The Dissertation Committee for Juan Federico Sequeda
certifies that this is the approved version of the following dissertation:

Integrating Relational Databases with the Semantic Web

Committee:

Daniel P. Miranker, Supervisor

Marcelo Arenas

Don Batory

Phil Cannata

William Cook

Integrating Relational Databases with the Semantic Web

by

Juan Federico Sequeda, B.S.C.S

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2015

Dedicated to my parents Mami y Papi,
my brother Quico and
my best friend Adriana

Acknowledgments

I am extremely lucky and fortunate to have crossed paths with Dan Miranker. I first met Dan when I transferred to UT Austin as an undergrad in Fall 2006. Due to my initial interest in the Semantic Web, he had a simple question for me: what does it mean to integrate Relational Databases with the Semantic Web? Very quickly I realized that I wanted to pursue a PhD because I wanted to continue working on this topic and I wanted to work with Dan. 8 years later, I have an answer to Dan's question.

Dan saw in me a potential that I didn't even know that I had. Even in the darkest moments, he stood by me and didn't give up on me. He trusted me. He believed in me. He challenged me. He would let me do things my own way, even though he new that I wasn't on the right track, so I can figure things out on my own. I learned the hard way to *focus, focus, focus.* He is a true advisor and mentor.

Dan taught me to *strive for excellence.* Be the best I can be. To ask myself, what does success look like. For example, if a paper was rejected, it was because the paper wasn't excellent. I learned that the hard way. I lost count on how many times the Ultrawrap paper was rejected. Evidence to Dan's teaching, for my final piece of research, the paper not only got accepted the first time it was submitted, it won the best student research paper award.

I am also very grateful to cross paths with Marcelo Arenas, who very quickly became my mentor and friend. Thanks to Marcelo, I gained love and passion for

the theoretical aspects of computer science (and beer too).

Thanks to the rest of my committee, Don Batory, Phil Cannata and William Cook, who also challenged me to be the best researcher I could be.

I have received valuable advice from many throughout this journey. Special thanks to Lorenzo Alvisi and J Moore.

It was a pleasure working with Zach Elkins and the rest of the Constitute team. I am extremely grateful to have had the opportunity to integrate my research with such an impactful project.

It has been great to work with amazing people in the Miranker Lab throughout these years: Aibo, Mayank, Lee, Nathan, Carlos, Albert, Slavcho, Rick, Saral, Ferner, Rudy, Smriti, Hamid.

My research interest in the Semantic Web is thanks Oscar Corcho. I met Oscar in 2005 when he gave a tutorial while I was studying at the Universidad del Valle in Cali, Colombia. Thanks to him, I learned about the Semantic Web. If it weren't for Oscar, who knows if I would have even been interested in pursuing a PhD. Thanks also to all the folks at UPM, specially Asun, and for all the fun times we had at Cercedilla.

I am honored and thankful to have met Jim Hendler and received so much advice from him throughout these years. Everytime Jim and I meet for dinner, it's in a different part of the world. I hope we can continue this tradition for years to come.

I am very lucky to be part of a vibrant research community where I have had the opportunity to learn so much. I am very proud to consider many researchers as my friends. Olaf Hartig, Tom Heath and Michael Hausenblas were one of the first people I met at ISWC 2008. Olaf and I became very good friends and started the COLD workshop series. Thanks Tom for all your advice (and pizza is on me next time:). It was great working with Michael for years on the W3C RDB2RDF working

group. It's very hard to name everybody, but special thanks to Peter Mika, Barry Norton, Aidan Hogan, Andreas Harth, Cliff Joslyn and Chris Welty.

Thanks to all my Latin American friends in Austin for all the happy hours and Zacapa parties. Without you, this journey would have been boring.

Adriana, you have been by my side throughout the most crucial parts of this journey. Thank you for support, patience, love and sacrifice. You have been and always will be my best friend. Te amo baby.

Last but not least, thanks to my family. I would not be where I am if it weren't for them. My brother has always been there when I most needed help. My father told me that a true PhD is one that makes the world a better place. I hope that my work can live up to that goal. My mother always reminded me that life is about a balance. Not only should I strive for excellence in academics, but also at a personal level. My family has always wanted the best for me and have supported my dreams. This is to them.

Juan Federico Sequeda

The University of Texas at Austin
May 2015

Integrating Relational Databases with the Semantic Web

Publication No. _____

Juan Federico Sequeda, Ph.D.
The University of Texas at Austin, 2015

Supervisor: Daniel P. Miranker

An early vision in Computer Science was to create intelligent systems capable of reasoning on large amounts of data. Independent results in the areas of Description Logic and Relational Databases have advanced us towards this vision. Description Logic research has advanced the understanding of the tradeoff between the computational complexity of reasoning and the expressiveness of logic languages, and now underpins the Semantic Web. The Semantic Web comprises a graph data model (RDF), an ontology language for knowledge representation and reasoning (OWL) and a graph query language (SPARQL). Database research has advanced

the theory and practice of management of data, embodying features such as views and recursion which are capable of representing reasoning. Despite the independent advances, the interface between Relational Databases and Semantic Web is poorly understood.

This dissertation revisits this vision with respect to current technology and addresses the following question: *How and to what extent can Relational Databases be integrated with the Semantic Web?* The thesis is that much of the existing Relational Database infrastructure can be reused to support the Semantic Web. Two problems are studied.

Can a Relational Database be automatically virtualized as a Semantic Web data source? This paradigm comprises a single Relational Database. The first contribution is an automatic direct mapping from a Relational Database schema and data to RDF and OWL. The second contribution is a method capable of evaluating SPARQL queries against the Relational Database, per the direct mapping, by exploiting two existing relational query optimizations. These contributions are embodied in a system called Ultrawrap. Empirical analysis consistently yield that SPARQL query execution performance on Ultrawrap is comparable to that of SQL queries written directly for the relational representation of the data. Such results have not been previously achieved.

Can a Relational Database be mapped to existing Semantic Web ontologies and act as a reasoner? This paradigm comprises an OWL ontology including inheritance and transitivity, a Relational Database and mappings between the two. A third contribution is a method for Relational Databases to support inheritance and transitivity by compiling the ontology as mappings, implementing the mappings as SQL views, using SQL recursion and optimizing by materializing a subset of views. This contribution is implemented in an extension of Ultrawrap. Empirical analysis reveals that Relational Databases are able to effectively act as reasoners.

Contents

List of Tables

List of Figures

‘

Chapter 1

Introduction

An early vision in Computer Science was to create intelligent systems capable of *reasoning* on *large amounts of data*. *Reasoning* means to infer new knowledge from what is already known. *Large amounts of data* is usually understood to mean data sufficient to warrant the explicit use of a database management system[6]

The vision can be traced back to a landmark workshop on logic and databases organized by Gallaire, Minker and Nicolas in 1977 [59]. The interest in logic and data accelerated through the 1980's. To support the computational workload, specialized parallel computers were developed[1]. A large-scale japanese initiative, the Fifth Generation Project [80], stimulated world-wide research[2]. Scientific publications appeared in mainstream Database and AI venues (e.g. SIGMOD and AAAI), as well as a number of specialized workshops and conferences; interest continued into the 21st century[3].

Despite the amount of research towards this vision, it is still largely unfulfilled. Nevertheless, independent results in the areas of Description Logic and Relational Databases have advanced towards this vision.

Description Logic research has advanced the understanding of the tradeoff between the computational complexity of reasoning and the expressiveness of logic languages. Description Logic underpins the Semantic Web, an extension to the Web that enables intelligent access to data on the Web. The technologies supporting the Semantic Web consist of a set of standards. Data on the Semantic Web is represented in a graph data model called RDF (Resource Description Framework). Ontologies are a form of knowledge representation and reasoning which provide semantic relationships, such as inheritance or transitivity, between concepts. The standardized ontology languages are called RDFS (RDF Schema) and OWL (Web

[1]See for example The DADO Production System Machine [123] and The Connection Machine [77]

[2]e.g. MCC in Austin, Texas and Alvey in the UK.

[3]See the proceedings of workshops on Expert Systems (1984-1988) [82, 81, 83], Deductive Databases (1990-1999) [43, 111, 110, 62, 50, 61, 58] and Knowledge Representation meets Databases (1994-2003) [15, 16, 17, 20, 25, 26, 56, 27, 91, 24, 34]

Ontology Language). Data can be accessed by a graph based query language called SPARQL. Figure 1.1 presents the Semantic Web languages as an inheritance stack.

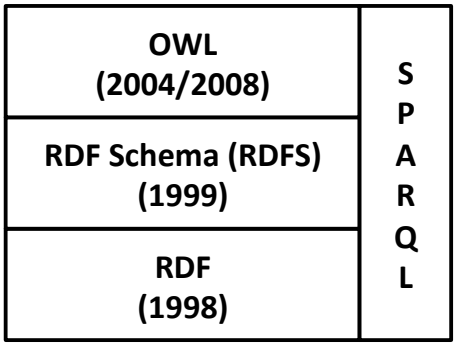


Figure 1.1: Semantic Web stack

Database research has advanced the theory and practice of storage and management of data. Modern Relational Database Management Systems (RDBMS) are founded on the relational model, which is based on first order logic. RDBMS are also comprised by a set of standards which have evolved over the past two decades.

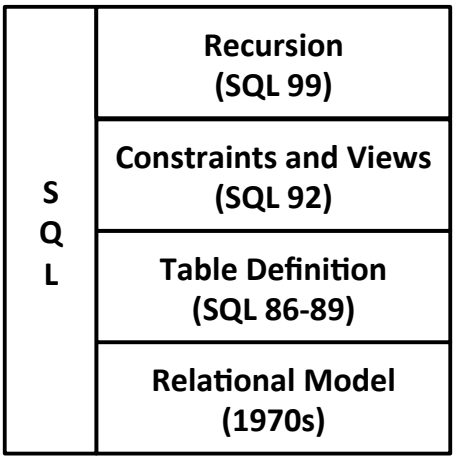


Figure 1.2: Relational Database stack

SQL 86-89 provided the capability to describe schemas. SQL 92 increased expressivity by adding constraints and views, which are mechanisms of adding domain

semantics. SQL 99 further increased the expressive power of RDBMS by supporting recursion. Figure 1.2 presents the RDBMS standards as a stack.

This dissertation revisits the early Computer Science vision of combining data and logic with the current technologies: Relational Databases and Semantic Web. The overarching question this dissertation investigates is:

> ***How*, and to *what extent*, can Relational Databases be integrated with the Semantic Web?**

The thesis is that the independent development of Description Logic, and the evolution of Relational Database Management Systems, each manifesting in standards, contain substantive overlap. That overlap enables much of the existing Relational Database infrastructure to be reused in support of the Semantic Web. The ultimate deliverable of this dissertation is an intelligent system capable of reasoning, through means of the Semantic Web, on large amounts of data stored in a Relational Database.

