^{stg} for STG patterns	55
4.2.1.3	Function <i>genPRSQL</i> ^{stg} for STG patterns	57
4.2.1.4	Function <i>trans</i> for translating STG patterns	58
4.2.1.5	Comparison of resulting translation of an STG query for RDB2RDF systems ontop and morph-RDB	60
4.2.2	Left Outer Join Elimination for OPTIONAL patterns	61
4.2.2.1	Comparison of resulting translation of an OSTG query for RDB2RDF systems ontop and morph-RDB	63
4.2.3	Phantom Triple Pattern Introduction	64
4.2.3.1	Comparison of resulting translation of Phantom TP Introduction query for RDB2RDF systems ontop and morph-RDB	64
4.2.4	IS NOT NULL checking	66
4.2.4.1	Comparison of resulting translation of a TP pattern for RDB2RDF systems ontop and morph-RDB	66
4.2.5	Other Optimisations	67
4.2.5.1	Table reordering	67
4.2.5.2	Union Reduction	67
4.2.5.3	Incompatible Term Types	68
4.2.5.4	Incompatible Datatype	69

4.3	Conclusion	70
5	On the Expressive Power of Direct Mapping and its Relationship with R2RML	71
5.1	<i>R2RML_{lite}</i>	72
5.1.1	<i>R2RML_{lite}</i> Mappings	72
5.1.2	<i>R2RML_{lite}</i> Normalization	72
5.1.2.1	1st Normal Form (1NF)	73
5.1.2.2	2nd Normal Form (2NF)	74
5.1.2.3	3rd Normal Form (3NF)	76
5.1.2.4	4th Normal Form (4NF)	77
5.1.3	<i>R2RML_{lite}</i> 4NF in Datalog	78
5.2	Fundamental Properties	81
5.3	<i>R2RML_{lite}</i> to Direct Mapping	82
5.3.1	Approach for solving IPP and QRPP	82
5.3.1.1	Solving IPP	82
5.3.1.2	Solving QRPP	85
5.3.1.3	Note: Solving IPP with query	86
5.4	Conclusion	87
6	MIRROR: An Automatic R2RML Mappings Generator	88
6.1	Automatic Generation of R2RML Mappings	91
6.1.1	A Catalogue of Typical Patterns in Relational Schemas	92
6.1.1.1	Catalogue creation process	92
6.1.1.2	Catalogue description	93
6.1.2	Algorithms for the Generation of R2RML Mappings	95
6.1.2.1	Subclass Identification	98
6.1.2.2	Object Property Identification	98
6.1.2.3	Datatype Property Identification	98
6.2	Conclusion	99
7	Experimentation	100
7.1	Experimentation with morph-RDB	101
7.1.1	Evaluation with BSBM	101
7.1.1.1	Discussion	103
7.1.2	Evaluation with RÉPENER Project	104
7.1.3	Evaluation with BizkaiSense Project	104
7.1.4	Evaluation with Integrate Project	106
7.2	Experimentation with MIRROR	110
7.2.1	Experimentation using the Direct Mapping test cases	111
7.2.2	Experimentation using D011B	111
7.3	Summary	112
8	Extensions of morph-RDB	116
8.1	morph-GFT	116
8.1.1	Architecture of morph-GFT	117
8.1.2	Example	118
8.1.3	Conclusion	119

8.2	morph-LDP	120
8.2.1	Mapping LDP components with R2RML	121
8.2.2	Implementation	122
8.2.3	Conclusion	123
9	Conclusions and Future Work	124
9.1	Summary on the first contribution and future work	124
9.2	Summary on the second contribution and future work	126
A	Mappings and Queries in in Section 4.2.1	128
A.1	R2RML Mappings in Section 4.2.1	129
A.2	Ontop Mappings in Section 4.2.1	130
A.3	Ontop SQL and Morph SQL in Section 4.2.1	131
A.3.1	Ontop STG SQL	131
A.3.2	Morph-RDB STG SQL	131
A.4	Ontop SQL and Morph SQL in Section 4.2.2	132
A.4.1	Ontop OSTG SQL	132
A.4.2	Morph-RDB OSTG SQL	132
A.5	Ontop SQL and Morph SQL in Section 4.2.3	133
A.5.1	Ontop PT SQL	133
A.5.2	Morph-RDB PT SQL	133
A.6	Ontop SQL and Morph SQL in Section 4.2.4	134
A.6.1	Ontop TP SQL	134
A.6.2	Morph-RDB TPSQL	134
B	Proofs for Self Join Elimination and Left Outer Join Elimination	135
B.1	Proof for Self Join Elimination	135
B.2	Proof for Left Outer Join Elimination	137
	Bibliography	140

List of Figures

1.1	Ontology Based Data Access [Len11]	2
2.1	Simple Database of Chess Players	9
2.2	A RDF Graph of Russian Grandmasters	11
2.3	<i>trans</i> function as defined by Chebotko et al. in [CLF09]	15
2.4	Overview of R2RML Triples Map	24
2.5	Overview of R2RML Term Map	25
4.1	Tables Patient and Stage	61
5.1	Overview of the structure of an R2RML Term Map in <i>R2RML_{lite}</i>	73
5.2	Relational Schema R and its Instance <i>I</i>	80
5.3	Approach for Solving the Information Preservation Problem	85
5.4	Approach for Solving the Query Result Preservation Problem	86
6.1	Graphical Representation of D011B Database.	89
6.2	A general overview of the R2RML mapping generation process in MIRROR.	91
6.3	Obtaining the Catalogue for Guiding the R2RML Mapping Generation.	93
7.1	BSBM query evaluations (normalized time to native query). N=Native SQL, C=naïve translation queries, SQE=Subquery Elimination, SJE=Self-Join elimination, SQE+SJE=Subquery+Self-Join Elimination	102
7.2	SEiS - query evaluation time (in seconds)	105
7.3	BizkaiSense - query evaluation time (in seconds)	106
7.4	Running time for selected Integrate queries on morph-RDB and D2R (in seconds)	109
7.5	Integrate Queries Evaluation Time in Cold Mode	109
7.6	Integrate Queries Evaluation Time in Warm Mode	110
8.1	Architecture of morph-GFT	117
8.2	The morph-LDP use case	120
8.3	An example of morph-LDP in action, using the SPARQL query mode.	122

List of Tables

2.1	Example of Table Triple for storing RDF triples.	14
2.2	View ValidOffer	23
2.3	Example of table Offer	24
2.4	Comparison table of query translation techniques and the supporting systems, extended from [RMR15].	31
4.1	Example of computing mapping α for our running example.	44
4.2	Examples of computing mapping β for our running example	45
4.3	Examples of function φ in our running example.	51
4.4	Examples of function ψ in our running example.	52
4.5	An example of how to calculate mapping α^{stg}	55
4.6	An example of how to calculate function φ^{stg}	56
4.7	An example of computing function $genPRSQL^{stg}$	57
5.1	$\llbracket V \rrbracket_I$, the result of applying view V in Listing 5.6 over I in Figure 5.2 . . .	84
6.1	Correspondences between conceptual, logical and physical models (rows 1 to 5)	94
6.2	Correspondences between conceptual, logical and physical models (rows 6 to 9)	95
7.1	Grouping of all queries according to the use case queries	106

For/Dedicated to/To my family

Chapter 1

Introduction

“The beginning is the most important part of the work.”

Plato

In the last few years we have witnessed the explosion of data, in what has been more commonly known as Big Data. Many aspects characterize Big Data, such as volume, velocity, variety, and veracity. *Volume* speaks of the sheer amount of data, which can be measured by the size of it. *Velocity* corresponds to the speed of incoming or outgoing data, from the static data stored in relational databases, to the streaming data produced by sensor devices. *Variety* tells us about the diversity of how data may be stored in various types of data sources, such as relational databases (RDBs), spreadsheets or web-tables, among many others. *Veracity* refers to the degree of how we can trust the data being useful for our purpose.

The work presented in this thesis mostly focuses on the diversity aspect, driven by the need to access heterogeneous data sources, both from a technological point of view (different APIs or query languages for different data sources) and from a conceptual point of view (different terminology may be used in each data source to represent data).

A technique that is used to deal with the diversity aspect on the (Semantic) Web is to use Resource Description Framework (RDF) as a common data model for sharing and integrating information. An RDF triple consists of a subject and an object, connected by a predicate. The W3C Recommendation RDF 1.1 [CWL14] introduces two distinct concepts: *RDF graph* and *RDF dataset*. An RDF graph is a collection of RDF triples while an RDF dataset is a collection of RDF graphs. For conciseness, we refer to both of them as *RDF data* when the difference is not important. RDF data is usually stored in RDF triple stores, which are built either from scratch (*native triple stores*), or on top

of existing relational databases (*RDBMS-backed triple stores*). SPARQL [PS⁺08] is the W3C standard query language for querying RDF datasets.

The RDF data itself can be generated either from scratch or by transforming the content of existing data repositories by means of mappings, an approach commonly known as Ontology Based Data Access (OBDA) [Len11], as shown in Figure 1.1.

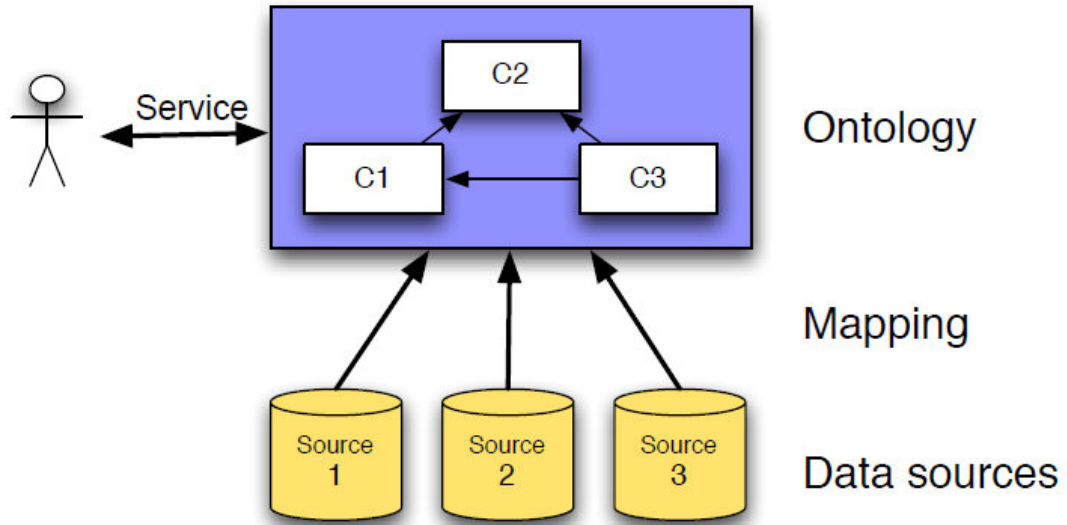


FIGURE 1.1: Ontology Based Data Access [Len11]

With the emergence of the Semantic Web and the Web of Data, Relational Database to Resource Description Framework (RDB2RDF) has been proposed as one particular class of an OBDA solution. In RDB2RDF solutions, data is stored in relational databases and it is viewed as RDF data, hence the name RDB2RDF. The RDF data that result from the application of RDB2RDF solutions can be generated either in a materialized or a virtualized mode. In the materialized mode, also known as ETL (Extract-Transform-Load) mode, SPARQL queries posed over the generated RDF dataset are evaluated directly over the materialized RDF data. In the virtualized mode no RDF data is stored and SPARQL queries have to be translated into SQL queries, taking into account the specified mappings.

Early RDB2RDF systems, such as ODEMapster [BGP06], D2R [BC06], Triplify [ADL⁺09a] and many others were accompanied by their own mapping languages. This situation led the W3C to start an initiative to standardize the mapping language. As a result, two recommendations were published in September 2012, Direct Mapping [MAS12] and R2RML [DSC12]. The Direct Mapping recommendation specifies the fixed set of transformation rules that generate RDF data from the database schema and its instances, without the

need of intervention from users. On the contrary, with R2RML, users are able to specify more complex transformation rules to generate the desired RDF data.

The W3C R2RML Recommendation does not specify the translation algorithm to be used to exploit mappings. Without having a formalized query translation technique, RDB2RDF engines may produce incorrect results or suffer from poor performance in terms of evaluation time due to inefficient SQL queries that result from the translation process.

The relationship between the two W3C Recommendations has not been studied formally either. This relationship can be studied bidirectionally: from Direct Mapping to R2RML, and from R2RML to Direct Mapping. From Direct Mapping to R2RML, we can consider that the input for Direct Mapping is a relational schema and its instance, which is transformed into RDF data using the fixed set of rules defined in the recommendation, without allowing users to modify those rules. Furthermore, these transformation rules do not take into account the implicit relationships encoded in the database schema. In the direction from R2RML to Direct Mapping, while intuitively one can consider that the R2RML Recommendation has more expressive power than the Direct Mapping, there are still some questions to be answered in this respect. For example, how do we define that one recommendation has the same or more expressive power than the other. Additionally, what is the fragment of the R2RML Recommendation that has the same expressive power as the Direct Mapping Recommendation.

In this thesis, we report our formalization of an algorithm for translating SPARQL queries into SQL queries, taking into account user-defined R2RML mappings. We extend an existing algorithm that was originally designed for RDB-backed triple stores. As the resulting SQL queries are not necessarily efficient to be evaluated in real settings, we also propose some optimizations that can be implemented to make the evaluation of such queries over relational databases more efficient. Furthermore, we also report our study regarding the relationship between R2RML and Direct Mapping. From the Direct Mapping to R2RML, we propose a technique for representing Direct Mapping rules together with the relationship in a database schema as R2RML Mappings. From the opposite direction, we identify a fragment of the R2RML Recommendation, which we name *R2RML_{Lite}*, together with its normalization process, and describe the necessary properties needed to measure the expressive power of these two W3C recommendations.

1.1 Thesis Structure

This thesis is structured as follows:

- In Chapter 2 we provide some background and state of the art of the Data Integration, and on its specific class, namely RDB2RDF system.
- In Chapter 3 we describe the research problems and objectives of this thesis.
- In Chapter 4 we present our proposal on the formalization of SPARQL to SQL query translation, together with some optimization techniques that can be implemented to generate more efficient queries.
- In Chapter 5 and 6 we analyze the relationship between the Direct Mapping and the R2RML Recommendations. In Chapter 5, we define two fundamental properties to specify the expressive power of Direct Mapping and R2RML. In Chapter 6, we propose a technique for automatically generating two sets of R2RML mappings from a relational database schema. The first set of mappings simulates the behavior of Direct Mapping, while the second set represents implicit knowledge encoded in the relationship between tables.
- In Chapter 7 we report the results of our experimentation using morph-RDB, our implementation of the technique that we described in Chapter 4, using a synthetic benchmark together with queries from real projects.
- In Chapter 8 we describe two extension of morph-RDB (morph-GFT and morph-LDP) and MIRROR, our implementation of the techniques that we presented in Chapter 7.
- In Chapter 9 we present a summary of our main findings and a conclusion, together with future direction of our work.

1.2 Dissemination Results

The main contributions of this thesis have been published in or have been submitted for publication to the following international conferences/workshops:

- morph-RDB, described in Chapters 4 and 7, has been published in Priyatna, F, Oscar C, and Sequeda, J. **Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph**. In Proceedings of the 23rd International Conference on World Wide Web 2014, pp. 479-490.
- Chapter 6 has been published in de Medeiros, L. F., Priyatna, F., and Corcho, O. **MIRROR: Automatic R2RML Mappings Generation From Relational Databases**. In Proceedings of the 15th International Conference On Web Engineering 2015, pp. 326-343.

- Part of Chapters 4 and 7 has been published in Priyatna, F, Alonso Calvo, R., Paraiso-Medina, S., Padron-Sanchez, G., Corcho, O. **R2RML-based access and querying to relational clinical data with morph-RDB** In Proceedings of Semantic Web Applications and Tools For Life Sciences, 2015.
- morph-GFT, part of Chapter 8, has been published in Priyatna, F, Buil-Aranda, C, and Corcho, O. **Applying SPARQL-DQP for Federated SPARQL Querying over Google Fusion Tables**. In The Semantic Web: ESWC 2013 Satellite Events, pp. 189-193. Springer Berlin Heidelberg, 2013.
- morph-LDP, part of Chapter 8, has been published in Mihindukulasooriya, N, Priyatna, P, Corcho, O, Garcia-Castro, R, and Esteban-Gutiérrez, M. **morph-LDP: An R2RML-based Linked Data Platform implementation**. In The Semantic Web: ESWC 2014 Satellite Events (pp. 418-423). Springer International Publishing.

Chapter 2

State of the Art

“If you steal from one author it’s plagiarism; if you steal from many it’s research.”

Wilson Mizner

We start this chapter by reviewing in Section 2.1 the concept of Ontology Based Data Access (OBDA). Next, each of its components (source schema, target schema, and mappings) will be discussed. Section 2.2 introduces relational database systems (RDBMS), which have proven their capability and robustness to store and organize data, and relational algebra, which serves as the foundation for the Standard Query language (SQL) commonly implemented in any RDBMS.

In Section 2.3, we describe the Resource Description Framework (RDF) data model, which is widely used for publishing and exchanging data on the web. Afterwards, we will be looking at SPARQL, the W3C standard query language for this data model. We close this chapter by reviewing the state of the art in query translation techniques. One of them, developed by Chebotko et al., will be the basis for our R2RML-based query translation, explained in Chapter 4.

Section 2.4 discusses the mapping approaches that have been proposed in the state of the art for OBDA, as well as several implementations in practice.

We close this chapter by reviewing the concept of query answering in Section 2.5, which consists of the following tasks: reformulation (query rewriting), unfolding (query translation) and query evaluation.

2.1 Ontology Based Data Access (OBDA)

As discussed in the introduction, the concept of variety is strongly related to the heterogeneity of data sources. Such heterogeneity may be related to:

- The type of the data source. Data be stored in multiple types of repositories, such as relational databases, spreadsheet documents, web tables, etc.
- The terminology used by users to organize data in such data sources. For example, one entity may be called **Person** in one data source, and **People** in another one.

One proposal to deal with such heterogeneity is to provide a unified view over data sources, and mappings that define the relationship between the unified view and the data source. Ontologies, which are defined as formal specifications of shared conceptualizations [Gru93], are often considered as a suitable choice to play the role of unified views. This concept is known as Ontology Based Data Access (OBDA) [PLC⁺08], and it is considered as a class of data integration system [Len02]. Formally, a data integration system or OBDA is defined as a triple of $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where:

- \mathcal{G} is the global schema.
- \mathcal{S} is the source schema.
- \mathcal{M} is the mapping between \mathcal{S} and \mathcal{G} .

In the context of OBDA, ontologies are used in the global schema layer. Since most of the data is stored in relational databases, they are naturally used as the source schema. Finally, the mapping layer in these systems may use various languages. In this document, we will focus on two recommendations by the W3C, namely Direct Mapping and R2RML.

In the following sections, we will discuss in more detail each of these components corresponding to RDB2RDF systems.

2.2 Source Layer in OBDA: Relational Databases (RDB)

In this document, we focus on relational databases [Cod70], a type of database systems that uses relations (or tables) for organizing its data. A system that manages relational databases is commonly called Relational Database Management System (RDBMS). This is the most popular type of database system adopted and has proven its robustness.

RDBMS are available either as non-commercial software such as MySQL, PostgreSQL, MonetDB, or as commercial systems, such as IBM DB2, Microsoft SQL Server, or Oracle Database.

Conceptually, a relational database schema consists of :

- A set of relation names \mathbf{R} . A relation is commonly called a *table*, which is a set of tuples that have the same attributes.
- A set of attribute names \mathbf{A} . An attribute is commonly called a *column*.
- $att(R)$ that represents the set of attribute names associated to relation name $R \in \mathbf{R}$.
- A set \mathbf{PK} whose elements are in the form of (R, A) where $R \in \mathbf{R}$ and $A \subseteq \mathbf{A}$, representing that the set of attributes A is the *primary key* associated to the relation R . The primary key of a relation is the unique identifier of tuples in that relation.
- A set \mathbf{FK} whose elements are in the form of (R_C, A_C, R_P, A_P) . Each element of \mathbf{FK} represents a *foreign key*, that the value of A_C of relation R_C is restricted to the values of A_P of relation R_P . In other words, that is to say that the column A_C is a foreign key in table R_C that refers to the column A_P in table R_P .

A tuple $t = (tid, v, R, A)$ where A is an attribute and R is a relation, specifies that v is a value for attribute A of relation R with tid as the identifier of the tuple. A tuple is commonly known as a *row*.

Example 2.1. *The following is a database schema having two tables ($PLAYER, COUNTRY$), whose attributes and primary/foreign keys can be seen below.*

- $R = \{PLAYER, COUNTRY\}$.
- $A = \{PID, PNAME, PRATING, PCOUNTRY, CID, CNAME\}$.
- $att(PLAYER) = \{PID, PNAME, PRATING, PCOUNTRY\}$.
- $att(COUNTRY) = \{CID, CNAME\}$.
- $PK = \{(PLAYER, \{PID\}), (COUNTRY, \{CID\})\}$.
- $FK = \{(PLAYER, \{PCOUNTRY\}, COUNTRY, \{CID\})\}$

One example instance of such database can be seen in Figure 2.1.

PLAYER				
<u>PID</u>	PNAME	PSEX	PRATING	PCOUNTRY
MCA	Magnus Carlsen	M	2865	NO
GKA	Garry Kasparov	M	2851	RU
JPO	Judit Polgar	F	2675	HU
AKA	Anatoly Karpov	M	2628	RU
AKO	Alexandra Konsteniuk	F	2529	RU

COUNTRY	
<u>CID</u>	CNAME
HU	Hungary
NO	Norway
RU	Russia

FIGURE 2.1: Simple Database of Chess Players

2.2.1 Relational Algebra and SQL

Relational algebra is used as the foundation for the query language for relational databases, which in practice is represented by SQL statements. Relational algebra contains many operators that can be applied under relational tuples. A relational tuple is a set of attributes together with their values. A database column is represented as tuple attributes while the instances are represented as tuple values. Some of the most commonly used relational algebra operators are:

- **Projection** (π). A projection $\pi_{a_1, \dots, a_n}(R)$ is a selection of attributes a_1, \dots, a_n over a set of relational tuples R . This corresponds to the SQL **SELECT** operator.
- **Selection** (σ). A selection σ over a set of relational tuples R returns a subset of R where the condition σ is true. This operator corresponds to the SQL **WHERE** operator.
- **Union** (\cup). This operator corresponds to the set union operator. That is to say, that the result of $A \cup B$ is the relation C which contains (without duplications) all elements of A plus all elements B , given that A and B share the same number of attributes. The corresponding SQL operator is **UNION**.

- **Difference** (\setminus). This operator corresponds to the set difference operator, that is to say that the result of $A \setminus B$ is a set C which contains all element in A but not in B .
- **Intersection** (\cap). This operator corresponds to the set intersection operator, that is to say that the result of $A \cap B$ is a set C which contains all element that exists both in A and in B .
- **Natural Join** (\bowtie). A natural join $A \bowtie B$ returns a new set whose elements are sets of tuples in A and B that share common attribute values.
- **Cross Join** (\times). A cross join $A \times B$ returns a new set of tuples which contains attributes $\langle a_1, \dots, a_n, b_1, \dots, b_m \rangle$ where $\langle a_1, \dots, a_n \rangle$ are all attributes of A and $\langle b_1, \dots, b_m \rangle$ are all attributes of B . This corresponds to the SQL JOIN operator.

2.3 Target Layer in OBDA: Resource Description Framework (RDF)

Resource Description Framework (RDF) [CWL14] is a W3C recommendation that is used as a language to represent resources and their attributes. A resource in RDF, which is identified by its URI, may represent physical things such as people or represent virtual things such as a webpage. A resource has some attributes/properties and relationships with other resources. The resource identification is done via RDF statements, which are also called triples because they consist of three elements. Those elements are:

- **Subject**. The subject of an RDF statement, either an IRI (**I**) or a blank node (**B**), is the resource that is being identified by the statement. The subject is either a new object, that means we are specifying a new object, or existing object which means we are extending the description of that resource.
- **Predicate**. The predicate, an IRI (**I**), represents the relationship or property between the subject and the object in the statement.
- **Object**. The object of an RDF statement is either an IRI (**I**), a literal (**L**), or a blank Node (**B**).

A set of RDF triples is called RDF graph (because subjects/objects can be represented as nodes and predicates as edges). An RDF dataset is a collection of RDF graphs. RDF graphs can be serialized using various formats, such as RDF/XML, N-Triples, Turtle, and N3.

Example 2.2. Listing 2.1 is an *N-Triples* representation of an *RDF* graph that consists of several triples, which encode some information regarding chess players. Figure 2.2 is the graphical representation of such dataset.

```

1 <http://ex.com/PLAYER/GKA> rdf:type <http://ex.com/Grandmaster> .
2 <http://ex.com/PLAYER/GKA> foaf:name "Garry Kasparov" .
3 <http://ex.com/PLAYER/GKA> dbpedia:birthplace <http://ex.com/COUNTRY/RU> .
4 <http://ex.com/PLAYER/GKA> ex:sex "M" .
5 <http://ex.com/PLAYER/AKA> rdf:type <http://ex.com/Grandmaster> .
6 <http://ex.com/PLAYER/AKA> foaf:name "Anatoly Kasparov" .
7 <http://ex.com/PLAYER/AKA> dbpedia:birthplace <http://ex.com/COUNTRY/RU> .
8 <http://ex.com/PLAYER/GKA> ex:sex "M" .
9 <http://ex.com/PLAYER/AKO> rdf:type <http://ex.com/Grandmaster> .
10 <http://ex.com/PLAYER/AKO> foaf:name "Alexandra Konsteniuk" .
11 <http://ex.com/PLAYER/AKO> dbpedia:birthplace <http://ex.com/COUNTRY/RU> .
12 <http://ex.com/PLAYER/GKA> ex:sex "F" .

```

LISTING 2.1: A RDF Dataset of Russian Grandmasters

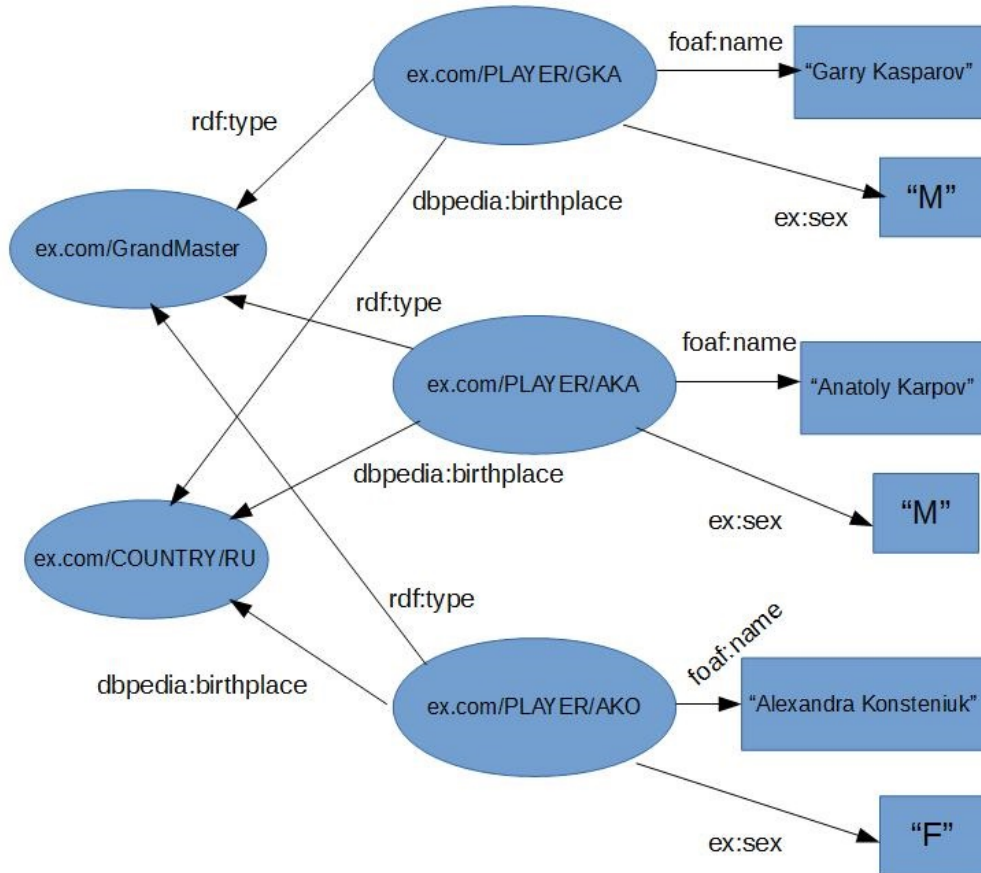


FIGURE 2.2: A RDF Graph of Russian Grandmasters

2.3.1 SPARQL

We have seen that RDF triples can be seen as a graph, which implies that to query RDF dataset, we need a graph query language. SPARQL is the W3C standard query language for querying RDF datasets.

To define the syntax of SPARQL, we use the work in [PAG09], which defines SPARQL graph patterns recursively as follows:

- A tuple from $(\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ is a graph pattern (a triple pattern).
- If P_1 and P_2 are graph patterns, then expressions $(P_1 \text{ AND } P_2)$, $(P_1 \text{ OPT } P_2)$ and $(P_1 \text{ UNION } P_2)$ are graph patterns.
- If P is a graph pattern and R is a SPARQL built-in condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern.
- If P is a graph pattern and W is a finite set of variables, then $(\text{SELECT } W \ P)$ is a graph pattern.

A SPARQL built-in condition is constructed using elements of the set $(\mathbf{I} \cup \mathbf{V})$ and constants, logical connectives (\neg , \wedge , \vee), inequality symbols ($<$, \leq , \geq , $>$), the equality symbol ($=$), unary predicates such as `bound`, `isBlank`, and `isIRI` (see [PS⁺08, GSP13] for a complete list). In this paper, we restrict to the fragment where the built-in condition is a Boolean combination of terms constructed by using `=` and `bound`, that is: (1) if $?X, ?Y \in \mathbf{V}$ and $c \in \mathbf{I}$, then `bound(?X)`, `?X = c` and `?X = ?Y` are built-in conditions, and (2) if R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$ and $(R_1 \wedge R_2)$ are built-in conditions.

2.3.1.1 Semantics of SPARQL

To define the semantics of SPARQL, we need to introduce some extra terminology. A *mapping* μ is a partial function $\mu : \mathbf{V} \rightarrow \mathbf{U}$. Abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables occurring in t according to μ . The *domain* of μ , denoted by $\text{dom}(\mu)$, is the subset of \mathbf{V} where μ is defined. Two mappings μ_1 and μ_2 are *compatible*, denoted by $\mu_1 \sim \mu_2$, when for all $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, it is the case that $\mu_1(?X) = \mu_2(?X)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping. Moreover, given a mapping μ and a set of variables W , the *restriction of μ to W* , denoted by $\mu|_W$, is a mapping such that $\text{dom}(\mu|_W) = (\text{dom}(\mu) \cap W)$ and $\mu|_W(?X) = \mu(?X)$ for every $?X \in (\text{dom}(\mu) \cap W)$.

To define the semantics of graph patterns, we first need to introduce the notion of satisfaction of a built-in condition by a mapping, and then we need to introduce some operators for mappings. More precisely, given a mapping μ and a built-in condition R , we say that μ *satisfies* R , denoted by $\mu \models R$, if (omitting the usual rules for Boolean connectives): (1) R is $\text{bound}(?X)$ and $?X \in \text{dom}(\mu)$; (2) R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$; and (3) R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$. Moreover, given sets Ω_1 and Ω_2 of mappings, the *join* of, the *union* of, the *difference* between and the *left outer-join* between Ω_1 and Ω_2 are defined as follows:

$$\begin{aligned}\Omega_1 \bowtie \Omega_2 &= \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\}, \\ \Omega_1 \cup \Omega_2 &= \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}, \\ \Omega_1 \setminus \Omega_2 &= \{\mu \in \Omega_1 \mid \forall \mu' \in \Omega_2 : \mu \not\sim \mu'\}, \\ \Omega_1 \Join \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2).\end{aligned}$$

Then given an RDF graph G and a graph pattern P , the evaluation of P over G , denoted by $\llbracket P \rrbracket_G$, is recursively defined as follows: (1) if P is a triple pattern t , then $\llbracket P \rrbracket_G = \{\mu \mid \text{dom}(\mu) \text{ is equal to the set of variables occurring in } t \text{ and } \mu(t) \in G\}$; (2) if P is $(P_1 \text{ AND } P_2)$, then $\llbracket P \rrbracket_G = \llbracket P_1 \rrbracket_G \bowtie \llbracket P_2 \rrbracket_G$; (3) if P is $(P_1 \text{ OPT } P_2)$, then $\llbracket P \rrbracket_G = \llbracket P_1 \rrbracket_G \Join \llbracket P_2 \rrbracket_G$; (4) if P is $(P_1 \text{ UNION } P_2)$, then $\llbracket P \rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$; (5) if P is $(P_1 \text{ FILTER } R)$, then $\llbracket P \rrbracket_G = \{\mu \mid \mu \in \llbracket P_1 \rrbracket_G \text{ and } \mu \models R\}$; and (6) if P is $(\text{SELECT } W \text{ } P_1)$, then $\llbracket P \rrbracket_G = \{\mu|_W \mid \mu \in \llbracket P_1 \rrbracket_G\}$.

2.3.2 RDB-based Triple Stores

RDF data (graphs/datasets) can be stored in triple stores. There are at least two categories of triple stores: those built from scratch, and those built on top of relational database systems. Some examples of the first category are: in-memory Jena, Sesame and AllegroGraph. As for the latter category, the motivation is to benefit from the results of the work conducted in the last decades in the field of relational databases. Jena SDB, 3store, Virtuoso, and Oracle fall into this category. There are multiple ways to implement triple stores using relational database systems, using a table of three columns being the simplest approach. Several techniques have been proposed to improve the performance of triple stores, such as to use multiple tables and partition the triples based on the types or the predicates.

For example, the RDF graph in Figure 2.2 can be stored in a table `TRIPLES` having 3 columns (s, p, o) , which can be seen in Table 2.1.

TABLE 2.1: Example of Table **Triple** for storing RDF triples.

Triples		
s	p	o
http://ex.com/PLAYER/GKA	rdf:type	http://ex.com/Grandmaster
http://ex.com/PLAYER/GKA	foaf:name	Garry Kasparov
http://ex.com/PLAYER/GKA	dbpedia:birthplace	http://ex.com/COUNTRY/RU
http://ex.com/PLAYER/GKA	ex:sex	M
http://ex.com/PLAYER/AKA	rdf:type	http://ex.com/Grandmaster
http://ex.com/PLAYER/AKA	foaf:name	Anatoly Karpov
http://ex.com/PLAYER/AKA	dbpedia:birthplace	http://ex.com/COUNTRY/RU
http://ex.com/PLAYER/AKA	ex:sex	M
http://ex.com/PLAYER/AKO	rdf:type	http://ex.com/Grandmaster
http://ex.com/PLAYER/AKO	foaf:name	Alexandra Konsteniuk
http://ex.com/PLAYER/AKO	dbpedia:birthplace	http://ex.com/COUNTRY/RU
http://ex.com/PLAYER/AKO	ex:sex	F

In this second type of systems, SPARQL queries have to be translated into semantically equivalent SQL queries. By semantically equivalent, we mean that the expected result of evaluating SQL queries over RDBMS-based triplestores, should return the same result as evaluating SPARQL queries over native triple stores.

One of the approaches proposed to formalise this translation process is that of Chebotko and colleagues [CLF09]. This approach proposes the use of a set of mappings (α and β) and functions (*genCondSQL*, *genPRSQL*, and *name*) to translate SPARQL into SQL.

- Mapping $\alpha : TP \rightarrow REL$ that given a set of possible triple patterns $TP = (IVL) \times (IV) \times (IVL)$ and a set of relations REL , maps triple pattern $tp \in TP$ to a relation $rel \in REL$ such that rel stores triples that match tp .
- Mapping $\beta : TP \times POS \rightarrow ATR$ that given a set of possible triple patterns $TP = (IVL) \times (IV) \times (IVL)$, a set $POS = \{sub, pre, obj\}$, and a set of relational attributes ATR , map a triple pattern $tp \in TP$ and a position $pos \in POS$ to an attribute $atr \in ATR$, such that the values in atr match tp at position pos .
- Function *name* that given a term in IVL generates a unique attribute name.
- Function *GenCondSQL* that given a set of possible triple patterns $TP = (IVL) \times (IV) \times (IVL)$ and mapping *beta*, takes a triple pattern tp and generates an SQL expression *cond* whose evaluation is *true* iff $\beta(tp, sub)$, $\beta(tp, pre)$, and $\beta(tp, obj)$ matches tp .
- Function *GenPRSQL* that given a set of possible triple patterns $TP = (IVL) \times (IV) \times (IVL)$, function *name* and mapping *beta*, takes a triple pattern $tp \in TP$

and generates an SQL expression that projects relational attributes correspond to subject, predicate, and object of the triple pattern tp .

Once the mappings and functions have been properly defined, the translation of SPARQL queries can be done by calling the function *trans* that takes query patterns, as can be seen in Figure 2.3.

```

trans(tp, α, β) =
    Select Distinct genPR-SQL(tp, β, name) From α(tp) Where genCond-SQL(tp, β);

trans(gp1 AND gp2, α, β) =
    Select Distinct name(a), [a|a ∈ (terms(gp1) - terms(gp2))] name(b), [b|b ∈ (terms(gp2) - terms(gp1))]
    Coalesce(r1.name(c), r2.name(c)) As name(c), [c|c ∈ (terms(gp1) ∩ terms(gp2))]
    From ( trans(gp1, α, β) ) r1 Inner Join ( trans(gp2, α, β) ) r2
    On (True And [c|c ∈ (terms(gp1) ∩ terms(gp2))])
    (r1.name(c)=r2.name(c) Or r1.name(c) Is Null Or r2.name(c) Is Null));
    where r1 = alias() and r2 = alias().

trans(gp1 OPT gp2, α, β) =
    Select Distinct name(a), [a|a ∈ (terms(gp1) - terms(gp2))] name(b), [b|b ∈ (terms(gp2) - terms(gp1))]
    Coalesce(r1.name(c), r2.name(c)) As name(c), [c|c ∈ (terms(gp1) ∩ terms(gp2))]
    From ( trans(gp1, α, β) ) r1 Left Outer Join ( trans(gp2, α, β) ) r2
    On (True And [c|c ∈ (terms(gp1) ∩ terms(gp2))])
    (r1.name(c)=r2.name(c) Or r1.name(c) Is Null Or r2.name(c) Is Null);
    where r1 = alias() and r2 = alias().

trans(gp1 UNION gp2, α, β) =
    Select name(a)[a|a ∈ A], name(b)[b|b ∈ B], r1.name(c)[c|c ∈ C] As name(c)
    From (trans(gp1, α, β)) r1 Left Outer Join (trans(gp2, α, β)) r2 On (False)
    Union
    Select name(a)[a|a ∈ A], name(b)[b|b ∈ B], r3.name(c)[c|c ∈ C] As name(c)
    From (trans(gp2, α, β)) r3 Left Outer Join (trans(gp1, α, β)) r4 On (False);
    where r1, r2, r3, and r4 = alias(); A, B, and C are ordered sets (terms(gp1) - terms(gp2)),
    (terms(gp2) - terms(gp1)), and (terms(gp1) ∩ terms(gp2)), respectively.

trans(gp FILTER expr, α, β) =
    Select * From ( trans(gp, α, β) ) alias() Where transexpr(expr);

trans(SELECT (v1, v2, ..., vn) WHERE(gp), α, β) =
    Select Distinct name(v1), name(v2), ..., name(vn) From ( trans(gp, α, β) ) alias();

```

FIGURE 2.3: *trans* function as defined by Chebotko et al. in [CLF09]

We now give an example of how those mappings and functions work.

Example 2.3. Consider again the triple table *Triples*(*s*, *p*, *o*) in Table 2.1, that uses its columns to store the subject, predicate, and object elements of RDF triples. A triple pattern $tp1 = ?x \text{ ex:sex } "M"$ corresponds to the following mappings/functions:

- Mapping α returns the table that holds the triples that correspond to this triple pattern $tp1$. That is, $\alpha(tp1) = \textit{Triples}$.

- Mapping β returns the column that corresponds to each triple pattern position (*sub*, *pre* or *obj*). That is, $\beta(tp1, sub) = s$, $\beta(tp1, pre) = p$, and $\beta(tp1, obj) = o$.
- Function *genCondSQL* filters the table *Triples* returned by $\alpha(tp1)$ so that only those records that match the triple pattern *tp* are returned. $genCondSQL(tp1, \beta) = \{p = rdfs:label \text{ AND } o = "M"\}$.
- Function *name* generates alias for each triple pattern element. One possible way to implement function *name*(*n*) is to return "*var_*" + *n* if *n* is a variable, "*iri_*" + *n* if *n* is an IRI, or *n* if *n* is a constant. That is, $name(x) = var_x$, $name(rdf:type) = iri_rdf_type$, and $name("M") = "M"$.
- Function *genPRSQL* projects the β column as the value returned by function *name* for each triple pattern position. That is, $genPRSQL(tp1, \beta, name) = \{s \text{ AS } var_x, p \text{ AS } iri_rdf_type, o \text{ AS } "M"\}$.
- Function *trans*(*tp1*, α , β) finally returns the SQL query using the values returned by the previous mappings and functions. That is, $trans(tp1, \alpha, \beta) = SELECT \text{ DISTINCT } s \text{ AS } var_x, p \text{ AS } iri_rdf_type, o \text{ AS } "M" \text{ FROM } TRIPLES \text{ WHERE } p = rdf:type \text{ AND } o = "M"$.¹

Notice that the algorithm shown in Figure 2.3 produces nested subqueries for every pattern, except for the most basic one, the triple pattern. Elliot et al. [ECTOO09] proposed a flexible SQL model that is used to generate flat SQL queries and this is especially useful when the database engine used does not support subquery elimination.

2.4 Mapping Layer in OBDA

The next OBDA component that we discuss is the mapping component. We start this section by discussing existing approaches for the design of mappings. Next, we see that mappings can be implemented in various ways, some of which are proprietary formats, while some others are W3C Recommendations.

2.4.1 OBDA Mapping Approaches

The mapping layer contains information about how the source layer and the target layer are related. Two basic approaches for defining mappings are Local As View (LAV) and Global As View (GAV), each having its own pros and cons. A combination of the

¹For the sake of simplicity, we omit the aliases that have to be prefixed on each column name.

two approaches, called Global Local As View (GLAV), has also been proposed in the literature. Next, we discuss each of the approaches in more detail.

2.4.1.1 Local As View (LAV)

In LAV [Ull00] each element of the source schema \mathcal{S} is mapped to a query $Q_{\mathcal{G}}$ over the global schema \mathcal{G} . That can be expressed as $s \rightsquigarrow q(\mathcal{G})$ where s is element of \mathcal{S} and $q(\mathcal{G})$ is a query over global schema \mathcal{G} . This approach supports for easy addition of new sources since there is no need to change the query processing component, thus this approach should be taken when the global schema \mathcal{G} is already stable but the local schema \mathcal{S} may be modified over time or new sources may be added.

Example 2.4. Consider the following example:

- The global schema \mathcal{G} with a class $Person(PID)$ together with two properties that are: $hasName(PID, PName)$ and $hasSport(PID, PSport)$.
- The source schema \mathcal{S} contains two relations that are: $Grandmaster(SID, SName)$ and $SoccerPlayer(PID, PName)$.
- The LAV mapping between \mathcal{G} and \mathcal{S} can be defined as:

$$Grandmaster(x, y) \rightsquigarrow \{\langle x, y \rangle | Person(x) \wedge hasName(x, y) \wedge hasSport(x, "Chess")\}$$

$$SoccerPlayer(x, y) \rightsquigarrow \{\langle x, y \rangle | Person(x) \wedge hasName(x, y) \wedge hasSport(x, "Soccer")\}$$

2.4.1.2 Global As View (GAV)

In GAV [Hal01] each element of the global schema \mathcal{G} is mapped to a query over the source schema \mathcal{S} . That can be expressed as $g \rightsquigarrow q(\mathcal{S})$ where g is element of \mathcal{G} and $q(\mathcal{S})$ is a query over global schema \mathcal{S} . Since this approach is the opposite of LAV, then the reverse properties of LAV hold. This approach is ideal when the set of sources is already stable.

Example 2.5. Consider the following

- The global schema \mathcal{G} contains a class $SpanishGrandmaster(Id)$.
- The source schema \mathcal{S} contains a relation $ChessPlayer(Id, Name, Country, Title)$.
- The GAV mapping between \mathcal{G} and \mathcal{S} can be defined as

$$SpanishGrandmaster(x) \rightsquigarrow \{\langle x \rangle | ChessPlayer(x, y, "ES", "GM")\}$$

2.4.1.3 Global Local As View (GLAV)

Besides the two main approaches (LAV and GAV), there is another approach which combines those two approaches. That approach is called GLAV [FLM99]. So in GLAV, the mapping has the form $q(\mathcal{S}) \rightsquigarrow q(\mathcal{G})$ which means that the query over the source schema is represented as a query over the global schema.

Example 2.6. *Consider the following example:*

- The global schema \mathcal{G} with a class $ChessPlayer(PID)$ together with its properties: $hasName(PID, PName)$ and $hasCountry(PID, PCountry)$.
- The source schema \mathcal{S} contains a relation: $Spaniard(PID, PName, PSport)$.
- The GLAV mapping between \mathcal{G} and \mathcal{S} can be defined as:

$$Spaniard(x, y, "Chess") \rightsquigarrow \{ \langle x, y \rangle | ChessPlayer(x) \wedge hasName(x, y) \wedge hasCountry(x, "ES") \}$$

2.4.2 Proprietary Mapping Languages

Prior to the creation of the RDB2RDF W3C Working Group, many mapping languages had been created for the purpose of implementing mappings between relational databases and ontologies. Next, we discuss two of the most relevant ones: R_2O and D2RQ.

2.4.2.1 R_2O

R_2O [Bar07] is a declarative mapping language developed by the Ontology Engineering Group. It falls on the category of Global As View (GAV) mapping approach. The main components of this mapping language are:

- **Concept Map**, which consists of the following elements: target class **name**, IRI generation expression **uri-as**, conditional expression **applies-if**, join condition expression **joins-via**, a set of attribute maps **attributemap-def**, and a set of relation maps **relationmap-def**. For example, the following Concept Map $CM - Grandmaster$ is defined as follows:
 - **name** specifies to which class the generated instances belong to, for example class `ex:Grandmaster`.
 - **uri-as** specifies the expression for generating the IRIs of the instances. For example, the expression `CONCAT("http://ex.com/Grandmaster/", PLAYER.PID)`

specifies that the IRI of the generated instances come from the concatenation value of the base IRI `http://ex.com/Grandmaster/` with values from the column `PID` of table `PLAYER`.

- **applies-if** is used to filter rows from the participating tables. For example, the conditional expression `PLAYER.ACTIVE=Y` only generate instances for active players.
 - **joins-via** specifies the join condition of tables participating in the mapping.
 - **attributemap-def** specifies a set of attribute maps for generating data properties associated to the instances. This will be explained more detail below.
 - **relationmap-def** specifies a set of relation maps for generating object properties associated to the generated instances. This will be explained more detail below.
- **Attribute Map**, which is used to associate a data property with the generated instances. An Attribute map consists of the following elements: concept map **CM**, target property **name**, conditional expression **applies-if**, join condition expression **joins-via**, and value transformation expression **aftertransform**. For example, the following Attribute Map *PM – hasRating* is defined as follows:
 - **name** specifies the data property of instances to be generated, for example, `ex:hasSex`.
 - **applies-if** specifies the conditional expression to be evaluated, such as `PLAYER.PSEX <> ""`.
 - **aftertransform** specifies the transformation expression to be performed, for example `IF(PSEX="M") THEN "Male" ELSE "Female"`.
 - **joins-via** specifies the join expression needed if multiple tables are participating.
 - **Relation Map**, which is used to associate an object property with the generated instances, by relating them with another concept map. A Relation map consists of the following elements: **name**, **toconcept**, **applies-if**, and **joins-via**. For example, the following relation map *RM – hasCountry* is defined as follows:
 - **name** that specifies the object property of instances to be generated, for example, `ex:hasCountry`.
 - **toconcept** that specifies the other concept map that whose instances will be used as the value of the generated object properties, for example, *CM – Country*.

- `applies-if` that specifies the condition expression to be fulfilled, for example, `USER.UCOUNTRY <> ""`.
- `joins-via` that specifies the join condition between the two concept maps, for example, `USER.UCOUNTRY = COUNTRY.CID`.

2.4.2.2 D2RQ

One of the most widely used RDB2RDF systems is the D2RQ platform (D2RQ mapping language [BS04], D2RQ engine and D2RServer [BC06]). Like *R₂O*, this mapping language also falls on the GAV category. The D2RQ mapping language is a declarative XML-based language that permits mappings between relational database schema with an ontology. The D2RQ engine is a processor using the Jena API or Sesame API that rewrites an RDF query into SQL queries. D2RServer is an HTTP server that permits viewing the RDF data as Linked Data. There is no formal description of the semantics of the D2RQ mapping language and no literature available that elaborates on the process of query translation performed by the D2RQ engine. Two main mapping components of D2RQ are `ClassMap` and `PropertyBridge`, which we explain in more detail below.

`ClassMap` is used to specify how instances of the mapped class are generated. Some important elements of `ClassMap` are:

- `d2rq:class` specifies the target class to which the generated instances belong, for example `ex:Grandmaster`
- `d2rq:uriColumn` specifies the column to be used as the URI for the generated instances, for example, `"PERSON.PWEBPAGE"`.
- `d2rq:uriPattern` specifies the expression for the generation of the URI for the generated instances, for example, `"http://www.ex.com/Grandmaster/@@Person.PID@@"`.
Note that `d2rq:uriPattern` can not be used together with `d2rq:uriColumn`.

`PropertyBridge` is used to generate pair of property and object for instances of the mapped classes. Some important elements of `PropertyBridge` are:

- `dr2q:property` specifies the IRI of the property to be generated, for example, `foaf:name`.
- `d2rq:belongsToClassMap` specifies which class the generated property belongs to, for example, the property bridge `foaf:name` may be defined to have `d2rq:belongsToClassMap foaf:Person`. `d2rq:column` specifies the value of the object corresponding to

the generated property, for example, the property bridge `foaf:name` may have `d2rq:column "PERSON.PNAME"`.

- `d2rq:pattern` specifies the expression for generating literal values, for example, the property bridge `ex:fullname` may have `d2rq:pattern "@@PERSON.LASTNAME@@ , @@PERSON.FIRSTNAME@@"`.

For a complete list of D2RQ elements, we refer the reader to <http://d2rq.org/d2rq-language>.

2.4.3 W3C Recommendations on Mapping Languages

In 2007, a workshop on RDF Access to Relational Databases sponsored by the W3C was held, gathering researchers reporting their works and experiences with RDB2RDF systems. The workshop was concluded by a common agreement that there was a need to have a common mapping language for RDB2RDF systems, thus, an W3C Incubator Group was founded in the following year having a two-fold mission:

”To examine and classify existing approaches to mapping relational data into RDF and decide whether standardization is possible and/or necessary in this area and, if so, the direction such standardization could take. The goal is to specify how to generate RDF triples from one or more Relational tables without loss of information. Further, a default mapping should not be used, but, instead, it should be possible for the the mapping to be customized by the user.”

and

”To examine and classify existing approaches to mapping OWL classes to Relational data, or, more accurately, SQL queries, moving towards the goal of defining a standard in this area. Each OWL class would be associated with one or more SQL queries which may be run on separate databases. The results from these queries would then be integrated into a single Relational table. This would be transformed into RDF using the approach defined as a result of the first initiative.”

In 2009, having published their survey of approaches for RDB2RDF [SHH⁺09], the incubator group was closed and the W3C RDB2RDF Working Group was formed, with the mission to standardize languages for mapping relational data and relational database

schemas into RDF and OWL. In 2012, this working group published its main results: Direct Mapping [MAS12] and R2RML [DSC12], both classified as the W3C Recommendations.

2.4.3.1 Direct Mapping

Direct Mapping defines a default and automatic transformation of relational data to an RDF graph. The input to a direct mapping is a relational schema R and an instance I of that schema. The output is an RDF graph, denoted by $\llbracket \mathcal{DM} \rrbracket_I$, which is called *direct graph* [ABP⁺13, SAM12]. One implementation of Direct Mapping engine is Ultrawrap developed by Sequeda and colleagues [SM13]. In addition to performing transformation process from database content into RDF, Ultrawrap also employs an efficient SPARQL query answering by pushing the optimizations as much as possible to the underlying relational database systems.

The transformation rules defined in the Direct Mapping recommendation are:

- *Row node*. If a table has a primary key, then each relational table row is identified by an IRI, which is generated by concatenating the `TableName` with the value of the primary key columns. If there is no primary key defined, then a blank node is generated.
- *Table triples* generation rule. Each row in a relational table produces a *table triple* (`s rdf:type TableName`) where `s` is the row node (IRI or blank node) and `TableName` is the name of the table. A base IRI can be appended to `TableName`.
- *Literal triples* generation rule. For each cell in a relational table row, the *literal triple* (`s ColumnName TableName`) is generated, where `s` is the row node (IRI or blank node); `ColumnName` is the concatenation of `TableName` and the name of the column. `TableName` is as explained above.
- *Reference triples* generation rule. For each cell, in a relational table row, that belongs to a foreign key column in that table, a *reference triple* (`s RefColumnName o`) is generated, where `s` is the row node (IRI or blank node); `RefColumnName` is the concatenation of `TableName` and the name of the referenced column. Finally, `o` is the row node (IRI or blank node) corresponding to the referenced row.

Example 2.7. Consider a view *ValidOffer* in Table 2.2, having three columns (*OID*, *PID*, and *Price*) without any primary keys. The application of Direct Mapping of the first row of this view can be seen in line 1-4 of Listing 2.2 and is explained below:

ValidOffer		
OID	PID	Price
1	1	1.5
2	2	2.3

TABLE 2.2: View ValidOffer

- *Row node.* As no primary keys defined, a fresh blank node will be generated as the row node. In our example, one possible generated blank node is
:ValidOffer/OID=1.PID=1.Price=1.5.
- *Table Triple.* The table triple for the first row is:
:ValidOffer/OID=1.PID=1.Price=1.5 a <ValidOffer>.
- *Literal triples.* Three literal triples will be generated for this row, each corresponds to an attribute of this view.
 - :ValidOffer/OID=1.PID=1.Price=1.5 <ValidOffer/OID> 1.
 - :ValidOffer/OID=1.PID=1.Price=1.5 <ValidOffer/PID> 1.
 - :ValidOffer/OID=1.PID=1.Price=1.5 <ValidOffer/Price> 1.5.

```

1 :ValidOffers/OID=1.PID=1.Price=1.5 a <ValidOffers> ;
2   <ValidOffers#OID> 1 ;
3   <ValidOffers#PID> 1 ;
4   <ValidOffers#Price> 1.5 .
5
6 :ValidOffers/OID=2.PID=2.Price=2.3 a <ValidOffers> ;
7   <ValidOffers#OID> 2 ;
8   <ValidOffers#PID> 2 ;
9   <ValidOffers#Price> 2.3 .

```

LISTING 2.2: Output Dataset of evaluating Direct Mapping over a view in Table 2.2

2.4.3.2 R2RML

R2RML allows specifying rules for transforming relational database content into an *R2RML output dataset*, the resulting graph from applying R2RML mappings. The transformation rules are defined in an R2RML mapping document, which contains a set of Triples Maps (`rr:TriplesMap`). Triples Maps are used to generate RDF triples from logical tables. The components of Triples Maps are:

- one logical table (`rr:LogicalTable`) that specifies the source relational table/view.
- one subject map (`rr:SubjectMap`) that specifies the target URI classes

- optionally multiple predicate-object maps (`rr:PredicateObjectMap`). Predicate-object map is composed by one or many predicate maps (`rr:PredicateMap`), and one or many object maps (`rr:ObjectMap`), or reference object maps (`rr:RefObjectMap`) when joins with another logical table are needed.

Unless explicitly specified using (`rr:GraphMap`), the resulting triples will be stored in a default graph. Figure 2.4 gives an overview of R2RML Triples Map.

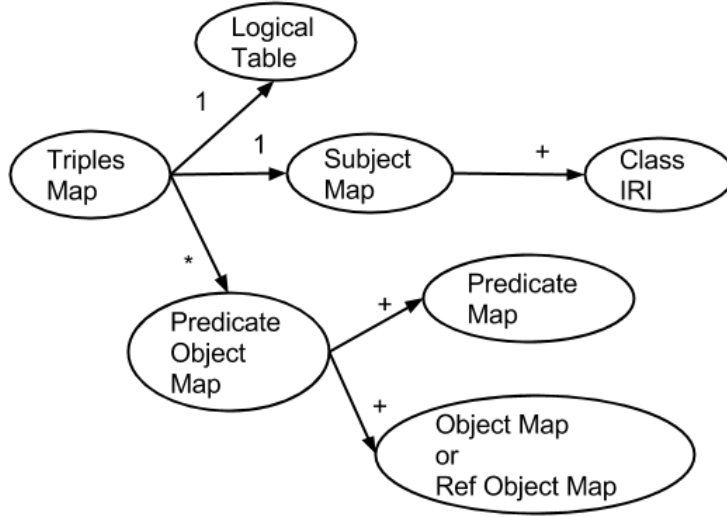


FIGURE 2.4: Overview of R2RML Triples Map

Subject Map, Predicate Map, and Object Map are defined as Term Maps (`rr:TermMap`). Term Maps are used to generate RDF terms, either as IRIs (`rr:IRI`), Blank Nodes (`rr:BlankNode`), or literal (`rr:Literal`). The values of the term maps can be specified using constant-valued map (`rr:Constant`), column-valued map (`rr:Column`), or template-valued map (`rr:Template`). Furthermore, additional information such as language (`rr:Language`) or datatype (`rr:Datatype`) can also be attached to term maps. Figure 2.5 gives an overview of R2RML Term Map.

Example 2.8. Consider the table *Offer* which can be seen in Table 2.3.

Offer				
nr	product	label	price	validFrom
1	1	offer on pen	1.5	2030-12-31
2	2	offer on water	2.3	2020-12-31
3	1	exclusive deal on pen	1.2	1999-12-31

TABLE 2.3: Example of table *Offer*

One possible R2RML *TriplesMap* for that table can be seen in Listing 2.3 and is explained below:

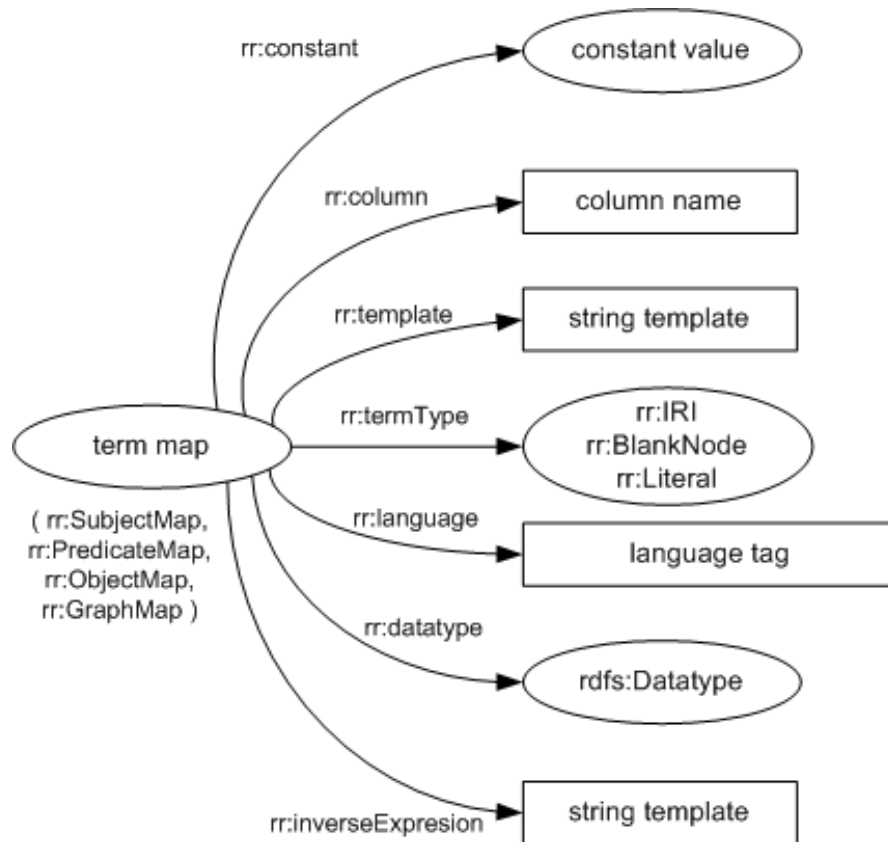


FIGURE 2.5: Overview of R2RML Term Map

- Line 2 declares the table *Offer* as a logical table.
- Line 6 declares that the output of this triples map are instances of type *bsbm:Offer*.
- Line 7 specifies the URI of the generated instances.
- Lines 10-13 declare that for each instance, there is a triple with predicate *rdfs#label* and the value of the column *label* as its object.
- Lines 15-18 declare that for each instance, there is a triple with predicate *bsbm:price* and the value of the column *price* as its object.
- Lines 20-23 declare that for each instance, there is a triple with predicate *bsbm:hasProduct* and the concatenation of *Product* URI and the column *product* as the value of its object.

The application of the R2RML mapping in Listing 2.3 over the table *Offer* will produce the R2RML output dataset that can be seen in Listing 2.2.

Recently, there have been some attempts to extend R2RML to support data sources beyond relational databases. For example, Calbimonte et. al., extend R2RML so that it

```

1 <TMOffer> a rr:TriplesMap;
2   rr:logicalTable [ rr:tableName "Offer" ];
3
4   rr:subjectMap [
5     a rr:Subject;
6     rr:class bsbm:Offer;
7     rr:template "http://localhost:2020/resource/Offer/{nr}";
8   ];
9
10  rr:predicateObjectMap [
11    rr:predicateMap [ rr:constant rdfs:label ];
12    rr:objectMap [ rr:column "label" ];
13  ];
14
15  rr:predicateObjectMap [
16    rr:predicateMap [ rr:constant bsbm:price ];
17    rr:objectMap [ rr:column "price" ];
18  ];
19
20  rr:predicateObjectMap [
21    rr:predicateMap [ rr:constant bsbm:hasProduct ];
22    rr:objectMap [ rr:template "http://localhost:2020/resource/Product/{product}" ];
23  ];
24 ].

```

LISTING 2.3: R2RML Mappings Correspond to Table **Offer**.

```

1 <Offer/1> a bsbm:Offer ;
2   rdfs:label " offer  on pen" ;
3   bsbm:price 1.5 ;
4   bsbm:hasProduct <Product/1> .
5
6 <Offer/2> a bsbm:Offer ;
7   rdfs:label " offer  on water" ;
8   bsbm:price 2.3 ;
9   bsbm:hasProduct <Product/2> .
10
11 <Offer/3> a bsbm:Offer ;
12   rdfs:label " exclusive deal on pen" ;
13   bsbm:price 1.2 ;
14   bsbm:hasProduct <Product/1> .

```

LISTING 2.4: Output Dataset of evaluating R2RML Mappings in Listing 2.3 over the Table **Offer**

supports window operators commonly used on sensor data processing [CCG10]. Michel et. al., have proposed xR2RML [MDFZM15], an R2RML extension that supports XML, object oriented databases, and NoSQL databases. In parallel, Dimou et. al., have proposed RML [DVSC⁺14], which supports CSV, XML, and JSON. In Section 8.1, we will also describe morph-GFT [PAC13], an extension of morph-RDB, our R2RML engine, that supports data coming from Google Fusion Tables and SPARQL endpoints.

2.4.4 Mapping Generation Approaches

Several works in the state of the art have dealt with the automatic or manual generation of RDB2RDF mappings.

The first group of systems that we can refer to is that of early RDB2RDF systems (e.g. ODEMapster [Bar07], D2R Server [BC06], Triplify [ADL⁺09b]), which used their own mapping languages to transform relational database content to RDF. They also had associated functionalities to ease the generation of such mappings, either manually or automatically. For example, the ODEMapster GUI was a NeOn Toolkit plugin that allowed specifying in a graphical manner the most common types of mappings that may be declared in the R2O language. D2R Server provides an automatic mapping generation functionality based on table and column names, as well as constraints such as primary and foreign keys, what is very similar to the one provided for the W3C Direct Mapping. This implementation was first available for the D2R language and later for R2RML. Triplify provides mapping templates for some well-known Web applications (e.g. WordPress, Joomla!, or Drupal).

There are also other approaches that are independent from the mapping language and tool used to specify and run mappings. For instance, the authors in [STCM11, SAM12] analyse the different types of relationships that exist between relational tables, using primary and foreign keys, so as to determine the classes and properties of the expected results in RDF. However, no mappings are generated as a result of this analysis, what means that this work cannot be reused by other RDB2RDF engines. More recent work [SAM14a] has proposed a fixed set of rules for saturating mappings, once that the mappings between the relational database and an ontology have been defined by a domain expert.

In [PVC⁺14] the authors proposed a semi-automatic mapping generation process, where R2RML mappings are generated based on a set of semantic correspondences (for classes and properties) defined by domain experts. And the authors in [SHSH13] present a GUI-based R2RML mapping editor to help non-expert users to create and modify their mappings.

In the context of ontology alignment, QQDI [TSM13] uses the input query as context for dynamically generates mapping between two ontologies. By taking queries as context, QQDI is shown to be able to deal with two challenges: ambiguous mapping and missing mapping.

However, none of the aforementioned approaches and systems deals with the automatic generation, from relational databases, of R2RML mappings that encode the implicit information that can be obtained from the relational database schema.

2.4.5 Relationship Between Direct Mapping and R2RML

Both recommendations (Direct Mapping and R2RML) were created in parallel, with the purpose of transforming content of relational database into RDF. However, they differ in the approach they take.

The Direct Mapping recommendation specifies the transformation rules to be applied to generate automatically an RDF dataset that reflects the structure and content of the relational database. Since this may not be always adequate or optimal, especially in those cases when the relational database content needs to be transformed into RDF according to an existing ontology, the R2RML recommendation allows customising the transformation rules to be applied.

The Direct Mapping approach employs the *one-click* paradigm, in the sense that no intervention from users is needed for the transformation process² because those rules are specified in the recommendation for generating an RDF dataset that reflects the structure and content of the relational database. Since this may not be always adequate or optimal, especially in those cases when the relational database content needs to be transformed into RDF according to an existing ontology, the R2RML recommendation allows customising the transformation rules to be applied.

Hence R2RML provides more flexibility than its counterpart, the Direct Mapping specification. However, this comes at a cost for users interested in generating RDF from their relational databases: they need to learn how to create those R2RML mappings. Several tools have been made available to facilitate the task of mapping generation, but either they produce mappings in earlier RDB2RDF languages (e.g. the ODEMapster GUI, which produces R2O mappings) or are not usable enough (e.g. form-based tools that only provide syntactic sugar to users, who still require a good knowledge of R2RML). An alternative approach to ease the burden of R2RML creation from users, making them more efficient, is to bootstrap the process with the creation of an initial R2RML mapping document that reflects the behaviour of the Direct Mapping specification, and then allow users to edit that document further, e.g. in a text editor. This has generally proven to be useful in our own work, since in many cases a large percentage of triple maps inside an R2RML mapping document are reused. This has also an additional positive side effect, which is the fact that any R2RML engine (e.g. morph-RDB) can be used to produce RDF following the Direct Mapping specification.

Intuitively, one can consider that the expressive power of the Direct Mapping recommendation is less than that of R2RML. However, there hasn't been any study reported about the expressive power of these two recommendations. Such study may facilitate

²Hence, *one-click* refers to the only click needed to initiate the transformation process.

better understanding of the relationship between the two recommendations both theoretically and practically. For example, if we can infer that the Direct Mapping has the same expressive power as the R2RML recommendation, then software developers may reuse an existing Direct Engine mapping for developing an R2RML engine.

2.5 Query Answering in OBDA

Consider an OBDA $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, where \mathcal{O} is an ontology acting as the global schema, \mathcal{S} is the source schema, and \mathcal{M} is the mapping between \mathcal{O} and \mathcal{S} . When a query q posed over \mathcal{O} , there are two approaches for answering this query: *bottom-up* and *top-down* [PLC⁺08]. The first option is to generate and materialize the RDF graph of the ontology \mathcal{O} from the source \mathcal{S} according to \mathcal{M} . Hence, this approach is sometimes called *materialized approach*. While this approach is fairly simple, the drawback is the need to redo the materialization process whenever there is an update over the data source \mathcal{S} .

The top-down approach, on the contrary, avoids the materialization process and instead rewrites the original query q defined over target schema \mathcal{O} to a new query $q_{\mathcal{S}}$ defined over the source schema \mathcal{S} , taking into account both \mathcal{O} and \mathcal{M} . This approach is commonly known as *virtualized approach*. The three main steps of this approach are:

1. **Reformulation.** First, the query q is rewritten (hence, this step is also called *query rewriting*) according to the ontology \mathcal{O} , resulting in query $q_{\mathcal{O}}$.
2. **Unfolding.** Next, the query $q_{\mathcal{O}}$ is translated (hence, this step is also called *query translation*) using the mapping \mathcal{M} , resulting in query $q_{\mathcal{S}}$, that is evaluable over the source schema \mathcal{S} .
3. **Query Evaluation.** Finally, the query $q_{\mathcal{S}}$ is evaluated over the schema \mathcal{S} , and the results of this translation are evaluated again using the mapping \mathcal{M} as the final result of answering the original query q .

The R2RML recommendation does not provide any formalisation of the SPARQL to SQL query translation process that needs to be followed by R2RML-aware query translators, and none of the aforementioned implementations describe the formalizations of their query translation algorithms. In any case, some attempts have been done in the past to provide such a formalisation. For instance, Garrote and colleagues [GG11] provide a draft of the transformation of SPARQL SELECT queries to SQL based on the query translation algorithm defined in [CLF09]. As their focus is to provide an R2RML-based RESTful writable API, the query translation algorithm that they describe is limited.

For example, projections and conditions for those queries are built only for variable components of the triple pattern. However, a triple pattern may involve URIs and constants in their components, and these have to be taken into account when translating SPARQL queries. Unbehauen and colleagues [USA13] define the process of binding triple patterns to the mappings and the process of generating column groups (a set of columns) for every RDF term involved in the graph pattern. Using the calculated bindings and column groups, they define how to translate each of the SPARQL operators (AND, OPTIONAL, FILTER, and UNION) into SQL queries. However, the function *joinCond*(*s1*,*s2*), in which two patterns are joined, is not clearly defined. In fact, one fundamental aspect that distinguishes SPARQL queries from SQL queries is the semantics of the joins, as discussed in [Cyg05, CLF09], which corresponds to the treatment of NULL values in the join conditions. In recent works, Rodriguez-Muro and Rezk have presented *ontop* [RMR15], which translates R2RML mappings and SPARQL queries into a set of Datalog rules, where optimizations based on query containment and Semantic Query Optimisation are applied, before transforming them into SQL queries. In Section 4.2, we will see in more detail the SQL queries resulting from the optimization techniques, and the differences between the SQL queries generated by *ontop* and by our approach.

Furthermore, the performance of virtual RDF datasets based on RDB2RDF mapping languages has not always been satisfactory, as reported by Gray et al. [GGO09]. This experience has also been confirmed empirically by us in projects like Répener [SNMM13], BizkaiSense³, or Integrate⁴, for which we have had access to the data sources and mappings in languages like D2R. In all cases, some of the SQL queries produced by the translation algorithm could not be evaluated by the underlying relational database engine (e.g., too many joins are involved) or their evaluation took too much time to complete, what made their use in a virtual RDF dataset context unfeasible. The main reason for this is that the resulting queries are not sufficiently optimised to be efficiently evaluated over the underlying database engines. Thus, some optimization techniques are needed to produce efficient SQL queries, especially if we cannot depend on the underlying database systems to perform the necessary optimizations. Examples of such optimizations are: self-join elimination, sub-query elimination and detection of unsatisfiability condition (such as IS NOT NULL over primary key columns). Table 2.4, which is an extension of the table presented in [RMR15] and presents the summary of those techniques and the supporting systems. We will see in more detail such optimizations in Section 4.2.

³<http://www.tecnologico.deusto.es/projects/bizkaisense/>

⁴<http://www.fp7-integrate.eu>

	Chebotko [CLF09]	Elliot [ECTOO09]	D2R [BC06]	ODEMapster [Bar07]	ontop [RMR15]	Ultrawrap [SM13]	morph-RDB [PCS14]
Type	Triple Stores	Triple Stores	OBDA	OBDA	OBDA	OBDA	OBDA
Mapping Language	No	No	D2RQ	R2O	ontop R2RML	Direct Mapping	R2RML
Self-join elimination	No	No	No	No	yes	yes	yes
Push join into union	No	No	No	No	yes	no	no
Detection of unsatisfiable condition	No	No	No	No	yes	yes	yes
Remove URIs from join	No	No	No	No	yes	yes	yes
Subquery elimination	No	Yes	Yes	No	yes	no	yes

TABLE 2.4: Comparison table of query translation techniques and the supporting systems, extended from [RMR15].

Recall that in Section 2.3.2, we have discussed some works of translating SPARQL into SQL [CLF09, ECTOO09] in the context of using relational databases as the backend for triple stores. This means that these techniques are oblivious to mappings. In Chapter 4 we will describe how we formalize the query translation algorithm taking into account R2RML mappings.

2.6 Summary

In this chapter we have seen the state of the art of OBDA. We reviewed the concept of OBDA in general and RDB2RDF in particular, together with its SPARQL to SQL query translation techniques. We turned our attention to the mappings component of RDB2RDF, in which we reviewed proprietary and W3C Recommendation mapping languages. We focused on the W3C Recommendation mapping languages, Direct Mapping and R2RML. We have seen that there are two main problems in the state of the art. The first problem is the lack of formalization of query translation techniques for RDB2RDF systems that uses R2RML mapping language together with its optimization. The second problem is the study of the relationship between the two mapping language recommendations. In the next chapter, we formalize the research problems, our hypotheses, and the objectives of this thesis document.

Chapter 3

Objectives and Contributions

“It takes a lot of people to make a winning team. Everybody’s contribution is important.”

Gary David Goldberg

In this chapter, we present the research problems that we want to address and our objectives. We also state the hypotheses that we want to validate, together with our main assumptions, and provide the context for this work and the limitations that can be associated to the proposed solutions.

3.1 Problem Statement and Objectives

The primary aim of this work is to provide theoretical foundations and technological support for accessing and querying R2RML-mapped data sources, using a sound and efficient algorithm. To achieve this aim, the following research problems have to be addressed:

- P1. The existing algorithms for R2RML-based SPARQL to SQL query translation have not been formalized ¹.
- P2. Some of the SQL queries that result from existing SPARQL to SQL query translation algorithms exhibit a very low performance when evaluated in the underlying relational database management systems.

¹In the final stage of the writing of this thesis document, Rodriguez-Muro and Rezk published their recent work in [RMR15] where they have also formalised an R2RML-based SPARQL-to-SQL query translation.

The first problem leads us to the following research question: *How can we formalise a sound SPARQL to SQL query translation algorithm, taking into account R2RML mapping documents?* Our hypothesis (H1) is that the formalisation of an existing query translation algorithm originally designed to work with non R2RML-aware systems, such as RDBMS-backed triple stores, can be extended to deal with R2RML-mapping based SPARQL to SQL query translation. Our objective **(O1)** is to **propose and formalize an algorithm for translating SPARQL to SQL queries that takes into account R2RML mappings.**

For the second problem, the following research question is to be addressed: *How do we optimize the proposed algorithm in order to produce SQL queries that are efficient enough to be evaluated over relational database systems in a virtualized approach for a set of the most common types of queries?* Our hypotheses are: (H2) the SQL queries that result from translating SPARQL query patterns using existing query translation algorithm are not necessarily efficient, what yields to poor performance in terms of query evaluation time and (H3) there are optimization techniques known in the relational database domain that can be applied to generate more efficient SQL queries. Our objective **(O2)** is to **apply optimization techniques to the proposed SPARQL to SQL query translation algorithm so that queries can be evaluated more efficiently over the underlying database systems while preserving soundness.**

The secondary aim of this work is to gain a clearer understanding of the relationship between the W3C Direct Mapping and R2RML recommendations. This aim is driven by the following research problem: (P3) lack of a formal study of the relationship between the W3C Direct Mapping and R2RML recommendations.

We divide the corresponding questions related to this research problem into two types: from R2RML to DM, and from DM to R2RML.

- From R2RML to DM, the corresponding research questions to be addressed are: *How can the W3C R2RML and Direct Mapping recommendations be related?* More specifically, *which fragment of R2RML has the same expressive power with Direct Mapping* and furthermore, *how do we express such relationship?* Our hypotheses are that (H4) there is a particular fragment of R2RML that has the same expressive power as the W3C Direct Mapping recommendation, and (H5) two fundamental properties of Direct Mapping can be used to express the relationship between the W3C R2RML and Direct Mapping recommendations.

Our objective associated to this research problem is **(O3) to formalize the relationship between the Direct Mapping and R2RML recommendations.**

- From DM to R2RML, the corresponding research questions are: *How do we represent Direct Mapping as R2RML mappings so as to bootstrap the mapping generation process while at the same time to extend the generated R2RML mappings with the implicit knowledge encoded in a database schema.* Our hypothesis is that (H6) we can encode the Direct Mapping rules and (H7) the implicit knowledge encoded in a relational schema as R2RML.

Our objective associated to this research problem is **(O4) to propose an approach of representing Direct Mapping together with the implicit knowledge encoded in a relational database schema as R2RML mappings.**

3.2 Assumptions and Limitations

The work in this thesis is based on the following set of assumptions.

- The time required for translating SPARQL to SQL can be generally considered neglectable in comparison with the time used for evaluating the SQL queries.
- R2RML mapping documents have been correctly defined by domain experts and reflect the relationship between the underlying relational databases and the RDF model to be generated.
- Users are familiar and able to formulate SPARQL queries.
- There is a set of SPARQL query patterns that are the most commonly used ones by SPARQL-aware information system.
- The impact of the proposed optimization techniques is more relevant on non-commercial relational databases than in commercial ones.
- Statistics of data source content are not expected to be available all the time. However, when available, they may be used to improve the performance of the resulting translated queries.
- It is possible to define views on the underlying database, as well as IRI replacement functions over the generated virtual RDF dataset/SPARQL query result bindings.
- Primary keys are NOT NULL and UNIQUE.
- Relational database schemes have been defined according to 3rd Normal Form.

Furthermore, the work is bound on the following limitations/restrictions:

- Ontology-based query rewriting is out of the scope of this thesis. Some recent work in this direction can be found at [MRC14].

3.3 Contributions

This thesis aims to provide both conceptual and technological solutions to the research problems mentioned above. Chapters 4, 5 and 6 describe the conceptual solutions, while Chapter 8 presents the technological solutions.

The conceptual contributions of this thesis are:

- (C1) **Formalisation** of R2RML-based SPARQL to SQL query translation together with its (C2) **optimizations**, which are described in Chapter 4.
- (C3) A definition of an R2RML fragment called **R2RML_{lite}**, and procedures to normalize that fragment into a simpler form, together with (C4) a formal study of the **relationship** between **R2RML_{lite}** and the W3C Direct Mapping recommendation.
- (C5) An approach to represent Direct Mapping and implicit knowledge encoded in database schemes as R2RML mappings.

The technological contributions are bundled inside the **morph** ontology-based data access suite, which consists of:

- (C6) **morph-RDB**, an R2RML engine that implements our contributions (C1) and (C2).
- (C7) **morph-GFT**, an extension of morph-RDB that evaluates federated SPARQL queries over Google Fusion tables and other SPARQL endpoints.
- (C8) **morph-LDP**, an extension of morph-RDB that is able to perform updates on the underlying R2RML-mapped relational data.
- (C9) **MIRROR**, an R2RML mappings generator system that implements our contribution (C5).

Chapter 4

Formalization and Optimizations of R2RML-based SPARQL to SQL Query Translation

“My big thesis is that although the world looks messy and chaotic, if you translate it into the world of numbers and shapes, patterns emerge and you start to understand why things are the way they are.”

Marcus du Sautoy

In this chapter we present our first contribution, the extension of an existing SPARQL to SQL query translation algorithm [CLF09] originally designed to work with RDBMS-backed triple stores, so that it takes into account R2RML mappings for the generation of equivalent SQL queries.

We divide this chapter into two main parts. The first part (Section 4.1) is where we formalize the R2RML-based query translation technique. In that section, we describe how we relate the mappings/functions defined in the aforementioned algorithm with the relevant parts of R2RML mappings. This allows the algorithm to work with arbitrary relational database layouts, such as the ones normally used in legacy systems. The second part (Section 4.2) is the optimisation of our initial query translation technique, where we describe how to eliminate unnecessary self-joins, left-join elimination, and other optimisations.

4.1 An R2RML-based extension of Chebotko's approach

In Section 2.3.2, we have seen how the algorithm defined in [CLF09] is used to translate SPARQL queries into SQL queries for RDBMS-backed triple stores. Before we present the detail of how to extend that algorithm to work with R2RML mappings, we give an illustrative example of R2RML mappings and a SPARQL graph pattern that will be used throughout this chapter.

4.1.1 Illustrative Example

Example 4.1. Consider the R2RML mapping document in Listing 4.1, which specifies a mapping from the table *Product(nr, label)* into an ontology concept *bsbm:Product*. The column *nr* is mapped as part of the URI of the *bsbm:Product* instances and the column *label* is mapped to the property *rdfs:label*. Now consider the SPARQL graph pattern shown in Listing 4.2. Both the R2RML mappings and the SPARQL query will be used throughout this chapter.

The first triple pattern ($tp1 = :Product/1 \text{ rdfs:label } ?lbl$) in line 2 corresponds the following mappings/functions:

- **Mapping** α returns the table that holds the triples that correspond to this triple pattern $tp1$. $\alpha(tp1) = \textit{Product}$.
- **Mapping** β returns the column/constant that corresponds to each triple pattern position (*sub*, *pre* or *obj*). That is, $\beta(tp1, \textit{sub}) = \textit{nr}$, $\beta(tp1, \textit{pre}) = \textit{'rdfs:label'}$, and $\beta(tp1, \textit{obj}) = \textit{label}$.
- **Function** *genCondSQL* filters the table *Product* returned by $\alpha(tp1)$ so that only the records that match the triple pattern $tp1$ are returned. That is, $\textit{genCondSQL}(tp1, \beta) = \{\textit{nr} = 1 \text{ AND } \textit{label} \text{ IS NOT NULL}\}$.
- **Function** *name* generates alias for the triple pattern element. In this case, $\textit{name}(:Product/1) = \textit{iri_Product1}$, $\textit{name}(\textit{rdfs : label}) = \textit{iri_rdfs_label}$, and $\textit{name}(lbl) = \textit{var_lbl}$.
- **Function** *genPRSQL* projects the β column as the alias name of each triple pattern element. That is, $\textit{genPRSQL}(tp1, \beta, \textit{name}) = \{\textit{nr AS iri_Product1}, \textit{'rdfs:label' AS iri_rdfs_label}, \textit{label AS var_lbl}\}$.
- **Function** *trans* finally returns the SQL query using the values returned by the previous mappings and functions. That is, $\textit{trans}(tp1, \alpha, \beta) = \textit{SELECT nr AS iri_Product1},$

```

1  @prefix rr: <http://www.w3.org/ns/r2rml#> .
2  @prefix bsbm: <http://localhost:2020/resource/vocab/> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
5
6  <TMProduct> a rr:TriplesMap;
7    rr:logicalTable [ rr:tableName "Product" ];
8
9    rr:subjectMap [ a rr:Subject;
10      rr:class bsbm:Product;
11      rr:template "http://www.example.com/Product/{nr}";
12    ];
13
14    rr:predicateObjectMap [
15      rr:predicateMap [ rr:constant rdfs:label ];
16      rr:objectMap [ rr:column "label" ];
17    ];
18
19    rr:predicateObjectMap [
20      rr:predicateMap [ rr:constant rdfs:comment ];
21      rr:objectMap [ rr:column "comment" ];
22    ];
23
24    rr:predicateObjectMap [
25      rr:predicateMap [ rr:constant bsbm:propText4 ];
26      rr:objectMap [ rr:column "propertyTextual4" ];
27    ];
28
29
30    rr:predicateObjectMap [
31      rr:predicateMap [ rr:constant bsbm:hasOffer ];
32      rr:objectMap [
33        rr:parentTriplesMap <TriplesMapOffer>;
34        rr:joinCondition [ rr:child "nr" ; rr:parent "product" ; ]
35      ];
36    ];
37  ].
38
39  <TMOffer> a rr:TriplesMap;
40    rr:logicalTable [ rr:tableName "Offer" ];
41
42    rr:subjectMap [ a rr:Subject;
43      rr:class bsbm:Offer;
44      rr:template "http://www.example.com/Offer/{nr}";
45    ];
46
47    rr:predicateObjectMap [
48      rr:predicateMap [ rr:constant rdfs:label ]; rr:objectMap [ rr:column "label" ];
49    ];
50
51    rr:predicateObjectMap [
52      rr:predicateMap [ rr:constant bsbm:price ]; rr:objectMap [ rr:column "price" ];
53    ];
54
55    rr:predicateObjectMap [
56      rr:predicateMap [ rr:constant bsbm:deliveryDays ]; rr:objectMap [ rr:column "deliveryDays" ];
57    ];
58
59    rr:predicateObjectMap [
60      rr:predicateMap [ rr:constant bsbm:hasProduct ];
61      rr:objectMap [ rr:template "http://www.example.com/Product/{product}" ];
62    ];
63  ].

```

LISTING 4.1: An R2RML Mapping Document for Tables **Product** and **Offer**.

'rdfs:label' AS iri_rdfs_label, label AS var_lbl FROM Product WHERE nr = 1 AND label IS NOT NULL.

```

1 {
2   :Product/1 rdfs:label ?lbl . #tp1
3   :Product/1 bsbm:hasOffer :Offer/3847 . #tp2
4   OPTIONAL { :Product/1 bsbm:propText4 ?pt4 . } #tp3
5   OPTIONAL { :Offer/3847 bsbm:price ?offerPrice . } #tp4
6   OPTIONAL { :Offer/3847 bsbm:deliveryDays ?dd . } #tp5
7 }

```

LISTING 4.2: Example of SPARQL graph pattern

Next, we describe how we extend the translation function of a triple pattern in Chebotko's approach taking into account R2RML mappings. We then go into more detail explaining the mappings/functions used in the translation algorithm¹.

4.1.2 Function *trans* for translating SPARQL graph patterns

Definition 4.1. (Function *trans*) Given a set of all possible SPARQL graph patterns \mathbf{P} , a set of possible R2RML mapping documents \mathbf{M} , and a set of SQL queries \mathbf{Q} , function $trans : \mathbf{P} \times \mathbf{M} \rightarrow \mathbf{Q}$ takes a graph pattern $P \in \mathbf{P}$ and an R2RML mapping document $\mathcal{M} \in \mathbf{M}$ and generates an SQL query $Q \in \mathbf{Q}$ which can be executed by the underlying RDBMS to return the answer for P .