This dissertation resolves two controversies. The first is concerned with the applicability of Relational Database query execution methods for the Semantic Web. The second is concerned with the capability of a Relational Database to act as a reasoner.

Approximately 70% of websites have relational database back-ends [75]. The sheer number of websites suggests the success of the Semantic Web is tied to maintaining compatibility and consistency with legacy RDBMSs. Therefore, the goal is to have Semantic Web applications coexist with the legacy applications by executing SPARQL queries over the RDBMS instead of creating consistency problems by creating a replicated copy of the relational data as RDF. In 2008, it was shown that SPARQL is equivalent, from an expressive point of view, to relational algebra [8].

This result suggests that SPARQL can be optimized by the native SQL optimizer. However, in 2009, two studies evaluated three SPARQL to SQL systems and came to the opposite conclusion: existing SPARQL to SQL systems do not compete with traditional relational databases and require additional optimizations [23, 66].

The first goal was to resolve the apparent contradiction between theses results. Towards that end, the Ultrawrap system was built. In order to use as much of the native SQL optimizer as possible, a key intuition was to represent a mapping, from the relational data to RDF, as SQL views. SPARQL queries are then translated to SQL queries in terms of the views. The result of this approach is that existing query optimizations can be applied directly by the native SQL optimizer.

When it comes to reasoning, the goal is to answer SPARQL queries over an OWL ontology using mappings between the RDBMS and the OWL ontology. This paradigm has recently taken the name of *Ontology Based Data Access* (OBDA). Commonly, researchers have taken one of two approaches to develop OBDA systems: a materialization-based approach (forward chaining) or a query rewriting-based approach (backward chaining). In the query rewriting approach, a common assumption is that the RDBMS is only capable of relational algebra (i.e. SQL 92) [35]. Therefore, the research focuses on applying optimizations to queries outside of the RDBMS [105, 115, 85, 86, 92, 65, 45, 130, 54, 114]. Another effect of this assumption is that recursion in SQL 99 is not exploited. Therefore, existing OBDA systems do not support ontologies with transitivity.

The second goal was to identify how a Relational Database can act as a reasoner by exploiting advanced database capabilities. Towards that end, Ultrawrap was extended to support OBDA. The extension, Ultrawrap$^{\text{OBDA}}$, is designed to be a bidirectional evaluation engine; that is, a hybridization of query rewriting and materialization. In order to exploit the native SQL optimizer as a reasoner, a key intuition was that mappings can be unified with the ontology to create new mappings,

denoted as *saturated mappings*, and then represent them as SQL views. SPARQL queries are also translated into SQL queries in terms of those views. Inspired by database research on OLAP and view materialization for query optimization, a cost model was defined to determine which views to materialize in order to optimize query performance. The result of this approach is that the Relational Database is able to act as a reasoner for ontologies that include inheritance and transitivity.

This dissertation is organized as follows. The remainder of this chapter presents background, the methodology used in this research, and details of the challenges and contributions of this dissertation. Chapter 2 provides the definition and notation used throughout the dissertation. Chapter 3 defines a default and automatic mapping of Relational Databases to RDF and OWL, coined the *direct mapping*. Chapter 4 presents the Ultrawrap system capable of executing SPARQL queries over the Relational Database by taking advantage of the existing SQL optimizer. The ultimate goal of Ontology Based Data Access is presented in a single chapter, Chapter 5. We conclude and detail future work in Chapter 6.

1.1 Background

1.1.1 Relational Databases

A database is a collection of data which can be managed by a database management system (DBMS). A DBMS is a set of programs that enables data storage, modifications, and access from a database. In the 1970s, Codd proposed the relational model, which became the foundations of Relational Databases [46]. Based on First Order Logic, the relational model represents data in terms of tuples (rows), grouped into relations (tables). A relational database management system (RDBMS) is a DBMS that is based on the relational model. Research in databases have advanced the theory and practice of storage, querying, and managing data. Today, RDBMSs

continue to be the predominant choice for the storage and management of data.

SQL is a standardized language used to manage a Relational Database. Over the last two decades, Relational Databases have been evolving through an ongoing sequence of expanding standards [120].

SQL consists of a data definition language (DDL), a data manipulation language (DML) and a data query language (DQL). SQL-DDL is used to define and modify the schema of a database. SQL-DML is used to modify the records of the database. SQL-DQL is the query language, based on relational algebra, used to access data stored in the database.

SQL-DDL has had the following evolution. The initial SQL standard, SQL86-89 was limited to defining tables, attributes, datatypes and replaced abbreviated formal relational notations with full English words (e.g. `CREATE TABLE`). SQL 92 added data integrity constraints such as `CHECK`, `PRIMARY KEY`, `FOREIGN KEY` and `UNIQUE`. This was the first approach to increase expressivity and offer domain semantics by connecting relations and permitting a developer to know beforehand what values are allowed. The addition of views to SQL (i.e. `CREATE VIEW`) enabled the creation of derived relations defined by a query in terms of existing relations and/or other views.

Throughout the past decades, features encompassing Artificial Intelligence concepts have been incorporated into Relational Databases [125, 124]. An example is SQL Views. Views add deductive (reasoning) capabilities to a relational database, as evidenced by Datalog[4]. A view can be seen as an intensional predicate in the head of a Datalog rule, while the base tables in the Relational Databases are the extensional predicates in the body of a Datalog rule. Subsequently, recursion in Datalog inspired the addition of recursion to the SQL standard in 1999.

Views have also been studied in the context of a variety of data manage-

[4]The reader is referred to [6] for details on Datalog

ment problems such as query optimization, data integration, and data warehousing [70, 72, 98, 95, 42]. A common technique to improve query performance is to materialize a view. A materialized view is when the query result of a view is computed and persistently stored. Accessing a materialized view can be much faster than recomputing the defining query. Techniques have been developed to determine when views should be materialized and how an optimizer may rewrite a query to use materialized views [71]. These techniques have been implemented in commercial Relational Databases [22, 134, 64].

Relational Databases have also benefited from semantic knowledge for query optimization, known as Semantic Query Optimizations (SQO). The purpose is to use the semantics encoded in integrity constraints to transform a query into a semantically equivalent query with a lower execution cost. For instance, joins can be eliminated if the results are known a priori. For example, in the query `SELECT A.X, B.Z FROM A, B WHERE A.Y=B.Z`, where `A.Y` is a foreign key referencing `B.Z`, the join to the table `B` can be dropped because the value of `B.Z` is already known to be `A.Y`. Therefore, the semantically equivalent query, with a lower execution cost is `SELECT X, Y FROM A`. Query predicates inconsistent with integrity constraints can be eliminated a priori because the query will not have an answer. For example, even though the query `SELECT * FROM A WHERE ID = 1 AND ID = 2` is valid, the result will always be empty because an attribute can only have one value. These techniques were initially designed for deductive databases and have also been integrated in commercial relational databases [41].

1.1.2 Description Logic and the Semantic Web

Description Logics (DL) are a family of languages for knowledge representation and reasoning that are used to represent ontologies [18]. An ontology is *an explicit specification of a conceptualization* of a domain of interest [69]. DLs are a decidable

fragment of First Order Logic (FOL).

Ontologies, represented in DL, have two functionalities. First, as a descriptive language to express expert knowledge in an unambiguous and formal way. This is accomplished by representing knowledge as a set of concepts within a domain and the relationships between those concepts. A second functionality is as a logical language to infer implicit knowledge from the knowledge explicitly represented in the ontology. This aspect of inferring additional knowledge distinguishes ontologies from modeling languages such as UML.

Description Logics were introduced to solve the lack of formal logic-based semantics of two existing knowledge representation approaches from the late 1960s and early 1970s: semantic networks [109] and frames [97]. Hayes recognized that these approaches could be given semantics relying on FOL [74]. Brachman and Levesque later on realized that frames and semantic networks did not require all of FOL [28], and different features of a knowledge representation language would lead to different fragments of FOL. An important consequence of this fact is that reasoning could be accomplished by specialized techniques without requiring FOL theorem provers. Further, different fragments of FOL lead to computational problems of different complexity.

The first DL system was KL-ONE [30]. Studying the tradeoff between the expressiveness and computational complexity of reasoning of a DL language lead to three approaches: 1) limited expressiveness of the language but with complete reasoning algorithms; CLASSIC being the primary example [29] 2) expressive languages but reasoning with incomplete algorithms; LOOM being the primary example [93] and finally 3) expressive language with complete complete reasoning algorithm; KRIS being the primary example [19]. In this early work, the reasoning algorithms were not efficient. Recently, the focus has been to develop complete and efficient reasoning algorithms for expressive DL languages [78].

Description Logics have also influenced the World Wide Web. The growth of the World Wide Web attracted researchers to envision its next generation. The goal is to build systems allowing software agents to "semantically" query the web as if it were a database, rather than relying on human interpretation of the information on the web. In order to accomplish this goal, software agents must understand how to interact with information on the web. Thus the meaning of information must be captured. It was widely agreed that ontologies, grounded in Description Logics, play a key role in providing meaning to the information on the web [18]. This extension to the web is known as the *Semantic Web*.

The technologies comprising the Semantic Web consists of a set of languages to represent data based on a graph data model (RDF) and represent knowledge as ontologies based on Description Logic (RDFS and OWL). Additionally, there is also a query language for graphs based on pattern matching (SPARQL). This set of languages forms an inheritance stack. RDFS inherits from RDF, and OWL inherits from RDFS, as illustrated in Figure 1.1. RDFS and OWL ontologies can be represented as RDF graphs.