The algorithm for function *trans* can be seen in Listing 4.3. The next section describes the behavior of *trans* when it receives a triple pattern. The translation function *trans* for other patterns (AND, OPTIONAL, UNION, FILTER) are described in [CLF09].

Example 4.2. Consider the graph pattern *gp* as seen in Listing 4.2. Given the R2RML mapping document \mathcal{M} defined in Listing 4.1, the result of translating the SPARQL graph pattern *gp* can be seen in Listing 4.4.

4.1.3 Function *trans* for translating triple patterns

The algorithm (see Listing 4.5) for translating a triple pattern $TP \in \mathbf{TP}$ where \mathbf{TP} is a set of possible triple patterns in the form of $(\mathbf{IV}) \times (\mathbf{IV}) \times (\mathbf{IVL})$:

- A triple pattern can be mapped to several triples maps in the mapping document, thus all results from the possible mappings will be put in a single UNION query (line 1-10).
- If the predicate part is unbound (that is, having a variable as its value), then the triple pattern will be grounded according to available mappings, and results will

¹While the order of presenting the mappings/functions is not important, we start by explaining *trans(tp)*, so as to facilitate understanding.


```

1 trans(P, M, Q) :-
2   isTriplePattern(P), ...
3   (see Listing 4.3)
4
5 trans(P, M, Q) :-
6   andPattern(GP1, GP2, P),
7   transAND(GP1, GP2, M, Q).
8
9 trans(P, M, Q) :-
10  optPattern(GP1, GP2, P),
11  transOPT(GP1, GP2, M, Q).
12
13 trans(P, M, Q) :-
14  unionPattern(GP1, GP2, P),
15  transUNION(GP1, GP2, M, Q).
16
17 trans(P, M, Q) :-
18  filterPattern(GP, Expr, P),
19  transFILTER(GP, Expr, M, Q).
20
21 transAND(GP1, GP2, M, Q) :-
22  termsA(GP1, GP2, TermsA),
23  termsB(GP1, GP2, TermsB),
24  termsC(GP1, GP2, TermsC),
25  trans(GP1, M, Query1),
26  addAlias(Query1, R1),
27  trans(GP2, M, Query2),
28  addAlias(Query2, R2),
29  buildSelect(TermsA, TermsB, TermsC, R1, R2, SelectItems),
30  buildFrom(Query1, FromItem),
31  buildOn(TermsC, R1, R2, OnExpression),
32  buildJoin(innerjoin, Query2, OnExpression, JoinItem),
33  sqlQuery(SelectItems, FromItem, JoinItem, [], Q).
34
35 transOPT(GP1, GP2, M, Q) :-
36  termsA(GP1, GP2, TermsA),
37  termsB(GP1, GP2, TermsB),
38  termsC(GP1, GP2, TermsC),
39  trans(GP1, M, Query1),
40  addAlias(Query1, R1),
41  trans(GP2, M, Query2),
42  addAlias(Query2, R2),
43  buildSelect(TermsA, TermsB, TermsC, R1, R2, SelectItems),
44  buildFrom(Query1, FromItem),
45  buildOn(TermsC, R1, R2, OnExpression),
46  buildJoin(leftouterjoin, Query2, OnExpression, JoinItem),
47  sqlQuery(SelectItems, FromItem, JoinItem, [], Query).
48
49 transUNION(GP1, GP2, M, Q) :-
50  termsA(GP1, GP2, TermsA),
51  termsB(GP1, GP2, TermsB),
52  termsC(GP1, GP2, TermsC),
53  trans(GP1, M, Query1),
54  addAlias(Query1, R1),
55  trans(GP2, M, Query2),
56  addAlias(Query2, R2),
57  buildSelect(TermsA, TermsB, TermsC, R1, R2, SelectItemsR1R2),
58  buildFrom(Query1, FromItem1),
59  buildJoin(leftouterjoin, Query2, false, JoinItem2),
60  sqlQuery(SelectItemsR1R2, FromItem1, JoinItem2, [], QueryR1R2).
61  trans(GP2, M, Query3),
62  addAlias(Query3, R3),
63  trans(GP1, M, Query4),
64  addAlias(Query4, R4),
65  buildSelect(TermsA, TermsB, TermsC, R3, R4, SelectItemsR3R4),
66  buildFrom(Query3, FromItem3),
67  buildJoin(leftouterjoin, Query4, false, JoinItem4),
68  sqlQuery(SelectItemsR3R4, FromItem3, JoinItem4, [], QueryR3R4),
69  genSQLExpr(union, [QueryR1R2, QueryR3R4], Query).
70
71 transFILTER(GP, Expr, M, Q) :-
72  trans(GP, M, Q1),
73  addAlias(Q1, R1),
74  transExpr(Expr, WhereExpr),
75  sqlQuery(*, Q1, [], WhereExpr, Q).

```

LISTING 4.3: Function *trans* for translating SPARQL graph pattern

be merged as a UNION query (line 12-21). This ensures that all the mappings/-functions (α , β , *genCondSQL* and *genPRSQL*) that we will describe next only accept an IRI as the value of the triple pattern's predicate.

- (line 23-27) In the base case where the triples map is defined and the predicate of the triple pattern is bounded, *transTP* builds a SQL query using the mapping α

```

1 SELECT g3.iri_Product1, g3.iri_rdfsLabel, g3.var_lbl
2   , g3.iri_bsbmHasOffer, g3.iri_Offer3847
3   , g3.iri_bsbmPropText4, g3.var_propText4
4   , g3.iri_bsbmPrice, g3.var_offerPrice
5   , t5.iri_bsbmDeliveryDays, t5.var_dd
6 FROM (
7   SELECT g2.iri_Product1, g2.iri_rdfsLabel, g2.var_lbl
8     , g2.iri_bsbmHasOffer, g2.iri_Offer3847
9     , g2.iri_bsbmPropText4, g2.var_propText4
10    , t4.iri_bsbmPrice, t4.var_offerPrice
11 FROM (
12   SELECT g1.iri_Product1, g1.iri_rdfsLabel, g1.var_lbl
13     , g1.iri_bsbmHasOffer, g1.iri_Offer3847
14     , t3.iri_bsbmPropText4, t3.var_propText4
15 FROM (
16   SELECT t1.iri_Product1, t1.iri_rdfsLabel, t1.var_lbl
17     , t2.iri_bsbmHasOffer, t2.iri_Offer3847
18 FROM (
19   SELECT Product.nr AS iri_Product1, rdfs:label AS iri_rdfsLabel, Product.label AS var_lbl
20   FROM Product
21   WHERE Product.nr = 1 AND Product.label IS NOT NULL
22 ) t1 -- translation result of tp1
23 INNER JOIN (
24   SELECT Product.nr AS iri_Product1, bsbm:hasOffer AS iri_bsbmHasOffer, Offer.nr AS iri_Offer3847
25   FROM Product, Offer
26   WHERE Product.nr = 1 AND Offer.nr = 3847 AND Product.nr = Offer.product
27 ) t2 -- translation result of tp2
28 ON transTP1.iri_Product1 = transTP2.iri_Product1
29 ) g1 -- translation result of tp1 AND tp2 } OPT tp3
30 LEFT OUTER JOIN (
31   SELECT Product.nr AS iri_Product1, bsbm:propText4 AS iri_bsbm_propText4, Product.propertyTextual4 AS var_pt4
32   FROM Product
33   WHERE Product.nr = 1 AND Product.propertyTextual4 IS NOT NULL
34 ) t3 -- translation result of tp3
35 ON g1.iri_Product1 = t3.iri_Product1
36 ) g2 -- translation result of { tp1 AND tp2 } OPT tp3
37 LEFT OUTER JOIN (
38   SELECT Offer.nr AS iri_Offer3847, bsbm:price AS iri_bsbmPrice, Offer.price AS var_offerPrice
39   FROM Offer
40   WHERE Offer.nr = 3847 AND Offer.price IS NOT NULL
41 ) t4 -- translation result of tp4
42 ON g2.iri_Offer3847 = t4.iri_Offer3847
43 ) g3 -- translation result of { tp1 AND tp2 } OPT tp3 OPT t4
44 LEFT OUTER JOIN (
45   SELECT Offer.nr AS iri_Offer3847, bsbm:deliveryDays AS iri_bsbmDeliveryDays, Offer.deliveryDays AS var_dd
46   FROM Offer
47   WHERE Offer.nr = 3847 AND Offer.deliveryDays IS NOT NULL
48 ) t5 -- translation result of tp5
49 ON g3.iri_Offer3847 = t5.iri_Offer3847;

```

LISTING 4.4: Translation result of a SPARQL graph pattern in Listing 4.2

(Section 4.1.4), mapping β (Section 4.1.5), function *genCondSQL* (Section 4.1.6), and function *genPRSQL* (Section 4.1.7).

Example 4.3. Consider the triple patterns in Listing 4.2. Given an R2RML mapping document \mathcal{M} defined in Listing 4.1, the SQL queries resulting from the translation results of these triple patterns are:

- $trans(tp1, \mathcal{M}) = SELECT Product.nr AS iri_Product1, 'rdfs:label' AS iri_rdfsLabel, Product.label AS var_lbl FROM Product WHERE Product.nr = 1 AND Product.label IS NOT NULL.$
- $trans(tp2, \mathcal{M}) = SELECT Product.nr AS iri_Product1, 'bsbm:hasOffer' AS iri_bsbmHasOffer, Offer.nr AS iri_Offer3847 FROM Product, Offer WHERE Product.nr = 1 AND Offer.nr = 3847 AND Product.nr = Offer.product.$
- $trans(tp3, \mathcal{M}) = SELECT Product.nr AS iri_Product1, 'bsbm:propText4' AS iri_bsbm_propText4, Product.propertyTextual4 AS var_pt4$

```

1 trans(TP, M, Queries) :-
2   isTriplePattern(TP),
3   triplesMaps(TP, M, TMList),
4   transTMList(TP, TMList, Queries).
5
6 transTMList(TP, [], []).
7 transTMList(TP, [TMHead | TMTail], Q) :-
8   transTM(TP, TMHead, QHead),
9   transTMList(TP, TMTail, QTail),
10  genSQLExpr(union, [QHead, QTail], Q).
11
12 transTM(TP, TM, Q) :-
13   groundPredicate(TP.Predicate, TM, PreURIList),
14   groundTriple(TP, PreURIList, TPList),
15   transTPList(TPList, TM, Q).
16
17 transTPList([], TM, []).
18 transTPList([TPHead \mid TPTail], TM, Q) :-
19   transTP(TPHead, TM, QHead),
20   transTPList(TPTail, TM, QTail),
21   genSQLExpr(union, [QHead, QTail], Q).
22
23 transTP(TP, TM, Query) :-
24    $\psi$ (TP, TM, PRSQLResult),
25    $\alpha$ (TP, TM, AlphaResult),
26    $\varphi$ (TP, TM, CondSQLResult),
27   sqlQuery(PRSQLResult, AlphaResult, CondSQLResult, Query).

```

LISTING 4.5: Function *trans* for translating triple patterns

FROM Product WHERE Product.nr = 1 AND Product.propertyTextual4 IS NOT NULL.

- *trans(tp4, M) = SELECT Offer.nr AS iri_Offer3847, 'bsbm:price' AS iri_bsbmPrice, Offer.price AS var_offerPrice FROM Offer WHERE Offer.nr = 3847 AND Offer.price IS NOT NULL.*
- *trans(tp5, M) = SELECT Offer.nr AS iri_Offer3847, 'bsbm:deliveryDays' AS iri_bsbmdeliveryDays, Offer.deliveryDays AS var_dd FROM Offer WHERE Offer.nr = 3847 AND Offer.deliveryDays IS NOT NULL.*

In the next sections, we will see more detail the mappings α , mapping β , function φ and function ψ .

4.1.4 Mapping *alpha* for triple patterns

Definition 4.2. (Mapping α) Given a set of all possible bounded predicate triple patterns $\mathbf{TP}^{\mathbf{bp}} = (\mathbf{IV}) \times (\mathbf{I}) \times (\mathbf{IVL})^2$, a set of relations \mathbf{REL} , and a set of all possible triples maps \mathbf{TM} defined in R2RML, a mapping α is a many-to-one mapping $\alpha : \mathbf{TP} \times \mathbf{TM} \rightarrow \mathbf{REL}$, if given a triple pattern $TP \in \mathbf{TP}^{\mathbf{bp}}$ and a triples map $TM \in \mathbf{TM}$,

²For compactness, we write IRI as **I**, Variable as **V**, Literal as **L**, and combinations of those letters.

$\alpha(TP, TM)$ is a set of relations $\mathbf{R} \subseteq \mathbf{REL}$ in which all the triples that may match TP are stored.

```

1 |  $\alpha(TP, TM, [\text{AlphaSub} \mid \text{AlphaPreObj}]) :-$ 
2 |    $\alpha_{sub}^{tp}(TM, \text{AlphaSub}),$ 
3 |    $\alpha_{preobj}^{tp}(TP, TM, \text{AlphaPreObj}).$ 
4 |
5 |  $\alpha_{sub}^{tp}(TM, TN) :-$ 
6 |   logicalTable(TM, LT),
7 |   tableName(LT, TN).
8 |
9 |  $\alpha_{preobj}^{tp}(TP, TM, \text{AlphaPreObj}) :-$ 
10 |   predicateObjectMap(TM, TP.Predicate, POMap),
11 |   referenceObjectMap(POMap, ROMap),
12 |   parentTriplesMap(ROMap, ParentTM),
13 |   logicalTable(ParentTM, AlphaPreObj).
14 |
15 |  $\alpha_{preobj}^{tp}(TP, TM, []) :-$ 
16 |   predicateObjectMap(TM, TP.Predicate, POMap),
17 |   objectMap(POMap, OMap).

```

LISTING 4.6: Mapping α for triple patterns

The algorithm for computing mapping α under a set of R2RML mappings is provided in Listing 4.6. The output of this algorithm is used as the FROM part of the generated SQL query. This algorithm is divided into two main parts; calculating α for the subject (α_{sub}^{tp}) (line 5-7), and for the predicate-object part α_{preobj}^{tp} (line 9-17).

- (line 5-7) The function α_{sub}^{tp} returns the logical table property *logicalTable* of a triples map TM .
- (line 9-13) A logical table from a triple may be joined with another logical table through the `refObjectMap` property. This case is handled by an auxiliary function α_{preobj}^{tp} that returns the parent logical table of the `refObjectMap` property.
- (line 15-17) If an object map is specified instead of a reference object map, then no additional table is added.

The output of mapping α is a set of logical tables with the result from the two auxiliary functions (line 1-3).

Example 4.4. Given the R2RML mapping document \mathcal{M} in Listing 4.1 and the SPARQL graph pattern gp in Listing 4.2, Table 4.1 presents the results of computing mapping α for each of the triple patterns in gp .

TP	TM	α_{sub}^{tp}	α_{preobj}^{tp}	$\alpha(TP, TM)$
:Product/1 rdfs:label ?lbl	TMProduct	{PRODUCT}	{ }	{PRODUCT}
:Product/1 bsbm:hasOffer :Offer/3847	TMProduct	{PRODUCT}	{OFFER}	{PRODUCT, OFFER}
:Product/1 bsbm:propText4 ?pt4	TMProduct	{PRODUCT}	{ }	{PRODUCT}
:Offer/3847 bsbm:price ?offerPrice	TMOffer	{OFFER}	{ }	{OFFER}
:Offer/3847 bsbm:deliveryDays ?dd	TMOffer	{OFFER}	{ }	{OFFER}

TABLE 4.1: Example of computing mapping α for our running example.

4.1.5 Mapping β for triple patterns

Definition 4.3. (Mapping β) Given a set of all possible bounded predicate triple patterns $\mathbf{TP}^{bp} = (\mathbf{IV}) \times (\mathbf{I}) \times (\mathbf{IVL})$, a set of positions in a triple pattern $\mathbf{POS} = \{sub, pre, obj\}$, a set of relational attributes \mathbf{ATTR} , and a set of all possible R2RML triple maps \mathbf{TM} , a mapping β is a many-to-one mapping $\beta : \mathbf{TP}^{bp} \times \mathbf{POS} \times \mathbf{TM} \rightarrow \mathbf{ATTR}$, if given a triple pattern $TP \in \mathbf{TP}^{bp}$, a position $pos \in POS$, and an R2RML triples map $TM \in \mathbf{TM}$, $\beta(TP, pos, TM)$ is a relational attribute $ATTR \in \mathbf{ATTR}$ whose value may match TP at position pos .

```

1   $\beta(TP, sub, TM, BetaSub) :-$ 
2    subjectMap(TM, SM),
3    referencedColumn(SM, BetaSub).
4
5   $\beta(TP, pre, TM, TP.Predicate).$ 
6
7   $\beta(TP, obj, TM, BetaObj) :-$ 
8    predicateObjectMap(TM, TP.Predicate, POMap),
9     $\beta_{obj}^{tp}(POMap, BetaObj).$ 
10
11  $\beta_{obj}^{tp}(POMap, BetaOM) :-$ 
12   objectMap(POM, OMap),
13   referencedColumn(OMap, BetaOM).
14
15  $\beta_{obj}^{tp}(POMap, BetaROM) :-$ 
16   refObjectMap(POMap, ROMap),
17   parentSubjectMap(ROMap, ParentTM),
18   subjectMap(ParentTM, ParentSM),
19   referencedColumn(ParentSM, BetaROM).
```

LISTING 4.7: Mapping β for triple patterns

The algorithm for computing the mapping β under M is provided in Listing 4.7 and some examples of β results can be seen on Table 4.2. The output of mapping β is used in the functions *genPRSQL* and *genCondSQL* for selecting the relational attribute corresponding to the triple pattern position.

- (line 1-3) If the position pos is subject, then β returns the corresponding relational attributes defined in the subject map SM of *TriplesMap*.
- (line 5) If the position pos is predicate, then the IRI of the predicate is returned.

- (line 7-19) If the position pos is object, then the function β_{obj} checks whether the argument $POMap$ contains a Reference Object Map $ROMap$.
 - (line 11-13) If $POMap$ contains an object map $objectMap$, the corresponding relational attributes of $objectMap$ are returned.
 - (line 15-19) If $POMap$ contains a reference object map $ROMap$, the parent triple map $ParentTriplesMap$ of the $ROMap$ is retrieved and the relational attributes corresponding to the subject map of $ParentTriplesMap$ are returned.

Example 4.5. Consider the R2RML mapping document \mathcal{M} in Listing 4.1 and the SPARQL graph pattern gp in Listing 4.2. The result of calculating β for each of the triple patterns in gp can be seen in Table 4.2.

TP	TriplesMap	$\beta(TP, sub, TM)$	$\beta(TP, pre, TM)$	$\beta(TP, obj, TM)$
:Product/1 rdfs:label ?lbl	TMProduct	{Product.nr}	{ 'rdfs:label' }	{Product.label}
:Product/1 bsbm:hasOffer :Offer/3847	TMProduct	{Product.nr}	{ 'bsbm:hasOffer' }	{Offer.nr}
:Product/1 bsbm:propText4 ?pt4	TMProduct	{Product.nr}	{ 'bsbm:propText4' }	{Product.propertyTextual4}
:Offer/3847 bsbm:price ?offerPrice	TMOffer	{Offer.nr}	{ 'bsbm:price' }	{Offer.price}
:Offer/3847 bsbm:deliveryDays ?dd	TMOffer	{Offer.nr}	{ 'bsbm:deliveryDays' }	{Offer.deliveryDays}

TABLE 4.2: Examples of computing mapping β for our running example

4.1.6 Function $genCondSQL$ for triple patterns

Definition 4.4. (Function φ) Given a set of all possible triple patterns $\mathbf{TP}^{bp} = (\mathbf{IV}) \times (\mathbf{I}) \times (\mathbf{IVL})$ and a set of all possible R2RML triples maps \mathbf{TM} , φ takes a triple pattern $TP \in \mathbf{TP}^{bp}$ and an R2RML triples map $TM \in \mathbf{TM}$ and generates an SQL boolean expression that is evaluated to true if and only if the tuple $(TP.subject, TP.predicate, TP.object)$ matches a tuple represented by relational attributes $(\beta(TP, sub, TM), \beta(TP, pre, TM), \beta(TP, obj, TM))$.

The algorithm of function φ is provided in Listing 4.8. This function calls and returns the results of its auxiliary functions: φ_{sub}^{tp} (line 2), φ_{preobj}^{tp} (line 3, 7-10), and φ_{tp}^{tp} (line 4). The output of this function is used as the WHERE part of the generated SQL query.

4.1.6.1 Function $genCondSQL_{sub}^{tp}$

Given a triple pattern TP and an R2RML triples map TM , function φ_{sub}^{tp} (Listing 4.9) generates the SQL conditional expression that matches with the subject of the triple pattern. First, this function obtains the subject map SM of TM (line 2) and calculates the β value of that triple pattern for the sub position (line 3). Then, it calculates the

```

1 |  $\varphi(\text{TP}, \text{TM}, \text{CondSQL}) :-$ 
2 |    $\varphi_{sub}^{tp}(\text{TP}, \text{TM}, \text{CondSub}),$ 
3 |    $\varphi_{preobj}^{tp}(\text{TP}, \text{TM}, \text{CondPreObj})$ 
4 |    $\varphi_{tp}^{tp}(\text{TP}, \text{TM}, \text{CondTP}),$ 
5 |    $\text{genSQLExpr}(\text{and}, [\text{CondSub}, \text{CondPreObj}, \text{CondTP}], \text{CondSQL}).$ 
6 |
7 |  $\varphi_{preobj}^{tp}(\text{TP}, \text{TM}, \text{CondPreObj}) :-$ 
8 |    $\varphi_{pre}^{tp}(\text{TP}, \text{TM}, \text{CondPre}),$ 
9 |    $\varphi_{obj}^{tp}(\text{TP}, \text{TM}, \text{CondObj}),$ 
10 |   $\text{genSQLExpr}(\text{and}, [\text{CondPre}, \text{CondObj}], \text{CondPreObj}).$ 

```

LISTING 4.8: Function φ for triple patterns

conditional expression for the triple's pattern subject, depending on whether the subject is a variable or an IRI.

- (line 6-8) If the triple's subject is a variable, then φ_{sub}^{tp} returns the conditional expression for ensuring that the β associated to the subject is not null.
- (line 10-24) If the subject is an IRI then it checks the type of the subject map, whether it is a template map, a column map, or a constant map.
 - (line 10-14) If the subject map is a template map, then the subject's identifier (*SubjectID*) is obtained by a function *inverseExpr*, which takes an IRI and returns only the corresponding values that can be related to database values. For example, *inverseExpression* will return 1 or 3847 for IRIs *ex.com/Product/1* or *ex.com/Offer/3847*, respectively. Then the equality expression is generated for *SubjectID* and the database column associated with the template map.
 - (line 16-19) If it is an IRI and the subject map is a column map, then the equality expression is generated as the value of the column specified in the column map and the triple's subject.
 - (line 21-24) If it is an IRI and the subject map is a constant map, then the equality expression is generated as the value of the constant map and the triple's subject.

4.1.6.2 Function $\text{genCondSQL}_{pre}^{tp}$

Function φ_{pre}^{tp} (Listing 4.10) works in a similar way as function φ_{sub}^{tp} , with the exception of the unbound predicate (when the predicate is a variable), which is handled by the *trans* function described above. So, the only check that is done is the type of the predicate map (whether constant, column, or template) with an IRI as the value of the triple pattern's predicate.

```

1   $\varphi_{sub}^{tp}(TP, TM, CondSub) :-$ 
2    subjectMap(TM, SM),
3     $\beta(TP, sub, TM, BetaSub)$ ,
4     $\varphi_{sub}^{tp}Aux(TP, BetaSub, SM, CondSub)$ .
5
6   $\varphi_{sub}^{tp}Aux(TP, BetaSub, SM, CondSub) :-$ 
7    var(TP.subject),
8    genSQLExpr(isNotNull, [BetaSub], CondSub).
9
10  $\varphi_{sub}^{tp}Aux(TP, BetaSub, SM, CondSub) :-$ 
11   iri (TP.subject),
12   templateMap(SM, SMTemplateMap),
13   inverseExpr(TPSubject, SMTemplate, TPSubjectID),
14   genSQLExpr(equals, [BetaSub, TPSubjectID, CondSub]).
15
16  $\varphi_{sub}^{tp}Aux(TP, BetaSub, SM, CondSub) :-$ 
17   iri (TP.subject),
18   columnMap(SM, SMColumnMap),
19   genSQLExpr(equals, [BetaSub, TPSubject], CondSub).
20
21  $\varphi_{sub}^{tp}Aux(TP, BetaSub, SM, CondSub) :-$ 
22   iri (TP.subject),
23   constantMap(SM, SMConstantMap),
24   genSQLExpr(equals, [BetaSub, TPSubject], CondSub).

```

LISTING 4.9: φ_{sub}^{tp}

```

1   $\varphi_{pre}^{tp}(TP, TM, CondPre) :-$ 
2    predicateObjectMap(TM, TP.predicate, PMap),
3    predicateMap(PMap, PMap),
4     $\beta(TP, pre, TM, BetaPre)$ ,
5     $\varphi_{pre}^{tp}Aux(TP, PMap, BetaPre, CondPre)$ .
6
7   $\varphi_{pre}^{tp}Aux(TP, PMap, BetaPre, CondPre) :-$ 
8    iri (TP.predicate),
9    templateMap(PMap, TemplateMap),
10   inverseExpr(TPPredicate, TemplateMap, TPPredicateID),
11   genSQLExpr(equals, TPPredicateID, BetaPre, CondPre).
12
13  $\varphi_{pre}^{tp}Aux(TPPredicate, PMap, BetaPre, CondPre) :-$ 
14   iri (TP.predicate),
15   columnMap(PMap, ColumnMap),
16   genSQLExpr(equals, TPPredicate, BetaPre, CondPre).
17
18  $\varphi_{pre}^{tp}Aux(TPPredicate, PMap, BetaPre, CondPre) :-$ 
19   iri (TP.predicate),
20   constantMap(PMap, ConstantMap),
21   genSQLExpr(equals, TPPredicate, BetaPre, CondPre).

```

LISTING 4.10: φ_{pre}^{tp}


```

1   $\varphi_{obj}^{tp}(TP, TM, CondObj) :-$ 
2    predicateObjectMap(TM, TP.predicate, POM),
3    objectMap(POMap, OM),
4     $\beta(TP, obj, TM, BetaObj)$ ,
5     $\varphi_{obj}^{tp}OM(TP, BetaObj, OM, CondObj)$ .
6
7   $\varphi_{obj}^{tp}(TP, TM, CondObj) :-$ 
8    predicateObjectMap(TM, TP.predicate, POM),
9    refObjectMap(POM, ROM),
10    $\beta(TP, obj, TM, BetaObj)$ ,
11   joinCondition(ROM, JoinCondition),
12   joinConditionChild(JoinCondition, ChildColumn),
13   joinConditionParent(JoinCondition, ParentColumn),
14   genSQLExpr(equals, [ChildColumn, ParentColumn], CondJoin),
15    $\varphi_{obj}^{tp}ROM(TPObject, BetaObj, ROM, CondObj2)$ .
16   genSQLExpr(and, [CondObj1, CondObj2], CondObj).
17
18   $\varphi_{obj}^{tp}OM(TP, BetaObj, OM, Cond) :-$ 
19    var(TP.object),
20    genSQLExpr(isNotNull, [BetaObj], Cond).
21
22   $\varphi_{obj}^{tp}OM(TP, BetaObj, OM, Cond) :-$ 
23    iriOrLiteral(TP.object),
24    templateMap(OM, OMTemplateMap),
25    inverseExpr(OMTemplateMap, TP.object, TPObjID),
26    genSQLExpr(equals, [BetaObj, TPObjID], Cond).
27
28   $\varphi_{obj}^{tp}OM(TP, BetaObj, OM, Cond) :-$ 
29    iriOrLiteral(TP.object),
30    constantOrColumnMap(OM),
31    genSQLExpr(equals, [BetaObj, TPObjID], Cond).
32
33   $\varphi_{obj}^{tp}ROM(TP, ROM, []) :-$ 
34    var(TP.object).
35
36   $\varphi_{obj}^{tp}ROM(TP, ROM, Cond) :-$ 
37    iri(TP.object),
38    parentTriplesMap(ROM, ParentTM),
39     $\beta(TP, sub, ParentTM, ParentBetaSub)$ ,
40    subjectMap(ParentTM, ParentSM),
41    constantOrColumnMap(ParentSM),
42    genSQLExpr(equals, [TP.object, ParentBetaSub], Cond).
43
44   $\varphi_{obj}^{tp}ROM(TP, ROM, Cond) :-$ 
45    iri(TP.object),
46    parentTriplesMap(ROM, ParentTM),
47     $\beta(TP, sub, ParentTM, ParentBetaSub)$ ,
48    subjectMap(ParentTM, ParentSM),
49    templateMap(ParentSM, ParentSMTemplateMap),
50    inverseExpr(ParentSMTemplateMap, TP.object, TPObjID),
51    genSQLExpr(equals, [ObjId, ParentBetaSub], Cond).

```

LISTING 4.11: φ_{obj}^{tp}

4.1.6.3 Function $genCondSQL_{obj}^{tp}$

Given a triple pattern TP and an R2RML triples map TM , function φ_{obj}^m (Listing 4.11) generates the SQL filter that matches the triple pattern's object. First, this function obtains the predicate object map POM of the triples map TM associated to the predicate's URI, and it checks whether the corresponding predicate object map contains an object map or a reference object map.

If an ObjectMap is defined (line 1-5, 18-31), then the β value for the *obj* position is calculated and the function checks the type of the triple pattern's predicate, whether it is a variable, an IRI, or a literal.

- (line 18-20) If the predicate is a variable, then the SQL expression stating that the β value IS NOT NULL is returned. The reason for this is that the R2RML specification states that the result of a term map is empty if it has a NULL value.
- (line 22-31) If the predicate is an IRI or a literal, two cases need to be evaluated, depending on whether the object map is a template map, a constant map, or a column map.
 - (line 22-26) If the ObjectMap is a template map, first *OBJID* as the identifier of *tp.object* is evaluated using the inverse expression. The function returns an SQL equality expression of *OBJID* and β_{obj}^m mapping of *POMap* (line 38-41).
 - (line 28-31) If the ObjectMap is either a constant map or a column map, then a SQL conditional expression is generated that states that the β associated to the ObjectMap is equal to the triple pattern's object.

If a RefObjectMap is defined (line 7-16), first, then a SQL expression that represents the *JoinCondition* property from *RefObjectMap* is obtained and a SQL expression for joining the child and parent specified in the *JoinCondition* is generated (line 14). This function then calls $\varphi_{obj}^{tp}ROM$ that checks the type of the triple's pattern object.

- (line 33-34) no additional expression is generated when the triple pattern's object is a variable.
- (line 36-51) If it is an IRI, then the subject map *ParentSM* of the parent triples map is extracted, and betaSub for *ParentSM* is calculated.
 - (line 36-42) If *ParentSM* is a type of either constant or column map, then the SQL equality condition for betaSub and the value of triple's pattern object is generated.
 - (line 44-51) If *ParentSM* is a type of template map, then the object identifier *TPObjID* is obtained via the function *inverseExpr* and a SQL equality expression for betaSub and *TPObjID*.

```

1 strongMatchedMapping(TermMap1, TermMap2) :-
2   isConstantMap(TermMap1),
3   isConstantMap(TermMap2).
4
5 strongMatchedMapping(TermMap1, TermMap2) :-
6   isConstantMap(TermMap1),
7   isColumnMap(TermMap2).
8
9 strongMatchedMapping(TermMap1, TermMap2) :-
10  isConstantMap(TermMap1),
11  isTemplateMap(TermMap2).
12
13 strongMatchedMapping(TermMap1, TermMap2) :-
14  isColumnMap(TermMap1),
15  isConstantMap(TermMap2).
16
17 strongMatchedMapping(TermMap1, TermMap2) :-
18  isColumnMap(TermMap1),
19  isColumnMap(TermMap2).
20
21 strongMatchedMapping(TermMap1, TermMap2) :-
22  isColumnMap(TermMap1),
23  isTemplateMap(TermMap2).
24
25 strongMatchedMapping(TermMap1, TermMap2) :-
26  isTemplateMap(TermMap1),
27  isConstantMap(TermMap2).
28
29 strongMatchedMapping(TermMap1, TermMap2) :-
30  isTemplateMap(TermMap1),
31  isColumnMap(TermMap2).
32
33 strongMatchedMapping(TermMap1, TermMap2) :-
34  isTemplateMap(TermMap1),
35  isTemplateMap(TermMap2),
36  templateMap(Base1, Col1, TermMap1),
37  templateMap(Base2, Col2, TermMap2),
38  equals(Base1, Base2).

```

LISTING 4.12: *strongMatchedMappings*

4.1.6.4 Function $genCondSQL_{tp}^{tp}$

Before we describe the function φ_{tp}^{tp} , we start with the notion of strongly matched mappings, which can be seen in Listing 4.12. Intuitively, this notion states that for two template mappings, they can be compared only and only if the those mappings share the same base URI. Constant and column mappings can be compared to any other mapping type. The notion of strongly matched mappings can be seen in Listing 4.12.

Function φ_{tp}^{tp} ensures that:

- (Listing 4.13 line 2, Listing 4.19 line 1-4) If $TP.subject = TP.predicate$ and the subject map strongly matches with the predicate map, then $\beta(TP, sub, TM) = \beta(TP, pre, TM)$.
- (Listing 4.13 line 3, Listing 4.19 line 6-14) If $TP.subject = TP.object$ and the subject map strongly matches with the object map, then $\beta(TP, sub, TM) = \beta(TP, obj, TM)$.

```

1  $\varphi_{tp}^{tp}(TP, TM, AuxSPO) :-$ 
2    $\varphi_{2tp}^{2tp}(TP, sub, TP, pre, AuxSP),$ 
3    $\varphi_{2tp}^{2tp}(TP, sub, TP, obj, AuxSO),$ 
4    $\varphi_{2tp}^{2tp}(TP, pre, TP, obj, AuxPO),$ 
5    $genSQLExpr(and, [AuxSP, AuxSO, AuxPO], AuxSPO).$ 

```

LISTING 4.13: φ_{tp}^{tp}

- (Listing 4.13 line 4, Listing 4.19 line 26-34) If $TP.predicate = TP.object$ and the predicate map strongly matches with the object map, then $\beta(TP, pre, TM) = \beta(TP, obj, TM)$.

Example 4.6. Consider the R2RML mapping document \mathcal{M} in Listing 4.1 and the SPARQL graph pattern in Listing 4.2. The result of computing function φ can be seen in Table 4.3.

TP	TriplesMap	φ_{sub}^{tp}	φ_{preobj}^{tp}	φ_{tp}^{tp}
:Product/1 rdfs:label ?lbl	TMProduct	Product.nr = 1	Product.label IS NOT NULL	-
:Product/1 bsbm:hasOffer :Offer/3847	TMProduct	Product.nr = 1	Product.nr = Offer.product AND Offer.product=3847	-
:Product/1 bsbm:propText4 ?pt4	TMProduct	Product.nr = 1	Product.propertyTextual4 IS NOT NULL	-
:Offer/3847 bsbm:price ?offerPrice	TMOffer	Offer.nr = 3847	Offer.price IS NOT NULL	-
:Offer/3847 bsbm:deliveryDays ?dd	TMOffer	Offer.nr = 3847	Offer.deliveryDays IS NOT NULL	-

TABLE 4.3: Examples of function φ in our running example.

4.1.7 Function $genPRSQL$ for triple patterns

Definition 4.5. (Function ψ) Given a set of all possible triple patterns $\mathbf{TP}^{bp} = (\mathbf{IV}) \times (\mathbf{I}) \times (\mathbf{IVL})$ and a set of all possible R2RML triples maps \mathbf{TM} , ψ takes a triple pattern $TP \in \mathbf{TP}^{bp}$ and a triples map $TM \in \mathbf{TM}$ and generates a SQL expression which projects only those relational attributes that correspond to distinct $TP.subject$, $TP.predicate$, $TP.object$ and renames the projected attributes as:

- $\beta(TP, sub, TM) \rightarrow name(TP.subject)$
- $\beta(TP, pre, TM) \rightarrow name(TP.predicate)$
- $\beta(TP, obj, TM) \rightarrow name(TP.object)$

The algorithm for computing $genPRSQL$ under M is provided in Listing 4.14. This function calls and collects the results from three auxiliary functions: ψ_{sub}^{tp} , ψ_{pre}^{tp} , and ψ_{obj}^{tp} .

- (line 6-9) ψ_{sub}^{tp} projects the β value of the subject as the result of the function $name$.

```

1   $\psi(TP, TM, [PRSQLSub, PRSQLPre, PRSQLObj]) :-$ 
2     $\psi_{sub}^{tp}(TP, TM, PRSQLSub),$ 
3     $\psi_{pre}^{tp}(TP, TM, PRSQLPre),$ 
4     $\psi_{obj}^{tp}(TP, TM, PRSQLObj).$ 
5
6   $\psi_{sub}^{tp}(TP, TM, PRSQLSub) :-$ 
7     $\beta(TP, sub, TM, BetaSub),$ 
8     $name(TP.subject, SubjectAlias),$ 
9     $genSQLExpr(as, [BetaSub, SubjectAlias], PRSQLSub).$ 
10
11  $\psi_{pre}^{tp}(TP, TM, PRSQLPre) :-$ 
12    $notEquals(TP.subject, TP.predicate),$ 
13    $\beta(TP, pre, TM, BetaPre),$ 
14    $name(TPPredicate, PreAlias),$ 
15    $genSQLExpr(as, [BetaPre, PreAlias], PRSQLPre).$ 
16
17  $\psi_{obj}^{tp}(TP, TM, PRSQLObj) :-$ 
18    $notEquals(TP.object, TP.subject),$ 
19    $notEquals(TP.object, TP.predicate),$ 
20    $\beta(TP, obj, TM, BetaObj),$ 
21    $name(TPObject, ObjectAlias),$ 
22    $genSQLExpr(as, [PRSQL, ObjectAlias], PRSQLObj).$ 

```

LISTING 4.14: Function *genPRSQL* for triple patterns

- (line 11-15) If the predicate is not same as the subject, then ψ_{pre}^{tp} projects the β value of the subject as the result of the function *name*.
- (line 17-22) If the object is not same as the subject and the predicate, then ψ_{obj}^{tp} projects the β value of the subject as the result of the function *name*.

The outputs of this function are used in the SELECT part of the generated SQL as the select items of the SQL queries together with its aliases, which are generated by the function *name*. Given a term in **IVL**, the function *name* generates a unique name, such that the generated name conforms to the SQL syntax for relational attribute names. One way to implement the function *name(t)* is to return:

- *var_{t}* if *t* is a variable. For example, the result of *name(?lbl)* is *var_lbl*.
- *iri_{t}* if *t* is an IRI. For example, the result of *name(rdfs : label)* is *iri_rdfsLabel*.
- *{t}* if *t* is a literal. For example, the result of *name('keyboard')* is *'keyboard'*.

Example 4.7. Given the R2RML mapping document in Listing 4.1 and the SPARQL graph pattern in Listing 4.2, the result of calculating function ψ for each triple pattern in the graph pattern can be seen in Table 4.4.

TP	TriplesMap	ψ_{sub}^{tp}	ψ_{pre}^{tp}	ψ_{obj}^{tp}
.Product/1 rdfs:label ?lbl	TMPProduct	{PRODUCT.nr AS iri_Product1}	{'rdfs:label' AS iri_rdfs_label}	{PRODUCT.label AS var_lbl }
.Product/1 bsbm:hasOffer :Offer/3847	TMPProduct	{PRODUCT.nr AS iri_Product1}	{'bsbm:hasOffer' AS iri_bsbm_hasOffer}	{OFFER.nr AS iri_Offer3847 }
.Product/1 bsbm:propText4 ?pt4	TMPProduct	{PRODUCT.nr AS iri_Product1}	{'bsbm:propText4' AS iri_bsbm_propText4}	{PRODUCT.propertyTextual4 AS var_pt4 }
.Offer/3847 bsbm:price ?offerPrice	TMOffer	{OFFER.nr AS iri_Offer3847}	{'bsbm:price' AS iri_bsbm_price}	{OFFER.price AS var_offerPrice }
.Offer/3847 bsbm:deliveryDays ?dd	TMOffer	{OFFER.nr AS iri_Offer3847}	{'bsbm:deliveryDays' AS iri_bsbm_deliveryDays}	{OFFER.deliveryDays AS var_dd }

TABLE 4.4: Examples of function ψ in our running example.

4.2 Optimisations

The previous section described our extension of Chebotko et al.'s approach so as to handle R2RML mappings. In this section, we present some optimisation techniques that can be performed in order to generate more efficient SQL queries.

We start in Section 4.2.1 by describing how the algorithm can be extended by reducing the number of self-joins when the SPARQL queries contain a pattern called Subject Triple Group (STG). In Section 4.2.2 we present an optimisation technique when the SPARQL queries contain a pattern called Subject Triple Group with Optional (OSTG). The proofs for these optimizations are given in Appendix B.

We close this section by presenting other optimizations such as: phantom triple pattern introduction, exploiting database metadata when possible (such as for checking the necessity of IS NOT NULL expression or reordering the tables used in join queries), and union reduction to generate more optimised SQL queries.

4.2.1 Self-join Elimination

A common pattern used in SPARQL queries is a set of triple patterns having the same subject, connected by AND operators. That pattern is also called a *subject triple group* (STG) pattern.

Definition 4.6. A subject triple group (STG) pattern is defined recursively as follows:

- If TP_1 and TP_2 are triple patterns and $TP_1.subject = TP_2.subject$, then $(TP_1 \text{ AND } TP_2)$ is an STG pattern.
- If P_1 is an STG pattern, TP is a triple pattern, and $P_1.subject = TP.subject$, then $(P_1 \text{ AND } TP)$ is an STG pattern.

Using the *trans* algorithm defined previously, the process of translating STG patterns having n triple patterns is defined by recursively calling the function *trans*(AND) of $n - 1$ triple patterns with the n th triple pattern.

Definition 4.7. The function *trans* Subject Triple Group (STG) is defined recursively as:

- $trans(tp_1 \text{ AND } tp_2) = trans(tp_1) \bowtie trans(tp_2)$.
- $trans(\{tp_1 \text{ AND } tp_2 \text{ AND } \dots \text{ AND } tp_{n-1}\} \text{ AND } tp_n) = trans(\{tp_1 \text{ AND } tp_2 \text{ AND } \dots \text{ AND } tp_{n-1}\}) \bowtie trans(tp_n)$.

Example 4.8. One example of an STG pattern is the graph pattern *tp1* AND *tp2* as part of the graph pattern in Listing 4.2 lines 2-3. The result of translating this STG pattern can be seen in line 16-29 of Listing 4.4.

The translation algorithm for STG described above implies that the number of triple patterns in the STG determines the number of joins occurring in the SQL queries. The resulting SQL queries are not necessarily very efficient when evaluated in RDBMS that do not implement optimisations like self-join elimination (e.g., MySQL, PostgreSQL).

Records of tables in an RDB2RDF context are already arranged in tabular fashion. Exploiting this fact, we can design a more optimised version of the algorithm *trans* in cases where the SPARQL query contains STG patterns, so that we do not need to join all tables coming from the triple patterns. The idea is to extend the functions defined in the previous section (α , *genCondSQL*, *genPRSQL*, and *trans*), so that they accept STG patterns and process separately the subject and the predicate-object parts of those triple patterns in the STG patterns. Next, we will elaborate in more detail how the extension of those functions work.

4.2.1.1 Mapping α for STG patterns

Definition 4.8. (Mapping α^{stg}) Given a set of all possible STG patterns **STG**, a set of relations **REL**, and a set of all possible triples maps **TM** defined in an R2RML mapping document, a mapping α^{stg} is a many-to-one mapping $\alpha^{stg} : \mathbf{STG} \times \mathbf{TM} \rightarrow \mathbf{REL}$, if given a subject triple group $STG \in \mathbf{STG}$ and $TM \in \mathbf{TM}$, $\alpha^{stg}(STG, TM)$ is a set of relations $\mathbf{R} \subseteq \mathbf{REL}$ in which all the triples that may match *STG* are stored.

```

1 |  $\alpha^{stg}([TP \mid STGTail], TM, [AlphaSub \mid AlphaPreObj]) :-$ 
2 |    $\alpha_{sub}^{tp}(TM, AlphaSub),$ 
3 |    $\alpha_{preobj}^{stg}([TP \mid STGTail], TM, AlphaPreObj).$ 
4 |
5 |  $\alpha_{preobj}^{stg}([], TM, []).$ 
6 |
7 |  $\alpha_{preobj}^{stg}([TP \mid STGTail], TM, [ResultHead \mid ResultTail]) :-$ 
8 |    $\alpha_{preobj}^{tp}(TP, TM, ResultHead).$ 
9 |    $\alpha_{preobj}^{stg}(STGTail, TM, ResultTail).$ 

```

LISTING 4.15: Mapping α^{stg} for STG patterns

The algorithm for computing mapping α^{stg} is provided in Listing 4.15. The output of this function will be used as the FROM part of the generated SQL query. The function first retrieves the relational source of the corresponding first triples map (line 2). After that, it iterates over all the triple patterns in the STG pattern to get the relational source of its predicate-object (line 3, 5-9).

Example 4.9. Consider an STG pattern $\{tp1 \text{ AND } tp2\}$, where $tp1 = :Product/1 \text{ rdfs:label ?lbl}$ and $tp2 = :Product/1 \text{ bsbm:hasOffer :Offer/3847}$. The result of calculating α^{stg} over this STG pattern can be seen in Table 4.5.

STG	TM	α_{sub}^{tp}	α_{preobj}^{stg}	α^{stg}
$\{ :Product1 \text{ rdfs:label ?lbl. AND } :Product1 \text{ :hasOffer :Offer3847 } \}$	TMProduct	$\{ \text{Product} \}$	$\{ \text{Offer} \}$	$\{ \text{Product, Offer} \}$

TABLE 4.5: An example of how to calculate mapping α^{stg}

4.2.1.2 Function $genCondSQL^{stg}$ for STG patterns

Definition 4.9. (Function φ^{stg}) Given a set of possible STG patterns **STG** and a set of possible R2RML triples maps **TM**, function φ^{stg} takes an STG pattern $STG = \{TP_1 \text{ AND } TP_2 \text{ AND } \dots \text{ AND } TP_n\} \in \mathbf{STG}$ and a triples map $TM \in \mathbf{TM}$ and generates an SQL boolean expression which is evaluated to true if and only if the tuples $(TP_1.subject, TP_1.predicate, \dots, TP_n.predicate, TP_1.object, \dots, TP_n.object)$ matches the tuples represented by relational attributes $(\beta(TP_1, sub, TM), \beta^m(TP_1, pre, TM), \dots, \beta^m(TP_n, pre, TM), \beta^m(TP_1, obj, TM), \dots, \beta^m(TP_n, obj, TM))$.

```

1 |  $\varphi^{stg}([STGHead \mid STGTail], TM, CondSTG) :-$ 
2 |    $\varphi_{sub}^{tp}(STGHead, TM, CondSub),$ 
3 |    $\varphi_{preobj}^{stg}([STGHead \mid STGTail], TM, CondPreObj),$ 
4 |    $\varphi_{tp}^{stg}([STGHead \mid STGTail], TM, Cond1TP),$ 
5 |    $\varphi_{2tp}^{stg}([STGHead \mid STGTail], [STGHead \mid STGTail], TM, Cond2TP),$ 
6 |    $genSQLExpr(\text{and}, [CondSub, CondPreObj, Cond1TP, Cond2TP], CondSTG).$ 
7 |
8 |  $\varphi_{preobj}^{stg}([], TM, []).$ 
9 |
10 |  $\varphi_{preobj}^{stg}([STGHead \mid STGTail], TM, CondPreObj) :-$ 
11 |    $\varphi_{preobj}^{tp}(STGHead, TM, CondHead),$ 
12 |    $\varphi_{preobj}^{stg}(STGTail, TM, CondTail),$ 
13 |    $genSQLExpr(\text{and}, [CondHead, CondTail], CondPreObj).$ 

```

LISTING 4.16: Function φ^{stg} for STG patterns

The algorithm of function φ^{stg} is provided in Listing 4.16. First the subject condition expression for the first triple pattern is obtained by calling the function φ_{sub}^{tp} (line 2). Afterwards, the predicate and object condition expression for each of the triple patterns is obtained (line 3, line 8-13). This function also calls two auxiliary functions: φ_{tp}^{stg} (line 4) and φ_{2tp}^{stg} (line 5), which will be explained in the following. Then all the results are collected as the final result (line 6).

Function φ_{tp}^{stg} ensures that the equality expression for a triple pattern described in Listing 4.13 holds for every triple pattern in the STG pattern.


```

1 |  $\varphi_{tp}^{stg}([], TM, []).$ 
2 |
3 |  $\varphi_{tp}^{stg}([H | T], TM, [ResultH | ResultT]) :-$ 
4 |    $\varphi_{tp}^{tp}(H, TM, ResultH),$ 
5 |    $\varphi_{tp}^{stg}(T, TM, ResultT).$ 

```

LISTING 4.17: Function φ_{tp}^{stg} over a triple pattern.

Function φ_{2tp}^{stg} ensures that for every pair of triple patterns $TP1, TP2 \in STG$, the following conditional expression holds:

- (Listing 4.18 line 7, Listing 4.19 line 1-4) $TP1.subject = TP2.predicate$, then $\beta(TP1, sub, TM) = \beta(TP2, pre, TM)$.
- (Listing 4.18 line 8, Listing 4.19 line 6-14) $TP1.subject = TP2.object$, then $\beta(TP1, sub, TM) = \beta(TP2, obj, TM)$.
- (Listing 4.18 line 9, Listing 4.19 line 16-19) $TP1.predicate = TP2.subject$, then $\beta(TP1, pre, TM) = \beta(TP2, sub, TM)$.
- (Listing 4.18 line 10, Listing 4.19 line 21-25) $TP1.predicate = TP2.predicate$, then $\beta(TP1, pre, TM) = \beta(TP2, pre, TM)$.
- (Listing 4.18 line 11, Listing 4.19 line 27-37) $TP1.predicate = TP2.object$, then $\beta(TP1, pre, TM) = \beta(TP2, obj, TM)$.
- (Listing 4.18 line 12, Listing 4.19 line 39-47) $TP1.object = TP2.subject$, then $\beta(TP1, obj, TM) = \beta(TP2, sub, TM)$.
- (Listing 4.18 line 13, Listing 4.19 line 49-59) $TP1.object = TP2.predicate$, then $\beta(TP1, obj, TM) = \beta(TP2, pre, TM)$.
- (Listing 4.18 line 14, Listing 4.19 line 61-83) $TP1.object = TP2.object$, then $\beta(TP1, obj, TM) = \beta(TP2, obj, TM)$.

Example 4.10. Consider an STG pattern $\{tp1 \text{ AND } tp2\}$, where $tp1 = :Product/1 \text{ rdfs:label ?lbl AND } :Product1 :hasOffer :Offer3847 \}$ and $tp2 = :Product/1 \text{ bsbm:hasOffer :Offer/3847}$. Table 4.6 illustrates the result of calculating φ^{stg} .

STG	TM	φ_{sub}^{tp}	φ_{preobj}^{stg}	φ_{tp}^{stg}	φ_{2tp}^{stg}
$\{ :Product1 \text{ rdfs:label ?lbl AND } :Product1 :hasOffer :Offer3847 \}$	TMPProduct	$\{PRODUCT.nr = 1\}$	$\{PRODUCT.label \text{ IS NOT NULL AND } PRODUCT.nr = OFFER.product \text{ AND } OFFER.product=3847\}$	-	-

TABLE 4.6: An example of how to calculate function φ^{stg}

```

1 |  $\varphi_{2tp}^{stg}([H1 \mid T1], [H2 \mid T2], TM, [ResultH \mid ResultT]) :-$ 
2 |    $\varphi_{2tp}^{tp}(H1, H2, TM, ResultH),$ 
3 |    $\varphi_{2tp}^{stg}(T1, [H2 \mid T2], TM, ResultT).$ 
4 |
5 |  $\varphi_{2tp}^{2tp}(TP1, TP2, TM, Cond2TP) :-$ 
6 |   notEquals(TP1, TP2),
7 |    $\varphi_{2tp}^{2tp}(TP1, sub, TP2, pre, TM, Aux1),$ 
8 |    $\varphi_{2tp}^{2tp}(TP1, sub, TP2, obj, TM, Aux2),$ 
9 |    $\varphi_{2tp}^{2tp}(TP1, pre, TP2, sub, TM, Aux3),$ 
10 |   $\varphi_{2tp}^{2tp}(TP1, pre, TP2, pre, TM, Aux4),$ 
11 |   $\varphi_{2tp}^{2tp}(TP1, pre, TP2, obj, TM, Aux5),$ 
12 |   $\varphi_{2tp}^{2tp}(TP1, obj, TP2, sub, TM, Aux6),$ 
13 |   $\varphi_{2tp}^{2tp}(TP1, obj, TP2, pre, TM, Aux7),$ 
14 |   $\varphi_{2tp}^{2tp}(TP1, obj, TP2, obj, TM, Aux8),$ 
15 |  genSQLExpr(and, [Aux1, Aux2, Aux3, Aux4, Aux5, Aux6, Aux7, Aux8], Cond2TP).
```

LISTING 4.18: Function φ_{2tp}^{stg} over a pair of triple patterns

4.2.1.3 Function $genPRSQL^{stg}$ for STG patterns

Definition 4.10. (Function ψ^{stg}) Given a set of possible STG patterns **STG**, a set of possible triple maps **TM** defined in R2RML, ψ^{stg} takes a subject triple group $STG \in \mathbf{STG}$ where $STG = \{TP_1 \text{ AND } TP_2 \text{ AND } \dots \text{ AND } TP_n\}$ and a triples map $TM \in \mathbf{TM}$, and generates an SQL expression which can be used to project only those relational attributes that correspond to distinct $TP_1.subject$, $TP_1.predicate$, \dots , $TP_n.predicate$, $TP_1.object$, \dots , $TP_n.object$ and renames the projected attributes as $\beta(TP_1, sub, TM) \rightarrow name(TP_1.subject)$, $\beta(TP_1, pre, TM) \rightarrow name(TP_1.predicate), \dots, \beta(TP_n, pre, TM) \rightarrow name(TP_n.predicate)$, $\beta(TP_1, obj, TM) \rightarrow name(TP_1.object), \dots, \beta(TP_n, obj, TM) \rightarrow name(TP_n.object)$.

The algorithm of function ψ^{stg} is provided in Listing 4.20. First, it executes ψ_{sub}^{tp} of the first triple pattern in the pattern STG (line 2). Then, each of the triple patterns in STG is iterated, calculating ψ_{pre}^{tp} (line 3, 7-10) and ψ_{obj}^{tp} (line 4, 12-15) of each triple pattern. All the intermediate results are collected as the final result (line 5).

Example 4.11. Given the R2RML mapping document \mathcal{M} in Listing 4.1 and the SPARQL graph pattern $tp1 \text{ AND } tp2$ in Listing 4.2, the result of calculating function ψ^{stg} for gp can be seen in Table 4.7.

STG	Triples Map	ψ_{sub}^{tp}	ψ_{pre}^{stg}	ψ_{obj}^{stg}
{ :Product1 rdfs:label ?lbl. AND :Product1 :hasOffer :Offer3847 }	TMProduct	{ PRODUCT.nr AS iri_Product1 }	{ 'rdfs_label' AS iri_rdfs_label, 'bsbm_hasOffer' AS iri_bsbm_hasOffer }	{ PRODUCT.label AS var_lbl, OFFER.nr AS iri_Offer3847 }

TABLE 4.7: An example of computing function $genPRSQL^{stg}$

```

1   $\varphi_{2tp}^{2tp}(TP1, sub, TP2, pre, TM, Cond) :- equals(TP1.subject, TP2.predicate),$ 
2    subjectMap(TM, SM1),
3    predicateObjectMap(TM, TP2.predicate, POM2), predicateMap(POM2, PM2),
4    stronglyMatchMapping(SM1, PM2),  $\beta(TP1, sub, TM, B1), \beta(TP2, pre, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
5
6   $\varphi_{2tp}^{2tp}(TP1, sub, TP2, obj, TM, Cond) :- equals(TP1.subject, TP2.object),$ 
7    subjectMap(TM, SM1),
8    predicateObjectMap(TM, TP2.predicate, POM2), objectMap(POM2, OM2),
9    stronglyMatchMapping(SM1, OM2),  $\beta(TP1, sub, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
10
11  $\varphi_{2tp}^{2tp}(TP1, sub, TP2, obj, TM, Cond) :- equals(TP1.subject, TP2.object),$ 
12   subjectMap(TM, SM1),
13   predicateObjectMap(TM, TP2.predicate, POM2), refObjectMap(POM2, ROM2), parentSubjectMap(ROM2, SM2),
14   stronglyMatchMapping(SM1, SM2),  $\beta(TP1, sub, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
15
16  $\varphi_{2tp}^{2tp}(TP1, pre, TP2, sub, TM, Cond) :- equals(TP1.predicate, TP2.subject),$ 
17   predicateObjectMap(TM, TP1.predicate, POM1), predicateMap(POM1, PM1),
18   subjectMap(TM, SM2),
19   stronglyMatchMapping(PM1, SM2),  $\beta(TP1, pre, TM, B1), \beta(TP2, sub, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
20
21  $\varphi_{2tp}^{2tp}(TP1, pre, TP2, pre, TM, Cond) :- equals(TP1.predicate, TP2.predicate),$ 
22   predicateObjectMap(TM, TP1.predicate, POM1), predicateMap(POM1, PM1),
23   predicateObjectMap(TM, TP2.predicate, POM2), predicateMap(POM2, PM2),
24   stronglyMatchMapping(PM1, PM2),  $\beta(TP1, pre, TM, B1), \beta(TP2, pre, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
25
26  $\varphi_{2tp}^{2tp}(TP1, pre, TP2, obj, TM, Cond) :- equals(TP1.predicate, TP2.object),$ 
27   predicateObjectMap(TM, TP1.predicate, POM1), predicateMap(POM1, PM1),
28   predicateObjectMap(TM, TP2.predicate, POM2), objectMap(POM2, OM2),
29   stronglyMatchMapping(PM1, OM2),  $\beta(TP1, pre, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
30
31  $\varphi_{2tp}^{2tp}(TP1, pre, TP2, obj, TM, Cond) :- equals(TP1.predicate, TP2.object),$ 
32   predicateObjectMap(TM, TP1.predicate, POM1), predicateMap(POM1, PM1),
33   predicateObjectMap(TM, TP2.predicate, POM2), refObjectMap(POM2, ROM2), parentSubjectMap(ROM2, SM2),
34   stronglyMatchMapping(PM1, SM2),  $\beta(TP1, pre, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
35
36  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, sub, TM, Cond) :- equals(TP1.object, TP2.subject),$ 
37   predicateObjectMap(TM, TP1.predicate, POM1), objectMap(POM1, OM1),
38   subjectMap(TM, SM2),
39   stronglyMatchMapping(OM1, SM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, sub, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
40
41  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, sub, TM, Cond) :- equals(TP1.object, TP2.subject),$ 
42   predicateObjectMap(TM, TP1.predicate, POM1), refObjectMap(POM1, ROM1), parentSubjectMap(ROM1, SM1),
43   subjectMap(TM, SM2),
44   stronglyMatchMapping(SM1, SM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, sub, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
45
46  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, pre, TM, Cond) :- equals(TP1.object, TP2.predicate),$ 
47   predicateObjectMap(TM, TP1.predicate, POM1), objectMap(POM1, OM1),
48   predicateObjectMap(TM, TP2.predicate, POM2), predicateMap(POM2, PM2),
49   stronglyMatchMapping(OM1, PM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, pre, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
50
51  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, pre, TM, Cond) :- equals(TP1.object, TP2.predicate),$ 
52   predicateObjectMap(TM, TP1.predicate, POM1), refObjectMap(POM1, ROM1), parentSubjectMap(ROM1, SM1),
53   predicateObjectMap(TM, TP2.predicate, POM2), predicateMap(POM2, PM2),
54   stronglyMatchMapping(SM1, PM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, pre, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
55
56  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, obj, TM, Cond) :- equals(TP1.object, TP2.object),$ 
57   predicateObjectMap(TM, TP1.predicate, POM1), objectMap(POM1, OM1),
58   predicateObjectMap(TM, TP2.predicate, POM2), objectMap(POM2, OM2),
59   stronglyMatchMapping(OM1, OM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
60
61  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, obj, TM, Cond) :- equals(TP1.object, TP2.object),$ 
62   predicateObjectMap(TM, TP1.predicate, POM1), objectMap(POM1, OM1),
63   predicateObjectMap(TM, TP2.predicate, POM2), refObjectMap(POM2, ROM2), parentSubjectMap(ROM2, SM2),
64   stronglyMatchMapping(OM1, SM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
65
66  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, obj, TM, Cond) :- equals(TP1.object, TP2.object),$ 
67   predicateObjectMap(TM, TP1.predicate, POM1), refObjectMap(POM1, ROM1), parentSubjectMap(ROM1, SM1),
68   predicateObjectMap(TM, TP2.predicate, POM2), objectMap(POM2, OM2),
69   stronglyMatchMapping(SM1, OM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 
70
71  $\varphi_{2tp}^{2tp}(TP1, obj, TP2, obj, TM, Cond) :- equals(TP1.object, TP2.object),$ 
72   predicateObjectMap(TM, TP1.predicate, POM1), refObjectMap(POM1, ROM1), parentSubjectMap(ROM1, SM1),
73   predicateObjectMap(TM, TP2.predicate, POM2), refObjectMap(POM2, ROM2), parentSubjectMap(ROM2, SM2),
74   stronglyMatchMapping(SM1, SM2),  $\beta(TP1, obj, TM, B1), \beta(TP2, obj, TM, B2), genSQLExpr(equals, [B1, B2], Cond).$ 

```

LISTING 4.19: φ_{2tp}^{2tp}

4.2.1.4 Function *trans* for translating STG patterns

Now we are ready to extend the original *trans* algorithm to accept an STG pattern, without recursively using the translation of AND pattern. The algorithm for translating

```

1 |  $\psi^{stg}([H \mid T], TM, [PRSub \mid PRPreObj]) :-$ 
2 |    $\psi_{sub}^{tp}(H, TM, PRSub),$ 
3 |    $\psi_{pre}^{stg}([H \mid T], TM, PRPre),$ 
4 |    $\psi_{obj}^{stg}([H \mid T], TM, PRObj),$ 
5 |   append(PRPre, PRObj, PRPPreObj).
6 |
7 |  $\psi_{pre}^{stg}([], TM, []).$ 
8 |  $\psi_{pre}^{stg}([H \mid T], TM, [PRPreHead, PRPreTail]) :-$ 
9 |    $\psi_{pre}^{tp}(H, TM, PRPreHead),$ 
10 |   $\psi_{pre}^{stg}(T, TM, PRPreTail).$ 
11 |
12 |  $\psi_{obj}^{stg}([], TM, []).$ 
13 |  $\psi_{obj}^{stg}([H \mid T], TM, [PRObjHead, PRObjTail]) :-$ 
14 |   $\psi_{obj}^{tp}(H, TM, PRObjHead),$ 
15 |   $\psi_{obj}^{stg}(T, TM, PRObjTail).$ 

```

LISTING 4.20: Function $genPRSQL^{stg}$ for STG patterns

an STG pattern can be seen in Listing 4.21.

```

1 | trans(STG, M, Queries) :-
2 |   isSTGPattern(STG),
3 |   triplesMaps(STG, M, TMList),
4 |   transTMListstg(STG, TMList, Queries).
5 |
6 | transTMListstg(STG, [], []).
7 |
8 | transTMListstg(STG, [TMHead | TMTail], Queries) :-
9 |   transSTG(STG, TMHead, QueryHead),
10 |  transTMListstg(STG, TMTail, QueryTail),
11 |  getSQLExpr(union, [QueryHead, QueryTail], Queries).
12 |
13 | transSTG(STG, TM, Query) :-
14 |    $\alpha^{stg}(STG, TM, AlphaSTGResult),$ 
15 |    $\psi^{stg}(STG, TM, PRSQLSTGResult),$ 
16 |    $\varphi^{stg}(STG, TM, CondSQLSTGResult),$ 
17 |   sqlQuery(PRSQLSTGResult, AlphaSTGResult, CondSQLSTGResult, Query).

```

LISTING 4.21: Function *trans* for translating STG patterns

Example 4.12. Consider again the STG pattern *tp1* AND *tp2* in Listing 4.2. With the optimized version *trans* function described in Listing 4.21, the result of translating that STG pattern is:

```

SELECT Product.nr AS iri_Product1, 'rdfs:label' AS iri_rdfsLabel
      , Product.label AS var_lbl, 'bsbm:hasOffer' AS iri_bsbmHasOffer
      , Offer.nr AS iri_Offer3847
FROM Product, Offer
WHERE Product.nr = 1 AND Product.label IS NOT NULL AND
      Product.nr = Offer.nr AND Offer.nr = 3847;

```

Now with the optimised version, the complete translation result of the pattern in Listing 4.2 can be seen in Listing 4.22. Lines 16-21 in Listing 4.22 correspond to lines 16-29 in Listing 4.4.

```

1 SELECT g3.iri_Product1, g3.iri_rdfsLabel, g3.var_lbl
2   , g3.iri_bsbmHasOffer, g3.iri_Offer3847
3   , g3.iri_bsbmPropText4, g3.var_propText4
4   , g3.iri_bsbmPrice, g3.var_offerPrice
5   , t5.iri_bsbmDeliveryDays, t5.var_dd
6 FROM (
7   SELECT g2.iri_Product1, g2.iri_rdfsLabel, g2.var_lbl
8     , g2.iri_bsbmHasOffer, g2.iri_Offer3847
9     , g2.iri_bsbmPropText4, g2.var_propText4
10    , t4.iri_bsbmPrice, t4.var_offerPrice
11  FROM (
12    SELECT g1.iri_Product1, g1.iri_rdfsLabel, g1.var_lbl
13      , g1.iri_bsbmHasOffer, g1.iri_Offer3847
14      , t3.iri_bsbmPropText4, t3.var_propText4
15    FROM (
16      SELECT Product.nr AS iri_Product1, rdfs:label AS iri_rdfsLabel
17        , Product.label AS var_lbl, bsbm:hasOffer AS iri_bsbmHasOffer
18        , Offer.nr AS iri_Offer3847
19      FROM Product, Offer
20      WHERE Product.nr = 1 AND Product.label IS NOT NULL AND Product.nr = Offer.nr AND Offer.nr = 3847
21    ) g1 -- translation result of stg pattern { tp1 AND tp2 }
22  LEFT OUTER JOIN (
23    SELECT Product.nr AS iri_Product1, bsbm:propText4 AS iri_bsbm_propText4, Product.propertyTextual4 AS var_pt4
24    FROM Product
25    WHERE Product.nr = 1 AND Product.propertyTextual4 IS NOT NULL
26  ) t3 --translation result of tp3
27  ON g1.iri_Product1 = t3.iri_Product1
28  ) g2 -- translation result of { tp1 AND tp2 } OPT tp3
29  LEFT OUTER JOIN (
30    SELECT Offer.nr AS iri_Offer3847, bsbm:price AS iri_bsbmPrice, Offer.price AS var_offerPrice
31    FROM Offer
32    WHERE Offer.nr = 3847 AND Offer.price IS NOT NULL
33  ) t4 --translation result of tp4
34  ON g2.iri_Offer3847 = t4.iri_Offer3847
35  ) g3 -- translation result of { tp1 AND tp2 } OPT tp3 OPT t4
36  LEFT OUTER JOIN (
37    SELECT Offer.nr AS iri_Offer3847, bsbm:deliveryDays AS iri_bsbmDeliveryDays, Offer.deliveryDays AS var_dd
38    FROM Offer
39    WHERE Offer.nr = 3847 AND Offer.deliveryDays IS NOT NULL
40  ) t5 --translation result of tp5
41  ON g3.iri_Offer3847 = t5.iri_Offer3847;

```

LISTING 4.22: Example of translating SPARQL graph pattern with *trans* function that accepts STG patterns

4.2.1.5 Comparison of resulting translation of an STG query for RDB2RDF systems ontop and morph-RDB

In addition to the running example that we use for this chapter, starting from this section, we also provide one example for each of the optimizations presented, to show the queries produced by two different R2RML engines: ontop using the mappings in Appendix A.2 and morph-RDB (an R2RML engine implementing the query translation techniques that we see in this chapter) using the mappings in Appendix A.1. Both mappings are mapped to the following tables: `tbl_patient(patientid, name, type, stage)` and `tbl_stage(stageid, stagename)`. `patientid` and `stageid` are the primary keys of the tables `tbl_patient` and `tbl_stage`, respectively, and `stageid` is a foreign key that refers to the column `patientid`. The instances of the tables can be seen in Figure 4.1.

Consider the following graph pattern

```

{ ?p rdf:type :Patient .
  ?p :hasName ?pname .
  ?p :hasStage ?s . }

```

patientid	name	type	stage
1	mary	0	4
2	john	1	7

stageid	stagename
1	NSCLC Stage I
2	NSCLC Stage II
3	NSCLC Stage III
4	NSCLC Stage IIIa
5	NSCLC Stage IIIb
6	NSCLC Stage IV
7	SCLC Stage Limited
8	SCLC Stage Extensive

FIGURE 4.1: Tables Patient and Stage

The SQL query translated by ontop is:

```
SELECT patientid, name, stagename
FROM tbl_patient T1, tbl_stage T2
WHERE T1.patientid IS NOT NULL AND T1.name IS NOT NULL
AND T1.stage=T2.stageid AND T2.stagename IS NOT NULL
```

The SQL query translated by morph-RDB is:

```
SELECT patientid, name, stagename
FROM tbl_patient T1 INNER JOIN tbl_stage T2 ON (T1.stage=T2.stageid)
WHERE T1.name IS NOT NULL AND T2.stagename IS NOT NULL
```

Both queries return the following results:

```
(Patient/1, "Mary", Stage-NSCLC-Stage-IIIA)
(Patient/2, "John", Stage-SCLC-Stage-Limited)
```

See Appendix [A.3](#) for the actual queries produced by the two systems.

4.2.2 Left Outer Join Elimination for OPTIONAL patterns

Another common pattern of SPARQL queries is an STG pattern having OPTIONAL blocks containing only a single triple pattern, and where the object of the OPTIONAL pattern is not referred by any other graph pattern, what we call subject triple group with optional (OSTG).

Definition 4.11. A subject triple group with optional (OSTG) is defined recursively as follows:

- If TP_1 and TP_2 are triple patterns, $TP_1.subject = TP_2.subject$, and $TP_2.object \notin var(TP_1)$ then $(TP_1 \text{ OPT } TP_2)$ is a subject triple group with optional where $var(P)$ refers to a set of variables in the pattern P and $P.subject$ refers to the subject of the pattern P .
- If STG is a subject triple group, TP is a triple pattern, $STG.subject = TP.subject$, and $TP.object \notin var(STG)$, then $(STG \text{ OPT } TP)$ is a subject triple group with optional.
- If $OSTG$ is a subject triple group with optional, TP is a triple pattern, $OSTG.subject = TP.subject$, and $TP.object \notin var(OSTG)$, then $(OSTG \text{ OPT } TP)$ is a subject triple group with optional.

Example 4.13. The graph pattern $tp1 \text{ AND } tp2 \text{ OPT } tp3$ in Listing 4.2 lines 2-6 is an OSTG pattern. The result of translating this OSTG pattern can be seen in Listing 4.4 lines 12-36.

Using the translation algorithm and the self-join elimination optimisation that we presented in the previous sections, the translation of an OSTG pattern will contain a left outer join between the STG and the TP pattern. We note that further optimisation can be obtained by eliminating the left outer join and dropping the IS NOT NULL condition from the *genCondSQL* for the object part of the TP. We present the algorithm for translating OSTG patterns in Listing 4.23 where φ^{ostg} is basically the same as φ^{stg} except that it does not generate the IS NOT NULL expression for the object of the triple pattern in the OSTG pattern.

```

1  trans(OSTG, M, Queries) :-
2    isOSTG(OSTG),
3    triplesMaps(OSTG, TMList),
4    transTMListostg(OSTG, TMList, Queries).
5
6  transTMListostg(OSTG, [], []).
7
8  transTMListostg(OSTG, [TMSHead | TMSTail], Queries) :-
9    transOSTG(OSTG, TMSHead, QueryHead),
10   transTMListostg(OSTG, TMSTail, QueryTail),
11   genSQLExpr(union, [QueryHead, QueryTail], Queries).
12
13  transOSTG(OSTG, TM, Query) :-
14     $\psi^{stg}$ (OSTG, TM, SelectPart),
15     $\alpha^{stg}$ (OSTG, TM, FromPart),
16     $\varphi^{ostg}$ (OSTG, TM, WherePart),
17    removeISNOTNULL(WherePart, WherePart2),
18    sqlQuery(SelectPart, FromPart, WherePart2, Query).

```

LISTING 4.23: Function *trans(OSTG)*

Example 4.14. Using the optimised *trans* function that accepts OSTG pattern, the result of translating the graph pattern *tp1 AND tp2 OPT tp3* in Listing 4.2 lines 2-6 is:

```
SELECT Product.nr AS iri_Product1, 'rdfs:label' AS iri_rdfsLabel
, Product.label AS var_lbl, 'bsbm:hasOffer' AS iri_bsbmHasOffer
, Offer.nr AS iri_Offer3847, Product.propertyTextual4 AS var_pt4
FROM Product, Offer
WHERE Product.nr = 1 AND Product.label IS NOT NULL
AND Product.nr = Offer.nr AND Offer.nr = 3847
```

The complete translation result of the pattern in Listing 4.2 can be seen in Listing 4.24. Note that lines 12-17 in Listing 4.24 correspond to lines 12-28 in Listing 4.22.

```
1 SELECT g3.iri_Product1, g3.iri_rdfsLabel, g3.var_lbl
2   , g3.iri_bsbmHasOffer, g3.iri_Offer3847
3   , g3.iri_bsbmPropText4, g3.var_propText4
4   , g3.iri_bsbmPrice, g3.var_offerPrice
5   , t5.iri_bsbmDeliveryDays, t5.var_dd
6 FROM (
7   SELECT g2.iri_Product1, g2.iri_rdfsLabel, g2.var_lbl
8     , g2.iri_bsbmHasOffer, g2.iri_Offer3847
9     , g2.iri_bsbmPropText4, g2.var_propText4
10    , g2.iri_bsbmPrice, t4.var_offerPrice
11  FROM (
12    SELECT Product.nr AS iri_Product1, rdfs:label AS iri_rdfsLabel
13      , Product.label AS var_lbl, bsbm:hasOffer AS iri_bsbmHasOffer
14      , Offer.nr AS iri_Offer3847, Product.propertyTextual4 AS var_pt4
15    FROM Product, Offer
16    WHERE Product.nr = 1 AND Product.label IS NOT NULL AND Product.nr = Offer.nr AND Offer.nr = 3847
17  ) g2 -- translation result of { tp1 AND tp2 } OPT tp3
18  LEFT OUTER JOIN (
19    SELECT Offer.nr AS iri_Offer3847, bsbm:price AS iri_bsbmPrice, Offer.price AS var_offerPrice
20    FROM Offer
21    WHERE Offer.nr = 3847 AND Offer.price IS NOT NULL
22  ) t4 --translation result of tp4
23  ON g2.iri_Offer3847 = t4.iri_Offer3847
24  ) g3 -- translation result of { tp1 AND tp2 } OPT tp3 OPT t4
25  LEFT OUTER JOIN (
26    SELECT Offer.nr AS iri_Offer3847, bsbm:deliveryDays AS iri_bsbmDeliveryDays, Offer.deliveryDays AS var_dd
27    FROM Offer
28    WHERE Offer.nr = 3847 AND Offer.deliveryDays IS NOT NULL
29  ) t5 --translation result of tp5
30  ON g3.iri_Offer3847 = t5.iri_Offer3847;
```

LISTING 4.24: Example of translating SPARQL graph pattern with *trans* function that accepts OSTG patterns

4.2.2.1 Comparison of resulting translation of an OSTG query for RDB2RDF systems ontop and morph-RDB

Consider the following graph pattern

```
{ ?p rdf:type :Patient .
OPTIONAL { ?p :hasName ?pname . }
}
```

The SQL query translated by ontop is:


```

SELECT T1.patientid, T2.name
FROM tbl_patient T1
LEFT OUTER JOIN tbl_patient T2 ON T1.patientid=T2.patientid
AND T1.patientid IS NOT NULL AND T2.name IS NOT NULL
WHERE T1.patientid IS NOT NULL

```

The SQL query translated by morph-RDB is:

```

SELECT T2.patientid, T2.name
FROM tbl_patient T2

```

Both queries return the following results:

```

(Patient/1, "Mary")
(Patient/2, "John")

```

See Appendix [A.4](#) for the actual queries produced by the two systems.

4.2.3 Phantom Triple Pattern Introduction

Consider the SPARQL graph pattern in Listing [4.2](#), which do not contain either an STG pattern or an OSTG pattern. Thus, none of the aforementioned optimisations can be applied on this query. In order to exploit those optimisations we have presented so far, this query has to be transformed into another query whose resulting query translation can be optimised. To do that, we use the fact that for every IRI x , the fact $(x \text{ a } \textit{rdf:Resource})$ holds, so that we can safely add this triple pattern to the query without changing the semantics of the query. We call such triple pattern a *phantom triple pattern*. The result of adding the phantom triple pattern can be seen in Listing [4.25](#) now the optimisations that we have presented are available to be applied and the translation result of the new graph pattern can be seen in Listing [4.26](#).

4.2.3.1 Comparison of resulting translation of Phantom TP Introduction query for RDB2RDF systems ontop and morph-RDB

Consider the following graph pattern

```

SELECT * WHERE {
?p rdf:type :Patient .

```

```

1 {
2   :Product/1 rdfs:label ?lbl . #tp1
3   :Product/1 bsbm:hasOffer :Offer/3847 . #tp2
4   OPTIONAL {
5     :Product/1 bsbm:productPropertyTextual4 ?pt4 . #tp3
6   }
7   :Offer/3847 a rdf:Resource . #tp0
8   OPTIONAL { :Offer/3847 bsbm:price ?offerPrice . #tp4
9   }
10  OPTIONAL { :Offer/3847 bsbm:deliveryDays ?dd . #tp5
11  }
12 }

```

LISTING 4.25: Example of SPARQL graph pattern with phantom triple pattern

```

1 SELECT g2.iri_Product1, g2.iri_rdfsLabel, g2.var_lbl
2 , g2.iri_bsbmHasOffer, g2.iri_Offer3847
3 , g2.iri_bsbmPropText4, g2.var_propText4
4 , g4.iri_bsbmPrice, g4.var_offerPrice, g4.var_dd
5 FROM (
6   SELECT Product.nr AS iri_Product1, rdfs:label AS iri_rdfsLabel
7   , Product.label AS var_lbl, bsbm:hasOffer AS iri_bsbmHasOffer
8   , Offer.nr AS iri_Offer3847, Product.propertyTextual4 AS var_pt4
9   FROM Product, Offer
10  WHERE Product.nr = 1 AND Product.label IS NOT NULL AND Product.nr = Offer.nr AND Offer.nr = 3847
11 ) g2 -- translation result of { tp1 AND tp2 } OPT tp3
12 INNER JOIN (
13   SELECT Offer.nr AS iri_Offer3847, bsbm:price AS iri_bsbmPrice, Offer.price AS var_offerPrice, Offer.deliveryDays AS var_dd
14   FROM Offer
15   WHERE Offer.nr = 3847
16 ) g4 -- translation result of tp0 OPT tp4 OPT tp5
17 ON g2.iri_Offer3847 = g4.iri_Offer3847;

```

LISTING 4.26: Example of translating SPARQL graph pattern with *trans* function that accepts phantom triple patterns

```

?p :hasStage ?s .
OPTIONAL { ?s :hasStageName ?stagename . }
}

```

The SQL query translated by *ontop* is:

```

SELECT T1.patientid, T2.stagename
FROM (
  tbl_patient T1 JOIN tbl_stage T2
  ON T1.patientid IS NOT NULL AND (T1.stage = T2.stageid)
  AND T2.stagename IS NOT NULL
)
LEFT OUTER JOIN tbl_stage T3
ON (T2.stagename = T3.stagename) AND T3.stagename IS NOT NULL);

```

The SQL query translated by *morph-RDB* is:

```

SELECT T1.patientid, T2.stagename
FROM tbl_patient T1

```

```
INNER JOIN tbl_stage T2 ON (T1.stage = T2.stageid)
WHERE T1.stage IS NOT NULL
```

Both queries return the following results:

```
(Patient/1, Stage/4, "NSCLC Stage IIIa")
(Patient/2, Stage/7, "SCLC Stage Limited")
```

See Appendix A.5 for the actual queries produced by the two systems.

4.2.4 IS NOT NULL checking

Recall that the function *genCondSQL* described in Section 4.1.6 requires an IS NOT NULL expression for the database values returned. However, the database metadata may contain constraints that certain columns cannot have NULL values, such as id columns. Thus, upon reading this constraint, we can safely drop the IS NOT NULL condition in the *genCondSQL* expression.

Example 4.15. *If a constraint `Product.label IS NOT NULL` is specified in the database schema, then there is no need for the expression `Product.label` for the translation of the triple pattern `(:Product1 rdfs:label ?lbl)`. The result of translating the graph pattern in Listing 4.2 now can be seen in Listing 4.27.*

```

1 SELECT g2.iri_Product1, g2.iri_rdfsLabel, g2.var_lbl
2   , g2.iri_bsbmHasOffer, g2.iri_Offer3847
3   , g2.iri_bsbmPropText4, g2.var_propText4
4   , g4.iri_bsbmPrice, g4.var_offerPrice, g4.var_dd
5 FROM (
6   SELECT Product.nr AS iri_Product1, rdfs:label AS iri_rdfsLabel
7   , Product.label AS var_lbl, bsbm:hasOffer AS iri_bsbmHasOffer
8   , Offer.nr AS iri_Offer3847, Product.propertyTextual4 AS var_pt4
9   FROM Product, Offer
10  WHERE Product.nr = 1 AND NULL AND Product.nr = Offer.nr AND Offer.nr = 3847
11 ) g2 -- translation result of { tp1 AND tp2 } OPT tp3
12 INNER JOIN (
13   SELECT Offer.nr AS iri_Offer3847, bsbm:price AS iri_bsbmPrice, Offer.price AS var_offerPrice, Offer.deliveryDays AS var_dd
14   FROM Offer
15   WHERE Offer.nr = 3847
16 ) g4 -- translation result of tp0 OPT tp4 OPT tp5
17 ON g2.iri_Offer3847 = g4.iri_Offer3847;
```

LISTING 4.27: Example of translating SPARQL graph pattern with *trans* function when `Product.label` is specified as NOT NULL

4.2.4.1 Comparison of resulting translation of a TP pattern for RDB2RDF systems atop and morph-RDB

Consider the following graph pattern

```
SELECT * WHERE {  
  ?p :hasName ?n .  
}
```

The SQL query translated by *ontop* is:

```
SELECT T1.patientid, T1.name  
FROM tbl_patient T1  
WHERE T1.patientid IS NOT NULL AND T1.name IS NOT NULL;
```

Because *patientid* is a primary key of the table *tbl_patient* that implies that *patientid* can not be NULL, then there is no need to include the expression *patientid IS NOT NULL*. The SQL query translated by *morph-RDB* is:

```
SELECT T1.patientid, T1.name  
FROM tbl_patient T1  
WHERE T1.name IS NOT NULL
```

Both queries return the following results:

```
(Patient/1, "Mary")  
(Patient/2, "John")
```

See Appendix [A.6](#) for the actual queries produced by the two systems.

4.2.5 Other Optimisations

4.2.5.1 Table reordering

When the translation query contains multiple joins, we can also read the metadata for the size/number of rows of the joined tables. We reorder the tables so that the smaller tables are joined first. In doing so, we may help the database optimiser to reduce the number of rows to be joins in the intermediate results.

4.2.5.2 Union Reduction

We have seen in Listing [4.5](#) that when an unbounded predicate triple pattern (that is, when a triple pattern has a variable in its predicate component) is translated, all

the possible mapped predicates are translated and put together as a UNION query. Depending on the number of properties mapped in the R2RML mapping document, the generated UNION query may be unnecessarily large. In order to reduce the number of elements in the UNION part of the query, it is possible to analyze the combination of mappings and queries whose resulting translation can be safely removed. In the following, we show two of the potential combinations that can appear in this case, with the optimisations that would be performed to reduce such number of unnecessary parts in the queries.

4.2.5.3 Incompatible Term Types

```

1  -- sparql query --
2  SELECT * WHERE {
3    :Product1 ?p "water" . #tp
4  }
5
6  trans(tp) = SQL1 UNION SQL2 UNION SQL3
7
8  -- SQL1: translation from (:Product1 rdfs:label "water")
9  SELECT nr AS iri_Product1, rdfs:label AS var_p, label AS lit_water
10 FROM PRODUCT
11 WHERE PRODUCT.nr = 1 AND label = water
12
13 -- SQL2: translation from (:Product1 rdfs:comment "water")
14 SELECT nr AS iri_Product1, rdfs:comment AS var_p, comment AS lit_water
15 FROM PRODUCT
16 WHERE PRODUCT.nr = 1 AND comment = water
17
18 -- SQL3: translation from (:Product1 bsbm:productOffer "water")
19 SELECT PRODUCT.nr as iri_Product1, :hasOffer AS var_p, label AS lit_water
20 FROM PRODUCT, OFFER WHERE PRODUCT.nr = OFFER.product
21 AND PRODUCT.nr = 1 AND OFFER.nr = water

```

LISTING 4.28: RefObjectMap vs Literal

Consider the SPARQL query in Listing 4.28, having an unbounded predicate triple pattern. According to the triples map *TriplesMapProduct* specified in our R2RML document that contains three `rr:PredicateObjectMap` mappings, the triple pattern $tp = (:Product1 ?p "water")$ will be interpreted as the union of bounded triple patterns:

- $(:Product1 \text{ rdfs:label } "water")$ which is translated into *SQL1*.
- $(:Product1 \text{ rdfs:comment } "water")$ which is translated into *SQL2*.
- $(:Product1 \text{ bsbm:hasOffer } "water")$ which is translated into *SQL3*.

The translation of the triple pattern tp will be the UNION of *SQL1*, *SQL2*, and *SQL3*. However, one can note that the R2RML mapping specified for *bsbm:hasOffer* is a `rr:RefObjectMap` mapping, whose output is expected to be a IRI (the subject of the

parent triple map) but the object specified in the query ("water") is not a valid URI. Thus, we can easily remove *SQL3* from the UNION query.