1.2 Methodology

To understand the relationship between Relational Databases and Semantic Web, we adopt a methodology where each technology is decomposed into corresponding layers [128]. This enables us to identify and exploit similarities. Understanding the historical context, it is easy to decompose Relational Databases into a stack representing increasing expressive power (see Figure 1.2) and perhaps not by coincidence, one that corresponds to the Semantic Web stack (see Figure 1.1). Figure 1.3 shows the resulting relationship between Relational Databases and the Semantic Web. The following observations are made:

- The foundational layer of Relational Databases and Semantic Web is the data

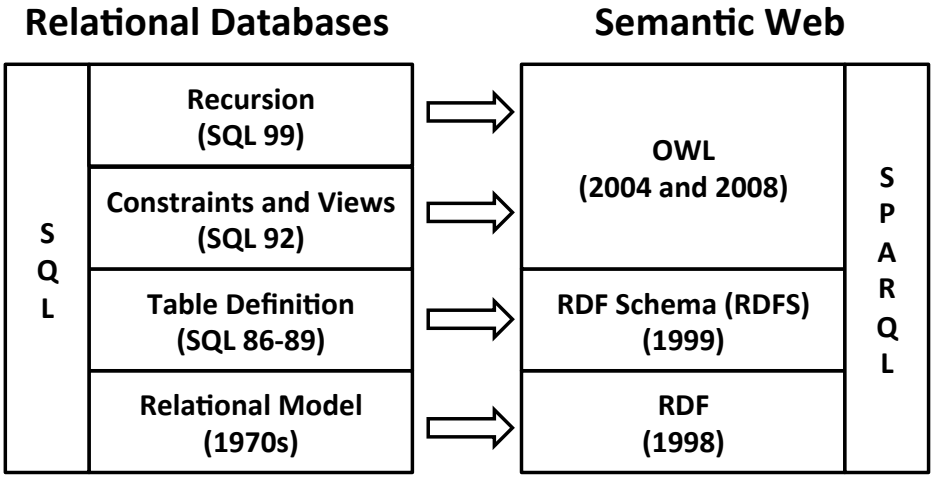


Figure 1.3: Two Layer Cake mapping between Semantic Web and Relational Databases

model (Relational Model and RDF).

- The next layer provides a language to describe schemas (Table Definitions and RDFS).
- The top layers increases the expressivity of the language (Constraints, Views, Recursion and OWL).
- Both technologies have a query language (SQL and SPARQL).
- Each layer has evolved over time.

The direct correspondence observed between the two layer cakes in Figure 1.3 suggests that a framework can be defined to support an automatic way of virtualizing a Relational Database as a Semantic Web data source. A Semantic Web data source consists of RDF data and an OWL ontology with the capability of evaluating SPARQL queries. By virtualization we mean that the data continues to reside in a Relational Database instead of physically creating the new Semantic Web data source.

Subsequently, it is hypothesized that such framework can also support reasoning over a Relational Database by mapping to OWL ontologies created independent of the database. The research question of this dissertation is tackled by addressing the following two problems.

1.3 Problem 1: Relational Database as a Semantic Web data source

In this problem, the goal is to automatically virtualize the Relational Database as a Semantic Web data source. The input is a single Relational Database. Therefore, the research question is the following:

> **Given a Relational Database, how can the Relational Database be automatically virtualized as a Semantic Web data source?**

Figure 1.4 depicts the framework to support the automatic virtualization of a Relational Database as a Semantic Web data source. Given a single Relational Database, an automatic mapping is defined which maps the relational data to RDF and the relational schema to OWL. The automatic mapping is called a *direct mapping*. Given that the OWL ontology is derived from the relational schema, it is dependent of the relational schema. SPARQL queries evaluated over the Semantic Web representation of the Relational Database are translated to SQL queries and subsequently evaluated over the Relational Databases.

Two challenges are identified. First, define a direct mapping and study the mapping with respect to correctness properties. Second, define a method for SPARQL queries to be efficiently evaluated by the Relational Database.

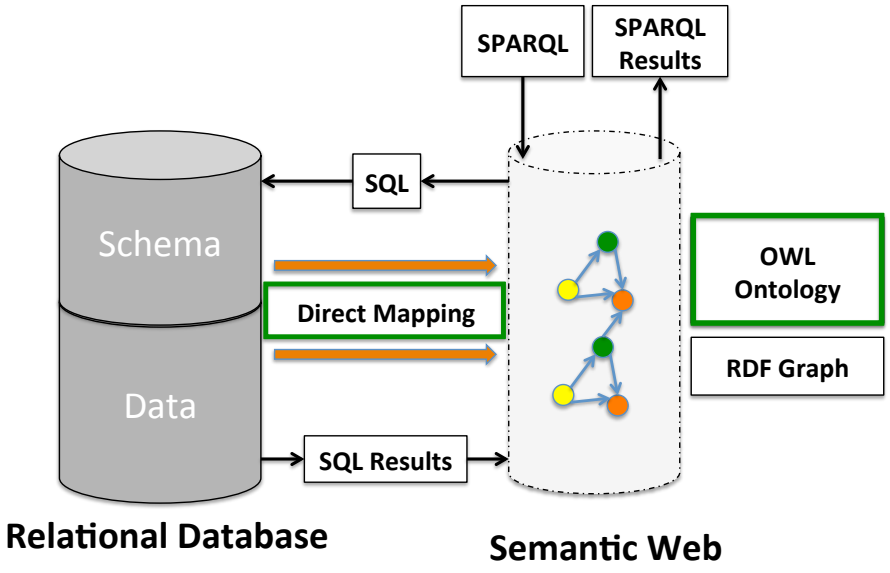


Figure 1.4: Framework 1

1.3.1 Challenge 1: Automatic Mapping of Relational Databases to Semantic Web

The challenge is to identify a mapping that can serve as an automatic and default way of translating a relational database schema to an OWL ontology and relational databases instances to RDF.

In order to understand the state of the art of automatically mapping Relational Databases to the Semantic Web, we surveyed seven different approaches [119]. Two shortcomings were identified:

1. The focus of existing approaches are direct mappings of a relational schema to an OWL ontology.

2. Existing direct mappings of relational schemas to OWL are expository in nature, in other words, based on examples without formal semantics.

Our early work was to first to define a direct mapping of a relational schema

to an OWL ontology with formal semantics defined using Datalog [129]. However, the challenge of defining a direct mapping from a relational instance to RDF has not been fulfilled. Additionally, such mappings should be studied from a theoretical perspective in order to understand properties that can serve as correctness criteria.

1.3.2 Contribution 1: Direct Mapping

The first contribution of this dissertation addresses Challenge 1. A direct mapping is presented, which in addition to generating an OWL ontology from the relational schema, it also generates RDF from the relational instance. The semantics of the direct mapping is defined using non-recursive Datalog. This direct mapping is the first automatic mapping of Relational Databases to RDF and OWL to be formally defined.

Four properties are presented in order to define a correctness criteria for the direct mapping. Two properties are fundamental: information preservation and query preservation. Information preservation speaks to the ability of reconstructing the original Relational Database from the result of the direct mapping. In other words, the mapping does not lose information. Query preservation means that every query over a Relational Database can be translated into an equivalent query over the result of the direct mapping. In other words, the mappings does not lose queries.

Two properties are desirable: monotonicity and semantics preservation. Monotonicity is a desired property because it assures that re-computation of the entire mapping is not needed after inserts to the database. Finally, a direct mapping is semantics preserving if the satisfaction of a set of integrity constraints are encoded in the mapping result.

A direct mapping is proposed and studied with respect to these four properties. The proposed direct mapping is monotone, information preserving and query

preserving even in the general and practical scenario where relational databases contain null values. Additionally, it is semantics preserving for consistent databases, meaning databases where integrity constraints are satisfied. However, it is not semantics preserving for inconsistent databases, meaning databases where integrity constraints are not satisfied. This is due to the closed world assumption of Relational Databases and the open world assumption of the Semantic Web (see Section 3.6.2).

The direct mapping presented in this dissertation served as the foundation of a W3C standard: *A Direct Mapping of Relational Data to RDF* [13].

1.3.3 Challenge 2: Execute SPARQL on Relational Data

The challenge is to define a method to execute SPARQL queries against a Relational Database, per the direct mapping. The goal is that such method should make use of optimizations already incorporated in Relational Databases.

To clarify the focus of this challenge, consider the taxonomy in Figure 1.5. In RDF data management there are efforts that concern Triplestores and those that concern legacy Relational Databases. Triplestores are DBMS for RDF data, and support SPARQL execution against the stored contents. Native triplestores are those that are implemented from scratch [32, 102, 132]. RDBMS-backed Triplestores are built by adding an application layer to an existing RDBMS. Within that literature is a discourse concerning the best database schema, SPARQL to SQL query translations, indexing methods and even storage managers, (i.e. column stores vs. row stores) [4, 44, 55, 57, 133]. NoSQL Triplestores are also being investigated as possible RDF storage managers [57, 79, 88, 47]. In all three cases, RDF is the primary data model.

The other spectrum consists of systems that integrate legacy Relational Databaes with the Semantic Web, a.k.a Relational Database to RDF (RDB2RDF) Within that, there are systems concerning ETL (Extract-Transform-Load) and

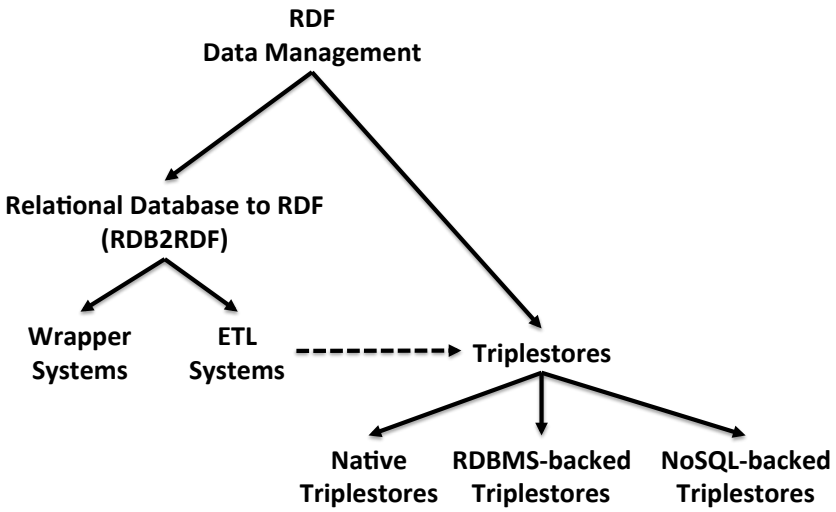


Figure 1.5: Taxonomy of RDF Data Management

Wrappers. An ETL system extracts data from the Relational Database, physically translates to RDF so it can then be loaded into a Triplestore.

Wrapper systems present a virtual RDF representation of the relational data physically stored in an RDBMS. Thus, no copy of the relational data is made to RDF, as opposed to ETL systems. It follows that SPARQL queries need to be translated to SQL queries through the use of mappings. The research in this dissertation is concerned with Wrapper systems.

Since both RDBMS-backed Triplestores and Wrapper systems involve Relational Databases and translation from SPARQL to SQL, there is a potential for confusion. The difference is that RDBMS-backed Triplestores translate SPARQL queries to SQL queries that are executed on database schemas that model and store RDF data. For example, many RDBMS-backed Triplestores use the triple table schema: a table with three attributes, containing one row for each triple [44]. Wrapper systems on the other hand translate SPARQL queries to SQL queries that are executed on legacy database schemas that model and store relational data (not RDF data) and where mappings play a key role.