4.2.5.4 Incompatible Datatype

Consider the SPARQL query in Listing 4.29, having an unbounded predicate triple pattern.

```

1  -- sparql query --
2  SELECT * WHERE {
3    :Offer1 ?p "limited" . #tp
4  }
5
6  trans(tp) = SQL1 UNION SQL2 UNION SQL3
7
8  -- SQL1: translation from (:Offer1 rdfs:label "limited")
9  SELECT nr AS iri.Offer1, rdfs:label AS var_p, label AS lit_limited
10 FROM OFFER
11 WHERE OFFER.nr = 1 AND label = limited
12
13 -- SQL2: translation from (:Offer1 bsbm:price "limited")
14 SELECT nr AS iri.Product1, bsbm:price AS var_p, label AS lit_limited
15 FROM PRODUCT
16 WHERE PRODUCT.nr = 1 AND price = limited
17
18 -- SQL3: translation from (:Offer1 bsbm:hasProduct "limited")
19 SELECT nr AS iri.Offer1, bsbm:hasProduct AS var_p, label AS lit_limited
20 FROM OFFER
21 WHERE OFFER.nr = 1 AND product = limited

```

LISTING 4.29: Incompatible Datatype

Because the triples map *TriplesMapOffer* contains three `rr:PredicateObjectMap` mappings, then the triple pattern $tp = (:Offer1 ?p "limited")$ will be interpreted the union of bounded triple patterns:

- $(:Offer1 \text{ rdfs:label } "limited")$ that produces *SQL1*.
- $(:Offer1 \text{ bsbm:price } "limited")$ that produces *SQL2*.
- $(:Offer1 \text{ bsbm:hasProduct } "limited")$ that produces *SQL3*.

The translations will be the UNION of *SQL1*, *SQL2* and *SQL3*. We can easily obtain, by looking at the original relational database, that the columns `price` and `product` are of datatype *double*. Since the object component of the triple pattern ("limited") cannot be converted to a numerical datatype, then we can safely remove *SQL2* and *SQL3* from the UNION query.

4.3 Conclusion

In this chapter we have shown that the query translation approaches used so far for handling RDB2RDF mapping languages may be inefficient. We have proposed an extension of one of the most detailed approaches for query rewriting, Chebotko's, which was not originally conceived for RDB2RDF query translation but for RDBMS-backed triple stores. Now we consider R2RML mappings in the query translation process. In Chapter 7 we will see through our empirical evaluation that in all of the synthetic benchmark and real cases queries, our approach behaves in general similarly to native queries, and better than other existing approaches.

There is still room for improvement in our work, which we will address in the near future. For instance, some of the optimised translated queries perform better than the native ones, due to the introduction of additional predicates, what is common in the area of Semantic Query Optimization (SQO). In this sense, we map deepen in the design of our algorithm so as to take into account other SQO techniques that can be useful in query translation as proposed by the work of Rodriguez et. al. [RMR15]. Our evaluation setup will be also made available as a service for other researchers to use, so that they can evaluate their R2RML query rewriting implementations with low effort in a number of RDBMSs.

Furthermore, in Chapter 8, we also describe two extensions of morph-RDB: morph-GFT that is able to support Google Fusion Table and SPARQL endpoints, and morph-LDP that provides read-write Linked Data access as to support Linked Data Platform (LDP) protocols.

Chapter 5

On the Expressive Power of Direct Mapping and its Relationship with R2RML

The two W3C Recommendations for generating RDF data from relational databases that we have described in Section 2.4.3 were published at the same time in 2012. To the best of our knowledge, there has not been any exhaustive formal study yet about the relationship between these two recommendations. Intuitively, one would consider Direct Mapping as a subset of R2RML, given the expressive power provided by the latter. However, the lack of such formal study does not allow concluding that this assumption is necessarily correct, neither which are the actual relationships between both recommendations.

Sequeda [Seq13] introduced a fragment of R2RML called $R2RML_{core}$, motivated by the observation that there are types of mappings that are the most commonly used mappings in his experience. This work was concluded by a hypothesis that $R2RML_{core}$ is as expressive as the Direct Mapping, if views are allowed as input.

This chapter presents our contribution along the lines of that work. First, we introduce $R2RML_{lite}$, an extension of $R2RML_{core}$, that still focuses on the essential part of R2RML. Second, we extend and validate the hypothesis in [Seq13] so as to claim that $R2RML_{lite}$ is as expressive as the Direct Mapping, as long as views are allowed as input together with an IRI *replacement* function.

In Section 5.1, we review $R2RML_{core}$ and introduce its extension, the $R2RML_{lite}$ fragment, together with the normalization process of $R2RML_{lite}$ mappings into a simpler form. In Section 5.2, we discuss the notions of information and query result preservation

properties, which will be used in Section 5.3 to analyze the relationship between Direct Mapping and $R2RML_{lite}$. Finally, in Section 5.4, we present some conclusions of our work.

5.1 $R2RML_{lite}$

5.1.1 $R2RML_{lite}$ Mappings

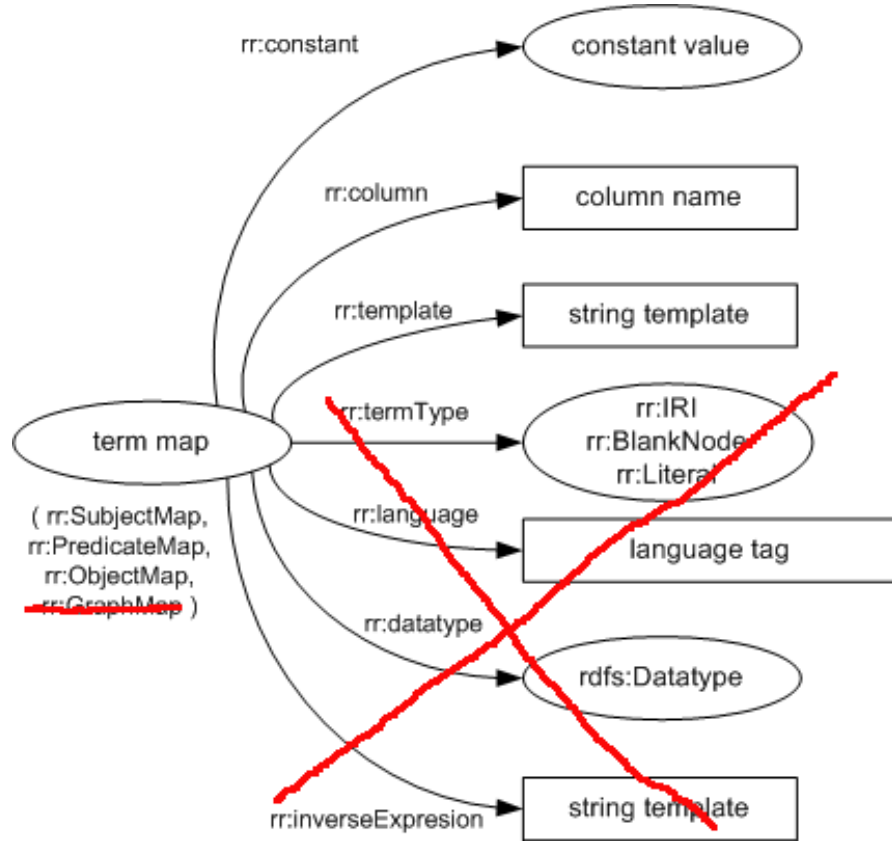
In order to study the properties of R2RML, we need to restrict the original R2RML specification, so as to focus our attention on a fragment that allows generating RDF triples without additional features such as graph, datatype, termtype and language. Motivated by this reason, Sequeda introduced a fragment of R2RML called $R2RML_{core}$ [Seq13]. That fragment was chosen by the observation that this is the fragment of mappings that is most commonly used in real use cases. In this fragment, Subject Maps are only generated by template-valued mappings, Predicate Maps by constant-value mappings, and Object Maps by column-valued mappings. Furthermore, in order to associate mappings with datalog rules, $R2RML_{core}$ implicitly assumed that a template mapping is to be in the form of `rr:template "c/{attr}"` where `c` is a constant specified by a user and `attr` is a database column name. The work in [Seq13] was concluded by a hypothesis that $R2RML_{core}$ is as expressive as the Direct Mapping as long as views are allowed as input.

In this section we introduce $R2RML_{lite}$, which extends $R2RML_{core}$ by allowing values of term map to be generated using all possible mappings (constant-valued, column-valued, and template-valued). Figure 5.1 gives an overview of the structure of an R2RML Term Map in $R2RML_{lite}$.

We introduce procedures that transform any $R2RML_{lite}$ mappings into simpler syntactic forms, and finally we close this section by presenting datalog rules associated to $R2RML_{lite}$.

5.1.2 $R2RML_{lite}$ Normalization

$R2RML$ allows multiple ways to express mappings that generate the same output, and this is inherited by $R2RML_{lite}$. In order to simplify the expressions that we can create in $R2RML_{lite}$, while preserving the semantics of the mappings, we propose the generation of a *normal form*, which is introduced in [KRRM⁺14, RMR15]. We extend this work by describing four normal forms and the procedure that can be followed to transform

FIGURE 5.1: Overview of the structure of an R2RML Term Map in *R2RML_{lite}*

mappings from the lowest to the highest normal form. The higher the normal form is, the simpler it is (in the sense of the cardinality and primitives used) while the more mappings are needed to express the same mapping specified in the lower form. The first and second normal mapping forms are used to reduce the cardinalities associated to mapping elements. The third normal mapping form is used to replace some mapping elements into an equivalent form (such as `rr:class` with a mapping of `rdf:type`, or `rr:RefObjectMap` with `rr:sqlQuery`), so that every mapping is independent and not related to other mappings. The fourth normal form is used to replace all constant and column mappings into template mappings.

5.1.2.1 1st Normal Form (1NF)

An R2RML mapping document \mathcal{M} is in the First Normal Form (1NF) if every `rr:PredicateObjectMap` contains only one pair of `rr:PredicateMap` and `rr:ObjectMap`.

R2RML Mappings that are not in normal form can be transformed into equivalent R2RML mappings in the first normal form by the following procedure:

1. Replace a Triples Map TM having `rr:predicateObjectMap` that contains m mappings of `rr:predicateMap` and n mappings of `rr:objectMap`, with $m \times n$ Triples Map so that each of the Triples Map $TM_{i,j}$, $0 < i < m$, $0 < j < n$ having `rr: predicateObjectMapi,j` that contains `rr : predicateMapi` and `rr : objectMapj`.

The R2RML mapping document shown in Listing 5.2 is the equivalent 1st normal form of the mapping in Listing 5.1.

```

1 @prefix rr: <http://www.w3.org/ns/r2rml#> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4 @prefix ex: <http://www.example.com#> .
5
6 <TMEmployee> a rr:TriplesMap;
7   rr:logicalTable [ rr:tableName "EMP" ];
8
9   rr:subjectMap [
10     rr:template "http://ex.com/Person/{EID}";
11     rr:class ex:Employee; ];
12
13   rr:predicateObjectMap [
14     rr:predicateMap [ rr:constant ex:name; ];
15     rr:objectMap [ rr:column "LNAME"; ];
16     rr:objectMap [ rr:column "FNAME"; ];];
17
18   rr:predicateObjectMap [
19     rr:predicateMap [ rr:constant ex:fullname; ];
20     rr:objectMap [ rr:template "{LNAME},{FNAME}"; ]; ];
21
22   rr:predicateObjectMap [
23     rr:predicateMap [ rr:constant ex:worksIn; ];
24     rr:objectMap [
25       rr:parentTriplesMap <TMDepartment>;
26       rr:joinCondition [ rr:child "DEPTID" ; rr:parent "DID" ; ];
27     ];
28 ].
29
30 <TMDepartment> a rr:TriplesMap;
31   rr:logicalTable [ rr:tableName "DEPT" ];
32
33   rr:subjectMap [
34     rr:template "http://ex.com/Department/{DID}{DNAME}";
35     rr:class ex:Department; ];
36
37   rr:predicateObjectMap [
38     rr:predicateMap [ rr:constant ex:name ];
39     rr:objectMap [ rr:column "DNAME"; ]; ];
40 ].

```

LISTING 5.1: Example of R2RML Mapping Document

5.1.2.2 2nd Normal Form (2NF)

The 2nd Normal Form extends the 1st Normal Form so that every triples map has only one `rr:logicalTable`, one `rr:Subjectmap` and one `rr:PredicateObjectMap`, made up from a pair of `rr:PredicateMap` and `rr:Objectmap`.

```

1  @prefix rr: <http://www.w3.org/ns/r2rml#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix ex: <http://www.example.com#> .
5
6  <TMEmployee1> a rr:TriplesMap;
7    rr:logicalTable [ rr:tableName "EMP" ];
8
9    rr:subjectMap [ rr:class ex:Employee; rr:template "http://ex.com/Person/{EID}"; ];
10
11    rr:predicateObjectMap [
12      rr:predicateMap [ rr:constant ex:name; ]; rr:objectMap [ rr:column "LNAME"; ]; ];
13
14    rr:predicateObjectMap [
15      rr:predicateMap [ rr:constant ex:fullname; ]; rr:objectMap [ rr:template "{LNAME},{FNAME}"; ]; ];
16
17    rr:predicateObjectMap [
18      rr:predicateMap [ rr:constant ex:worksIn; ];
19      rr:objectMap [ rr:parentTriplesMap <TMDepartment>;
20        rr:joinCondition [ rr:child "DEPTID" ; rr:parent "DID" ; ];
21      ];
22    ];
23  ].
24
25  <TMEmployee2> a rr:TriplesMap;
26    rr:logicalTable [ rr:tableName "EMP" ];
27
28    rr:subjectMap [ rr:class ex:Employee; rr:template "http://ex.com/Person/{EID}"; ];
29
30    rr:predicateObjectMap [
31      rr:predicateMap [ rr:constant ex:name; ]; rr:objectMap [ rr:column "FNAME"; ]; ];
32
33    rr:predicateObjectMap [
34      rr:predicateMap [ rr:constant ex:fullname; ]; rr:objectMap [ rr:template "{LNAME},{FNAME}"; ]; ];
35
36    rr:predicateObjectMap [
37      rr:predicateMap [ rr:constant ex:worksIn; ];
38      rr:objectMap [ rr:parentTriplesMap <TMDepartment>;
39        rr:joinCondition [ rr:child "DEPTID" ; rr:parent "DID" ; ];
40      ];
41    ];
42  ].
43
44  <TMDepartment> a rr:TriplesMap;
45    rr:logicalTable [ rr:tableName "DEPT" ];
46
47    rr:subjectMap [ rr:class ex:Department; rr:template "http://ex.com/Department/{DID}{DNAME}"; ];
48
49    rr:predicateObjectMap [
50      rr:predicateMap [ rr:constant ex:name ]; rr:objectMap [ rr:column "DNAME"; ]; ];
51  ].

```

LISTING 5.2: Example of R2RML Mapping Document in 1st NF

R2RML mappings in the first normal form can be transformed into equivalent R2RML mappings in the second normal form with the following procedure:

1. Replace a Triples Map *TM* having *n* numbers of `rr:predicateObjectMap` mappings into *n* Triples Maps each having one `rr:predicateObjectMap` mapping.

The R2RML mapping document shown in Listing 5.3 is the equivalent 2nd normal form of the mapping in Listing 5.1.

```

1  @prefix rr: <http://www.w3.org/ns/r2rml#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix ex: <http://www.example.com#> .
5
6  <TMEmployee1> a rr:TriplesMap;
7    rr:logicalTable [ rr:tableName "EMP" ];
8    rr:subjectMap [ rr:class ex:Employee;
9      rr:template "http://ex.com/Person/{EID}";
10   ];
11   rr:predicateObjectMap [
12     rr:predicateMap [ rr:constant ex:name ];
13     rr:objectMap [ rr:column "LNAME"; ];
14   ];
15
16  <TMEmployee2> a rr:TriplesMap;
17    rr:logicalTable [ rr:tableName "EMP" ];
18    rr:subjectMap [ rr:class ex:Employee;
19      rr:template "http://ex.com/Person/{EID}";
20   ];
21   rr:predicateObjectMap [
22     rr:predicateMap [ rr:constant ex:name ];
23     rr:objectMap [ rr:column "FNAME"; ];
24   ];
25
26  <TMEmployee3> a rr:TriplesMap;
27    rr:logicalTable [ rr:tableName "EMP" ];
28    rr:subjectMap [ rr:class ex:Employee;
29      rr:template "http://ex.com/Person/{EID}";
30   ];
31   rr:predicateObjectMap [
32     rr:predicateMap [ rr:constant ex:name ];
33     rr:objectMap [ rr:template "{LNAME},{FNAME}"; ];
34   ];
35
36  <TMEmployee4> a rr:TriplesMap;
37    rr:logicalTable [ rr:tableName "EMP" ];
38    rr:subjectMap [ rr:class ex:Employee;
39      rr:template "http://ex.com/Person/{EID}";
40   ];
41   rr:predicateObjectMap [
42     rr:predicateMap [ rr:constant ex:worksIn; ];
43     rr:objectMap [
44       rr:parentTriplesMap <TMDepartment>;
45       rr:joinCondition [
46         rr:child "DEPTID" ; rr:parent "DID" ;
47       ];
48     ];
49
50  <TMDepartment1> a rr:TriplesMap;
51    rr:logicalTable [ rr:tableName "DEPT" ];
52    rr:subjectMap [ rr:class ex:Department;
53      rr:template "http://ex.com/Department/{DID}{DNAME}";
54   ];
55   rr:predicateObjectMap [
56     rr:predicateMap [ rr:constant ex:name ];
57     rr:objectMap [ rr:column "DNAME"; ];
58   ];

```

LISTING 5.3: Example of R2RML Mapping Document in 2nd NF

5.1.2.3 3rd Normal Form (3NF)

The 3rd Normal Form extends the 2NF Normal Form so that `rr:RefObjectMap` is not used in any predicate-object mappings and no subject map contains `rr:class C`.

R2RML mappings in the second normal form can be transformed into equivalent R2RML mappings in the third normal form with the following procedure:

1. Replace a Triples Map TM having m numbers of `rr:class ex:SomeClass` mappings into m Triples Maps each having one `rr:predicateObjectMap` mapping with `rdf:type` in the predicate map $rr : predicateMap$ and `ex:SomeClass` in the object map $rr : objectMap$.
2. Replace a Triples Map TM having `rr:refObjectMap` that contains a join to parent Triples Map TM_{Parent} with another Triples Map TM' whose logical table is: `SELECT ChildColumns, ParentColumns FROM TM.LogicalTable WHERE JoinCondition`, where *ChildColumns* and *ParentColumns* are columns that referred in TM and TM_{Parent} respectively, and *JoinCondition* is the join condition specified in the Reference Object Map of TM .

The R2RML mapping document shown in Listing 5.4 is the 3rd normal form of the mapping in Listing 5.1.

In the 3rd normal form, there are 27 possible combinations of mappings. Each of subject-map, predicate-map, or object-map can take three possible mappings (constant, column, or template).

5.1.2.4 4th Normal Form (4NF)

Finally, we present the fourth level of the normalization, the 4th Normal Form (4NF). In this normal form, we use only template-valued mapping, as any constant-valued or column-valued mapping can be transformed into a template mapping. The procedure to transform any R2RML mappings in the third normal form into the fourth normal form is as follows:

1. Replace a constant mapping in the form of (`rr:constant "someConstant"`) with a template mapping in the form of (`rr:template "someConstant"`).
2. Replace a column mapping in the form of (`rr:column "someColumn"`) with a template mapping in the form of (`rr:template "{someColumn}"`).

The R2RML mapping document shown in Listing 5.5 is the equivalent 4th normal form of the mapping in Listing 5.1.

```

1  @prefix rr: <http://www.w3.org/ns/r2rml#> .
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4  @prefix ex: <http://www.example.com#> .
5
6  <TMEmployee1> a rr:TriplesMap;
7    rr:logicalTable [ rr:tableName "EMP" ];
8    rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
9    rr:predicateObjectMap [
10     rr:predicateMap [rr:constant rdf:type]; rr:objectMap [rr:constant ex:Employee]; ];
11
12  <TMEmployee2> a rr:TriplesMap;
13    rr:logicalTable [ rr:tableName "EMP" ];
14    rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
15    rr:predicateObjectMap [
16     rr:predicateMap [ rr:constant ex:name ]; rr:objectMap [ rr:column "LNAME"; ]; ];
17
18  <TMEmployee3> a rr:TriplesMap;
19    rr:logicalTable [ rr:tableName "EMP" ];
20    rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
21    rr:predicateObjectMap [
22     rr:predicateMap [ rr:constant ex:name ]; rr:objectMap [ rr:column "FNAME"; ];
23    ]; ];
24
25  <TMEmployee4> a rr:TriplesMap;
26    rr:logicalTable [ rr:tableName "EMP" ];
27    rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
28    rr:predicateObjectMap [
29     rr:predicateMap[rr:constant ex:name]; rr:objectMap[rr:template "{LNAME},{FNAME}"; ]; ];
30
31  <TMEmployee5> a rr:TriplesMap;
32    rr:logicalTable [ rr:sqlQuery """
33    SELECT EMP.EID, DEPT.DID, DEPT.DNAME
34    FROM EMP JOIN DEPT ON EMP.DEPTID = DEPT.DID""" ];
35    rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
36    rr:predicateObjectMap [ rr:predicateMap [rr:constant ex:worksIn];
37     rr:objectMap [ rr:template "http://ex.com/Department/{DID}{DNAME}"; ];
38    ];
39  ].
40
41  <TMDepartment1> a rr:TriplesMap;
42    rr:logicalTable [ rr:tableName "DEPT" ];
43    rr:subjectMap [ rr:template "http://ex.com/Department/{DID}{DNAME}"; ];
44    rr:predicateObjectMap [
45     rr:predicateMap [rr:constant rdf:type]; rr:objectMap [rr:constant ex:Department]; ];
46  ].
47
48  <TMDepartment2> a rr:TriplesMap;
49    rr:logicalTable [ rr:tableName "DEPT" ];
50    rr:subjectMap [ rr:template "http://ex.com/Department/{DID}{DNAME}"; ];
51    rr:predicateObjectMap [
52     rr:predicateMap [rr:constant ex:name]; rr:objectMap [rr:column "DNAME"; ]; ];
53  ].

```

LISTING 5.4: Example of R2RML Mapping Document in 3rd NF

5.1.3 $R2RML_{lite}$ 4NF in Datalog

Because the 4NF $R2RML_{lite}$ uses only template map, then every Triples Map has the form of:

```

<TM> a rr:TriplesMap;
rr:logicalTable [ rr:tableName Alpha ];

```

```

1 <TMEmployee1> a rr:TriplesMap;
2   rr:logicalTable [ rr:tableName "EMP" ];
3   rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
4   rr:predicateObjectMap [
5     rr:predicateMap [rr:template "rdf:type"]; rr:objectMap [rr:template "ex:Employee"];
6   ];
7 ];
8
9 <TMEmployee2> a rr:TriplesMap;
10  rr:logicalTable [ rr:tableName "EMP" ];
11  rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
12  rr:predicateObjectMap [
13    rr:predicateMap [ rr:template "ex:name" ]; rr:objectMap [ rr:template "{LNAME}"; ];
14 ];
15
16 <TMEmployee3> a rr:TriplesMap;
17  rr:logicalTable [ rr:tableName "EMP" ];
18  rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
19  rr:predicateObjectMap [
20    rr:predicateMap [ rr:template "ex:name" ]; rr:objectMap [ rr:template "{FNAME}"; ];
21 ];
22 ];
23
24 <TMEmployee4> a rr:TriplesMap;
25  rr:logicalTable [ rr:tableName "EMP" ];
26  rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
27  rr:predicateObjectMap [
28    rr:predicateMap [ rr:template "ex:fullName" ]; rr:objectMap [ rr:template "{FNAME} {LNAME}"; ];
29 ];
30 ];
31
32 <TMEmployee5> a rr:TriplesMap;
33  rr:logicalTable [ rr:sqlQuery """
34    SELECT EMP.EID, DEPT.DID, DEPT.DNAME
35    FROM EMP JOIN DEPT ON EMP.DEPTID = DEPT.DID""" ];
36  rr:subjectMap [ rr:template "http://ex.com/Person/{EID}"; ];
37  rr:predicateObjectMap [
38    rr:predicateMap [rr:template "ex:worksIn"];
39    rr:objectMap [ rr:template "http://ex.com/Department/{DID}" ];
40 ];
41 ];
42
43 <TMDepartment1> a rr:TriplesMap;
44  rr:logicalTable [ rr:tableName "DEPT" ];
45  rr:subjectMap [ rr:template "http://ex.com/Department/{DID}"; ];
46  rr:predicateObjectMap [
47    rr:predicateMap [rr:template "rdf:type"]; rr:objectMap [rr:template "ex:Department"];
48 ];
49 ];
50
51 <TMDepartment2> a rr:TriplesMap;
52  rr:logicalTable [ rr:tableName "DEPT" ];
53  rr:subjectMap [ rr:template "http://ex.com/Department/{DID}"; ];
54  rr:predicateObjectMap [
55    rr:predicateMap [rr:template "ex:name"]; rr:objectMap [rr:template "{DNAME}"];
56 ];
57 ];

```

LISTING 5.5: Example of R2RML Mapping Document in 4th NF

```

rr:subjectMap [ rr:template SubjectBase/{SubjectColumn} ];
rr:predicateObjectMap [
rr:predicateMap [ rr:template PredicateBase/{PredicateColumn} ];
rr:objectMap [ rr:template ObjectBase/{ObjectColumn} ];
];

```


];

Mappings in 4NF can be associated to the datalog predicate `triple`. Before we continue further in this direction, we describe some auxiliary datalog predicates that we will use to represent R2RML mappings.

- `CONCAT(B, C, T)` holds if `T` is the result of concatenating `B` with `C`.
- `VALUE(v, a, t, r)` is used to store the tuples in an relational instance I of a relational schema \mathbf{R} . `VALUE(v, a, t, r)` indicates that v is the value of an attribute a in a tuple with identifier t in a relation r (that belongs to \mathbf{R}); e.g. a tuple $t1$ of table `EMP` such that `t1.EMPNO = "1"` and `t1.ENAME = "John"` is stored by using the facts `VALUE("1","EMPNO","id1","EMP")` and `VALUE("John","ENAME","id1","EMP")`, assuming that `id1` is the identifier of tuple $t1$.

With the predicates defined above, we are now ready to associate any 4NF $R2RML_{lite}$ mapping with the following datalog rule:

```
triple(S, P, O) :-
  CONCAT(SubjectBase, SubjectColumn, SubjectTemplate),
  VALUE(SubjectColumnValue, SubjectColumn, t, Alpha),
  CONCAT(SubjectBase, SubjectColumnValue, S),
  CONCAT(PredicateBase, PredicateColumn, PredicateTemplate),
  VALUE(PredicateColumnValue, PredicateColumn, t, Alpha),
  CONCAT(PredicateBase, PredicateColumnValue, P),
  CONCAT(ObjectBase, ObjectColumn, ObjectTemplate),
  VALUE(ObjectColumnValue, ObjectColumn, t, Alpha),
  CONCAT(ObjectBase, ObjectColumnValue, O),
```

EMP

<u>PK</u>				->DEPT(DID)
<u>EID</u>	LNAME	FNAME	JOB	DEPTID
<u>7369</u>	SMITH	ADAM	CLERK	10

DEPT

<u>PK</u>		
<u>DID</u>	DNAME	LOC
<u>10</u>	APPSERVER	NEW YORK

FIGURE 5.2: Relational Schema \mathbf{R} and its Instance I

Example 5.1. Consider the R2RML Mappings in Listing 5.5. Given a database instance in Figure 5.2, the following predicates will be generated.

```
triple(":Person/7369", "rdf:type", "ex:Employee").
triple(":Person/7369", "ex:name", "SMITH").
triple(":Person/7369", "ex:name", "ADAM").
triple(":Person/7369", "ex:worksIn", ":Department/10").
triple(":Department/10", "rdf:type", "ex:Department").
triple(":Department/10", "ex:name", "APPSERVER").
```

In Section 5.3 we will continue the example by generating a view such that the application of Direct mapping over that view will generate the equivalent datalog predicates above.

5.2 Fundamental Properties

In this section we define two relevant properties that we will use in the rest of the chapter: information preservation property and query result preservation property. They are necessary to define whether two mappings have the same expressive power. For this, we need to start defining what we understand by IRI replacement.

We define an IRI replacement, or simply *replacement*, $\sigma : IRI \rightarrow IRI$ as a mapping between two IRIs. For example, applying the replacement $\sigma = \{\text{foaf:givenName} \mapsto \text{foaf:firstName}\}$ to an RDF triple $(:x \text{ foaf:givenName } ?y)$ will produce an output $(:x \text{ foaf:firstName } ?y)$.

Originally, the direct mapping \mathcal{DM} rules are applied over a relational schema \mathcal{R} and its instance I . For our purpose, we also allow a view as the input for the direct mapping. We denote by $\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I}$ the direct graph resulting from the application of \mathcal{DM} rules over $\llbracket V \rrbracket_I$ (the result of applying a view V over I).

Definition 5.1 (Information preservation). A view V over a relational schema \mathbf{R} is information preserving with respect to an R2RML mapping document \mathcal{M} if for every instance I of \mathbf{R} , there exists a *replacement* σ , such that $\llbracket \mathcal{M} \rrbracket_I = \sigma(\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I})$.

Intuitively, a view is information preserving with respect to an R2RML mapping document over a relational instance if from the direct graph, there is a way to obtain the result from the R2RML output dataset.

Definition 5.2 (Query result preservation). A view V is query result preserving with respect to an R2RML mapping document \mathcal{M} if for every SPARQL query P_1 , there exists

a SPARQL query P_2 , such that for every instance I of \mathbf{R} , there exists a *replacement* σ satisfying the following condition: $\llbracket P_1 \rrbracket_{\llbracket \mathcal{M} \rrbracket_I} = \sigma(\llbracket P_2 \rrbracket_{\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I}})$.

Intuitively, a view definition is query result preserving with respect to an R2RML mapping document over a relational instance if every query over the R2RML output dataset can be translated into an *equivalent query* over the direct graph. Two SPARQL queries are equivalent if they return the same mappings.

Definition 5.3. An R2RML mapping \mathcal{M} over a relational schema R and its instance I has the same expressive power as the Direct Mapping \mathcal{DM} over a view V over R if V satisfies the information preservation and query result preservation properties with respect to \mathcal{M} .

5.3 $R2RML_{lite}$ to Direct Mapping

Theorem 5.4. For every relational schema R and every $R2RML_{Lite}$ mapping \mathcal{M} over R , there exists a view V over R so that the Direct Mapping \mathcal{DM} over V has the same expressive power as \mathcal{M} .

Proof. Given an R2RML mapping document \mathcal{M} and a relational schema \mathbf{R} , our problem consists in generating a view V , such that the information preservation and query result preservation properties hold. This problem can be decomposed into two problems: Information Preservation Problem (IPP) and Query Result Preservation Problem (QRPP). \square

Definition 5.5 (Information Preservation Problem). Given a relational schema \mathbf{R} and an R2RML mapping document \mathcal{M} , find a view V and a *replacement* σ that satisfy the Information Preservation Property.

Definition 5.6 (Query Result Preservation Problem). Given a relational schema \mathbf{R} and an R2RML mapping document \mathcal{M} , and a SPARQL query P_1 , find a view V , a *replacement* σ , and a SPARQL query P_2 that satisfy the Query Result Preservation Property.

5.3.1 Approach for solving IPP and QRPP

5.3.1.1 Solving IPP

Our proposal for generating V is to associate every Triples Map in an R2RML mapping document to a SQL query, and union them in such a way that the application of Direct

Mapping rules over V can produce triples that are semantically equivalent to the ones generated by the application of R2RML rules over I .

Each of the Triples Map in 4NF corresponds to the following SQL query.

```
SELECT (SubjectBase || SubjectColumn) AS rdf:subject
(PredicateBase || PredicateColumn) AS rdf:predicate
(ObjectBase || ObjectColumn) AS rdf:object
FROM Alpha
WHERE ObjectColumn IS NOT NULL
```

```
1 CREATE VIEW rdf:Statement AS (
2   -- SQL query corresponds to TMEmployee1
3   SELECT ("http://ex.com/Person/" || EID) AS rdf:subject
4     , ("rdf:type" || "") AS rdf:predicate
5     , ("ex:Employee" || "") AS rdf:object
6   FROM EMP
7
8   -- SQL query corresponds to TMEmployee2
9   SELECT "http://ex.com/Person/" || EID) AS rdf:subject
10     , ("ex:name" || "") AS rdf:predicate
11     , (" " || LNAME) AS rdf:object
12   FROM EMP WHERE LNAME IS NOT NULL
13
14   -- SQL query corresponds to TMEmployee3
15   SELECT ("http://ex.com/Person/" || EID) AS rdf:subject
16     , ("http://ex.com/name" || "") AS rdf:predicate
17     , (" " || FNAME) AS rdf:object
18   FROM EMP WHERE FNAME IS NOT NULL
19
20   -- SQL query corresponds to TMEmployee4
21   SELECT ("http://ex.com/Person/" || EID) AS rdf:subject
22     , ("http://ex.com/name" || "") AS rdf:predicate
23     , (LNAME || FNAME) AS rdf:object
24   FROM EMP WHERE LNAME IS NOT NULL AND FNAME IS NOT NULL
25
26   -- SQL query corresponds to TMEmployee5
27   -- simplified version with subquery eliminated
28   SELECT ("http://ex.com/Person/" || EID) AS rdf:subject
29     , ("http://ex.com/worksIn" || "") AS rdf:predicate
30     , ("http://ex.com/Department/" || DID) AS rdf:object
31   FROM EMP JOIN DEPT ON DEPTID = DID
32   WHERE EMP.DEPTID IS NOT NULL
33
34   -- SQL query corresponds to TMDepartment1
35   SELECT ("<http://ex.com/Department/" || DID) AS rdf:subject
36     , ("rdf:type" || "") AS rdf:predicate
37     , ("http://ex.com/Department" || "") AS rdf:object
38   FROM DEPT
39
40   -- SQL query corresponds to TMDepartment2
41   SELECT ("http://ex.com/Department/" || DID) AS rdf:subject
42     , ("ex:name" || "") AS rdf:predicate
43     , (" " || DNAME) AS rdf:object
44   FROM DEPT WHERE DNAME IS NOT NULL
45 );
```

LISTING 5.6: SQL queries that correspond to the R2RML Mapping Document in Listing 5.5

Then, we define the view $V = \text{CREATE VIEW rdf:Statement AS } Q$, where Q is the UNION of SQL queries correspond to all Triples Maps in R2RML Mapping Document. Listing 5.6 shows the view V queries correspond to the R2RML mapping document defined in Listing 5.5 and the application of such view over I in Figure 5.2 can be seen in Table 5.1. The result of applying the direct mapping rules over $\llbracket V \rrbracket_I$ can be seen in Listing 5.7.

TABLE 5.1: $\llbracket V \rrbracket_I$, the result of applying view V in Listing 5.6 over I in Figure 5.2

rdf:Statement		
rdf:subject	rdf:predicate	rdf:object
http://ex.com/Person/7369	rdf:type	http://ex.com/Employee
http://ex.com/Person/7369	http://ex.com/name	SMITH
http://ex.com/Person/7369	http://ex.com/name	ADAM
http://ex.com/Person/7369	http://ex.com/fullname	ADAM SMITH
http://ex.com/Person/7369	http://ex.com/worksIn	http://ex.com/Department/10APPSERVER
http://ex.com/Department/10APPSERVER	rdf:type	http://ex.com/Department
http://ex.com/Department/10APPSERVER	http://ex.com/name	APPSERVER

```

1  .:a1 rdf:type rdf:Statement ;
2    rdf:Statement#rdf:subject <http://ex.com/Person/7369>;
3    rdf:Statement#rdf:predicate rdf:type;
4    rdf:Statement#rdf:object ex:Employee.
5
6  .:a2 rdf:type rdf:Statement ;
7    rdf:Statement#rdf:subject <http://ex.com/Person/7369>;
8    rdf:Statement#rdf:predicate ex:name;
9    rdf:Statement#rdf:object "SMITH".
10
11 .:a3 rdf:type rdf:Statement ;
12    rdf:Statement#rdf:subject <http://ex.com/Person/7369>;
13    rdf:Statement#rdf:predicate ex:name;
14    rdf:Statement#rdf:object "ADAM".
15
16 .:a4 rdf:type rdf:Statement ;
17    rdf:Statement#rdf:subject <http://ex.com/Person/7369>;
18    rdf:Statement#rdf:predicate ex:fullname;
19    rdf:Statement#rdf:object "ADAM SMITH".
20
21 .:a5 rdf:type rdf:Statement ;
22    rdf:Statement#rdf:subject <http://ex.com/Person/7369>;
23    rdf:Statement#rdf:predicate ex:worksIn;
24    rdf:Statement#rdf:object <http://ex.com/Department/10>.
25
26 .:b1 rdf:type rdf:Statement ;
27    rdf:Statement#rdf:subject <http://ex.com/Department/10>;
28    rdf:Statement#rdf:predicate rdf:type;
29    rdf:Statement#rdf:object ex:Department.
30
31 .:b2 rdf:type rdf:Statement ;
32    rdf:Statement#rdf:subject <http://ex.com/Department/10>;
33    rdf:Statement#rdf:predicate ex:name;
34    rdf:Statement#rdf:object "APPSERVER".

```

LISTING 5.7: $\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I}$

Given that $\sigma = \{ \text{rdf:Statement\#rdf:subject} \mapsto \text{rdf:subject},$
 $\text{rdf:Statement\#rdf:predicate} \mapsto \text{rdf:predicate},$
 $\text{rdf:Statement\#rdf:object} \mapsto \text{rdf:object} \}$. then $\llbracket \mathcal{M} \rrbracket_I =_{\text{reify}} \sigma(\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I})$, that
 is, $\llbracket \mathcal{M} \rrbracket_I$ is equal to $\sigma(\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I})$ under reification.

5.3.1.2 Solving QRPP

In a similar fashion, for solving the query result preservation problem, we also introduce the same view V . V and σ are defined as follows:

- $V = \text{CREATE VIEW rdf:Statement AS } Q, V = \text{CREATE VIEW rdf:Statement AS } Q$, where Q is then the UNION of all SQL queries correspond to all Triples Maps in R2RML Mapping Document.
- $\sigma = \{\}$
- $P_1 = \text{SELECT ?s ?p ?o WHERE } \{?s ?p ?o\}.$
- $P_2 = \text{SELECT ?s ?p ?o WHERE } \{$
 $\text{?st a rdf:Statement .}$
 $\text{?st rdf:subject ?s .}$
 $\text{?st rdf:predicate ?p .}$
 $\text{?st rdf:object ?o .}\}$

Thus, $\llbracket P_1 \rrbracket_{\llbracket \mathcal{M} \rrbracket_I} = \sigma(\llbracket P_2 \rrbracket_{\llbracket \mathcal{DM} \rrbracket_{\llbracket V \rrbracket_I}})$. Note that once we have the corresponding query P_2 for the triple pattern $(?s ?p ?o)$ in P_1 , the corresponding query for any graph patterns can be easily constructed using the triple pattern $(?s ?p ?o)$ as the building block.

Figure 5.3 and 5.4 illustrate our approach for solving the information preservation problem and query result preservation problem, respectively.

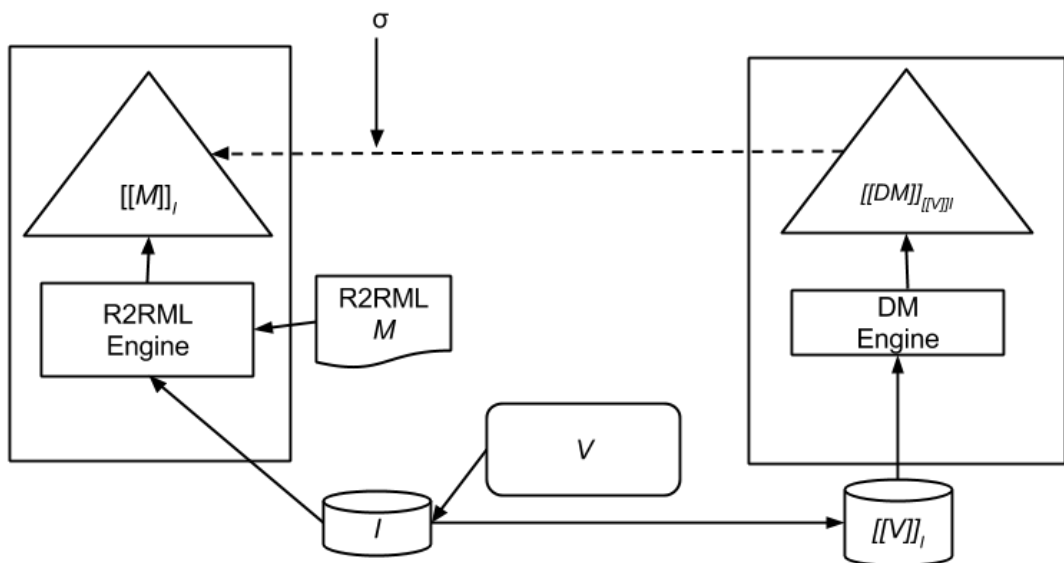


FIGURE 5.3: Approach for Solving the Information Preservation Problem

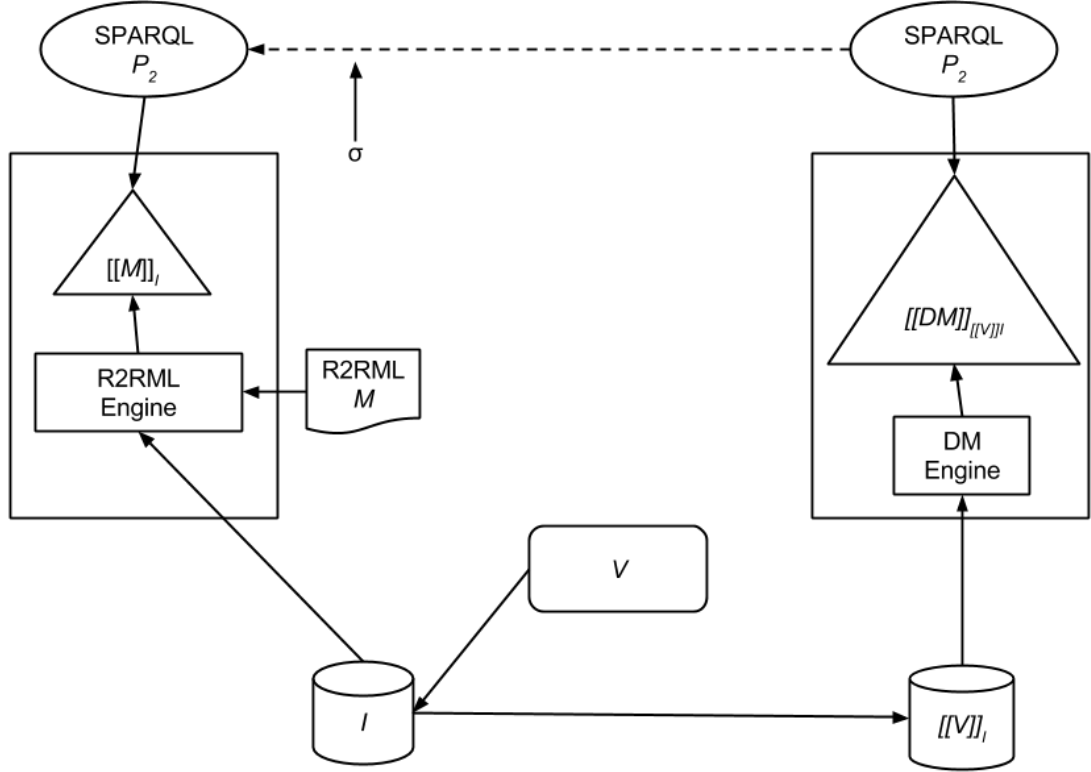


FIGURE 5.4: Approach for Solving the Query Result Preservation Problem

5.3.1.3 Note: Solving IPP with query

Using a slightly modified SPARQL query P_2 , we can actually solve the information preservation problem without the need of any *replacements* or reification. Consider:

- $P'_2 = \text{CONSTRUCT } ?s \ ?p \ ?o \ \text{WHERE } \{$
 $\quad ?st \ \text{a} \ \text{rdf:Statement} \ .$
 $\quad ?st \ \text{rdf:Statement\#rdf:subject} \ ?s \ .$
 $\quad ?st \ \text{rdf:Statement\#rdf:predicate} \ ?p \ .$
 $\quad ?st \ \text{rdf:Statement\#rdf:object} \ ?o \ . \}$
- $\sigma = \{\}$

Thus, $[[M]]_I = \sigma([P'_2]_{[[DM]]_{[[V]]_I}})$.