In 2008, Angles and Gutierrez proved that SPARQL is equivalent in expressive power to relational algebra [9]. Thus, one might expect that Wrapper systems would benefit from the native SQL optimizer to evaluate SPARQL queries.

In 2009, two overlapping, refereed studies compares three Wrapper systems (D2R [1] , SquirrelRDF [117], Virtuoso RDF Views [2]) with native SQL execution on the Relational Database [23, 66]. Bizer and Schultz compared D2R and Virtuoso RDF Views on MySQL [23]. Gray et al compared D2R and SquirrelRDF on MySQL [66]. The result of these experiments concluded that existing Wrapper systems do not exploit the native SQL optimizer and that rewriting algorithms should be improved [23, 66].

The March 2009 Berlin SPARQL Benchmark reported that SPARQL queries on the evaluated Wrapper systems and on the 100 million triple dataset were up to 1000 times slower that the native SQL queries. Today, those systems are still the most used in the Semantic Web community and no new system has been introduced and evaluated since then. Bizer and Schultz [23], creators of the Berlin SPARQL Benchmark, concluded that: *"Setting the results of the RDF stores and the SPARQL-to-SQL rewriters in relation to the performance of classical RDBMS unveiled an unedifying picture. Comparing the overall performance (100M triple, single client, all queries) of the fastest rewriter with the fastest relational database shows an overhead for query rewriting of 106%. This is an indicator that there is still room for improving the rewriting algorithms."*

Gray et al [66] tested D2R and SquirrelRDF on a scientific database. This study concluded that *"... current rdb2rdf systems are not capable of providing the query execution performance required to implement a scientific data integration system based on the rdf model. [...] it is likely that with more work on query translation, suitable mechanisms for translating queries could be developed. These mechanisms should focus on exploiting the underlying database systems capabilities to optimize*

queries and process large quantities of structured data, e.g. pushing the selection conditions to the underlying database system."

A motivation for this research is to resolve the apparent contradiction among the aforementioned papers.

1.3.4 Contribution 2: Ultrawrap

The second contribution of this dissertation is a method capable of evaluating SPARQL queries against the Relational Database, per the direct mapping, which pushes the optimization of queries into the existing SQL infrastructure. It is observed that two existing relational semantic query optimizations effect important query plan transformations for SPARQL queries. These are: detection of unsatisfiable conditions and self-join elimination.

The contribution is embodied in a system, Ultrawrap, which is organized as a set of four compilers that 1) extracts an OWL ontology from the relational schema, per the direct mapping, 2) encodes a logical representation of the direct mapping using SQL views, 3) syntactically translates SPARQL queries to SQL queries in terms of the views and 4) delegates optimizations to the SQL engine. Empirical analysis consistently yields that the performance of SPARQL query execution on Ultrawrap is comparable to the performance of SQL queries written directly for the relational representation of the data. Such results have not been achieved elsewhere.

1.4 Problem 2: Relational Database as a Reasoner

The framework in the first problem enables the automatic virtualization of a Relational Database as a Semantic Web data source. It has two distinguishing characteristics: 1) a direct mapping and 2) an OWL ontology derived from the relational schema, which lacks expressive features such as inheritance or transitivity.

Consider now the second problem. Given an OWL ontology with expressive

features such as inheritance or transitivity, created independent of a Relational Database; Can a Relational Database act as a reasoner in order to infer new facts from given facts? In this problem, the input consists of: 1) a Relational Database, 2) an OWL ontology independent of the Relational Database and 3) mappings from the Relational Database to the OWL ontology. SPARQL queries expressed in terms of the OWL ontology are to be answered by the Relational Database through use of the mappings. Therefore, the second question is:

Given a Relational Database, an OWL ontology with inheritance and transitivity, and a mapping between the two, how can a Relational Database act as a reasoner?

It is observed that the previous framework can support this new setting. First, the ontology derived from the relational schema can be replaced by an ontology created independent of the Relational Database. This ontology can contain expressive features such as inheritance and transitivity. Second, the direct mapping can be substituted for user defined mappings. Figure 1.6 depicts this new framework. Note that SPARQL evaluation over the relational database is not altered.

Similar to the previous problem, two aspects need to be considered: 1) the mapping and 2) the Relational Database optimizer as a way to efficiently evaluate SPARQL queries. The challenge is to identify a method such that mappings can incorporate features such as inheritance and transitivity from the ontology, and the existing Relational Database infrastructure can be reused as a reasoner in order to efficiently evaluate SPARQL queries in terms of the ontology.

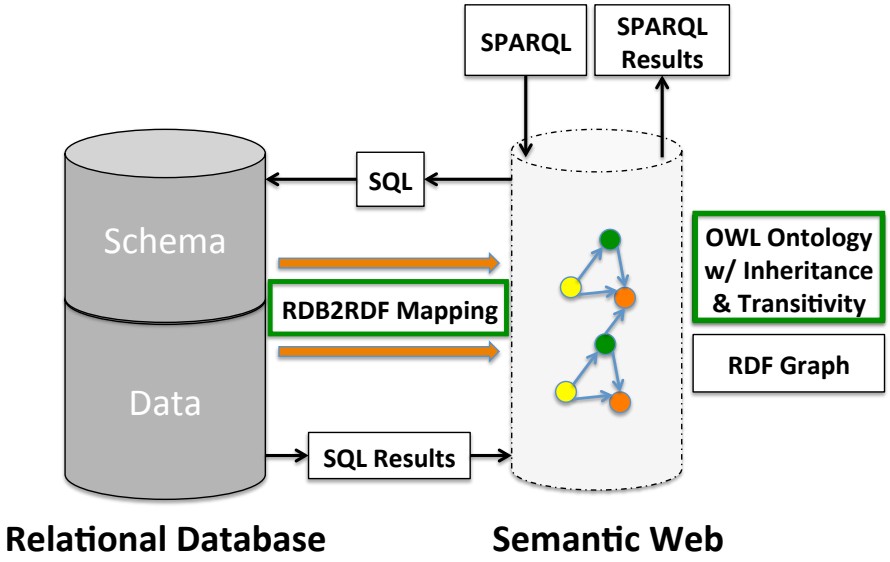


Figure 1.6: Framework 2

1.4.1 Challenge 3: Reasoning on a RDBMS

Combining a Relational Database, an OWL ontology, and mappings, in order to answer SPARQL queries over these three components, has recently taken the name of Ontology-Based Data Access (OBDA). Typically, researchers have taken one of two approaches to develop OBDA systems: a rewriting-based approach (backward chaining) or a materialization-based approach (forward chaining).

The rewriting-based approach has been the main focus of the research community in the past years. Three steps are executed, as shown in Figure 1.7. Given a SPARQL query and an OWL ontology, a new SPARQL query is generated which contains knowledge from the ontology. The new SPARQL query is considered the rewritten query. The majority of the OBDA literature focuses on this step [103]. Depending on the ontology, the size of the rewritten SPARQL query is worst case exponential w.r.t the size of the query and the ontology [36]. Therefore, even though each query can be evaluated efficiently on the database, an exponential number of

queries may need to be evaluated. Recent research has been devoted to understand the reasons for such large rewritings, focusing on generating a smaller rewritten query [105, 115, 85, 86, 92, 65, 45, 130, 54]. In the second step, the mapping is used to compile the rewritten SPARQL query into a SQL query which can then be evaluated over the database. Little research has been done on this step [106, 107]. Finally, the SQL query is evaluated on the relational database, which gives an the answer to the initial SPARQL query. To the best of our knowledge, in addition to Ultrawrap$^{\text{OBDA}}$, there is only one other system that completes the entire workflow [114].

The advantage of the rewriting approach is that reasoning is done over the original Relational Database. Therefore, query results reflect the latest up-to-date data. As previously described, the drawback is the overhead entailed by the rewriting of a SPARQL query.

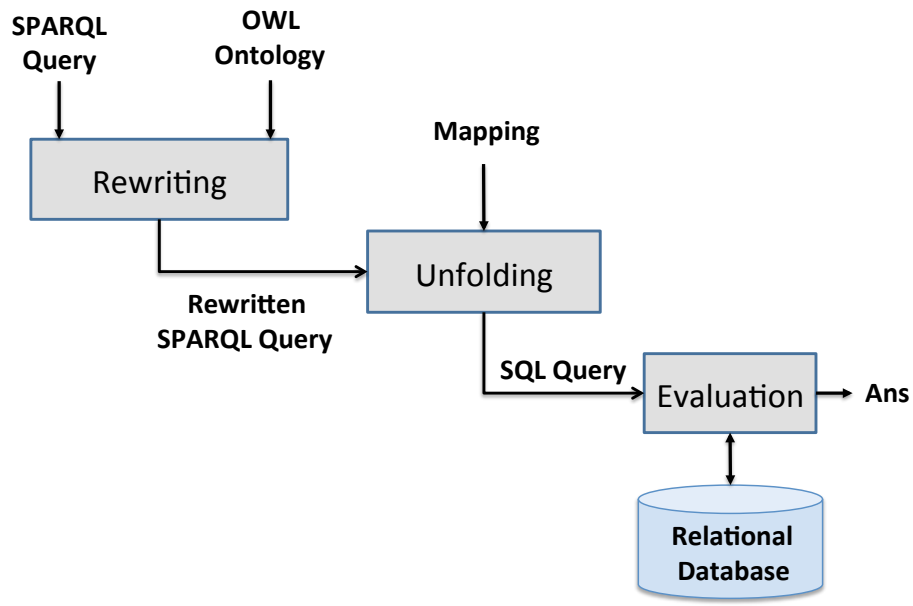


Figure 1.7: Query Rewriting

In the materialization-based approach, the inputs are the Relational Database,

an OWL ontology and a mapping. These are used to derive new facts as RDF that are then stored in a Triplestore, as shown in Figure 1.8. The RDF data in the Triplestore is considered to be the materialization of the data in the Relational Database given the mapping and the ontology. The answer to a SPARQL query over the target ontology is computed by directly posing the SPARQL query over the Triplestore [12].

With this approach, there is no overhead when a SPARQL query is evaluated over the Triplestore when compared to the rewriting approach. However, a drawback is that a copy of the original relational data needs to be maintained. If any updates occur on the original relational data, these need to be propagated to the RDF version. Additionally, depending on the ontology, the size of the materialized RDF data can be infinite.