5.4 Conclusion

In this chapter we have focused on the relationship between two W3C Recommendations for the transformation of relational database content into RDF, namely R2RML and Direct Mapping. The reason for this work is the fact that there has not been yet any formal and comprehensive study so far of their relationship, while there are several system implementations for each of them. To facilitate our study, we have defined *R2RML_{lite}*, a fragment of R2RML that does not consider elements such as datatype, language tag, term type, or graph map. With *R2RML_{lite}*, we have validated our hypothesis H3 described in Chapter 3 as the fragment of R2RML that has the same expressive power with the Direct Mapping. Furthermore, we have used two fundamental properties that we consider relevant for this comparison: Information Preservation Property and Query Result Preservation Property. Using these properties, we have shown that *R2RML_{lite}* is as expressive as Direct Mapping, as long as we allow the generation of relational views and the replacement of IRIs. This validate our hypothesis H5 described in Chapter 3.

While the theoretical results obtained by the study that we present in this paper allow determining the relationships between both recommendations, we are aware that the approach proposed in this chapter to understand such relationship is not necessarily efficient in terms of the complexity of the queries that are generated, or the time required to evaluate the view definition queries. In fact, our study is not intended to be used to replace R2RML engines with direct mapping engines.

This chapter presents our study on the relationship between the Direct Mapping to R2RML, on the direction from the R2RML to the Direct Mapping. In the next chapter, we will see our study on the opposite direction, from the Direct Mapping to R2RML.

Chapter 6

MIRROR: An Automatic R2RML Mappings Generator

In Section 2.4.3, we have seen the Direct Mapping and R2RML as two W3C Recommendation on transforming content of relational databases into RDF data. Furthermore, in the previous chapter, we studied a particular fragment of R2RML that can be represented as the Direct Mapping. In this chapter, we discuss the relationship between the two recommendations in the opposite way: how do we represent the Direct Mapping transformation rules as R2RML mappings. Furthermore, because the input of the Direct Mapping is a database schema (together with an instance), we also see how we infer relationships implicitly encoded in a database schema.

We discussed that the Direct Mapping Recommendation specifies the terms generation rules to be applied to generate automatically an RDF dataset that reflects the structure and content of the relational database. Since this may not be always adequate or optimal, especially in those cases when the relational database content needs to be transformed into RDF according to an existing ontology, R2RML allows customising the terms generation rules to be applied.

Hence R2RML provides more flexibility than its counterpart, the Direct Mapping specification. However, this comes at a cost for users interested in generating RDF from their relational databases: they need to learn how to create those R2RML mappings. Several tools have been made available to facilitate the task of mapping generation, as discussed in Section 2.4.4, but either they produce mappings in earlier RDB2RDF languages (e.g. the ODEMapster GUI, which produces R2O mappings) or are not usable enough (e.g. form-based tools that only provide syntactic sugar to users, who still require a good knowledge of R2RML). An alternative approach to ease the burden of R2RML creation from users, making them more efficient, is to bootstrap the process with the creation of

Person	
ID (PK)	SSN
10	1234510
11	1234511
12	1234512

Contact		
CID (PK)	SID (FK)	Email
1	10	venus@hotmail.com
2	10	venus@gmail.com
3	11	fernando@yahoo.com
4	12	david@msn.com

Student		
ID (PFK)	FirstName	LastName
10	Venus	Williams
11	Fernando	Alonso
12	David	Villa

Student_Sport	
ID_Student (PFK)	ID_Sport (PFK)
10	110
11	111
11	112
12	111

Sport	
ID (PK)	Description
110	Tennis
111	Football
112	Formula1

FIGURE 6.1: Graphical Representation of D011B Database.

an initial R2RML mapping document that reflects the behaviour of the Direct Mapping specification, and then allow users to edit that document further, e.g. in a text editor. This has generally proven to be useful in our own work, since in many cases a large percentage of triple maps inside an R2RML mapping document are reused. This has also an additional positive side effect, which is the fact that any R2RML engine (e.g. morph-RDB) can be used to produce RDF following the Direct Mapping specification.

Furthermore, we have already pointed out that the Direct Mapping generates an RDF dataset that reflects the structure and content of the relational database. However, there are some well-known and widely-applied relational database patterns that usually encode some additional information that may be useful in this transformation process. For instance, some combinations of primary and foreign keys in relational tables are commonly used to represent parent-child, 1-N and M-N relationships between tables. This means that we may be able to push our approach further by generating as well some of that implicit information (such as subclass-of relationships, some specific object properties, etc.), in addition to the mappings generated following the Direct Mapping specification.

Let us see an example, which will be used throughout the rest of the paper. Consider the database D011 from the W3C Direct Mapping Test Cases, with tables: **Student**(ID, FirstName, LastName) and **Sport**(ID, Description), where ID is the primary key in both cases; and **Student_Sport**(ID_Student, ID_Sport), where both columns form a composite primary key, and where ID_Student is a foreign key that refers to the column ID of the table **Student** and ID_Sport is a foreign key that refers to the column ID of the table **Sport**.

The constraints specified by the primary/foreign keys in the table **Student_Sport** represent an M-N relationship between table **Student** and table **Sport**. Thus, when transforming this database into RDF, one can expect that there will be an object property, for example **hasStudent**, with **Sport** as its domain and **Student** as its range, or viceversa with object property **hasSport**.

Now we will make the following modifications to that database, and will name the resulting database D011B. Figure 6.1 provides its graphical representation.

- Add another table **Person**(ID, SSN), with ID as the primary key. We also add a foreign key constraint to the column ID of table **Student**, which refers to the column ID of the table **Person**. Then, we can see that there is a parent-child relationship, being the table **Person** the parent, and the table **Student** the child. This relationship implies that every property available in the parent is also inherited by the child, so that when we transform this database into RDF, the instances resulting from table **Student** will also have properties corresponding to the column SSN of the parent table **Person**.
- Add another table **Contact**(CID, SID, Email) with CID as the primary key and SID as the foreign key that refers to the column ID in the table **Student**. The relationship between **Student** and **Contact** is a 1-N relationship, so we may expect to have an object property generated for this relationship, being **Student** the domain and **Contact** the range.

The main contribution of this chapter is the design and implementation of an algorithm that takes as an input a relational database and generates as an output an R2RML mapping document that includes two groups of mappings. The first group of mappings encodes the transformations that would be done by a Direct Mapping engine, with the only exception that the generated RDF will differ in the URIs that are generated for some RDF nodes. The second group of mappings encodes an additional set of transformations that exploit the implicit information that is normally contained in relational databases (e.g. subclass-of relationships, M-N relationships) and which are not exposed by directly following the Direct Mapping approach.

The rest of the chapter is structured as follows. In Section 6.1 we present the core of our approach for the automatic generation of R2RML mappings from a relational database schema. In Section 6.2 we provide some conclusions on this paper and our planned future work. In Section 7.2 of the next chapter, we present some experiments applied to the set of test cases provided by the W3C RDB2RDF working group.

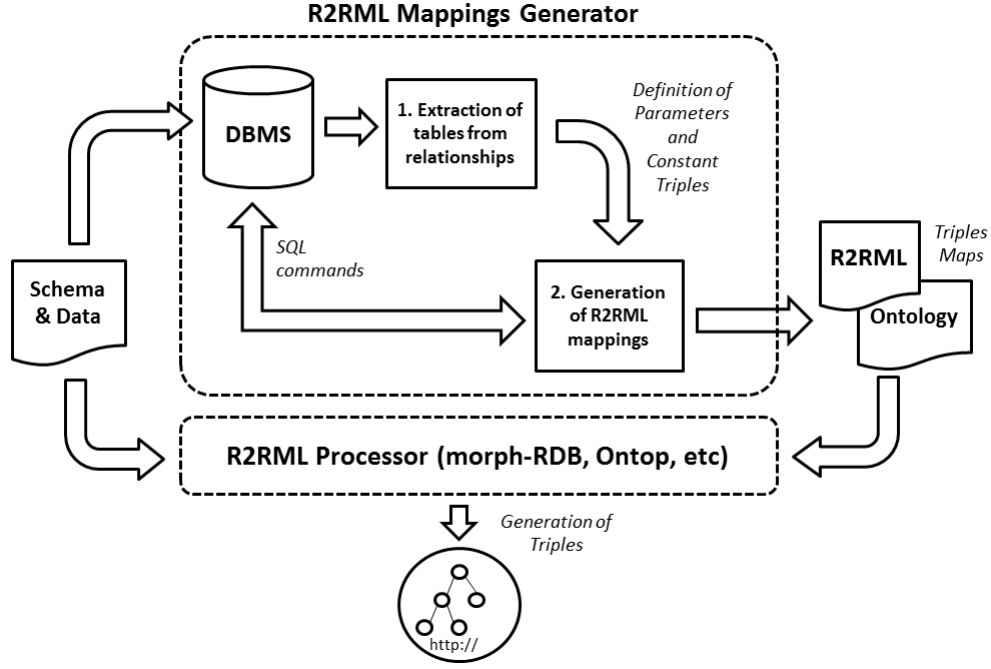


FIGURE 6.2: A general overview of the R2RML mapping generation process in MIRROR.

6.1 Automatic Generation of R2RML Mappings

Our process for automatically generating R2RML mappings from a relational database schema is depicted in Figure 6.2. The system receives as an input either the connection details to an existing database or a SQL file containing the database schema (represented by SQL DDL/DML statements). Then the process consists of two main steps:

1. **Identification of relationships between tables.** In this step, the relationships between tables are extracted, as well as their cardinality and the columns and constraints that are present in the database schema.
2. **Generation of R2RML mappings.** In this step, the R2RML triples maps that correspond to the patterns identified in the previous step are generated.

We consider two assumptions about the relational schema, so as to ensure completeness of the mapping process and preservation of information, as discussed in [SAM12]:

1. The relational schema must be normalized, at least, in third normal form (3NF).
2. Primary keys must be defined as not null and unique.

Next we discuss about the typical patterns that can be found in a relational database schema and our approach to convert them into R2RML mappings.

6.1.1 A Catalogue of Typical Patterns in Relational Schemas

A typical database modeling process considers three types of models: **conceptual** (where the elements are described using an Entity-Relationship or an Extended Entity-Relationship diagram), **logical** (which uses the relational model and is independent of the target database management system) and **physical** (which depends on the underlying database management system, defines how data is stored and declares constraints and keys).

While conceptual and logical models may in principle be the most adequate to understand the domain of a database, given their higher level of abstraction and technology independence, these models are not commonly available. Therefore, R2RML mapping generation algorithms need to be designed taking into account the information that can be obtained from the physical model: relations (tables) and relationships between them.

We have created a catalogue (see tables 6.1 and 6.2) describing nine types of relationships between two (or more) tables that may be found in the physical implementation of a relational database. We name these tables “parent“ and “child“. A parent table is the one that contains a primary key that is used as a foreign key by the child table.

6.1.1.1 Catalogue creation process

Figure 6.3 outlines the steps followed to generate our catalogue. First we consider all the possible pairs (16) that describes a relationship between two entities at the conceptual model, from the perspective of each entity: (Opt, 1), (Opt, N), (Mand, 1) and (Mand, N)¹. This list is then pruned as follows:

1. Using the commutative property (i.e., $1 - N \equiv N - 1$), the options are pruned to 10.
2. By assumption 2 (primary keys must be defined as not null and unique), options 10, 11, 12 and 16 are not allowed, because of the parent optional feature (Opt, X):
3. Finally, three special cases are added: i) **reciprocal relationships**, where one parent may be (optionally) related to only one child, and one child may be related to only one parent, respectively; ii) **self (or recursive) relationships**, where one instance of the parent may be related to another instance of the same parent; and iii) **n-ary relationships**, which involve many relationships, having many child tables connected to one parent table.

¹Opt and Mand refer to whether the relationship is optional or mandatory, and the second item refers to the maximum cardinality, which may be 1 or unspecified.

Steps	1	2	3	4	5	6	7	8
0) All Combinations	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Mand,N]	[Mand,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,1]	[Mand,N]	[Opt,1]	[Opt,N]
	Parent	[Opt,1]	[Opt,1]	[Opt,1]	[Opt,N]	[Opt,N]	[Opt,N]	[Opt,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,1]	[Mand,N]	[Opt,1]	[Opt,N]
1) After applying reflexive property	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Opt,1]	[Opt,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,N]	[Opt,N]	[Mand,N]	[Opt,N]
	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Opt,1]	[Opt,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,N]	[Opt,N]	[Mand,N]	[Opt,N]
2) After applying assumption 2	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Mand,N]	[Mand,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,N]	[Opt,N]	[Opt,N]	[Opt,N]
	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Mand,N]	[Mand,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,N]	[Opt,N]	[Opt,N]	[Opt,N]
3) After adding exceptions	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Mand,N]	[Mand,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,N]	[Opt,N]	[Opt,N]	[Opt,N]
	Parent	[Mand,1]	[Mand,1]	[Mand,1]	[Mand,N]	[Mand,N]	[Mand,N]	[Mand,N]
	Child	[Mand,1]	[Mand,N]	[Opt,1]	[Mand,N]	[Opt,N]	[Opt,N]	[Opt,N]

FIGURE 6.3: Obtaining the Catalogue for Guiding the R2RML Mapping Generation.

As a result, the catalogue is reduced to nine patterns, which need to be detected in the physical model of the database.

6.1.1.2 Catalogue description

These nine patterns are graphically depicted in tables 6.1 and 6.2, which shows how they are normally specified in the conceptual (already discussed), logical and physical models. In the **logical model** we specify the number of relations (tables) involved, as well as the type of relationship between them. In the **physical model** we describe whether null values are accepted in the key column (primary or foreign) of each relation.

TABLE 6.1: Correspondences between conceptual, logical and physical models (rows 1 to 5)

	Conceptual		Logical		Physical		Comments
	Parent	Child	Tables	Relationships	Parent	Child	
1	(Mand,1)	(Mand,1)	1	-	N	-	One table, no relationships
a	(Mand,1)	(Mand,1)	1	-	N	-	Used for subclass relationships (e.g. <i>Person</i> and <i>Student</i>)
2	(Mand,1)	(Opt,1)	2	1-N	N	Y	
3	(Mand,1)	(Opt,1)	2	1-N	N	Y	Reciprocal relationship
4	(Mand,1)	(Mand,N)	2	1-N	N	N	(e.g. <i>Student</i> and <i>Contact</i> , mandatory case)
a	(Mand,1)	(Opt,N)	2	1-N	N	Y	(e.g. <i>Student</i> and <i>Contact</i> , optional case)
b	(Mand,1)	(Opt,N)	3	M-N	N	Y	(e.g. <i>Student</i> , <i>Sport</i> and <i>Student_Sport</i> , if considering optional case)

Now we describe each of the rows of tables 6.1 and 6.2:

1. Rows 1 and 2a get transformed into a single table in the relational model (and hence also in the physical model).
2. Row 2b may be considered as a special case of row 5a. A special case of an IS-A relationship using two tables may be also considered here.
3. Row 3 indicates a reciprocal relationship case between two tables and it matches with twice 1:N case not mandatory, as row 5a also states.
4. Rows 7 and 8 can be considered as special cases using only two tables, settled in the row 9 as a more general form.
5. Row 6 describes a self-relationship.

The categories described next reflect the patterns of relationships that guide the SQL queries on the information schema that we will use in our algorithm, starting from, at least, two tables (patterns 1 and 2a, translated into only one table, are not considered):

TABLE 6.2: Correspondences between conceptual, logical and physical models (rows 6 to 9)

	Conceptual		Logical		Physical		Comments		
	Parent	Child	Tables	Relationships	Parent	Child			
6	a	(Mand, 1)	(Opt, N)	1	1-N	N	Y	<p>Parent ParentPK ParentCol1ParentCol2 ... ParentFK (1,1) (0,N)</p>	Self relationship (or recursive relationship)
	b			2	1-N	N	Y		
7	(Mand, N)	(Mand, N)	3	M-N	N	N		<p>Parent (1,1) ParentPK ParentCol1ParentCol2 ... (1,N) ParentFK ChildFK (1,N) Child (1,1) ChildPK ChildCol1 ChildCol2 ...</p>	(e.g. tables <i>Student</i> , <i>Sport</i> and <i>Student_Sport</i> , mandatory case)
8	(Mand, N)	(Opt, N)	3	M-N	N	Y		<p>Parent (1,1) ParentPK ParentCol1ParentCol2 ... (0,N) ParentFK ChildFK (0,N) Child (1,1) ChildPK ChildCol1 ChildCol2 ...</p>	Same as 5b
9	(Mand, N)	(Mand, N) or (Opt, N)	>=3	M-N	N	Y or N		<p>Child ChildPK ChildCol1 ChildCol2 ... (1,1) (X,N) Parent_Child1 (X,N) ParentFK Child1FK (1,1) Parent ParentFK ParentCol1ParentCol2 ... (X,N) Parent_ChildN ParentFK ChildNFK (X,N) Child (1,1) ChildNPK ChildNCol1ChildNCol2 ...</p>	N-ary relationships (i.e. many different combinations involving rows 7 and 8)

- 1:N, **optional** entity (rows 2b, 3, 5a, 6a and 6b).
- 1:N, **mandatory** entity (row 4)
- M:N having **2 tables**, optional or mandatory (rows 5b, 7 and 8)
- M:N having **more than 2 tables**, optional or mandatory (row 9)

6.1.2 Algorithms for the Generation of R2RML Mappings

Two different algorithms are proposed for R2RML mapping generation: one for 1:N relationships (Algorithm 6.1) and another one for M:N relationships (Algorithm 6.2).

In Algorithm 6.1 (Cardinality 1-N) the outer loop goes through all primary keys from the parent table, and executes three procedures that produce R2RML mapping components:

- *triplesMap*(n): it stores an ordered, auto-incremented triples maps, indexed by n , according to the template `<#TriplesMap{n}>`.
- *logicalTable*(RS): it stores the mapping component `rr:logicalTable` for the parent table RS .
- *subjectMap*(RS, KS): it stores the mapping component `rr:subjectMap`, taking in account the template `rr:template "http://IRI/RS/{KS}"`, where IRI is a parameter defined by the user.

The inner loop stores the mapping component `rr:predicateObjectMap`. It loops through all the columns that belong to the parent table (represented by argument *attr*(RS)). When the index n is incremented, the graph for the child table is generated, considering now:

- *triplesMap*(n): it has the same behaviour as for the parent table.
- *logicalTable*(RT): it stores the mapping component `rr:logicalTable` for the child table RT .
- *subjectMap*(RT, KT): it stores the mapping component `rr:subjectMap`, taking in account the template `rr:template "http://IRI/RT/{KT}"`.

After another inner loop with respect to the `rr:predicateObjectMap` for the child table, the algorithm registers the relationship, by means of the mapping component `rr:joinCondition`, linking the primary key KS from the parent table with the foreign key KT from the child table.

Algorithm 6.2 (Cardinality M-N) is different, since we use one more loop on all rows obtained from evaluation $\llbracket \phi \rrbracket_I$ (categories 3 and 4).

Algorithm 6.1 1-N Cardinality**Require:** $attr(\phi) = \{RS, KS, RT, KT\}$

```

1: if  $card(\llbracket \phi \rrbracket_I) = 1$  then
2:    $n = 1$ 
3:   for all  $KS$  do
4:      $triplesMap(n)$  ▷ Generates triples map for  $RS$  (parent table)
5:      $logicalTable(RS)$ 
6:      $subjectMap(RS, KS)$ 
7:     for all  $attr(RS)$  do
8:        $predicateObjectMap(attr(RS))$ 
9:     end for
10:     $n \leftarrow n + 1$ 
11:     $triplesMap(n)$  ▷ Generates triples map for  $RT$  (child table)
12:     $logicalTable(RT)$ 
13:     $subjectMap(RT, KT)$ 
14:    for all  $attr(RT)$  do
15:       $predicateObjectMap(attr(RT))$ 
16:    end for
17:     $joinCondition(KS, KT)$  ▷ Generates join condition
18:     $n \leftarrow n + 1$ 
19:  end for
20: end if

```

Algorithm 6.2 M-N Cardinality**Require:** $attr(\phi) = \{RS, KS, RT, KT\}$

```

1: if  $card(\llbracket \phi \rrbracket_I) > 1$  then
2:   for all tuples in  $\phi$  do
3:      $n = 1$ 
4:     for all  $KS$  do
5:        $triplesMap(n)$  ▷ Generates triples map for  $RS$  (parent table)
6:        $logicalTable(RS)$ 
7:        $subjectMap(RS, KS)$ 
8:       for all  $attr(RS)$  do
9:          $predicateObjectMap(attr(RS))$ 
10:      end for
11:       $n \leftarrow n + 1$ 
12:       $triplesMap(n)$  ▷ Generates triples map for  $RT$  (child table)
13:       $logicalTable(RT)$ 
14:       $subjectMap(RT, KT)$ 
15:      for all  $attr(RT)$  do
16:         $predicateObjectMap(attr(RT))$ 
17:      end for
18:       $joinCondition(KS, KT)$  ▷ Generates join condition
19:       $n \leftarrow n + 1$ 
20:    end for
21:  end for
22: end if

```

6.1.2.1 Subclass Identification

We use saturation to extend the set of R2RML mappings that has been initially created, exploiting subclass relationships that can be found in the database physical model. Unlike the work presented in [SAM14a], our work does not consider the use of an existing ontology to guide this saturation process. Our saturation approach considers two cases that can appear in the database physical model:

1. An IS-A relationship with cardinality 1-1 between a parent table and its child, having a common primary key table (row 2b from table 6.1).
2. An IS-A relationship with cardinality 1-N between a parent table and its child, becoming 1-1 after a data checking, testing whether any tuple in the parent table is related to only one tuple in the child table (it may happen with rows 4, 5a from 6.1, and 6a and 6b from table 6.2).

In these cases, the R2RML triple map for the child table is saturated with additional attributes from the parent table. An extra constant triple map is generated to feature explicitly the hierarchy, using `rdfs:subClassOf`.

6.1.2.2 Object Property Identification

Covering rows 5b from table 6.1; and 7, 8 and 9 from table 6.2, M-N relationships are represented by 3 tables. The binary table between parent and child tables, having the primary keys respectively as foreign keys, can be understood as an object property. Our R2RML mapping generator can create constant triples maps, templated as object property `ParentHasChild` and putting also its inverse, `ChildBelongsToParent`, using `owl:ObjectProperty`, `owl:inverseOf`, `rdfs:domain` and `rdfs:range`.

6.1.2.3 Datatype Property Identification

In the wake of the object properties handling, all columns of tables in the database schema are featured as datatype properties. The mapping generator creates constants triples maps (using `owl:DatatypeProperty`) considering the table to which the column belongs as the domain (using `rdfs:domain`), and the column data type as the range (using `rdfs:range`).

6.2 Conclusion

We have presented a set of algorithms and its corresponding implementations, for the automatic generation of R2RML mappings from a relational database schema. These mappings can be used by any R2RML processor to generate a set of RDF triples that is similar to those resulting from Direct Mapping. Several types of relationships between tables in a physical model have been categorised. By means of a core query and two algorithms that extract and organize this information, mappings are generated.

MIRROR has been integrated with morph-RDB, what allows experimenting with the generated R2RML mappings. In [Section 7.2](#) we will present the evaluation of MIRROR, which has been tested using the DM test cases suite, together with one extension of test cases, to cover more relationships.

Chapter 7

Experimentation

“There’s two possible outcomes: if the result confirms the hypothesis, then you’ve made a discovery. If the result is contrary to the hypothesis, then you’ve made a discovery.”

Enrico Fermi

In this chapter we describe the experimentations that we have performed to validate the hypotheses in Chapter 2. Recall that our hypotheses defined in the aforementioned chapter are:

- (H1) The formalisation of an existing query translation algorithm originally designed to work with non R2RML-aware systems, such as RDBMS-backed triple stores, can be extended to deal with R2RML-mapping based SPARQL to SQL query translation.
- (H2) The SQL queries that result from translating SPARQL query patterns using existing query translation algorithm are not necessarily efficient, what yields to poor performance in terms of query evaluation time and (H3) there are optimization techniques known in the relational database domain that can be applied to generate more efficient SQL queries.
- (H3) There are optimization techniques known in the relational database domain that can be applied to generate more efficient SQL queries.
- (H6 and H7) We can encode the Direct Mapping rules and the implicit knowledge encoded in a relational schema as R2RML.

This chapter is organized as follows. Hypotheses H1, H2 and H3 will be validated in Section 7.1, in which we report our results on evaluating morph-RDB with two groups of benchmark queries: synthetic and real cases. Hypotheses H6 and H7 will be validated in Section 7.2, in which we report our results on evaluating MIRROR with two types of databases: the Direct Mapping Test Suite provided by the W3C RDB2RDF Working Group which contains a set of databases covering varieties of database features, and an extension of one of the databases provided to cover all the implicit relationship that we discussed in Chapter 6.

7.1 Experimentation with morph-RDB

We separate our evaluations into two categories, using a synthetic BSBM benchmark (Section 7.1.1) and using three real projects (Sections 7.1.2 - 7.1.4). While the BSBM benchmark is considered as a standard way of evaluating RDB2RDF approaches, given the fact that it is very comprehensive, we were also interested in analysing real-world queries from projects that we had access to, and where there were issues with respect to the performance of the SPARQL to SQL query rewriting approach. In all the cases, we compare the queries generated by D2R Server, configured with `-fast` enabled, as this tool has been used by the projects prior to the time when we performed the evaluation. All the resulting queries together with their query plans are also available at <http://figshare.com/s/f55e5af27eee11e5b73f06ec4b8d1f61>. Their details are available in the corresponding sections. Our query translation algorithm has been implemented in our latest version of morph-RDB, which is available as a Scala open-source project in Github¹. The query translation types supported by morph-RDB are the naïve translation queries (C), which are the result of the query rewriting algorithm described in Section 4.1, together with three variants of it; with subquery elimination (SQE), self-join elimination (SJE), and both types of eliminations (SQE+SJE).

7.1.1 Evaluation with BSBM

The BSBM benchmark [BS08] focuses on the e-commerce domain and provides a data generation tool and a set of twelve SPARQL queries together with their corresponding SQL queries generated by hand. The data generator is able to generate datasets with different sizes containing entities normally involved in the domain (e.g., products, vendors, offers, reviews, etc). For the purpose of our benchmark, we work with the 100-million triple dataset configuration.

¹<https://github.com/oeg-upm/morph-rdb>

All these queries have been evaluated on the same machine, with the following configuration: Pentium E5200 2.5GHz processor, 4GB RAM, 320 GB HDD, and Ubuntu 13.04. The database server used for the synthetic benchmark queries is PostgreSQL 9.1.9. We normalize the evaluation time over the native evaluation time. We have run all queries with 20 times with different parameters, in warm mode run.

The BSBM SPARQL queries are designed in such a way that they contain different types of queries and operators, including SELECT/CONSTRUCT/DESCRIBE, OPTIONAL, UNION. In the same spirit, the corresponding SQL queries also consider various properties such as low selectivity, high selectivity, inner join, left outer join, and union among many others. Out of the 12 BSBM queries, we focus on all of the 10 SELECT queries (that is, we leave out DESCRIBE query Q09 and CONSTRUCT query Q12). We compare the native SQL queries (N), which are specified in the BSBM benchmark with the ones resulting from the translation of SPARQL queries generated by morph-RDB. Although not included here, we also evaluated those queries using D2R 0.8.1 with the *-fast* option enabled. The reason why we do not include here the results from these evaluations is because in many queries (such as in Q2, Q3, Q4, Q7, Q8, and Q11), D2R produces multiple SQL queries and then the join/union operations are performed at the application level, rather than in the database engine, what prevents us from doing direct comparisons. Nevertheless, this approach is clearly not scalable (e.g., in Q07 and Q08 the system returned an error while performing the operations, while the native and the translation queries could be evaluated over the database system).

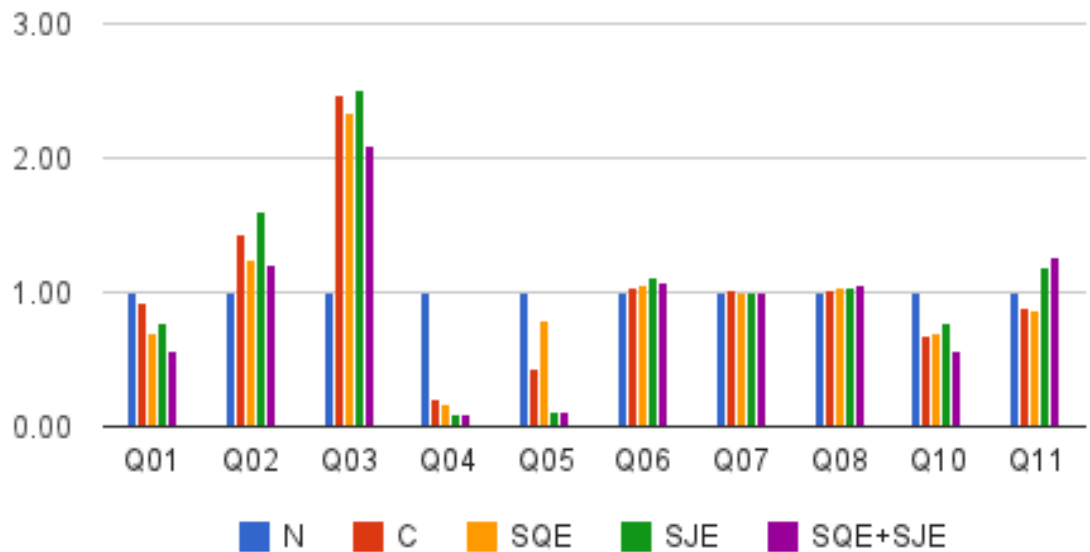


FIGURE 7.1: BSBM query evaluations (normalized time to native query). N=Native SQL, C=naïve translation queries, SQE=Subquery Elimination, SJE=Self-Join elimination, SQE+SJE=Subquery+Self-Join Elimination

7.1.1.1 Discussion

We can observe that all translation types (native, C, SQE, SJE, SQE+SJE) have similar performance in most of BSBM queries, ranging from 0.67 to 2.60 when normalized according to the native SQL queries. To understand this behaviour better, we analyzed the query plans generated by the RDBMS. Our observation tells us that in many of the queries (Q01, Q02, Q03, Q05, Q06, Q10, and Q11), C produces identical query plans to SQE's, and SJE produces identical query plans to SQE+SJE's. Additionally, SQE also produces identical query plans to SQE+SJE in Q07 and Q08. The reason for this is the capability of the database's optimizer to eliminate non-correlated subqueries. The difference between the query plans produced by C/SQE and the ones produced by SJE/SQE+SJE is the number of joins to be performed. However, as the join conditions are normally performed on indexed columns, there is small overhead in terms of their performance. This explains why all the translation results have similar performance.

However, in some queries the translation results show significant differences, such as in Q04 and Q05.

- BSBM SQL 4 contains a join between two tables (product and producttypeproduct) and three subqueries, two of them are used as OR operators. The SPARQL equivalent of this query is a UNION of two BGPs (a set of triple patterns). We note that the native query contains a correlated subquery and the generated query plan requires a table scan to find a specific row condition. The query plans generated by the translation algorithm, on the other hand, produce joins, instead of a correlated subquery, and the joins are able to exploit the indexes defined.
- BSBM SQL 5 is a join of four tables (product, product, productfeatureproduct, and productfeatureproduct). The size of table productfeatureproduct is significantly bigger than the table product (280K rows vs 5M rows). The generated query plan by the native query joins bigger tables (productfeatureproduct and productfeatureproduct) before joining the intermediate result with the smaller table (product and product). This join order is specified in the query itself. The join orders of the translation queries are different; C and SQE join based on the order of triple patterns in the graph, SJQ and SQE+SJE join based on the smaller tables first (which is an effect of the self-joins elimination process).

These explain why the translation queries perform better than the native queries in Q04 and Q05 and in fact show that the native queries proposed in the benchmark should have been better optimised when proposed for the benchmark.

7.1.2 Evaluation with RÉPENER Project

RÉPENER [SNMM13] is a Spanish national project that provides access to energy information using semantic technology, called SEiS². Two of the most common queries that are consulted through SEiS are the following:

- Q01 retrieves all buildings and their climatezone and building life cycle phase
- Q02 retrieves all buildings and their climatezone, building life cycle phase, and conditioned floor area.

The corresponding SPARQL queries are shown in Listing 7.1. All queries generated by D2R and morph-RDB are evaluated on a machine with the following specification: Intel Core 2 Quad Q9400 2.66 GHz, 4 GB RAM, Windows 7 Professional 32 bits, and MySQL 5.5 as the database server. As shown in Figure 7.2, all queries are successfully evaluated by the database server, with queries generated by morph-RDB being 2-3 faster than those generated by D2R server.

```

1  #Q01 Retrieve all buildings and their climatezone
2  # and building life cycle phase
3  SELECT DISTINCT *
4  WHERE {
5    ?a repener:hasBuilding ?building ; repener:value ?climatezone .
6    ?building a sumo:Building ; repener:hasProjectData ?projectData .
7    ?projectData repener:hasBuildingLifeCyclePhase ?buildingLifeCyclePhase .
8    ?buildingLifeCyclePhase repener:value ?phase .
9  }
10
11 #Q02. Retrieve all buildings and their building life cycle phase
12 # and conditioned floor area
13 SELECT DISTINCT *
14 WHERE {
15   ?b rdf:type sumo:Building ; repener:hasProjectData ?b1 .
16   ?b1 repener:hasBuildingLifeCyclePhase ?b2 .
17   ?b2 repener:value ?phase .
18   ?b repener:hasBuildingProperties ?b3 .
19   ?b3 repener:hasBuildingGeometry ?b4 .
20   ?b4 repener:hasConditionedFloorArea ?b5 .
21   ?b5 repener:conditionedFloorAreaValue ?conditionedFloorArea .
22 }
```

LISTING 7.1: SEiS queries

7.1.3 Evaluation with BizkaiSense Project

The BizkaiSense project is an effort to measure various environmental properties coming from sensors that are deployed throughout Greater Bilbao, Spain. Some of the most common queries that are needed in this project are:

²<http://arc.housing.salle.url.edu/repener/>

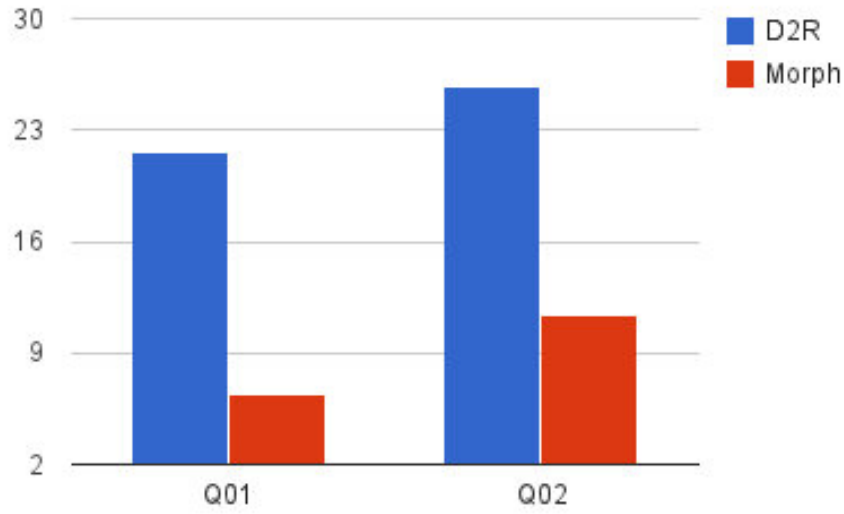


FIGURE 7.2: SEiS - query evaluation time (in seconds)

- Q01 obtains all observations coming from a particular weather or air quality station together with the time of the observation. We set 100, 1000, and 10000 as the maximum number of rows to be returned.
- Q02 extends Q01 by returning the sensor results generated from those observations obtained. We set 100, 1000, and 10000 as the maximum number of rows to be returned.
- Q03 returns the average measures by property for a given week in a given station (with the units of measure).
- Q04 returns the average measures by property for a given week in a given station (without the units of measure).
- Q05 returns the maximum measure in all the stations for each property in a given day (returning also the station in which it happened).
- Q06 returns the maximum measure in all the stations for each property in a given day (without the station in which it happened).
- Q07 returns the maximum measure in all the stations for a given property in a given day (returning the station in which it happened) -avoiding the use of the "MAX" clause.

The corresponding SPARQL queries can be seen in Listing 7.2. All queries generated by D2R and morph-RDB are evaluated on a machine with the following specification: Intel(R) Xeon(R) CPU E5640 2.67GHz x 16, 48GB RAM, Ubuntu 12.04 LTS - 64 bits

Representative query	Similar queries	TP	subjects	OPT	FILTER
Q01	1, 2, 5, 15, 19, 37, 41, 42	4	2	1	0
Q10	3, 10, 11, 40	25	9	21	4 (IN)
Q14	4, 6, 9, 14, 16, 17, 18, 19, 21, 35, 38, 39, 43	15	6	5	2 (IN, arithmetic)
Q34	22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 44	6	3	1	1 (IN)
Q45	7, 8, 12, 13, 20, 36, 45	14	5	4	2 (IN)

TABLE 7.1: Grouping of all queries according to the use case queries

and MySQL 5.5 with a 10 minutes timeout. The evaluation time of those queries (in some cases with a limit in the number of returned bindings) is shown in Figure 7.3. We note that the database server failed to evaluate the queries Q03, Q05, Q06, and Q07 generated from D2R Server and in all cases, the queries generated by morph-RDB take significantly less time to be evaluated compared to the ones generated by D2R Server.

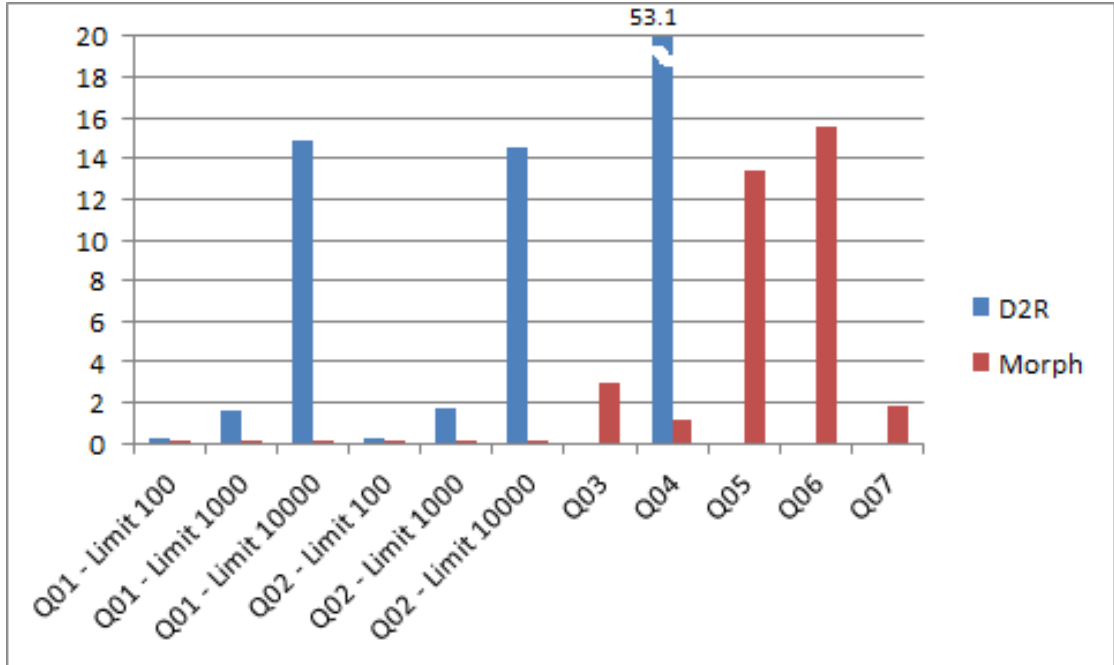


FIGURE 7.3: BizkaiSense - query evaluation time (in seconds)

7.1.4 Evaluation with Integrate Project

Integrate is an FP7 project for sharing and integrating clinical data using HL7-RIM [H⁺06], a model that represents entities and relationships commonly involved in clinical activities. We have collected a total of 45 SPARQL queries that are used in the patient recruitment and cohort selection scenario for breast cancer clinical trials. From this query list, we asked our domain experts to group the queries into a set of five groups that are representative of the whole set, and they are shown in Table 7.1.

The query characteristics of each the representative query are as follows:

```

1  #Q01 obtains all observations coming from a particular weather
2  # or air quality station together with the time of the observation.
3  SELECT DISTINCT ?medition ?date WHERE {
4  ?medition ssn:observedBy <http://localhost:2020/resource/station/ANORGA> .
5  ?medition dc:date ?date . } LIMIT 10 # 100, 1000
6
7  #Q02 extend Q01 by returning the sensor results generated
8  # from those observations obtained.
9  SELECT DISTINCT ?medition ?date ?res WHERE {
10 ?medition ssn:observedBy <http://localhost:2020/resource/station/ANORGA>.
11 ?medition dc:date ?date .
12 ?medition ssn:observationResult ?res . } LIMIT 10 # 100, 1000
13
14 #Q03 returns the average measures by property for a given week
15 # in a given station (with the units of measure).
16 SELECT (AVG(?datavalue) AS ?avg_month) ?prop ?unit WHERE {
17 ?medition ssn:observedBy <http://localhost:2020/resource/station/BETONO> ;
18 dc:date ?date ; ssn:observedProperty ?prop ; ssn:observationResult ?obsres .
19 ?obsres ssn:hasValue ?val .
20 ?val dul:hasDataValue ?datavalue ; dul:isClassifiedBy ?unit .
21 FILTER ( ?date >= "2011-01-01T00:00:00"^^xsd:date
22 && ?date <= "2011-01-07T00:00:00"^^xsd:date )
23 } GROUP BY ?prop ?unit
24
25 #Q04 returns the average measures by property for a given week
26 # in a given station (without the units of measure).
27 SELECT (AVG(?datavalue) AS ?avg_month) ?prop WHERE {
28 ?medition ssn:observedBy <http://localhost:2020/resource/station/BETONO> ;
29 dc:date ?date ; ssn:observedProperty ?prop ; ssn:observationResult ?obsres .
30 ?obsres ssn:hasValue ?val .
31 ?val dul:hasDataValue ?datavalue ; dul:isClassifiedBy ?unit .
32 FILTER ( ?date >= "2011-01-01T00:00:00"^^xsd:date
33 && ?date <= "2011-01-07T00:00:00"^^xsd:date )
34 } GROUP BY ?prop
35
36 #Q05 returns the maximum measure in all the stations for each property
37 # in a given day (returning also the station in which have happened).
38 SELECT (MAX(?datavalue) AS ?max) ?prop ?station WHERE {
39 ?medition a ssn:Observation ; dc:date ?date ; ssn:observedBy ?station ;
40 ssn:observedProperty ?prop ; ssn:observationResult ?obsres .
41 ?obsres ssn:hasValue ?val .
42 ?val dul:hasDataValue ?datavalue .
43 FILTER ( ?date >= "2011-01-01T00:00:00"^^xsd:date
44 && ?date < "2011-01-02T00:00:00"^^xsd:date )
45 } GROUP BY ?prop ?station
46
47 #Q06 returns the maximum measure in all the stations for each property
48 # in a given day (without the station in which have happened).
49 SELECT (MAX(?datavalue) AS ?max) ?prop WHERE {
50 ?medition a ssn:Observation ; dc:date ?date ; ssn:observedProperty ?prop ;
51 ssn:observationResult ?obsres .
52 ?obsres ssn:hasValue ?val .
53 ?val dul:hasDataValue ?datavalue .
54 FILTER ( ?date >= "2011-01-01T00:00:00"^^xsd:date
55 && ?date < "2011-01-02T00:00:00"^^xsd:date ) }
56
57 #Q07 returns the maximum measure in all the stations for a given property
58 # in a given day (returning the station in which have happened)
59 # –avoiding the use of "MAX" clause.
60 SELECT ?datavalue ?station WHERE {
61 ?medition a ssn:Observation ; dc:date ?date ;
62 ssn:observedBy ?station ; ssn:observationResult ?obsres .
63 ssn:observedProperty <http://sweet.jpl.nasa.gov/2.3/matrCompound.owl#NO>;
64 ?obsres ssn:hasValue ?val .
65 ?val dul:hasDataValue ?datavalue .
66 FILTER ( ?date >= "2011-01-01T00:00:00"^^xsd:date
67 && ?date < "2011-01-02T00:00:00"^^xsd:date )
68 } ORDER BY DESC(?datavalue) LIMIT 1

```

LISTING 7.2: BizkaiSense queries

- **Demographics query (Q01).** This query retrieves the information of all patients. It contains 4 triple patterns, 2 unique subjects, 1 triple pattern that is inside an OPTIONAL block, and 1 FILTER pattern.
- **Substance administration query (Q10).** This query retrieves the information of patients who were administered diphosphonate, including the information associated with the target site, the method used, and the approach site, if it exists. It consists of 35 triples patterns with 9 unique subjects. Most of the triple patterns are inside nested OPTIONAL blocks. There are 21 OPTIONAL blocks in this

query, some of which are nested under another OPTIONAL block. A FILTER pattern is used to filter results based on a certain condition.

- **Laboratory results query (Q14).** This query retrieves the information of patients who suffer anemia and whose body mass index is less than or equal to 30. It contains 15 triple patterns, with six unique subjects and 5 OPTIONAL patterns. Furthermore, some of the OPTIONAL blocks are nested inside a parent OPTIONAL block. This query also contains two FILTER patterns to filter results for particular code values and to perform some arithmetic calculations.
- **Procedure query (Q34).** This query retrieves the information of all patients who were administered chemotherapy. It consists of 6 triple patterns with one of them located inside an OPTIONAL pattern, and one FILTER pattern. There are 3 unique subjects in this query.
- **Observation query (Q45).** This query retrieves the information of patients who have been detected a category T2 breast tumor. It consists of 14 triple patterns and 5 unique subjects. There are 4 OPTIONAL patterns, one of them nested, and 2 FILTER patterns.

The machine used in our evaluation has the following specifications: CPU Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz, 8 GB of RAM, 750 GB HDD with Ubuntu Server 12.04 and MySQL Server 5.5. The dataset contains information of 3 months of historical clinical data, with 4674 patients and 65056 acts, among many other tables. The total size of the database is 105 MB. This database will be growing in the future, as more data is added as a result of the data integration processes carried out in the context of the projects where the database is being generated.

Figure 7.4 provides details for the five selected queries, which are also similar to the results obtained for the other queries in our query set. We can easily see that in most cases our total execution time is much lower than the one required for D2R Server. In some cases (queries Q14 and Q45) D2R Server was not able to produce results in less than five minutes.

We were also interested in how the SQL queries that result from the query rewriting approach perform in comparison to the SQL queries that would have been natively created by a SQL expert. For this reason, we asked a domain expert with good knowledge of the HL7 RIM relational database to construct SQL queries that were semantically equivalent to the corresponding SPARQL queries. In other words, without taking into account the mapping elements, such as template or URI generation, the SPARQL and SQL queries should return the same answer.

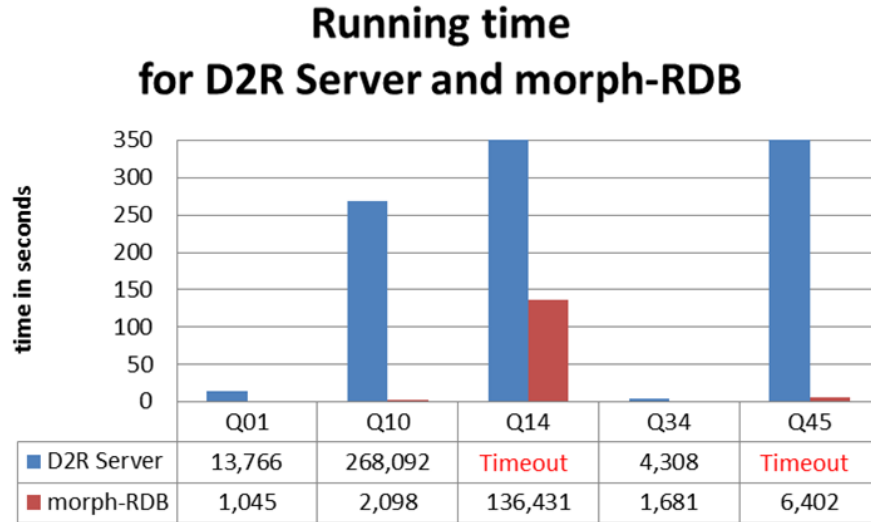


FIGURE 7.4: Running time for selected Integrate queries on morph-RDB and D2R (in seconds)

We evaluated each query 5 times in cold and warm modes. In the cold mode, we restart the server and empty the cache before we evaluate the next query. In the warm mode, we skip these steps and execute the queries directly one after the another. We measure the averages of query execution time and normalize the query evaluation time to the native query evaluation time. As an additional note, we can only do this type of evaluation using morph-RDB and native queries, as D2R Server produces multiple SQL queries in many cases and performs joins in memory, what makes it not comparable with the native or morph-RDB queries.

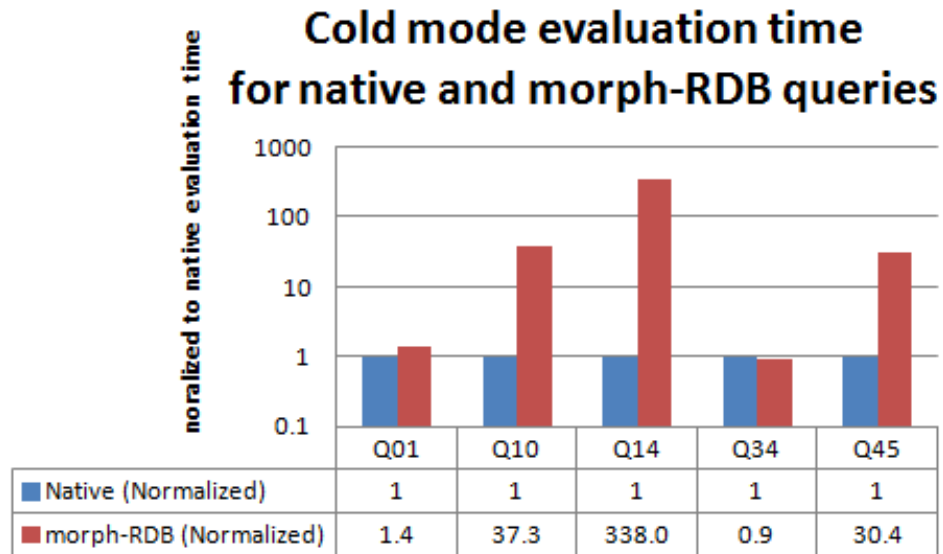


FIGURE 7.5: Integrate Queries Evaluation Time in Cold Mode

The results from both evaluation modes, which can be seen in Figures 7.5 and 7.6, show a similar trend. Furthermore, we observed that in the warm mode, the database

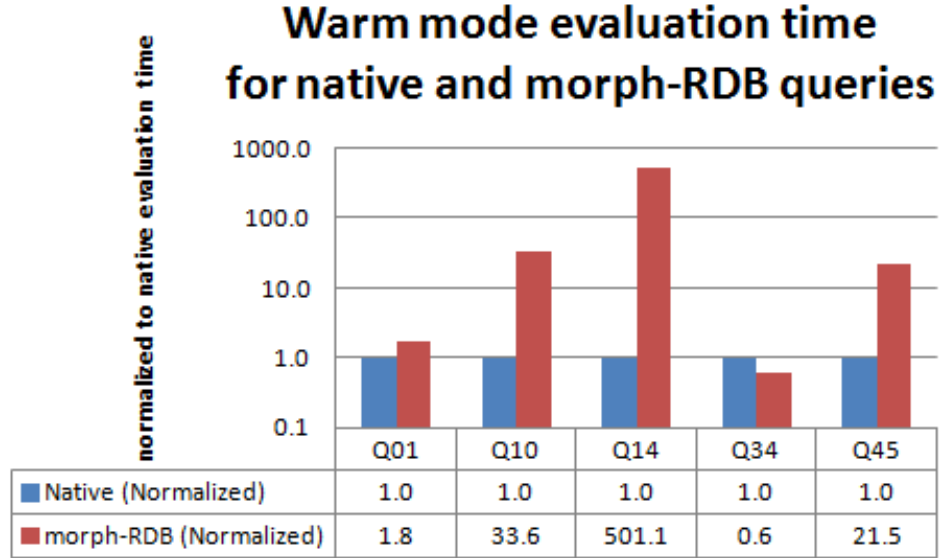


FIGURE 7.6: Integrate Queries Evaluation Time in Warm Mode

server doesn't lose its capability of reusing previous results of the query cache. This is reflected by the fact that only the first run of the query takes more time to complete, while subsequent queries can be evaluated with only a fraction of that time. Some of those queries produced by morph-RDB can be evaluated in a reasonable time. For example, the resulting query translation of query Q01 can be evaluated in a similar time as the native query Q01. Furthermore, the resulting query translation Q34 can be evaluated in less time than its corresponding native queries, which can be an indicator that there might be still room for improving the corresponding native query. Some other queries, such as Q10 and Q45, need more time to be evaluated, being in the range of 20-35x slower than the corresponding native queries, in which we still consider them as acceptable. The query Q14, however, needs more investigation, as it takes a lot of time to be evaluated, 380-500x slower in terms of normalized time to native queries. We suspect this is caused by the arithmetic operation that is performed over the resulting translation queries.

7.2 Experimentation with MIRROR

The algorithms described in Chapter 6 have been implemented in MIRROR³ (Mapping from Relational to Rdf generatOR) and have been integrated with morph-RDB⁴ [PCS14]. We have performed some experiments in order to show that our system can obtain R2RML mappings that encode the semantics in the W3C Direct Mapping specification,

³<https://github.com/oeg-upm/MIRROR>

⁴<https://github.com/oeg-upm/morph-rdb>

and furthermore that our system can generate R2RML mappings that also lift-up the implicit semantics encoded in the database.

We use two datasets in this experimentation: the set of databases provided in the Direct Mapping Test Cases, and we extend one of the databases (D011) to encode a parent-child relationship.

7.2.1 Experimentation using the Direct Mapping test cases

The Direct Mapping Test Cases⁵ is a test suite provided by the W3C RDB2RDF Working Group, which consists of a collection of test cases covering various database schemes such as databases without tables, databases with one table, with 1-N relationships, and with M-N relationships. In each of the test cases, the triples that can be expected as a result of applying Direct Mapping rules are provided. Thus, this test suite is a suitable source for us to evaluate our system, enabling us to see if our system produces the expected Direct Mapping results. In addition to that, we can also easily see the additional triples generated by the saturated mappings resulting from the identification of pattern relationships described in Chapter 6.

Here we discuss the database of Test Case D011, which consists of three tables **Student**, **Sport**, and **Student_Sport**. The table **Student_Sport** acts as a binary table that enables the M-N relationship between table **Student** and table **Sport**. The result of applying Direct Mapping rules over the database D011 can be seen in Listing 7.3.

When the same database schema and instance are passed to MIRROR, it generates the R2RML mappings shown in Listing 7.4.

Upon receiving these mappings, morph-RDB (or any other R2RML processor) generates a set of triples (see Listing 7.5) that correspond to the ones generated by Direct Mapping, hence we call them Direct Mapping triples.

In addition to the mappings above, additional mappings are also generated, as shown in Listing 7.6. These mappings produce the triples that can be seen in Listing 7.7, which specify the relationship between Student and Sport, which are not generated by the Direct Mapping specification.

7.2.2 Experimentation using D011B

For D011B, MIRROR generates the additional mappings shown in Listing 7.8:

⁵<http://www.w3.org/2001/sw/rdb2rdf/test-cases/>


```

1 <Student/ID=10> rdf:type <Student> .
2 <Student/ID=10> <Student#FirstName> "Venus" .
3 <Student/ID=10> <Student#ID> 10 .
4 <Student/ID=10> <Student#LastName> "Williams" .
5 <Student/ID=11> rdf:type <Student> .
6 <Student/ID=11> <Student#FirstName> "Fernando" .
7 <Student/ID=11> <Student#ID> 11 .
8 <Student/ID=11> <Student#LastName> "Alonso" .
9 <Student/ID=12> rdf:type <Student> .
10 <Student/ID=12> <Student#FirstName> "David" .
11 <Student/ID=12> <Student#ID> 12 .
12 <Student/ID=12> <Student#LastName> "Villa" .
13 <Student.Sport/ID_Student=10;ID_Sport=110> rdf:type <Student.Sport> .
14 <Student.Sport/ID_Student=10;ID_Sport=110> <Student.Sport#ID_Student> 10 .
15 <Student.Sport/ID_Student=10;ID_Sport=110> <Student.Sport#ref-ID_Student> <Student/ID=10> .
16 <Student.Sport/ID_Student=10;ID_Sport=110> <Student.Sport#ID_Sport> 110 .
17 <Student.Sport/ID_Student=10;ID_Sport=110> <Student.Sport#ref-ID_Sport> <Sport/ID=110> .
18 <Student.Sport/ID_Student=11;ID_Sport=111> rdf:type <Student.Sport> .
19 <Student.Sport/ID_Student=11;ID_Sport=111> <Student.Sport#ID_Student> 11 .
20 <Student.Sport/ID_Student=11;ID_Sport=111> <Student.Sport#ref-ID_Student> <Student/ID=11> .
21 <Student.Sport/ID_Student=11;ID_Sport=111> <Student.Sport#ID_Sport> 111 .
22 <Student.Sport/ID_Student=11;ID_Sport=111> <Student.Sport#ref-ID_Sport> <Sport/ID=111> .
23 <Student.Sport/ID_Student=11;ID_Sport=112> rdf:type <Student.Sport> .
24 <Student.Sport/ID_Student=11;ID_Sport=112> <Student.Sport#ID_Student> 11 .
25 <Student.Sport/ID_Student=11;ID_Sport=112> <Student.Sport#ref-ID_Student> <Student/ID=11> .
26 <Student.Sport/ID_Student=11;ID_Sport=112> <Student.Sport#ID_Sport> 112 .
27 <Student.Sport/ID_Student=11;ID_Sport=112> <Student.Sport#ref-ID_Sport> <Sport/ID=112> .
28 <Student.Sport/ID_Student=12;ID_Sport=111> rdf:type <Student.Sport> .
29 <Student.Sport/ID_Student=12;ID_Sport=111> <Student.Sport#ID_Student> 12 .
30 <Student.Sport/ID_Student=12;ID_Sport=111> <Student.Sport#ref-ID_Student> <Student/ID=12> .
31 <Student.Sport/ID_Student=12;ID_Sport=111> <Student.Sport#ID_Sport> 111 .
32 <Student.Sport/ID_Student=12;ID_Sport=111> <Student.Sport#ref-ID_Sport> <Sport/ID=111> .
33 <Sport/ID=110> rdf:type <Sport> .
34 <Sport/ID=110> <Sport#ID> 110 .
35 <Sport/ID=110> <Sport#Description> "Tennis" .
36 <Sport/ID=111> rdf:type <Sport> .
37 <Sport/ID=111> <Sport#ID> 111 .
38 <Sport/ID=111> <Sport#Description> "Football" .
39 <Sport/ID=112> rdf:type <Sport> .
40 <Sport/ID=112> <Sport#ID> 112 .
41 <Sport/ID=112> <Sport#Description> "Formula1" .

```

LISTING 7.3: The Result of Applying Direct Mapping Rules Over D011 Test Case Database.

- R2RML mappings that generate the triples that the instances of the class **Student** are also instances of the class **Person** (line 20-23) and that **SSN** is a property of **Student** because **Student** inherits properties of **Person** (line 24-27).
- R2RML mappings that generate relationships between **Student** and **Contact** (line 7-10).

7.3 Summary

In this chapter we described some evaluations that we have performed to validate the hypotheses described in Chapter 3. Morph-RDB is an R2RML engine that implements our query translation techniques described in Chapter 4 and is used to to validate the

```

1  <#TriplesMap5> a rr:TriplesMap;
2
3      rr:logicalTable [ rr:tableName "student"; ];
4
5      rr:subjectMap [
6          rr:class <Student>;
7          rr:template "Student/{ID}";
8          rr:termType rr:IRI;
9      ];
10
11      rr:predicateObjectMap [
12          rr:predicate <Student#ID>;
13          rr:objectMap [ rr:column "ID"; rr:datatype xsd:integer; ];
14      ];
15
16      rr:predicateObjectMap [
17          rr:predicate <Student#FirstName>;
18          rr:objectMap [ rr:column "FirstName"; rr:datatype xsd:string; ];
19      ];
20
21      rr:predicateObjectMap [
22          rr:predicate <Student#LastName>;
23          rr:objectMap [ rr:column "LastName"; rr:datatype xsd:string; ];
24      ];
25  .
26
27  <#TriplesMap10> a rr:TriplesMap;
28
29      rr:logicalTable [ rr:tableName "student_sport"; ];
30
31      rr:subjectMap [
32          rr:class <Student_Sport>;
33          rr:template "Student_Sport/{ID_Student}/{ID_Sport}";
34          rr:termType rr:IRI;
35      ];
36
37      rr:predicateObjectMap[
38          rr:predicate <Student_Sport#ID_Student>;
39          rr:objectMap[rr:column "ID_Student";rr:datatype xsd:integer;];
40      ];
41
42      rr:predicateObjectMap[
43          rr:predicate <Student_Sport#ID_Sport>;
44          rr:objectMap[rr:column "ID_Sport";rr:datatype xsd:integer;];
45      ];
46
47      rr:predicateObjectMap [
48          rr:predicate <Student_Sport#ref-ID_Student>;
49          rr:objectMap [
50              rr:parentTriplesMap <#TriplesMap5>;rr:joinCondition [ rr:child "ID_Student"; rr:parent "ID"; ];
51          ];
52      ];
53
54      rr:predicateObjectMap [
55          rr:predicate <Student_Sport#ref-ID_Sport>;
56          rr:objectMap [
57              rr:parentTriplesMap <#TriplesMap1>; rr:joinCondition [ rr:child "ID_Sport"; rr:parent "ID"; ];
58          ];
59      ];
60  .
61
62  <#TriplesMap1> a rr:TriplesMap;
63
64      rr:logicalTable [ rr:tableName "Sport"; ];
65
66      rr:subjectMap [
67          rr:class <Sport>;
68          rr:template "Sport/{ID}";
69          rr:termType rr:IRI;
70      ];
71
72      rr:predicateObjectMap [
73          rr:predicate <Sport#ID>;
74          rr:objectMap [ rr:column "ID"; rr:datatype xsd:integer; ];
75      ];
76
77      rr:predicateObjectMap [
78          rr:predicate <Sport#Description>;
79          rr:objectMap [ rr:column "Description"; rr:datatype xsd:string; ];
80      ];

```

LISTING 7.4: The Generated R2RML Mappings Correspond to Direct Mapping Triples.

hypotheses H1, H2, and H3. MIRROR is an automatic R2RML mappings generator that implements the techniques described in Chapter 6, and is used to validate hypotheses H6 and H7.

```

1 <Student/10> rdf:type <Student> .
2 <Student/10> <Student#FirstName> "Venus"^^xsd:string .
3 <Student/10> <Student#ID> "10"^^xsd:integer .
4 <Student/10> <Student#LastName> "Williams"^^xsd:string .
5 <Student/11> rdf:type <http://example.com/Student> .
6 <Student/11> <Student#FirstName> "Fernando"^^xsd:string .
7 <Student/11> <Student#ID> "11"^^xsd:integer .
8 <Student/11> <Student#LastName> "Alonso"^^xsd:string .
9 <Student/12> rdf:type <http://example.com/Student> .
10 <Student/12> <Student#FirstName> "David"^^xsd:string .
11 <Student/12> <Student#ID> "12"^^xsd:integer .
12 <Student/12> <Student#LastName> "Villa"^^xsd:string .
13 <Student.Sport/10/110> rdf:type <Student.Sport> .
14 <Student.Sport/10/110> <Student.Sport#ID_Student> "10"^^xsd:integer> .
15 <Student.Sport/10/110> <Student.Sport#ref-ID_Student> <Student/10> .
16 <Student.Sport/10/110> <Student.Sport#ID_Sport> "110"^^xsd:integer> .
17 <Student.Sport/10/110> <Student.Sport#ref-ID_Sport> <Sport/110> .
18 <Student.Sport/11/111> rdf:type <Student.Sport> .
19 <Student.Sport/11/111> <Student.Sport#ID_Student> "11"^^xsd:integer> .
20 <Student.Sport/11/111> <Student.Sport#ref-ID_Student> <Student/11> .
21 <Student.Sport/11/111> <Student.Sport#ID_Sport> "111"^^xsd:integer> .
22 <Student.Sport/11/111> <Student.Sport#ref-ID_Sport> <Sport/111> .
23 <Student.Sport/11/112> rdf:type <Student.Sport> .
24 <Student.Sport/11/112> <Student.Sport#ID_Student> "11"^^xsd:integer> .
25 <Student.Sport/11/112> <Student.Sport#ref-ID_Student> <Student/11> .
26 <Student.Sport/11/112> <Student.Sport#ID_Sport> "112"^^xsd:integer> .
27 <Student.Sport/11/112> <Student.Sport#ref-ID_Sport> <Sport/112> .
28 <Student.Sport/12/111> rdf:type <Student.Sport> .
29 <Student.Sport/12/111> <Student.Sport#ID_Student> "12"^^xsd:integer> .
30 <Student.Sport/12/111> <Student.Sport#ref-ID_Student> <Student/12> .
31 <Student.Sport/12/111> <Student.Sport#ID_Sport> "111"^^xsd:integer> .
32 <Student.Sport/12/111> <Student.Sport#ref-ID_Sport> <Sport/111> .
33 <Sport/110> rdf:type <Sport> .
34 <Sport/110> <Sport#ID> "110"^^xsd:integer> .
35 <Sport/110> <Sport#Description> "Tennis"^^xsd:string> .
36 <Sport/111> rdf:type <Sport> .
37 <Sport/111> <Sport#ID> "111"^^xsd:integer> .
38 <Sport/111> <Sport#Description> "Football"^^xsd:string> .
39 <Sport/112> rdf:type <Sport> .
40 <Sport/112> <Sport#ID> "112"^^xsd:integer> .
41 <Sport/112> <Sport#Description> "Formula1"^^xsd:string> .

```

LISTING 7.5: Direct Mapping triples of morph-RDB Result Over D011 Test Case Database.

```

1 <#TriplesMap15> a rr:TriplesMap;
2
3 rr:logicalTable [ rr:sqlQuery
4   "SELECT DISTINCT t_39025.ID_Student AS ID_Student, t_39025.ID_Sport AS ID_Sport
5     FROM (sport AS t_01724 JOIN student_sport AS t_39025 ON ((t_01724.ID=t_39025.ID_Sport)))
6     JOIN student AS t_83317 ON ((t_83317.ID=t_39025.ID_Student))" ];
7
8 rr:subjectMap [
9   rr:termType rr:IRI;
10  rr:template "Sport/{ID_Sport}";
11 ];
12
13 rr:predicateObjectMap [
14   rr:predicate <SportHasStudent>;
15   rr:objectMap [ rr:template "Student/{ID_Student}" ];
16 ]
17 .

```

LISTING 7.6: R2RML Mappings that Generate the Extra Triples.

```

1 <Sport/110> <SportHasStudent> <Student/10> .
2 <Sport/111> <SportHasStudent> <Student/11> .
3 <Sport/111> <SportHasStudent> <Student/12> .
4 <Sport/112> <SportHasStudent> <Student/11> .

```

LISTING 7.7: Extra triples of morph-RDB Result Over D011 Test Case Database.

```

1 <#TriplesMap12> a rr:TriplesMap;
2
3   rr:logicalTable [ rr:sqlQuery
4     "SELECT DISTINCT t_37839.ID, t_11900.CID
5     FROM (student AS t_37839 JOIN contact AS t_11900 ON ((t_37839.ID=t_11900.SID)))"
6   ];
7
8   rr:subjectMap [
9     rr:termType rr:IRI;
10    rr:template "http://example.com/Student/{ID}";
11  ];
12
13  rr:predicateObjectMap [
14    rr:predicate ex:StudentHasContact;
15    rr:objectMap [ rr:template "http://example.com/Contact/{CID}" ];
16  ];
17 ].
18
19 <#TriplesMap18> a rr:TriplesMap;
20
21   rr:logicalTable [ rr:sqlQuery
22     "SELECT t_76159.ID, t_76159.FirstName, t_76159.LastName, t_40951.SSN
23     FROM person AS t_40951 JOIN student AS t_76159 ON (t_40951.ID=t_76159.ID)"
24   ];
25
26   rr:subjectMap [
27     rr:class ex:Student;
28     rr:termType rr:IRI;
29     rr:template "http://example.com/student/{ID}";
30   ];
31
32   rr:predicateObjectMap[
33     rr:predicate rdf:type;
34     rr:objectMap [ rr:constant ex:Person ];
35   ];
36
37   rr:predicateObjectMap [
38     rr:predicate ex:Student#ssn;
39     rr:objectMap [ rr:datatype xsd:string; rr:column "SSN";]
40   ];
41
42   ...
43 ].

```

LISTING 7.8: R2RML Mappings Correspond to Subclass Generation and the Inherited SSN property in class **Student**.

Chapter 8

Extensions of morph-RDB

“Everything is perfect and there is
always room for improvement.”

Shunryu Suzuki

In this chapter we present two extensions of morph-RDB, namely morph-GFT and morph-LDP. morph-GFT, described in Section 8.1, extends morph-RDB by integrating it with federated query processor, so that it enables to access data coming from web-tables (Google Fusion Tables) and SPARQL endpoints. In Section 8.2 we describe morph-LDP, an extension of morph-RDB that works with a Linked Data Platform (LDP) implementation so that it exposes relational databases as LDP-aware applications.

8.1 morph-GFT

Announced in 2009, Google Fusion Tables [GHJ+10] is a data management service provided by Google to help users in working with their structured data. Users can upload their CSV data and host in the cloud on Google infrastructure such as Google Bigtable for scalability purposes. Hosting data in the cloud brings several benefits, such as enabling the data to be shared with other users, to be merged with other people’s data, and even allowing other users to collaborate by giving read/write permission. Google also integrates various visualisation tools so that users can share their data easily in various forms, such as on maps (using Google Maps), charts, or graphs. While the number of GFT tables users or available tables is not publicly advertised, it is believed that there are 400 millions active GMail users¹, and each of them potentially can use this service and contribute their own tables.

¹<http://www.quora.com/Gmail/How-many-total-active-Gmail-users-are-there>

As aforementioned, data hosted in GFT tables can be merged with other's people data. However, this can work only among GFT tables. Seen from outside the service, GFT tables consistute data silos that are not commonly linked to other data sources.

In this section, we present a system that exposes GFT tables as SPARQL endpoints by using R2RML [DSC12], and we use SPARQL-DQP to demonstrate that federated queries can be evaluated over them as if they were RDF datasets, thus taking benefit of the ease and scalability provided by the Google infrastructures while at the same time exploiting the power of querying over RDF enabled sources. The remainder of this section is as follows: Section 8.1.1 presents the architecture of our system and in Section 8.1.2 we see how the system works by using some examples. Finally in Section 8.1.3 we provide the conclusion and the future work.

8.1.1 Architecture of morph-GFT

The general architecture of the system is shown in Figure 8.1.

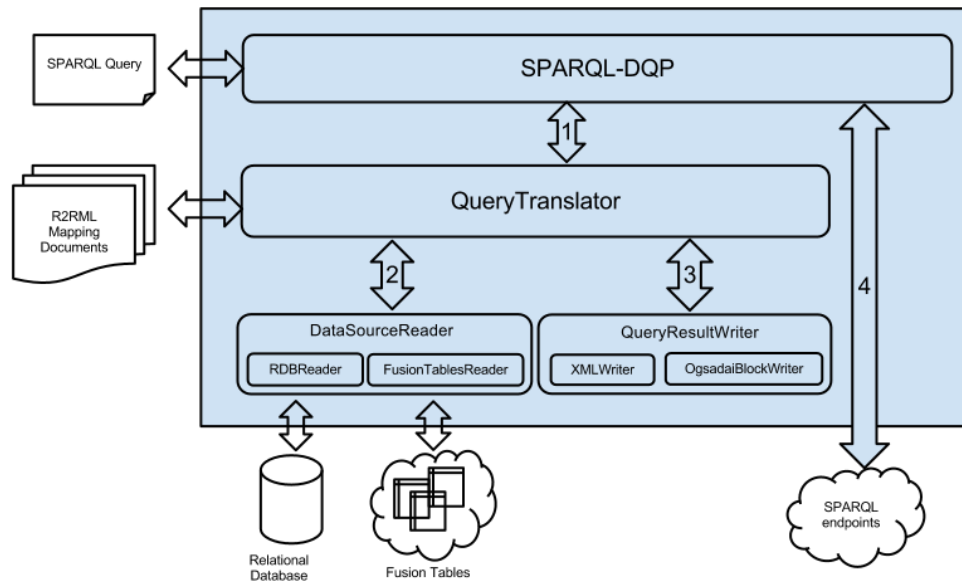


FIGURE 8.1: Architecture of morph-GFT

A federated SPARQL query sent by the user is received by SPARQL-DQP [BAAC11]. SPARQL-DQP is a SPARQL 1.1 Federated Query system based on OGSA-DAI/DQP system [AMG⁺04]. The SPARQL 1.1 Federated Query specification [PBA12] defines a SERVICE keyword as the location of the SPARQL endpoint that corresponds to the part of the query. By using SPARQL-DQP, we enable the SPARQL queries to query data not just from Google Fusion Tables, but also from available public SPARQL endpoints. The user query is a normal SPARQL 1.1 query in which the SERVICE keyword may

identify either a remote SPARQL endpoint or the R2RML mapping file that is used in the query translation process. The following processes occur when the system receives a SPARQL query:

- SPARQL-DQP divides the query into queries to be evaluated in each remote endpoint, as explained in [BAAC11], and sends each of them to the Query Translator (1) or directly to the corresponding SPARQL endpoint (4).
- The Query Translator translates (2) the SPARQL query from the user into an SQL query using the algorithm defined in [CLF09]. That new SQL query is sent to the Fusion Table service by the Data Source Reader which will also retrieve the results.
- These results will be converted into the internal SPARQL-DQP representation and streamed into the main data workflow (3). This data workflow is managed by SPARQL-DQP [BAAC11].

8.1.2 Example

We now explain the process when the system receives the SPARQL query “*give me all the members of the Ontology Engineering Group coming from a country whose capital is Madrid*”, shown in Listing 8.1.

```

1 SELECT ?n ?c
2 WHERE {
3   SERVICE <http://mappingpedia.linkeddata.es/mappings/fusiontables/
4     1pQBGUqR_g-j1WQavu-Fi1wGS7jsdRxomGc0DxMI/oegmembers.ttl>
5   {
6     ?m rdf:type foaf:Person.
7     ?m foaf:name ?n.
8     ?m ex:hasCountry ?c.
9   }
10  SERVICE <http://DBpedia.org/sparql> {
11    ?c <http://dbpedia.org/property/capital> <http://dbpedia.org/resource/Madrid>.
12  }
13 }
```

LISTING 8.1: Example of Federated SPARQL Query

That query asks for data about the members of the OEG research group (name and country) and also asks for data about Spain. For that, the query contains two SERVICE calls, one to the DBpedia SPARQL endpoint (asking about the country resource) and another one to a Google Fusion table containing the data about the members of this research group.

The part of the query that is addressed at the Fusion Table is sent to the Query Translator along with the mapping document specified in the SERVICE URI. The Query

Translator component translates the SPARQL query into a SQL query using the mapping information specified in the mapping document. The SPARQL and resulting SQL query can be seen in Listing 8.2. That SQL query is then sent to Fusion Table Reader component, which uses Google Fusion Table API to execute the query and process the results which are streamed into the SPARQL-DQP data workflow.

Meanwhile the SERVICE call to DBpedia is executed in parallel and SPARQL-DQP will join the results with the ones obtained from the GFT table query execution. All these remote query executions and data transfers form a single data workflow which is managed by SPARQL-DQP for finally send the results back to the user.

A video showing the example can be seen at this URL : <http://www.youtube.com/watch?v=qrTBJbnTHnc>.

```

1  -- SPARQL query sent to Query Translator
2  SELECT ?n ?c
3  WHERE {
4      ?m rdf:type foaf:Person.
5      ?m foaf:name ?name.
6      ?m ex:hasCountry ?c.
7  }
8
9  -- SQL query produced by Query Translator
10 SELECT name, country
11 FROM 1pQBGUqR_g-jlWQavu-Fi1wGS7jsdRxomGc0DxMI
12 WHERE name NOT EQUAL TO ""
13      AND country NOT EQUAL TO ""

```

LISTING 8.2: Query that is addressed at the GFT table

8.1.3 Conclusion

In this section we have presented a system that combines two features: first, exposing Google Fusion Tables as SPARQL endpoints using R2RML mapping documents in order to map the values from Fusion Tables as virtual RDF datasets; second, querying these virtual RDF datasets together with public SPARQL endpoints by using SPARQL-DQP.

In the current system, the user has to put the mapping document URL manually in the SPARQL query. In the future work, we will store those mapping documents in a triple store and enable a SPARQL endpoint. In that way, instead of providing the URL of the mapping document, a user can just post a SPARQL query like "get me all the mapping documents that map the concept foaf:person" in order to obtain the list of URL's correspond to the mappings of foaf:person.

8.2 morph-LDP

The W3C Linked Data Platform (LDP) candidate recommendation²[SAM14b] is focused on supporting read/write Linked Data by providing a standard HTTP-based RESTful protocol. The LDP specification extends the HTTP protocol with a set of new constraints, HTTP headers, and Link headers that are useful for read/write Linked Data applications. The two main types of concepts defined in LDP include:

- A Linked Data Platform Resource (LDPR) which is any HTTP resource that conforms to additional constraints defined in the LDP specification.
- A Linked Data Platform Container (LDPC) which is a specialization of an LDPR that acts as a collection resource that helps organizing LDPRs and creating new LDPRs as its members.

An obvious way of generating an LDP implementation is by storing RDF data in a triple store and internally doing SPARQL SELECT and UPDATE queries. However, there are contexts in which organizations are interested in continuing with their existing relational databases instead of transforming data into an RDF format and managing it from a triple store. Hence, the work that has been done in the past at W3C on the definition of the R2RML language, as well as the implementations available that support such language (e.g., Ontop [CCKE⁺15], morph-RDB [PCS14], D2R [BC06]), may be useful as an alternative implementation for LDP, as shown in Figure 8.2.

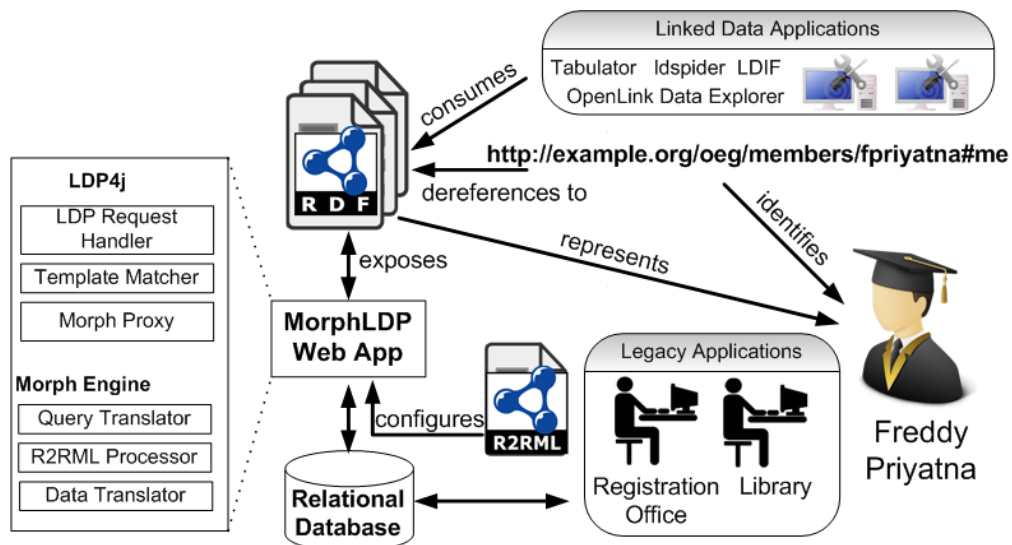


FIGURE 8.2: The morph-LDP use case

In this section we show how we can make use of the W3C R2RML recommendation as the underlying support for providing read/write Linked Data access to relational

²<http://www.w3.org/TR/ldp/>

databases. Database administrators only need to generate a R2RML mapping document and morph-LDP exposes the relational data as Linked Data. Such Linked Data is not only dereferenceable and available through the HTTP GET operation, but can also be updated using write operations such as PUT, POST, PATCH, and DELETE.

There is some previous work on the provision of update functionalities to R2RML-based systems (e.g., [GG11]). However, so far Linked Data generated by such an approach cannot be dereferenced using HTTP GET but rather have to be queried through a SPARQL endpoint. This hinders the usage of the follow-your-nose approach³ to traverse links found in Linked Data because it requires out-of-band information such as the SPARQL endpoint in addition to the link URI. In contrast, Linked Data generated by morph-LDP can be dereferenced and updated using URIs conforming to the HTTP-based LDP protocol. Moreover, morph-LDP does not require the clients to be SPARQL-aware, hence making the morph-LDP approach more attractive to those Web developers who are new to Semantic Web technologies. Finally, the usage of morph-RDB [PCS14] underneath makes the query evaluation process more efficient.

Another approach consists in mapping relational data to Web resources following REST principles and making those resources accessible via the HTTP protocol. Some systems that follow this approach are: HTTP database connector (HDBC) [MWL10], sql-REST⁴, and restSQL⁵. Nevertheless, none of those systems provides RDF representations. HDBC generates data according to the Atom format, while both sqlREST and restSQL generate XML outputs using W3C XLink and custom SQL Resources. While following the same approach, morph-LDP provides RDF representations (Turtle, RDF/XML, N-Triples and JSON-LD utilizing HTTP content negotiation). In addition, another advantage of morph-LDP is the use of a standard mapping language.

8.2.1 Mapping LDP components with R2RML

In this section, we explain how the concepts of LDP are mapped to R2RML so that the two standards can be combined to provide read/write Linked Data access to relational data.

The second Linked Data principle⁶ mandates the use of HTTP URIs so that people can look up those names. By inspecting the rules specified in SubjectMaps, these URIs can be mapped to Linked Data resources (LDPR) enabling read/write access through the LDP protocol.

³<http://patterns.dataincubator.org/book/follow-your-nose.html>

⁴<http://sqlrest.sourceforge.net/>

⁵<http://restsql.org/>

⁶<http://www.w3.org/DesignIssues/LinkedData.html>

PredicateObjectMaps allow generating the information about a specific resource, so that *when someone looks up a URI*, morph-LDP can *provide useful information using RDF* following the third Linked Data principle. LDP not only allows to lookup those resources but also to update them using HTTP write operations.

In R2RML, TriplesMaps are used to generate RDF triples out of the rows of logical tables. In LDP, LDP Containers are used as collection resources to organize LDPs. Thus logical tables and their corresponding TripleMaps can be mapped to LDP Containers.

8.2.2 Implementation

Our implementation⁷ is based on the result of two existing projects: **morph-RDB** and **LDP4j**⁸. LDP4j⁸ is a Java-based middleware for the development of LDP-aware applications. LDP4j is being developed in the context of the ALM iStack project, where the LDP middleware is used to integrate Application Lifecycle Management tools [MGCEG13].

morph-LDP extends morph-RDB with an LDP layer provided by LDP4j. The LDP layer extracts the metadata from the HTTP request and send it as an input for morph-RDB. morph-RDB has two modes of operation: API mode and SPARQL query mode and handles the transformation between RDF and relational data. Figure 8.3 illustrates the process when morph-LDP receives an HTTP Request when using the SPARQL query mode.

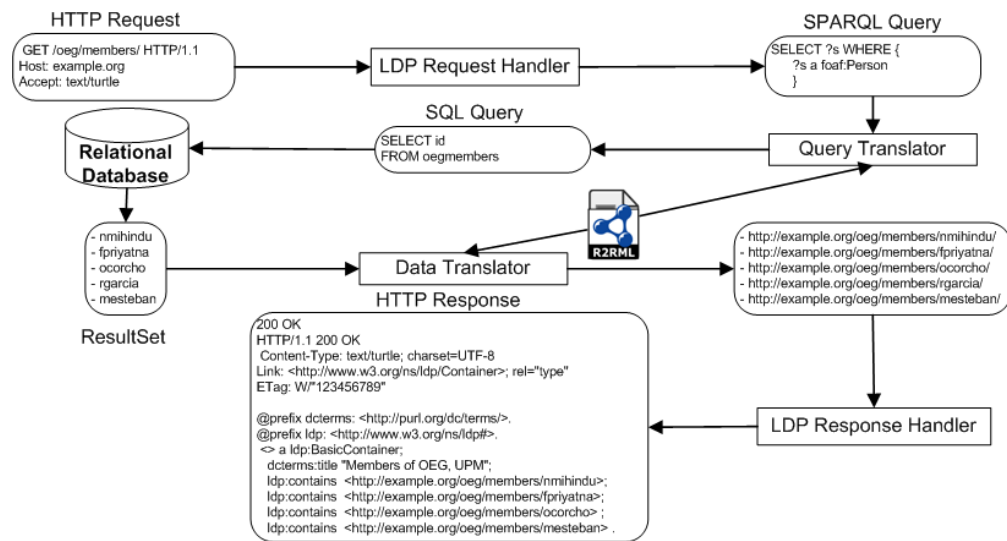


FIGURE 8.3: An example of morph-LDP in action, using the SPARQL query mode.

⁷<https://github.com/fpriyatna/morph-LDP>

⁸<http://ldp4j.org>

8.2.3 Conclusion

W3C LDP is in the final stages of the standardization process and we believe that integrating it with relational databases using W3C R2RML will help a wider adoption in the industry. In this section, we presented how LDP and R2RML can be combined to expose relational data as read/write Linked Data. Nevertheless, there is still some work to be done on providing support for Quality-of-Service requirements such as secure access and transactions. In addition to that, currently morph-LDP supports only simple R2RML mappings (no SQL view, no multiple mappings to a class/property) and we are working on giving support for more complex mappings.

Chapter 9

Conclusions and Future Work

“An investigator starts research in a new field with faith, a foggy idea, and a few wild experiments. Eventually the interplay of negative and positive results guides the work. By the time the research is completed, he or she knows how it should have been started and conducted.”

Donald Cram

This chapter concludes this thesis by presenting the summary of our contributions to the state of the art, as described in previous chapters. These contributions address the research problems that are described in Chapter 3. The main focus of this thesis is the creation of techniques and methods for the generation and exploitation of two W3C Recommendations for generating RDF from relational databases: the Direct Mapping and R2RML.

9.1 Summary on the first contribution and future work

The R2RML recommendation by the W3C allows users to specify transformation rules for transforming content of relational databases into RDF data, either materialized or virtual. SPARQL queries posed to virtual RDF data have to be translated into SQL queries. However, without a proper formalization and optimisations, the generated SQL queries may not be correct nor efficient enough to be evaluated on the underlying relational database.