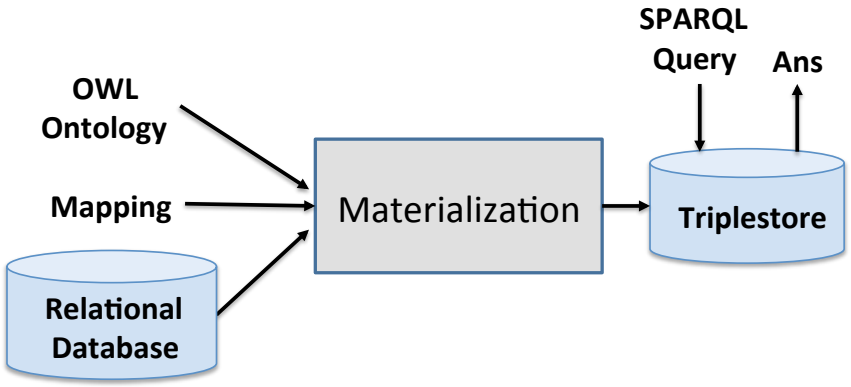


Figure 1.8: Materialization

1.4.2 Contribution 3

The third contribution is a bidirectional evaluation approach; that is, a hybridization of query rewriting (backward chaining) and materialization (forward chaining) for Ontology Based Data Access. The method enables Relational Databases to act as a reasoner supporting inheritance and transitivity by 1) unifying mappings with

the ontology to create new mappings, denoted as saturated mappings, 2) representing the saturated mappings as SQL views, 3) translating SPARQL queries in terms of those views and 4) a cost model to determine which views to materialize in order to optimize query performance. The result is that the underlying SQL optimizer is able to reduce the execution time of a SPARQL query by rewriting the query in terms of materialized views. This contribution is implemented in a system, Ultrawrap$^{\text{OBDA}}$. Empirical analysis reveals that by exploiting the capabilities beyond relational algebra, such as query rewriting using materialized views and SQL recursion, Relational Database are able to effectively act as reasoners for ontologies that include inheritance and transitivity.

Chapter 2

Preliminaries

This chapter presents the notation and definitions used. This section also defines the three standards comprising Semantic Web: RDF, the graph data model; OWL, the ontology language; and SPARQL, the query language for RDF. Subsequently, the expressivity of the OWL dialect used in this research is presented.

2.1 Relational Databases

A database is a collection of data. A Relational Database is a database founded on the relational model. The relational model represents data in terms of tuples (rows), grouped into relations (tables). Relational Algebra is used as a query language for Relational Databases.

Because nulls appear in practice in RDBMS, it is important to present a formal definition of Relational Databases and Relational Algebra with respect to null values. Assume, a countably infinite domain $\mathbf{D}$ of constants and a reserved symbol `NULL` that is not in $\mathbf{D}$. A *database schema* $\mathbf{R}$ is a finite set of relation names, where for each $R \in \mathbf{R}$, $att(R)$ denotes the nonempty finite set of attribute names associated with R. The arity of R, denoted as $arity(R)$, is the number of elements of the set $att(R)$. An instance I of $\mathbf{R}$ assigns to each relation symbol $R \in \mathbf{R}$, a finite set of tuples $R^I = \{t_1, \ldots, t_\ell\}$. Each tuple t_j $(1 \leq j \leq \ell)$ is a function that assigns to each attribute in $att(R)$ a value from $(\mathbf{D} \cup \{\texttt{NULL}\})$, denoted as $t : att(R) \rightarrow (\mathbf{D} \cup \{\texttt{NULL}\})$. The value of an attribute A in a tuple t_j is denoted by $t_j.A$. Moreover, $R(t_j)$ is a fact in I if $t_j \in R^I$. The notation $R(t_j) \in I$ is used in this case. We also view instances as sets of facts.

Example 1 The following university database schema is used as a running example throughout the dissertation. See Figure 2.1. The schema of this database consists of tables: `STUDENT(SID, NAME)`, `PROF(PID, NAME, TYPE)`, `COURSE (CID, TITLE, CODE)`, `DEPT(DID, NAME)`, `ENROLLED(SID, CID)` and `EMP(EID, NAME, TYPE, MAN)`.

Moreover, consider the following constraints about the schema of the university:

- `SID` is the primary key of `STUDENT`.
- `PID` is the primary key of `PROF`.
- `CID` is the primary key of `COURSE`.
- `DID` is the primary key of `DEPT`.
- `(SID,CID)` is the primary key of `ENROLLED`.
- `(EID)` is the primary key of `EMP`.
- `CODE` is a foreign key in `COURSE` that references attribute `DID` in `DEPT`.
- `SID` is a foreign key in `ENROLLED` that references attribute `SID` in `STUDENT`.
- `CID` is a foreign key in `ENROLLED` that references attribute `CID` in `COURSE`.
- `MAN` is a foreign key in `EMP` that references attribute `EID` in `EMP`.

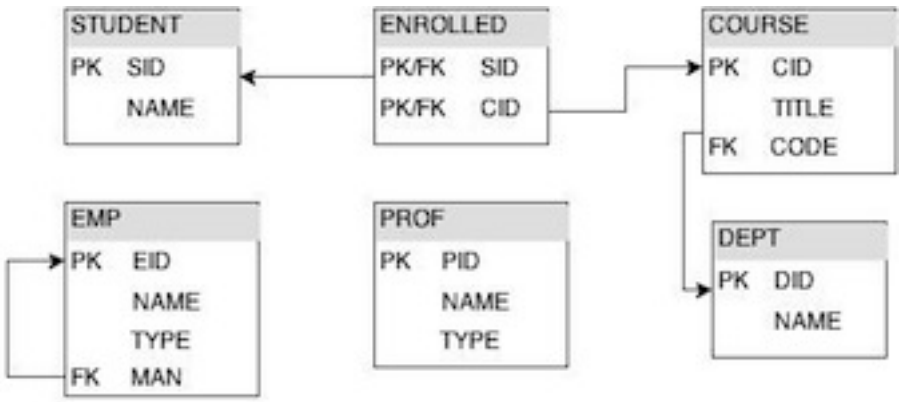


Figure 2.1: Example Relational Schema

2.1.1 Relational Algebra

Relational Algebra consists of operators which take one or two relations as operands and produce one relation as a result. The basic operators of relational algebra are: selection, projection, rename, join, union and difference. Relational Algebra operators can be composed into relational algebraic expressions. These relational algebraic expressions are then used to formulate queries over a Relational Database.

Recall that Relational Databases containing null values are considered. Therefore, the following details the syntax and semantics of Relational Algebra where null values play a role. For additional details of Relational Algebra that are not included in this section, the reader is referred to Abiteboul et al.'s *Foundations of Databases* [6] or Garcia Molina et al.'s *Database Systems: The Complete Book* [60]

Syntax of Relational Alegbra

Assume that $\mathbf{R}$ is a relational schema. The basic operands of relational algebra are a relation and the null value. The basic operations of relational algebra are: selection, projection, rename, join, union and difference. These operations can be composed into relational algebra expressions in order to formulate queries. A relational algebra expression φ over $\mathbf{R}$ and its set of attributes $att(\varphi)$ for the basic operands and operations are recursively defined as follows:

1. **Relation:** If $\varphi = R$ with $R \in \mathbf{R}$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(R)$.

2. **Null:** If $\varphi = \texttt{NULL}_A$, where A is an attribute, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = \{A\}$. Note that $\texttt{NULL}_A$ is simply a symbol to represent a null value. The semantics of this symbol is defined in the next section.

3. **Selection:** If ψ is a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$, $a \in \mathbf{D}$ and φ is any of the following expressions:

   (a) $\sigma_{A=a}(\psi)$

   (b) $\sigma_{A\neq a}(\psi)$

   (c) $\sigma_{\texttt{IsNull}(A)}(\psi)$

   (d) $\sigma_{\texttt{IsNotNull}(A)}(\psi)$

   then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(\psi)$.

4. **Projection:** If ψ is a relational algebra expression over $\mathbf{R}$, $U \subseteq att(\psi)$ and $\varphi = \pi_U(\psi)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = U$.

5. **Rename:** If ψ is a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$ and B is an attribute such that $B \notin att(\psi)$ and $\varphi = \delta_{A\to B}(\psi)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = (att(\psi) \setminus \{A\}) \cup \{B\}$.

6. **Join:** If ψ_1, ψ_2 are relational algebra expressions over $\mathbf{R}$ and $\varphi = (\psi_1 \bowtie \psi_2)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = (att(\psi_1) \cup att(\psi_2))$. The type of join consider is the *natural join*.

7. **Union:** If ψ_1, ψ_2 are relational algebra expressions over $\mathbf{R}$ such that $att(\psi_1) = att(\psi_2)$ and $\varphi = (\psi_1 \cup \psi_2)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(\psi_1)$.

8. **Difference:** If ψ_1, ψ_2 are relational algebra expressions over $\mathbf{R}$ such that $att(\psi_1) = att(\psi_2)$ and $\varphi = (\psi_1 \setminus \psi_2)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(\psi_1)$.

It is important to notice that operators such as $\sigma_{A=B}(\psi)$, selection expressions including boolean combinations in their conditions, equi-join, left-outer join, right-outer join and full-outer join can all be expressed using the previous operators. Additional types of conditional expressions such as $\sigma_{A>v}(\psi)$ and theta-join are defined in [6, 60]. For more details, the reader is referred to the Appendix and [6, 60].

Semantics of Relational Algebra

The following defines the semantics of relational algebra taking into consideration null values. Let $\mathbf{R}$ be a relational schema, I an instance of $\mathbf{R}$ and φ a relational algebra expression over $\mathbf{R}$. The evaluation of a relational expression φ over I, denoted by $[\![\varphi]\!]_I$, is defined recursively as follows:

1. If $\varphi = R$ with $R \in \mathbf{R}$, then $[\![\varphi]\!]_I = R^I$.

2. If $\varphi = \texttt{NULL}_A$, where A is an attribute, then $[\![\varphi]\!]_I = \{t\}$, where $t : \{A\} \rightarrow (\mathbf{D} \cup \{\texttt{NULL}\})$ is a tuple such that $t.A = \texttt{NULL}$.

3. **Selection:** The selection operator selects tuples from a relation satisfying a condition. Let ψ be a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$ and $a \in \mathbf{D}$. Let F be a condition of the form: $A = a$, $A \neq a$, $\texttt{IsNull}(A)$ or $\texttt{IsNotNull}(A)$. Then, the result of a selection operation $\sigma_F(\psi)$ is the set of tuples of ψ for which the formula F is true. Formally,

   (a) If $\varphi = \sigma_{A=a}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A = a\}$.

   (b) If $\varphi = \sigma_{A\neq a}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A \neq \texttt{NULL} \text{ and } t.A \neq a\}$.

   (c) If $\varphi = \sigma_{\texttt{IsNull}(A)}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A = \texttt{NULL}\}$.

   (d) If $\varphi = \sigma_{\texttt{IsNotNull}(A)}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A \neq \texttt{NULL}\}$.

4. **Projection:** The projection operator chooses a subset of the attributes of a relation. Let ψ be a relational algebra expression over $\mathbf{R}$ and $U \subseteq att(\psi)$. The

result of a projection operation $\pi_U(\psi)$, is the set of tuples of ψ which include values of attributes in U.

Formally, if $\varphi = \pi_U(\psi)$, then $[\![\varphi]\!]_I = \{t : U \to (\mathbf{D} \cup \{\texttt{NULL}\}) \mid$ there exists $t' \in [\![\psi]\!]_I$ such that for every $A \in U$: $t.A = t'.A\}$.

5. **Rename:** The rename operator enables changing the name of an attribute. Let ψ be a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$ and B an attribute such that $B \notin att(\psi)$. Then, the result of a rename operation $\delta_{A \to B}(\psi)$ is the set of tuples of ψ where the attribute A is now B.

   Formally, if $\varphi = \delta_{A \to B}(\psi)$, then $[\![\varphi]\!]_I = \{t : att(\varphi) \to (\mathbf{D} \cup \{\texttt{NULL}\}) \mid$ there exists $t' \in [\![\psi]\!]_I$ such that $t.B = t'.A$ and for every $C \in (att(\varphi) \setminus \{B\})$: $t.C = t'.C\}$.

6. **Join:** Join combines two relations into one on the basis of a condition. Let ψ_1 and ψ_2 be relational algebra expressions over $\mathbf{R}$. Then the result of a join operator $(\psi_1 \bowtie \psi_2)$ is the set of all possible pairs of tuples from ψ_1 and ψ_2 where the equality of all common attributes hold. This type of join is known as *natural join*.

   Formally, if $\varphi = (\psi_1 \bowtie \psi_2)$, then $[\![\varphi]\!]_I = \{t : att(\varphi) \to (\mathbf{D} \cup \{\texttt{NULL}\}) \mid$ there exist $t_1 \in [\![\psi_1]\!]_I$ and $t_2 \in [\![\psi_2]\!]_I$ such that for every $A \in (att(\psi_1) \cap att(\psi_2))$: $t.A = t_1.A = t_2.A \neq \texttt{NULL}$, for every $A \in (att(\psi_1) \setminus att(\psi_2))$: $t.A = t_1.A$, and for every $A \in (att(\psi_2) \setminus att(\psi_1))$: $t.A = t_2.A\}$.

7. **Union:** Union is the relation containing all tuples from both relations. Let ψ_1 and ψ_2 be relational algebra expressions over $\mathbf{R}$ such that $att(\psi_1) = att(\psi_2)$. Then the result of a union operator $(\psi_1 \cup \psi_2)$ is the set of tuples belonging to the union (in the set-theoretic sense) of ψ_1 and ψ_2.

   Formally, if $\varphi = (\psi_1 \cup \psi_2)$, then $[\![\varphi]\!]_I = [\![\psi_1]\!]_I \cup [\![\psi_2]\!]_I$.

8. **Difference:** Difference is the relation containing all tuples of the first relation that do not appear in the second relation. Let ψ_1 and ψ_2 be relational algebra expressions over $\mathbf{R}$ such that $att(\psi_1) = att(\psi_2)$. Then the result of a difference operator $(\psi_1 \setminus \psi_2)$ is the set of tuples belonging to the difference (in the set-theoretic sense) of ψ_1 and ψ_2.

   Formally, if $\varphi = (\psi_1 \setminus \psi_2)$, then $[\![\varphi]\!]_I = [\![\psi_1]\!]_I \setminus [\![\psi_2]\!]_I$.

The semantics of the additional types of conditional expressions for selection are presented in [6, 60]. The semantics of theta-join, equi-join, left-outer join, right-outer join and full-outer join are are defined analogous to the natural join. For details, see [6, 60]

2.1.2 Integrity Constraints

Integrity constraints are properties that are supposed to be satisfied by all instances of a relational schema. Two types of integrity constraints are considered: *keys* and *foreign keys*. Let $\mathbf{R}$ be a relational schema.

Key: A key is a set of attributes define whose values guarantee only one tuple exists for each value and such value can not be null. A key φ over $\mathbf{R}$ is an expression of the form $R[A_1, \ldots, A_m]$, where $R \in \mathbf{R}$ and $\emptyset \subsetneq \{A_1, \ldots, A_m\} \subseteq att(R)$. Given an instance I of $\mathbf{R}$, I satisfies key φ, denoted by $I \models \varphi$, if:

1. for every $t \in R^I$ and $k \in \{1, \ldots, m\}$, it holds that $t.A_k \neq \texttt{NULL}$, *and*
2. for every $t_1, t_2 \in R^I$, if $t_1.A_k = t_2.A_k$ for every $k \in \{1, \ldots, m\}$, then $t_1 = t_2$.

Foreign Key: A foreign key is a set of attributes in one relation that uniquely identifies a tuple in another relation. A foreign key over $\mathbf{R}$ is an expression of the form $R[A_1, \ldots, A_m] \subseteq_{\text{FK}} S[B_1, \ldots, B_m]$, where $R, S \in \mathbf{R}$, $\emptyset \subsetneq \{A_1, \ldots, A_m\} \subseteq att(R)$ and $\emptyset \subsetneq \{B_1, \ldots, B_m\} \subseteq att(S)$. Given an instance I of $\mathbf{R}$, I satisfies foreign key φ, denoted by $I \models \varphi$, if $I \models S[B_1, \ldots, B_m]$ and for every tuple t in R^I: either

1. there exists $k \in \{1, \ldots, m\}$ such that $t.A_k = \texttt{NULL}$, *or*

2. there exists a tuple s in S^I such that $t.A_k = s.B_k$ for every $k \in \{1, \ldots, m\}$.

Finally, given a relational schema $\mathbf{R}$, a set Σ of keys and foreign keys is said to be a *set of primary keys (PKs) and foreign keys (FKs) integrity constraints over* $\mathbf{R}$ if:

1. for every $\varphi \in \Sigma$, it holds that φ is either a key or a foreign key over $\mathbf{R}$, and

2. there are no two distinct keys in Σ of the form $R[A_1, \ldots, A_m]$ and $R[B_1, \ldots, B_n]$ (that is, that mention the same relation name R). In other words, a relation can only have one key.

Moreover, an instance I of $\mathbf{R}$ satisfies Σ, denoted by $I \models \Sigma$, if for every $\varphi \in \Sigma$, it holds that $I \models \varphi$.

2.2 Semantic Web

The Semantic Web is an extension to the Web that enables intelligent access to data on the Web. The technologies supporting the Semantic Web consist of a set of standards: RDF as the graph data model, OWL as the ontology language, and SPARQL as the query language

2.2.1 RDF

RDF stands for Resource Description Framework, which is a framework for representing information about resources in the Web. By resource, we mean anything in the world including physical things, documents, abstract concepts, etc[1]. RDF considers three types of values: resource identifiers (IRIs) to denote resources, literals to denote values such as strings, and blank nodes to denote the existence of

[1]The term "entity" can be considered synonymous to resource.

unnamed resources which are existentially quantified variables that can be used to make statements about unknown (but existent) resources.

Assume there are pairwise disjoint infinite sets $\mathbf{I}$ (IRIs), $\mathbf{B}$ (blank nodes) and $\mathbf{L}$ (literals). A tuple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object. A finite set of RDF triples is called an RDF graph. Assume that `triple` is a ternary predicate that stores RDF graphs in the obvious way: every triple $(a, b, c) \in G$ is stored as $\texttt{triple}(a, b, c)$. Moreover, assume the existence of an infinite set $\mathbf{V}$ of variables disjoint from the above sets, and assume that every element in $\mathbf{V}$ starts with the symbol "?".

Example 2 Consider representing the statement "There is a student whose name is Juan Sequeda" in RDF. This can be represented with two RDF triples. The first RDF triple

$$\texttt{triple}(\texttt{http://juansequeda.com\#me}, \texttt{rdf:type}, \texttt{foaf:Person})$$

states that the resource identified by `http://juansequeda.com#me` is of type Person. The type relationship is represented with `rdf:type`. Additionally, the concept Person is identified by the IRI `foaf:Person`. Note that `rdf:` and `foaf:` are being used instead of a full IRI. These are prefixes that replace a part of the IRI[2]. The second RDF triple

$$\texttt{triple}(\texttt{http://juansequeda.com\#me}, \texttt{foaf:name}, \texttt{"Juan Sequeda"})$$

states that `http://juansequeda.com#me` has a name which is "Juan Sequeda". The concept of name is identified by the IRI `foaf:name`.

[2]The prefix "`rdf:`" represents `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, hence the full IRI for `rdf:type` is `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. Additionally, the prefix "`foaf:`" represents `http://xmlns.com/foaf/0.1/`, hence the full IRI for `foaf:Person` is `http://xmlns.com/foaf/0.1/Person`

2.2.2 OWL

OWL stands for Web Ontology Language, which is the language to represent ontologies on the Web. In order to define the notion of ontology, the following set of reserved keywords are defined as **O**: {`subClass`, `subProp`, `dom`, `range`, `rdf:type`, `equivClass`, `equivProp`, `inverse`, `symProp`, `transProp`, `differentFrom`}. Assume that $\mathbf{O} \subseteq \mathbf{I}$. Two types of RDF triples are distinguished: ontological and assertional. Ontological RDF triples define the ontology. Assertional RDF triples define the facts. The formal definitions are the following:

Definition 1 (Ontological RDF Triple) *Following the definition presented by Weaver and Hendler [131], an RDF triple* (a, b, c) *is ontological if:*

1. $a \in (\mathbf{I} \setminus \mathbf{O})$, *and*
2. *either* $b \in (\mathbf{O} \setminus \{\texttt{rdf:type}\})$ *and* $c \in (\mathbf{I} \setminus \mathbf{O})$, *or* $b =$ `rdf:type` *and* c *is either* `symProp` *or* `transProp`.

In other words, an ontological RDF triple will always have as a subject an element in **I** but not in **O**. There are two types of ontological RDF triples. First, the predicate is an element in **O** but not `rdf:type` and the object is an element in **I** but not in **O**. Second, if the predicate is `rdf:type`, then the object is either `symProp` or `transProp`.

Definition 2 (Assertional RDF Triple) *An RDF triple* (a, b, c) *is assertional if it is not ontological.*

Definition 3 (Ontology) *An ontology* $\mathcal{O}$ *is defined as a finite set of ontological RDF triples.*

The semantics of an ontology $\mathcal{O}$ is usually defined by representing it as a set of description logic axioms, and then relying on the semantics of the logic [18] (which,

in turn, is derived from the semantics of first-order logic). It is more convenient to directly define a set of first-order formulae, denoted as $\Sigma_{\mathcal{O}}$, to encode the ontology $\mathcal{O}$. The semantics of each ontological triple of an ontology, $t \in \mathcal{O}$, is defined as a first-order formula φ_t over the predicate `triple`. Definitions 4 - 13 presents the first-order formula for ontological triples. Finally, the set $\Sigma_{\mathcal{O}}$ of first-order formulae encoding the ontology $\mathcal{O}$ is define as $\{\varphi_t \mid t \in \mathcal{O}\}$.