Our first contribution has been the formalization and optimizations of SPARQL to SQL queries using R2RML mappings. We have extended the SPARQL to SQL query translation proposed by Chebotko and colleagues. This query translation technique was originally developed to work with RDBMS-backed triple stores, and it consists of two auxiliary mappings α and β , and two auxiliary functions *genCondSQL* and *genPRSQL*, together with one main function *trans* that takes SPARQL graph pattern. First, we extended the auxiliary mappings/functions by adding the relevant R2RML mappings in their inputs. By doing so, we formalized the SPARQL to SQL query translation technique, which is described in Section 4.1. Although the generated SQL queries return the expected results, we are aware that these SQL queries may not be optimally evaluated over the underlying databases. This is because the granularity input of the original *trans* function, that only accepts triple patterns, basic graph patterns, optional patterns, and union patterns. However, between triple patterns and basic graph patterns, there is an intermediate pattern commonly used in SPARQL queries, that is, Subject Triple Group (STG) pattern. We have modified the *trans* function so that the function can receive an STG pattern as a first class citizen. This allows reducing the number of self-joins that is generated by the original *trans* function, and by doing so, generate a more optimal SQL queries. In addition to reducing the number of self-join, we presented other relevant optimization techniques, such as replacing left-outer-join with self-join, reducing the number of union, and removing the conditional expression `IS NOT NULL` when it is not necessary. All those optimization techniques have been reported in Section 4.2.

The formalization of our SPARQL to SQL query translation together with its optimization techniques have been implemented as morph-RDB, an open-source Scala application, which has received several contribution from research groups and companies around the world and has been used in real life applications. Our evaluation on morph-RDB that was reported in Chapter 7 validates our hypotheses H1, H2, and H3 described in Chapter 3 regarding the possibility to design an R2RML-based SPARQL to SQL query translation technique based on an existing approach and the need to generate more optimized SQL queries.

With our first contribution, we have confirmed the previous findings that RDB2RDF systems are not mature enough to be used in real life applications, but only if those systems are not designed to generate efficient SQL queries. Furthermore, we have shown that having formalization and optimizations of SPARQL to SQL query translations bring the performance of RDF2RDF systems to an acceptable level, and even in most cases, the resulting SQL queries are as efficient as the ones written manually by the domain experts.

In Chapter 8, we described two extensions of morph-RDB by integrating it with other existing systems. The first extension is morph-GFT, described in Section 8.1, deals with adding another type of data source beyond relational database. In this case we have added Google Fusion Tables as another type of data source, and we integrate morph-RDB with SPARQL Distributed Query Processor (SPARQL-DQP), so that it is also able to query any SPARQL endpoints, such as DBPedia, for example. The second extension is morph-LDP which is described in Section 8.2. morph-LDP extends morph-RDB by providing a correspondence between the Linked Data Platform (LDP) concepts to R2RML. By integrating morph-RDB with LDP4j (an LDP implementation), morph-LDP provides read/write Linked Data access to relational databases.

During the writing of this thesis, several approaches for an efficient SPARQL to SQL query translation using R2RML mapping have been proposed. The most recent one is by Rodriguez-Muro and Rezk [RMR15]. This approach has been implemented as *ontop* and published in March 2015. Although in Section 4.2 we include the queries generated by morph-RDB and *ontop* for comparison purposes, we have not made any intensive benchmark of these two systems. A possible future work is to design a benchmark of SPARQL queries and R2RML mappings, and to compare the generated SQL queries by existing R2RML engines such as morph-RDB or *ontop*.

xR2RML [MDFZM15] is one extension of R2RML to support additional data sources beyond relational databases. Such data sources supported by xR2RML are: CSV, JSON, XML, and noSQL data. The first xR2RML engine uses morph-RDB as the basis of the implementation for generating materialized RDF data from the supported data source types mentioned above. Because now relational databases is not the only data source supported, in the line of our first contribution, the challenge is to extend our query translation technique, so that it will generate queries corresponding to the target data sources.

9.2 Summary on the second contribution and future work

The Direct Mapping recommendation specifies a fixed set of transformation rules. While published together with R2RML at the same time by the W3C RDB2RDF working group, there was no extensive study of the relationship between these two recommendations. Our second contribution is the study of the relationship between these two recommendations. This study is divided into two directions: from R2RML to the Direct Mapping, and from Direct Mapping to R2RML, which have been discussed in Chapter 5 and Chapter 6, respectively.

On the direction from R2RML to the Direct Mapping, it is clear that R2RML itself is more expressive than Direct Mapping. Our strategy here was to identify a fragment of R2RML so that from a mapping document of this R2RML fragment, there is a way to generate a database view, in such a way that the application of the Direct Mapping over this view is equivalent to the application of R2RML to the corresponding R2RML mapping document. This technique has been reported in Chapter 5.

On the opposite direction, we start with representing the Direct Mapping rules in R2RML mappings. Because there are implicit relationships encoded in a database schema (such as parent-child, 1 to N, and M to N), we enrich the generated R2RML mappings with these relationships. By doing so, the application of R2RML over the corresponding mapping document generates RDF data, that not only represents the application of the Direct Mapping, but also the relationships. This technique has been reported in Chapter 6. The evaluation of this technique has been reported in Section 7.2. Together, in Chapters 5 and 6, we see how we validate hypotheses H4, H5, and H6.

With our second contribution, we have lifted most of the initial works needed to work with R2RML systems by generating the mappings that conform with the Direct Mapping specification, thus reflecting the database schema, but also enriching those mappings with relationships between tables, such as parent-child, 1-N, or M-N.

In this line of this work, a possible future work is to identify more possible relationships to be represented in the generated R2RML mappings. It will be also interesting to see the extension of this work beyond relational databases, such as xR2RML or RML.

Appendix A

Mappings and Queries in in Section [4.2.1](#)

A.1 R2RML Mappings in Section 4.2.1

```

1  @prefix rr: <http://www.w3.org/ns/r2rml#> .
2  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
4  @base <http://mappingpedia.org/rdb2rdf/r2rml/tc/> .
5
6  <TriplesMapPatient> a rr:TriplesMap;
7    rr:logicalTable [ rr:tableName "tbl_patient" ];
8
9    rr:subjectMap [ a rr:Subject; rr:termType rr:IRI;
10      rr:template "http://mappingpedia.linkeddata.es/resources/Patient/{patientid}";
11      rr:class <http://www.semanticweb.org/mrezk#Patient>;
12    ];
13
14    rr:predicateObjectMap [
15      rr:predicateMap [ rr:constant <http://www.semanticweb.org/mrezk#hasName> ];
16      rr:objectMap [ rr:termType rr:Literal; rr:column "name"; ];
17    ];
18
19    rr:predicateObjectMap [
20      rr:predicateMap [ rr:constant <http://www.semanticweb.org/mrezk#hasStage> ];
21      rr:objectMap [
22        rr:parentTriplesMap <TriplesMapStage>;
23        rr:joinCondition [ rr:child "stage" ; rr:parent "stageid" ; ]
24      ];
25    ];
26
27    rr:predicateObjectMap [
28      rr:predicateMap [ rr:constant <http://www.semanticweb.org/mrezk#hasNeoplasm> ];
29      rr:objectMap [ rr:termType rr:IRI;
30        rr:template "http://mappingpedia.linkeddata.es/resources/Neoplasm/{patientid}";
31      ];
32    ];
33 .
34
35  <TriplesMapStage> a rr:TriplesMap;
36    rr:logicalTable [ rr:tableName "tbl_stage" ];
37
38    rr:subjectMap [ a rr:Subject; rr:termType rr:IRI;
39      rr:template "http://mappingpedia.linkeddata.es/resources/Stage/{stagename}";
40      rr:class <http://www.semanticweb.org/mrezk#Stage>;
41    ];
42
43    rr:predicateObjectMap [
44      rr:predicateMap [ rr:constant <http://www.semanticweb.org/mrezk#hasStageName> ];
45      rr:objectMap [ rr:termType rr:Literal; rr:column "stagename"; ];
46    ];
47 .

```

LISTING A.1: An R2RML Mapping Document Correspond to Tables **Patient** and **Stage**.

A.2 Ontop Mappings in Section 4.2.1

```

1  [PrefixDeclaration]
2  :          http://www.semanticweb.org/mrezk#
3
4  [SourceDeclaration]
5  sourceUri    PatientDB
6  connectionUrl jdbc:h2:C:/Users/fpriyatna/test;TRACE_LEVEL_FILE=2
7  username     sa
8  password
9  driverClass   org.h2.Driver
10
11 [MappingDeclaration] @collection [[
12 mappingId     Patient
13 target        :db1/{patientid} a :Patient .
14 source        SELECT patientid FROM "tbl_patient"
15
16 mappingId     hasName
17 target        :db1/{patientid} :hasName {name} .
18 source        Select patientid ,name FROM "tbl_patient"
19
20 mappingId     Neop
21 target        :db1/{patientid} :hasNeoplasm :db1/neoplasm/{patientid} .
22 source        SELECT patientid FROM "tbl_patient"
23
24 mappingId     hasStage
25 target        :db1/{patientid} :hasStage :db1/stage-{stagenam} .
26 source        SELECT patientid, stagenam FROM "tbl_patient", "tbl_stage"
27                WHERE tbl_patient.stage = tbl_stage.stageid
28
29 mappingId     Stage
30 target        :db1/stage-{stagenam} a :Stage .
31 source        SELECT stageid FROM "tbl_stage"
32
33 mappingId     hasStageName
34 target        :db1/stage-{stagenam} :hasStageName {stagenam} .
35 source        Select stageid ,stagenam FROM "tbl_stage"
36
37 ]]

```

LISTING A.2: An Ontop Mapping Document Correspond to Tables *Patient* and *Stage*.

A.3 Ontop SQL and Morph SQL in Section 4.2.1

A.3.1 Ontop STG SQL

```

1 SELECT \n 1 AS \"pQuestType\", NULL AS \"pLang\", ('http://www.semanticweb.org/mrezk#db1/' || REPLACE(
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(CAST(QVIEW1.\"
  PATIENTID\" AS CHAR),',','%20'),!','%21'),'@','%40'),'#','%23'),'$','%24'),'&','%26'),'*','%42'),'(','%28'),')',
  '%29'),'[','%5B'),']','%5D'),','%2C'),':','%3B'),';','%3A'),'?','%3F'),'=','%3D'),'+','%2B'),''','%22'),'/','%2
  F')) AS \"p\", \n 3 AS \"pnameQuestType\", NULL AS \"pnameLang\", CAST(QVIEW1.\"NAME\" AS CHAR)
  AS \"pname\", \n 1 AS \"sQuestType\", NULL AS \"sLang\", ('http://www.semanticweb.org/mrezk#db1/stage-'
  || REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(CAST(QVIEW2
  .\"STAGENAME\" AS CHAR),',','%20'),!','%21'),'@','%40'),'#','%23'),'$','%24'),'&','%26'),'*','%42'),'(','%28'),
  ')', '%29'),'[','%5B'),']','%5D'),','%2C'),':','%3B'),';','%3A'),'?','%3F'),'=','%3D'),'+','%2B'),''','%22'),'/',
  '%2F')) AS \"s\" \n
2 FROM \"tbl.patient\" QVIEW1,\"tbl.stage\" QVIEW2
3 WHERE \nQVIEW1.\"PATIENTID\" IS NOT NULL AND\nQVIEW1.\"NAME\" IS NOT NULL AND\n(QVIEW1.\"
  STAGE\" = QVIEW2.\"STAGEID\") AND\nQVIEW2.\"STAGENAME\" IS NOT NULL;

```

LISTING A.3: Ontop STG SQL

A.3.2 Morph-RDB STG SQL

```

1 SELECT 'T3'.patientid AS "p",873993427 AS "mappingid_p",'http://www.semanticweb.org/mrezk#hasName' AS "
  uri_hasName736196735",'http://www.semanticweb.org/mrezk#hasStage' AS "uri_hasStage21860122424",'T3'.name AS "
  pname",1989132530 AS "mappingid_pname",'T4'.stagename AS "s",1414845278 AS "mappingid_s"
2 FROM tbl_patient T3 INNER JOIN tbl_stage T4 ON (T3.stage = T4.stageid)
3 WHERE 'T3'.name IS NOT NULL AND 'T4'.stagename IS NOT NULL

```

LISTING A.4: Morph-RDB STG SQL

A.4 Ontop SQL and Morph SQL in Section 4.2.2

A.4.1 Ontop OSTG SQL

```

1 SELECT \n 1 AS \"pQuestType\", NULL AS \"pLang\", ('http://www.semanticweb.org/mrezk#db1/' || REPLACE(
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
  PATIENTID\" AS CHAR), ' ', '%20'), '!', '%21'), '@', '%40'), '#', '%23'), '$', '%24'), '&', '%26'), '*', '%42'), '(', '%28'), ')', '%29'), '[', '%5B'], ']', '%5D'), ',', '%2C'), ':', '%3B'), ';', '%3A'), '?', '%3F'), '=', '%3D'), '+', '%2B'), ' ', '%20'), '/', '%2F')) AS \"p\", \n 3 AS \"nameQuestType\", NULL AS \"nameLang\", CAST(QVIEW2.\"NAME\" AS CHAR)
  AS \"name\" \n
2 FROM \n\n (\n\n \"tbl_patient\" QVIEW1\n
3 LEFT OUTER JOIN\n\n \"tbl_patient\" QVIEW2\n\n ON\n\n (QVIEW1.\"PATIENTID\" = QVIEW2.\"PATIENTID\") AND\n\n QVIEW1.\"PATIENTID\" IS NOT NULL AND\n\n QVIEW2.\"NAME\" IS NOT
  NULL\n\n )\n
4 WHERE \nQVIEW1.\"PATIENTID\" IS NOT NULL;

```

LISTING A.5: Ontop OSTG SQL

A.4.2 Morph-RDB OSTG SQL

```

1 SELECT 'T2','patientid' AS "p",'T2','name' AS "name",1829217853 AS "mappingid-p",636782475 AS "mappingid.name"
2 FROM tbl_patient T2

```

LISTING A.6: Morph-RDB OSTG SQL

A.5 Ontop SQL and Morph SQL in Section 4.2.3

A.5.1 Ontop PT SQL

```

1 SELECT \n 1 AS \"pQuestType\", NULL AS \"pLang\", ('http://www.semanticweb.org/mrezk#db1/' || REPLACE(
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(CAST(QVIEW1.\"
  PATIENTID\" AS CHAR),',','%20'),!','%21'),'@','%40'),'#','%23'),'$','%24'),'&','%26'),'*','%42'),'(','%28'),')',
  '%29'),'[','%5B'),']','%5D'),','%2C'),';','%3B'),':','%3A'),'?','%3F'),'=','%3D'),'+','%2B'),''','%22'),'/','%2
  F')) AS \"p\", \n 1 AS \"sQuestType\", NULL AS \"sLang\", ('http://www.semanticweb.org/mrezk#db1/stage-' ||
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(CAST(QVIEW2
  .\"STAGENAME\" AS CHAR),',','%20'),!','%21'),'@','%40'),'#','%23'),'$','%24'),'&','%26'),'*','%42'),'(','%28'),
  ')','%29'),'[','%5B'),']','%5D'),','%2C'),';','%3B'),':','%3A'),'?','%3F'),'=','%3D'),'+','%2B'),''','%22'),'/',
  '%2F')) AS \"s\", \n 3 AS \"stagenamQuestType\", NULL AS \"stagenamLang\", CAST(QVIEW2.\"
  STAGENAME\" AS CHAR) AS \"stagenam\"
2 FROM ((\"tbl-patient\" QVIEW1
3 JOIN \"tbl-stage\" QVIEW2 ON QVIEW1.\"PATIENTID\" IS NOT NULL AND (QVIEW1.\"STAGE\" = QVIEW2.\"
  STAGEID\") AND QVIEW2.\"STAGENAME\" IS NOT NULL )
4 LEFT OUTER JOIN \n
  \"tbl-stage\" QVIEW3 \n ON \n (QVIEW2.\"STAGENAME\" = QVIEW3.\"
  STAGENAME\") AND \n QVIEW2.\"STAGENAME\" IS NOT NULL \n );

```

LISTING A.7: Ontop PT SQL

A.5.2 Morph-RDB PT SQL

```

1 SELECT 1759250827 AS \"mappingid_p\", 'http://www.semanticweb.org/mrezk#hasStageName' AS \"
  uri_hasStageName1267936587\", 813823788 AS \"mappingid_s\", 'T2'.stage AS \"s\", 'T4'.stagenam AS \"stagenam\"
  , 2082557120 AS \"mappingid_stagenam\", 'T2'.patientid AS \"p\", 'http://www.semanticweb.org/mrezk#hasStage2' AS \"
  uri_hasStage21860122424\"
2 FROM tbl_patient T2 INNER JOIN tbl_stage T4 ON ('T2'.stage = 'T4'.stageid)
3 WHERE 'T2'.stage IS NOT NULL

```

LISTING A.8: Morph-RDB PT SQL

A.6 Ontop SQL and Morph SQL in Section 4.2.4

A.6.1 Ontop TP SQL

```

1 | SELECT \n 1 AS \"pQuestType\", NULL AS \"pLang\", ('http://www.semanticweb.org/mrezk#db1/' || REPLACE(
   | REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(CAST(QVIEW1.\"
   | REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(CAST(QVIEW1.\"
   | PATIENTID\" AS CHAR), ' ', '%20'), '!', '%21'), '@', '%40'), '#', '%23'), '$', '%24'), '&', '%26'), '*', '%42'), '(', '%28'), ')', '
   | %29'), '[', '%5B'), ']', '%5D'), ',', '%2C'), ':', '%3B'), ';', '%3A'), '?', '%3F'), '=', '%3D'), '+', '%2B'), '', '%22'), '/', '%2
   | F')) AS \"p\", \n 3 AS \"nameQuestType\", NULL AS \"nameLang\", CAST(QVIEW1.\"NAME\" AS CHAR)
   | AS \"name\" \n
2 | FROM \"tbl.patient\" QVIEW1
3 | WHERE QVIEW1.\"PATIENTID\" IS NOT NULL AND QVIEW1.\"NAME\" IS NOT NULL;

```

LISTING A.9: Ontop TP SQL

A.6.2 Morph-RDB TPSQL

```

1 | SELECT 'T1'.patientid AS \"p\", 'T1'.name AS \"name\", 300983713 AS \"mappingid_p\", 284686302 AS \"mappingid_name\"
2 | FROM tbl_patient T1
3 | WHERE 'T1'.name IS NOT NULL

```

LISTING A.10: Morph-RDB TP SQL

Appendix B

Proofs for Self Join Elimination and Left Outer Join Elimination

For the sake of readability, from now on, we refer α as *Alpha*, β as *Beta*, φ to *GenCondSQL* and σ to *GenPRSQL*.

B.1 Proof for Self Join Elimination

- **Base case.** Given a subject triple group stg_2 composed of a conjunction (AND) of two triple patterns tp_1 and tp_2 , such that $stg_2 = (tp_1 \text{ AND } tp_2)$ where $tp_1.s = tp_2.s$, $transAND(stg_2)$ is defined as $transTP(tp_1) \bowtie transTP(tp_2)$.
Now, we want to prove that $transAND(tp_1 \text{ AND } tp_2)$ is equivalent to $transSTG(stg_2)$.
By definition, $transTP(tp) = \pi_{a(tp)}(\sigma_{\varphi(tp)}(\alpha(tp)))$,
with $\alpha(tp) = \alpha_s(tp) \bowtie \alpha_{po}(tp)$, $a(tp) = \{a_s(tp), a_{po}(tp)\}$ and $\varphi(tp) = \varphi_s(tp) \wedge \varphi_{po}(tp)$.

$$\begin{aligned} & \text{By definition, } transAND(tp_1 \text{ AND } tp_2) = transTP(tp_1) \bowtie transTP(tp_2) \\ &= \pi_{a_s(tp_1), a_{po}(tp_1)}(\sigma_{\varphi_s(tp_1) \wedge \varphi_{po}(tp_1)}(\alpha_s(tp_1) \bowtie \alpha_{po}(tp_1))) \bowtie \\ & \pi_{a_s(tp_2), a_{po}(tp_2)}(\sigma_{\varphi_s(tp_2) \wedge \varphi_{po}(tp_2)}(\alpha_s(tp_2) \bowtie \alpha_{po}(tp_2))) \\ &= \pi_{a_s(tp_1), a_{po}(tp_1), a_s(tp_2), a_{po}(tp_2)}(\sigma_{\varphi_s(tp_1) \wedge \varphi_{po}(tp_1), \varphi_s(tp_2) \wedge \varphi_{po}(tp_2)}(\alpha_s(tp_1) \bowtie \alpha_{po}(tp_1) \bowtie \\ & \alpha_s(tp_2) \bowtie \alpha_{po}(tp_2)))) \end{aligned}$$

Because $tp_1.s = tp_2.s$, we have $\alpha_s(tp_1) = \alpha_s(tp_2)$, $\varphi_s(tp_1) = \varphi_s(tp_2)$, and $a_s(tp_1) = a_s(tp_2)$, so that

$$transAND(tp_1 \text{ AND } tp_2) = \pi_{a_s(tp_1), a_{po}(tp_1), a_{po}(tp_2)}(\sigma_{\varphi_s(tp_1) \wedge \varphi_{po}(tp_1), \wedge \varphi_{po}(tp_2)}(\alpha_s(tp_1) \bowtie \alpha_{po}(tp_1) \bowtie \alpha_{po}(tp_2))))$$

Recall that:

$$\alpha^{stg}(stg_2) = \alpha_s(tp_1) \bowtie \alpha_{po}(tp_1) \bowtie \alpha_{po}(tp_2)$$

$$a^{stg}(stg_2) = \{a_s(tp_1), a_{po}(tp_1), a_{po}(tp_2)\}$$

$$\varphi^{stg}(stg_2) = \varphi_s(tp_1) \wedge \varphi_{po}(tp_1) \wedge \varphi_{po}(tp_2)$$

$$\text{So } transAND(tp_1 \text{ AND } tp_2) = \pi_{a_{stg}(stg_2)}(\sigma_{\varphi_{stg}(stg_2)}(\alpha^{stg}(stg_2))) = transSTG(stg_2)$$

$$\text{We have proven that } transAND(tp_1 \text{ AND } tp_2) = transSTG(stg_2)$$

- **Inductive case.** Consider $stg_n = \{stg_{n-1} \text{ AND } tp_n\}$ where

$$stg_{n-1} = \{tp_1 \text{ AND } \dots \text{ AND } tp_{n-1} \text{ AND } tp_n\}.$$

Assume that $transAND(stg_{n-1} \text{ AND } tp_n) = trans_{stg}(stg_n)$ holds. Now we want to prove that for $stg_{n+1} = \{stg_n \text{ AND } tp_{n+1}\}$, $transAND(stg_n \text{ AND } tp_{n+1}) = trans_{stg}(stg_{n+1})$ also holds.

$$\begin{aligned} transAND(stg_n \text{ AND } tp_{n+1}) &= trans(stg_n) \bowtie transTP(tp_{n+1}) \\ &= transAND(stg_{n-1} \text{ AND } tp_n) \bowtie transTP(tp_{n+1}) \\ &= transSTG(stg_n) \bowtie transTP(tp_{n+1}) \\ &= \pi_{a^{stg}(stg_n)}(\sigma_{\varphi^{stg}(stg_n)}(\alpha^{stg}(stg_n))) \bowtie \pi_{a(tp_{n+1})}(\sigma_{\varphi(tp_{n+1})}(\alpha(tp_{n+1}))) \\ &= \pi_{a^{stg}(stg_n), a(tp_{n+1})}(\sigma_{\varphi^{stg}(stg_n) \wedge \varphi(tp_{n+1})}(\alpha^{stg}(stg_n) \bowtie \alpha(tp_{n+1}))) \\ &= \pi_{a^{stg}(stg_n), a_s(tp_{n+1}), a_{po}(tp_{n+1})}(\sigma_{\varphi^{stg}(stg_n) \wedge \varphi_s(tp_{n+1}) \wedge \varphi_{po}(tp_{n+1})}(\alpha^{stg}(stg_n) \bowtie \alpha_s(tp_{n+1}) \bowtie \alpha_{po}(tp_{n+1}))) \end{aligned}$$

Recall that:

$$a^{stg}(stg_{n+1}) = a_s(tp_1), a_{po}(tp_1), \dots, a_{po}(tp_n), a_{po}(tp_{n+1})$$

$$\varphi^{stg}(stg_{n+1}) = \varphi_s(tp_1) \wedge \varphi_{po}(tp_1) \wedge \dots \wedge \varphi_{po}(tp_n) \wedge \varphi_{po}(tp_{n+1})$$

$$\alpha^{stg}(stg_{n+1}) = \alpha_s(tp_1) \bowtie \alpha_{po}(tp_1) \bowtie \dots \bowtie \alpha_{po}(tp_n) \bowtie \alpha_{po}(tp_{n+1})$$

$$\text{Then } transAND(stg_n \text{ AND } tp_{n+1}) = \pi_{a^{stg}(stg_{n+1})}(\sigma_{\varphi^{stg}(stg_{n+1})}(\alpha^{stg}(stg_{n+1})))$$

$$transAND(stg_n \text{ AND } tp_{n+1}) = transSTG(stg_{n+1})$$

■

B.2 Proof for Left Outer Join Elimination

- **Base Case 1.** Given a subject triple group with optional $OSTG_1$ composed of a pattern P where P is either a triple pattern TP or subject triple group STG , and triple pattern TP_1 , such that $OSTG_1 = (P \text{ OPT } TP_1)$, where $P.s = TP_1.s$, $transOPT(P \text{ OPT } TP_1)$ is defined as $trans(P) \bowtie trans(TP_1)$.
Now we want to prove that, $transOPT(P \text{ OPT } TP_1) = transOSTG(OSTG_1)$.
 $transOPT(P \text{ OPT } TP_1) = trans(P) \bowtie transTP(TP_1)$.

For the sake of readability, we will use R to refer to $trans(P)$ and S to $transTP(TP_1)$, so that $transOPT(P \text{ OPT } TP_1) = R \bowtie S = (R \bowtie S) \cup ((R - \pi_{A,B}(R \bowtie S)) \times \{C : null\})$, where \mathbf{A} is the set of attributes exclusive to R , \mathbf{B} is the set of attributes common to R and S , \mathbf{C} is the set of attributes exclusive to S . In other words, $R(\mathbf{A}, \mathbf{B})$ and $S(\mathbf{B}, \mathbf{C})$. Consider $T1 = (R \bowtie S)$, and $T2 = (R - \pi_{A,B}(R \bowtie S)) \times \{C : null\}$ so that $transOPT(P \text{ OPT } TP_1) = T1 \cup T2$.

Be definition of function $genPRSQL$, $notnull(\mathbf{C})$ is part of $\varphi(P)$ of S .

Let $\varphi'(TP_1)$ denotes the result of taking out $notnull(\mathbf{C})$ from $\varphi(TP_1)$ and S' as the result of taking out $notnull(\mathbf{C})$ from S , so that $S' = \pi_{a(tp1)}(\sigma_{\varphi'(otp1)}(\alpha(otp1)))$ and $S = \sigma(notnull(\mathbf{C}))(S')$

Hence, $T1 = R \bowtie S = R \bowtie \sigma(notnull(\mathbf{C}))(S')$

Every instance of R either matches with $S = \sigma(notnull(\mathbf{C}))(S')$ or its complement, $\sigma(null(\mathbf{C}))(S')$, that is:

$$R = \pi_{A,B}(R \bowtie S) \cup \pi_{A,B}(R \bowtie \sigma(null(\mathbf{C}))(S')), \text{ or}$$

$$R - \pi_{A,B}(R \bowtie S) = \pi_{A,B}(R \bowtie \sigma(null(\mathbf{C}))(S'))$$

$$T2 = (R - \pi_{A,B}(R \bowtie S)) \times \{C : null\}$$

$$T2 = \pi_{A,B}(R \bowtie \sigma(null(\mathbf{C}))(S')) \times \{C : null\}$$

$$T2 = R \bowtie \sigma(null(\mathbf{C}))(S')$$

Therefore, we can obtain the following:

$$\begin{aligned} transOPT(P \text{ OPT } TP_1) &= T1 \cup T2. = (R \bowtie \sigma(notnull(\mathbf{C}))(S')) \cup (R \bowtie \sigma(null(\mathbf{C}))(S')) \\ &= R \bowtie (\sigma(notnull(\mathbf{C}))(S') \cup \sigma(null(\mathbf{C}))(S')) \\ &= R \bowtie S' \end{aligned}$$

if $P = TP$, then $R = transTP(P) = \pi_{a(TP)}(\sigma_{\varphi(TP)}(\alpha(TP)))$

$$transOPT(TP \text{ OPT } TP_1) = R \bowtie S'$$

$$\begin{aligned}
&= \pi_{a(TP)}(\sigma_{\varphi(TP)}(\alpha(TP))) \bowtie \pi_{a(TP_1)}(\sigma_{\varphi'(TP_1)}(\alpha(TP_1))) \\
&= \pi_{a(TP),a(TP_1)}(\sigma_{\varphi(TP) \wedge \varphi'(TP_1)}(\alpha(TP) \bowtie \alpha(TP_1))) \\
&= \pi_{a^{stg}(OSTG_1)}(\sigma_{\varphi^{stg}(OSTG_1)}(\alpha^{stg}(OSTG))) \\
transOPT(TP \text{ OPT } TP_1) &= transOSTG(OSTG_1)
\end{aligned}$$

$$\begin{aligned}
&\text{if } P = STG, \text{ then } R = transSTG(STG) = \pi_{a^{stg}(STG)}(\sigma_{\varphi^{stg}(STG)}(\alpha^{stg}(STG))) \\
transOPT(STG \text{ OPT } TP_1) &= R \bowtie S' \\
&= \pi_{a^{stg}(STG)}(\sigma_{\varphi^{stg}(STG)}(\alpha^{stg}(STG))) \bowtie \pi_{a(TP_1)}(\sigma_{\varphi'(TP_1)}(\alpha(TP_1))) \\
&= \pi_{a^{stg}(STG),a(TP_1)}(\sigma_{\varphi^{stg}(STG) \wedge \varphi'(TP_1)}(\alpha^{stg}(STG) \bowtie \alpha(TP_1))) \\
&= \pi_{a^{stg}(STG)}(\sigma_{\varphi^{stg}(OSTG_1)}(\alpha^{stg}(OSTG_1))) \\
transOPT(STG \text{ OPT } TP_1) &= transOSTG(OSTG_1)
\end{aligned}$$

- **Base Case 2.** Given a subject triple group with optional $ostg_1$ composed of a subject triple group stg and a triple pattern otp_1 , such that $ostg_1 = stg \text{ OPT } otp_1$, where $stg.s = otp_1.s$, $trans(ostg_1)$ is defined as $trans(stg) \bowtie trans(otp_1)$.
By using R to refer to $trans(stg)$ and S to refer to $trans(otp_1)$, and applying the same steps as the Base Case 1, we will reach the same result that $trans(ostg_1) = trans_{ostg}(ostg_1)$.
- **Inductive case.** Given a subject triple pattern with optional $ostg_n$, composed of a subject triple group with optional $ostg_{n-1} = \{stg \text{ OPT } otp_1 \text{ OPT } \dots \text{ OPT } otp_{n-1}\}$ and a triple pattern otp_n , such that $ostg_n = ostg_{n-1} \text{ OPT } otp_n$ and all the triple patterns in $ostg_n$ have the same subject, we want to prove that $trans(ostg_n) = trans_{ostg}(ostg_n)$.

Assume that $trans(ostg_{n-1}) = trans_{ostg}(ostg_{n-1})$ holds.

By definition, $trans(ostg_n) = trans(ostg_{n-1} \text{ OPT } otp_n) = trans(ostg_{n-1}) \bowtie trans(otp_n)$

For the sake of readability, we will use R to refer to $trans(ostg_{n-1})$ and S to $trans(otp_n)$.

Hence, $trans(ostg_n) = R \bowtie S = (R \bowtie S) \cup ((R - \pi_{A,B}(R \bowtie S)) \times \{C : null\}) = T1 \cup T2$,

where A is the set of attributes exclusive to R , B is the set of attributes common to R and S , C is the set of attributes exclusive to S , $T1 = (R \bowtie S)$, and $T2 = (R - \pi_{A,B}(R \bowtie S)) \times \{C : null\}$

Refer to the Base Case 1 that $T1 = R \bowtie S = R \bowtie \sigma(\text{nonnull}(C))(S')$, and $T2 = R \bowtie \sigma(\text{null}(C))(S')$ where S' as the result of taking out $\text{nonnull}(C)$ from $\varphi(tp_n)$ so that

$$\begin{aligned}
 \text{trans}(\text{ostg}_n) &= \text{trans}(\text{ostg}_{n-1} \text{ OPT } \text{otp}_n) = R \bowtie S' \\
 &= \text{trans}(\text{ostg}_{n-1}) \bowtie \pi_{a(\text{otp}_n)}(\sigma_{\varphi'(\text{otp}_n)}(\alpha(\text{otp}_n))) \\
 &= \pi_{a_{stg}(\text{ostg}_{n-1})}(\sigma_{\varphi_{ostg}(\text{ostg}_{n-1})}(\alpha_{stg}(\text{ostg}_{n-1}))) \bowtie \pi_{a(\text{otp}_n)}(\sigma_{\varphi'(\text{otp}_n)}(\alpha(\text{otp}_n))) \\
 &= \pi_{a_{stg}(\text{ostg}_{n-1}), a(\text{otp}_n)}(\sigma_{\varphi_{ostg}(\text{ostg}_{n-1}) \wedge \varphi'(\text{otp}_n)}(\alpha_{stg}(\text{ostg}_{n-1}) \bowtie \alpha(\text{otp}_n))) \\
 \text{trans}(\text{ostg}_n) &= \pi_{a_{stg}(\text{ostg}_n)}(\sigma_{\varphi_{ostg}(\text{ostg}_n)}(\alpha_{stg}(\text{ostg}_n)))
 \end{aligned}$$

By definition, $\text{trans}_{ostg}(\text{ostg}_n) = \pi_{a_{stg}(\text{ostg}_n)}(\sigma_{\varphi_{ostg}(\text{ostg}_n)}(\alpha_{stg}(\text{ostg}_n)))$

So we have proven that $\text{trans}(\text{ostg}_n) = \text{trans}_{ostg}(\text{ostg}_n)$.

Bibliography

- [ABP⁺13] Marcelo Arenas, Alexandre Bertails, Eric Prud, Juan Sequeda, et al. A direct mapping of relational data to RDF, W3C Recommendation 27 September 2012, 2013.
- [ADL⁺09a] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumüller. Triplify: lightweight Linked Data publication from relational databases (2009). In *18th International Conference on World Wide Web*, pages 621–630, 2009.
- [ADL⁺09b] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumüller. Triplify: light-weight linked data publication from relational databases. In *Proceedings of the 18th international conference on World wide web*, pages 621–630. ACM, 2009.
- [AMG⁺04] M Alpdemir, Arijit Mukherjee, Anastasios Gounaris, Norman Paton, Paul Watson, Alvaro Fernandes, and Desmond Fitzgerald. OGSA-DQP: A service for distributed querying on the grid. *Advances in Database Technology-EDBT 2004*, pages 3923–3923, 2004.
- [BAAC11] Carlos Buil-Aranda, Marcelo Arenas, and Oscar Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. *The Semantic Web: Research and Applications*, pages 1–15, 2011.
- [Bar07] J. Barrasa. *Modelo para la definición automática de correspondencias semánticas entre ontologías y modelos relacionales*. PhD thesis, Facultad de Informativa, Universidad Politecnica de Madrid., March 2007.
- [BC06] C. Bizer and R. Cyganiak. D2R Server : Publishing relational databases on the web. In *The 5th International Semantic Web Conference*, 2006.
- [BGP06] J. Barrasa and A. Gómez-Pérez. Upgrading relational legacy data to the semantic web. In *15th International Conference on World Wide Web*, pages 1069–1070. ACM, 2006.

- [BS04] C. Bizer and A. Seaborne. D2RQ-treating non-RDF databases as virtual RDF graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, 2004.
- [BS08] C. Bizer and A. Schulz. Benchmarking the performance of storage systems that expose sparql endpoints. In *4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS2008)*, 2008.
- [CCG10] J.P. Calbimonte, O. Corcho, and A. Gray. Enabling ontology-based access to streaming data sources. In *9th International Semantic Web Conference (ISWC 2010)*, pages 96–111, 2010.
- [CCKE⁺15] D Calvanese, B Cogrel, S Komla-Ebri, R Kontchakov, D Lanti, M Rezk, M Rodriguez-Muro, and G Xiao. Ontop: Answering sparql queries over relational databases. *Submitted to the Semantic Web Journal*, 2015.
- [CLF09] A. Chebotko, S. Lu, and F. Fotouhi. Semantics preserving SPARQL-to-SQL translation. *Data & Knowledge Engineering*, 68(10):973–1000, 2009.
- [Cod70] EF Codd. A relational model of data for large shared data banks. 1970.
- [CWL14] Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax. *W3C Recommendation*, 25:1–8, 2014.
- [Cyg05] R. Cyganiak. A relational algebra for sparql. Technical Report HPL-2005-170, Digital Media Systems Laboratory HP Laboratories Bristol., 2005. <https://www.hpl.hp.com/techreports/2005/HPL-2005-170.pdf>.
- [DSC12] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF mapping language, 2012. W3C Recommendation, <http://http://www.w3.org/TR/r2rml/>.
- [DVSC⁺14] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: a generic language for integrated RDF mappings of heterogeneous data. In *Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014)*, Seoul, Korea, 2014.
- [ECTO09] B. Elliott, E. Cheng, C. Thomas-Ogbuji, and Z.M. Ozsoyoglu. A complete translation from SPARQL into efficient SQL. In *Proceedings of the 2009 International Database Engineering & Applications Symposium*, pages 31–42. ACM, 2009.
- [FLM99] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of the National Conference on Artificial Intelligence*, pages 67–73. JOHN WILEY & SONS LTD, 1999.

- [GG11] A. Garrote and M.N.M. García. Restful writable APIs for the web of linked data using relational storage solutions. In *WWW 2011 Workshop: Linked Data on the Web (LDOW2011)*, 2011.
- [GGO09] A.J. Gray, N. Gray, and I. Ounis. Can RDB2RDF tools feasibly expose large science archives for data integration? In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, pages 491–505. Springer-Verlag, 2009.
- [GHJ⁺10] Hector Gonzalez, Alon Y Halevy, Christian S Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, Warren Shen, and Jonathan Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1061–1066. ACM, 2010.
- [Gru93] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.
- [GSP13] Steve H Garlik, Andy Seaborne, and Eric Prud. Sparql 1.1 query language. *World Wide Web Consortium*, 2013.
- [H⁺06] A Hasman et al. HL7 RIM: an incoherent standard. In *Ubiquity: Technologies for Better Health in Aging Societies, Proceedings of Mie2006*, volume 124, page 133. IOS Press, 2006.
- [Hal01] A.Y. Halevy. Answering queries using views: A survey. *The VLDB Journal The International Journal on Very Large Data Bases*, 10(4):270–294, 2001.
- [KRRM⁺14] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyashev. Answering SPARQL queries under the OWL 2 QL entailment regime. In *The Semantic Web–ISWC 2014*. Springer, 2014.
- [Len02] M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM New York, NY, USA, 2002.
- [Len11] Maurizio Lenzerini. Ontology-based data management. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 5–6. ACM, 2011.
- [MAS12] Eric Prud’hommeaux Marcelo Arenas, Alexandre Bertails and Juan Sequeda. A direct mapping of relational data to RDF, 2012. W3C Recommendation, <http://www.w3.org/TR/rdb-direct-mapping/>.

- [MDFZM15] Franck Michel, Loic Djimenou, Catherine Faron-Zucker, and Johan Montagnat. xR2RML: Non-relational databases to RDF mapping. Technical report, 2015.
- [MGCEG13] Nandana Mihindukulasooriya, Raúl Garcia-Castro, and Miguel Esteban-Gutiérrez. Linked Data Platform as a novel approach for Enterprise Application Integration. Oct 2013.
- [MRC14] Jose Mora, Riccardo Rosati, and Oscar Corcho. kyrie2: Query rewriting under extensional constraints in \mathcal{ELHIQ} . In *The Semantic Web-ISWC 2014*, pages 568–583. Springer, 2014.
- [MWL10] Alexandros Marinos, Erik Wilde, and Jiannan Lu. Http database connector (hdbc): Restful access to relational databases. In *Proceedings of the 19th international conference on World wide web*, pages 1157–1158. ACM, 2010.
- [PAC13] Freddy Priyatna, Carlos Buil Aranda, and Oscar Corcho. Applying sparql-dqp for federated sparql querying over google fusion tables. In *The Semantic Web: ESWC 2013 Demo*, pages 189–193. Springer, 2013.
- [PAG09] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.
- [PBA12] Eric Prud’hommeaux and Carlos Buil-Aranda. SPARQL 1.1 federated query. *World Wide Web Consortium, Proposed Recommendation (November 2012)*, 2012.
- [PCS14] Freddy Priyatna, Oscar Corcho, and Juan Sequeda. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In *Proceedings of the 23rd international conference on World wide web*, pages 479–490. International World Wide Web Conferences Steering Committee, 2014.
- [PLC⁺08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer, 2008.
- [PS⁺08] Eric Prud, Andy Seaborne, et al. SPARQL query language for RDF. *W3C recommendation*, 15, 2008.
- [PVC⁺14] Valéria M Pequeno, Vania MP Vidal, Marco A Casanova, Luís Eufrazio T Neto, and Helena Galhardas. Specifying complex correspondences between relational schemas and RDF models for generating customized R2RML

- mappings. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 96–104. ACM, 2014.
- [RMR15] Mariano Rodriguez-Muro and Martin Rezk. Efficient sparql-to-sql with r2rml mappings. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2015.
- [SAM12] Juan F Sequeda, Marcelo Arenas, and Daniel P Miranker. On directly mapping relational databases to RDF and OWL. In *Proceedings of the 21st International Conference on World Wide Web*, pages 649–658. ACM, 2012.
- [SAM14a] Juan F Sequeda, Marcelo Arenas, and Daniel P Miranker. OBDA: Query rewriting or materialization? in practice, both! In *The Semantic Web—ISWC 2014*, pages 535–551. Springer, 2014.
- [SAM14b] Steve Speicher, John Arwe, and Ashok Malhotra. Linked Data Platform 1.0, 2014. W3C Last Call Draft, <http://www.w3.org/TR/ldp/>.
- [Seq13] Juan Sequeda. On the semantics of R2RML and its relationship with the direct mapping. In *International Semantic Web Conference (Posters & Demos)*, volume 2013, pages 193–196. Citeseer, 2013.
- [SHH⁺09] Satya S Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. A survey of current approaches for mapping of relational databases to rdf. *W3C RDB2RDF Incubator Group Report*, pages 113–130, 2009.
- [SHSH13] Kunal Sengupta, Peter Haase, Michael Schmidt, and Pascal Hitzler. Editing R2RML mappings made easy. In *International Semantic Web Conference (Posters & Demos)*, pages 101–104, 2013.
- [SM13] Juan F Sequeda and Daniel P Miranker. Ultrawrap: Sparql execution on relational data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 22:19–39, 2013.
- [SNMM13] A. Sicilia, G. Nemirovskij, M. Massetti, and L. Madrazo. RÉPENER’s linked dataset (in revision). *Semantic Web Journal*, 2013.
- [STCM11] Juan F Sequeda, Syed Hamid Tirmizi, Oscar Corcho, and Daniel P Miranker. Survey of directly mapping SQL databases to the semantic web. *Knowledge Engineering Review*, 26(4):445–486, 2011.

- [TSM13] Aibo Tian, Juan F Sequeda, and Daniel P Miranker. Qodi: Query as context in automatic data integration. In *The Semantic Web–ISWC 2013*, pages 624–639. Springer, 2013.
- [Ull00] J. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.
- [USA13] Jörg Unbehauen, Claus Stadler, and Sören Auer. Accessing relational data on the web with sparqlmap. In *JIST*, pages 65–80. Springer, 2013.