Definition 4 (Subclass) If a is a subclass of b and x is an instance of a, then x is an instance of b. The first-order formula is:

$$\varphi_{(a,\texttt{subClass},b)} = \forall x \left(\texttt{triple}(x, \texttt{rdf:type}, a) \rightarrow \texttt{triple}(x, \texttt{rdf:type}, b)\right)$$

Definition 5 (Subproperty) If a is a subproperty of b, then all pairs of resources (x, y) which are related by a are also related by b. The first-order formula is:

$$\varphi_{(a,\texttt{subProp},b)} \quad = \quad \forall x \forall y \left(\texttt{triple}(x, a, y) \rightarrow \texttt{triple}(x, b, y)\right)$$

Definition 6 (Domain) If a has a domain b then any resource x that is related to a is an instance of b. The first-order formula is:

$$\varphi_{(a,\texttt{dom},b)} \quad = \quad \forall x \forall y \left(\texttt{triple}(x, a, y) \rightarrow \texttt{triple}(x, \texttt{rdf:type}, b)\right)$$

Definition 7 (Range) If a has a range b then any resource y that is related to a is an instance of b. The first-order formula is:

$$\varphi_{(a,\texttt{range},b)} \quad = \quad \forall x \forall y \left(\texttt{triple}(x, a, y) \rightarrow \texttt{triple}(y, \texttt{rdf:type}, b)\right)$$

Definition 8 (Equivalent Class) If a has an equivalent class of b and x is an instance of a, then x is an instance of b. Conversely, if x is an instance of b, then x

is an instance of a. The first-order formula is:

$$\varphi_{(a,\texttt{equivClass},b)} \quad = \quad \forall x \, (\texttt{triple}(x, \texttt{rdf:type}, a) \leftrightarrow \texttt{triple}(x, \texttt{rdf:type}, b))$$

Definition 9 (Equivalent Property) If a has an equivalent property of b, then all pairs of resources (x, y) which are related by a are also related by b. Conversely, all pairs of resources (x, y) which are related by b are also related by a. The first-order formula is:

$$\varphi_{(a,\texttt{equivProp},b)} \quad = \quad \forall x \forall y \, (\texttt{triple}(x, a, y) \leftrightarrow \texttt{triple}(x, b, y))$$

Definition 10 (Inverse Property) If a has an inverse property of b, then all pairs of resources (x, y) which are related by a are also related by b by the pair (y,x). Conversely, all pairs of resources (y, x) which are related by b are also related by a by the pair (x,y) The first-order formula is:

$$\varphi_{(a,\texttt{inverse},b)} \quad = \quad \forall x \forall y \, (\texttt{triple}(x, a, y) \leftrightarrow \texttt{triple}(y, b, x))$$

Definition 11 (Symmetric Property) If a is a symmetric property, then all pairs of resources (x,y) which are related by a are also related as the pair (y,x). The first-order formula is:

$$\varphi_{(a,\texttt{rdf:type},\texttt{symProp})} \quad = \quad \forall x \forall y \, (\texttt{triple}(x, a, y) \rightarrow \texttt{triple}(y, a, x))$$

Definition 12 (Transitive Property) If a is a transitive property, and for all pairs of resources (x,y) and (y, z) which are related by a then the pair (x,z) is also related by a. The first-order formula is:

$$\varphi_{(a,\texttt{rdf:type},\texttt{transProp})} \quad = \quad \forall x \forall y \forall z \, (\texttt{triple}(x, a, y) \wedge \texttt{triple}(y, a, z) \rightarrow \texttt{triple}(x, a, z))$$

Definition 13 (Different From) If a is different from b then $a \neq b$. The first-order formula is:

$$\varphi_{(a,\texttt{differentFrom},b)} \quad = \quad a \neq b$$

Given that the semantics of an ontology $\mathcal{O}$ has been defined as set of first order logic formulae $\Sigma_{\mathcal{O}}$ and a RDF graph G using the predicate `triple`, then $\Sigma_{\mathcal{O}} \cup G$ is consistent (and inconsistent) in the usual sense of First Order Logic.

Example 3 The following ontology is used as a running example which states that an Executive and ITEmployee are both Employees. Additionally that the property hasSuperior is a transitive relationship from an Employee to another Employee.

`triple(:Executive, subClass, :Employee)`

`triple(:Programmer, subClass, :ITEmployee)`

`triple(:SysAdmin, subClass, :ITEmployee)`

`triple(:ITEmployee, subClass, :Employee)`

`triple(:hasSuperior, rdf:type, transProp)`

`triple(:hasSuperior, dom, :Employee)`

`triple(:hasSuperior, range, :Employee)`

Ontology Profiles

The expressiveness of an ontology language can be specified by profiles. The Semantic Web technology stack specifies four ontology profiles: RDFS, OWL 2 EL, OWL 2 QL and OWL 2 RL [31, 99].

RDF Schema (RDFS) extends RDF as a schema language for RDF and a lightweight ontology language [31]. It includes constructs to declare classes, hierarchies between classes and properties and relate the domain and range of a property to a certain class. Ontological triples with `subClass`, `subProp`, `dom`, `range`, `rdf:type`,

`equivClass`, `equivProp` are in this profile. The following three profiles, OWL 2 EL, QL and RL, extend the expressiveness of RDFS.

OWL 2 EL profile is used to represent ontologies that define very large numbers of classes and/or properties with transitivity. This language has been tailored to model large life science ontologies, while still supporting efficient reasoning. OWL 2 EL is based on the EL++ Description Logic [14]. Ontological triples with `transProp` are in this profile.

OWL 2 QL provides constructs to express conceptual models such as UML class diagrams and ER diagrams. This language was designed so that data that is stored in a standard relational database system can be queried through an ontology via rewriting mechanisms. OWL 2 QL is based on the DL-Lite family of description logics [36]. Ontological triples with `inverse` and `symProp` are in this profile.

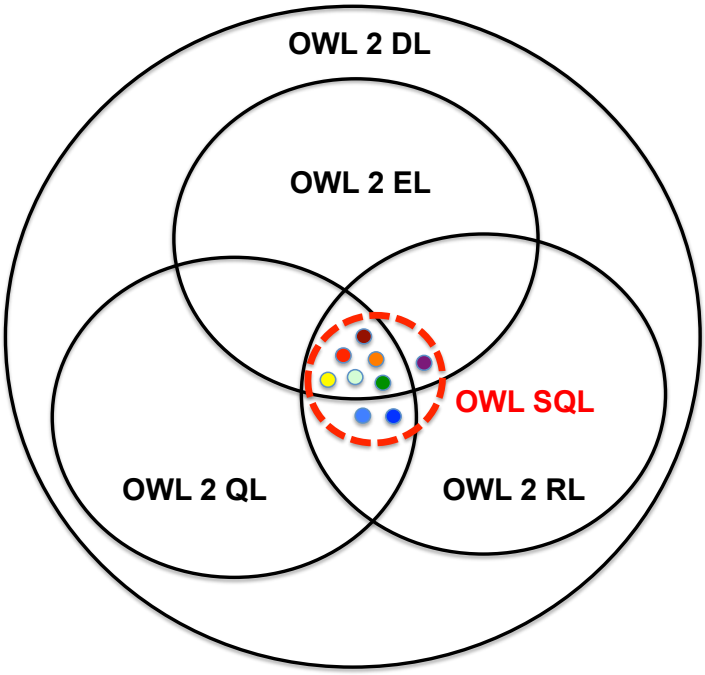
OWL 2 RL provides constructs to represent rules in ontologies. This language has been tailored for rule-based reasoning engines. OWL 2 RL is based on Description Logic Programs (DLP) [68]. Ontological triples with `inverse` and `symProp` are also in this profile.

The ontology expressivity considered in this dissertation (as defined in Definitions 4 - 13) is not specific to a single OWL profile. Thus, we propose a new ontology profile, OWL-SQL, which expresses the types of ontologies considered in this dissertation. Figure 2.2 denotes the expressivity of OWL-SQL with respect to the OWL 2 EL, QL and RL profiles.

The expressivity of OWL-SQL is subsumed by early ontology profile proposals known as RDFS-Plus [7], OWL-LD[63] and RDFS 3.0[76].

2.2.3 SPARQL

SPARQL is the standard query language for RDF. SPARQL is a graph pattern matching query language and has a syntax similar to SQL. A SPARQL query con-



		EL	QL	RL
●	subClass	X	X	X
●	subProp	X	X	X
●	domain	X	X	X
●	range	X	X	X
●	eqClass	X	X	X
●	eqProp	X	X	X
●	inverseProp		X	X
●	symProp		X	X
●	transProp	X		X

Figure 2.2: OWL-SQL, proposed OWL profile

tains a set of triple patterns called basic graph patterns. Triple patterns are similar to RDF triples with the exception that the subject, predicate or object can be variables (denoted by a leading question mark "?"). The result of a basic graph pattern query is a list of all variable bindings that cause a query pattern to match a subgraph of an RDF graph.

Example 4 Consider the RDF triples in Example 2. The following SPARQL query asks for all names of people.

```
SELECT ?n
WHERE {
  ?s rdf:type foaf:Person.
  ?s foaf:name ?n.
}
```

The basic graph pattern consists of two triple patterns. Matching these triple patterns with the RDF triples gives the answer `"Juan Sequeda"`.

The official syntax of SPARQL [108, 73] includes operators `OPTIONAL`, `UNION`, `FILTER`, `SELECT`, `AS` and concatenation via a point symbol (`.`), to construct graph pattern expressions. The syntax of the language uses brackets `{ }` to group patterns, and there are some implicit rules of precedence and association.

In order to avoid ambiguities in parsing, the approach proposed by Perez et al. [104] is used. The syntax of SPARQL graph patterns is presented in a traditional algebraic formalism, using operators AND (`.`), UNION (`UNION`), OPT (`OPTIONAL`), MINUS (`MINUS`), FILTER (`FILTER`), SELECT (`SELECT`) and AS (`AS`).

More precisely, a SPARQL graph pattern expression is defined recursively as follows.

1. { } is a graph pattern (the empty graph pattern).
2. 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)[3].
3. If P_1 and P_2 are graph patterns, then expressions $(P_1$ AND $P_2)$, $(P_1$ OPT $P_2)$, $(P_1$ UNION $P_2)$ and $(P_1$ MINUS $P_2)$ are graph patterns.
4. If P is a graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern.
5. If P is a graph pattern and $?A_1$, ..., $?A_m$, $?B_1$, ..., $?B_m$, $?C_1$, ..., $?C_n$ is a sequence of pairwise distinct elements from $\mathbf{V}$ ($m \geq 0$ and $n \geq 0$) such that none of the variables $?B_i$ ($1 \leq i \leq m$) are mentioned in P, then

$$(\text{SELECT } \{?A_1 \text{ AS } ?B_1, \ldots, ?A_m \text{ AS } ?B_m, ?C_1, \ldots, ?C_n\} \ P)$$

[3]Recall that $\mathbf{V}$ is an infinite set of variables disjoint from $\mathbf{I}$, $\mathbf{B}$ and $\mathbf{L}$ and that every element in $\mathbf{V}$ starts with the symbol "?". See Section 2.2.1.

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 [108, 73] for a complete list). A restriction is made 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.

The version of SPARQL used in this dissertation includes the following SPARQL 1.1 features: the operator MINUS, the possibility of nesting the SELECT operator and the operator AS [73].

The answer of a SPARQL query P over an RDF graph G is a finite set of *mappings*, where a mapping μ is a partial function from the set $\mathbf{V}$ of variables to $(\mathbf{I} \cup \mathbf{L} \cup \mathbf{B})$.

The semantics of SPARQL is defined as a function $[\![\cdot]\!]_G$ that, given an RDF graph G, takes a graph pattern expression and returns a set of mappings. The reader is referred to the Appendix for more detail.

Chapter 3

Direct Mapping Relational Databases to RDF and OWL

[1] This chapter presents how a Relational Database can be represented as a Semantic Web data source. The goal is to define a mapping of a Relational Database, including both its instances and its schema, to RDF and OWL. This mapping serves as a default and automatic way of representing a Relational Database as a Semantic Web data source. This mapping has been coined as the *direct mapping*. Four properties are presented as correctness criteria for the direct mapping.

Two properties are fundamental to a direct mapping; information preservation and query preservation. *Information preservation* speaks to the ability to reconstruct the original relational database from the result of the direct mapping. *Query preservation* means that every query over a relational database can be translated into an equivalent query over the result of the direct mapping.

In addition two desirable properties are considered; monotonicity and semantics preservation. *Monotonicity* assures that a re-computation of the mapping is not needed after any inserts to the database. A direct mapping is semantics preserving if the satisfaction of a set of integrity constraints (i.e primary keys and foreign keys) are encoded in the mapping result. However, semantics preservation of a direct mapping is complex due to different assumptions made by Relational Databases and the Semantic Web. Relational Databases make the Closed World Assumption (CWA) which means that a statement is inferred to be false if it is not know to be true. The Semantic Web makes an Open World Assumption (OWA) which means that a statement cannot be inferred to be false on the basis of failing to prove it. In other words, what causes an inconsistency in a Relational Database may result in inference of new knowledge in the Semantic Web.

Semantics preservation is analyzed from two perspectives. First, a direct mapping is said to be *positive semantics preserving* if the result of applying a direct

[1]Part of this chapter has been published as: Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. 2012. On directly mapping relational databases to RDF and OWL. In Proceedings of the 21st International Conference on World Wide Web (WWW '12). ACM, New York, NY, USA, 649-658. Marcelo Arenas and Daniel P. Miranker were advisors for this work.

mapping to a database, that satisfies the integrity constraints, results in a consistent RDF graph[2]. Second, a direct mapping is *negative semantics preserving* if the result of applying a direct mapping to a database, that does not satisfy the integrity constraints, results in an inconsistent RDF graph. If a direct mapping is both positive and negative semantics preserving, it is then *full semantics preserving*.

To the best of our knowledge, this is the first direct mapping from a Relational Database to RDF and OWL that has been thoroughly studied with respect to these properties. The proposed direct mapping is monotone, information preserving, query preserving and positive semantics preserving even in the general and practical scenario where relational databases contain null values. However, given an inconsistent database, meaning a database that violates an integrity constraint, the proposed direct mapping generates a consistent RDF graph, when it should return an inconsistent RDF graph. Therefore, it is not negative semantics preserving, nor is it full semantics preserving.

Why our direct mapping is not full semantics preserving is analyzed. The result is that monotonicity is an obstacle. It is first proved that if only primary keys are considered, it is possible to have a monotone direct mapping that is full semantics preserving. However this result is not sufficient because it dismisses foreign keys. Unfortunately, it is proven that no monotone direct mapping of a relational database, with primary keys and foreign keys, to RDF and OWL can be full semantics preserving because of the Relational Database's CWA and the Semantic Web's OWA. This result, that no monotone direct mapping can be full semantics preserving has an important implication in real world applications; if an inconsistent Relational Database is migrated to the Semantic Web using a monotone direct mapping, be prepared to experience consistency when what one would expect inconsistency. Finally, a non-monotone direct mapping that overcomes the aforementioned limitation

[2]Recall that consistency (and inconsistency) of an RDF graph is in the usual sense of First Order Logic (see Preliminaries)

is presented.

3.1 Direct Mapping Definition

A direct mapping is a default way to translate Relational Databases into RDF and OWL (without any input from the user on how the relational data should be translated). Let the input of a direct mapping, $\mathcal{M}$, be a relational schema $\mathbf{R}$, a set Σ of PKs and FKs over $\mathbf{R}$ and an instance I of $\mathbf{R}$. The output is an RDF graph and an OWL ontology[3].

Assume $\mathcal{G}$ is the set of all possible RDF graphs and $\mathcal{RC}$ is the set of all three-tuples of the form $(\mathbf{R}, \Sigma, I)$ such that $\mathbf{R}$ is a relational schema, Σ is a set of PKs and FKs over $\mathbf{R}$ and I is an instance of $\mathbf{R}$.

Definition 14 (Direct mapping) *A direct mapping $\mathcal{M}$ is a total function from $\mathcal{RC}$ to $\mathcal{G}$.*

3.2 Fundamental Properties

Two fundamental properties of direct mappings are information preservation and query preservation. Information preservation is a fundamental property because it guarantees that the mapping does not lose information. Query preservation is also a fundamental property because it guarantees that, everything that can be extracted from the relational database by a relational algebra query, can also be extracted from the resulting RDF graph by a SPARQL query.

3.2.1 Information Preservation

A direct mapping is information preserving if it does not lose any information about the relational instance being translated. That is, if there exists a way to recover

[3]Recall that an OWL ontology is defined as a set of finite ontological RDF triples

the original database instance from the RDF graph resulting from the translation process. Assume that $\mathcal{I}$ is the set of all possible relational instances.

Definition 15 (Information preservation) *A direct mapping $\mathcal{M}$ is* information preserving *if there exists a computable mapping $\mathcal{N} : \mathcal{G} \to \mathcal{I}$ such that for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$, and instance I of $\mathbf{R}$ satisfying Σ:*

$$\mathcal{N}(\mathcal{M}(\mathbf{R}, \Sigma, I)) \quad = \quad I$$

A mapping $\mathcal{N} : \mathcal{G} \to \mathcal{I}$ is computable if there exists an algorithm that, given $G \in \mathcal{G}$, computes $\mathcal{N}(G)$.

3.2.2 Query Preservation

A direct mapping is query preserving if every relational algebra query over a relational database can be translated into an equivalent SPARQL query over the RDF graph resulting from the mapping. That is, query preservation ensures that every relational query can be evaluated using the mapped RDF data.

To define query preservation, the focus is on relational queries that can be expressed in relational algebra [6] and RDF queries that can be expressed in SPARQL [108, 104]. Given the mismatch in the formats of these query languages, a function *tr* is used to convert tuples returned by relational algebra queries into mappings returned by SPARQL[4]. Given a relational schema $\mathbf{R}$, a relation name $R \in \mathbf{R}$, an instance I of $\mathbf{R}$ and a tuple $t \in R^I$, define $tr(t)$ as the mapping μ such that:

1. the domain of μ is $\{?A \mid A \in att(R)$ and $t.A \neq \texttt{NULL}\}$, and

[4]Recall that the answer of a SPARQL query is a finite set of mappings, where a mapping μ is a partial function from the set $\mathbf{V}$ of variables to $(\mathbf{I} \cup \mathbf{L} \cup \mathbf{B})$. See Section 2.2.3 for more details.

2. $\mu(?A) = t.A$ for every A in the domain of μ.

Example 5 Assume that a relational schema contains a relation name `STUDENT` and attributes `ID`, `NAME` and `AGE`. Moreover, assume that t is a tuple in this relation such that $t.\texttt{ID} = 1$, $t.\texttt{NAME} =$ John and $t.\texttt{AGE} = \texttt{NULL}$. Then, $tr(t) = \mu$, where the domain of μ is $\{\texttt{?ID}, \texttt{?NAME}\}$, $\mu(\texttt{?ID}) = 1$ and $\mu(\texttt{?NAME}) =$ John.

Definition 16 (Query preservation) *A direct mapping $\mathcal{M}$ is* query preserving *if for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and relational algebra query Q over $\mathbf{R}$, there exists a SPARQL query $Q^\star$ such that for every instance I of $\mathbf{R}$ satisfying Σ:*

$$tr([\![Q]\!]_I) \quad = \quad [\![Q^\star]\!]_{\mathcal{M}(\mathbf{R},\Sigma,I)}$$

It is important to notice that in the context of this research, information preservation and query preservation are incomparable properties. If a direct mapping $\mathcal{M}$ is information preserving, it does not guarantee that every relational algebra query Q can be rewritten into an equivalent SPARQL query over the translated data, as $\mathcal{M}$ could transform source relational databases in such a way that a more expressive query language is needed to express Q over the generated RDF graphs. On the other hand, a mapping $\mathcal{M}$ can be query preserving and not information preserving if the information about the schema of the relational database being translated is not stored. For example, a direct mapping that includes information about these relational schemas is presented in Section 3.4. It will become clear that if such information is not stored, then the direct mapping would be query preserving but not information preserving.

3.3 Desirable Properties

Two desirable properties of direct mappings are monotonicity and semantic preservation. Monotonicity permits the mapping to be calculated only over the new inserted data instead of recalculating the mapping for the entire database after each insert. Semantic preservation enables us to understand the expressive power of a direct mapping and, its ability to properly deal with integrity constraints.

3.3.1 Monotonicity

Given two database instances I_1 and I_2 over a relational schema $\mathbf{R}$, instance I_1 is said to be contained in instance I_2, denoted by $I_1 \subseteq I_2$, if for every $R \in \mathbf{R}$, $R^{I_1} \subseteq R^{I_2}$. A direct mapping $\mathcal{M}$ is monotone if for any such pair of instances, the result of mapping I_2 contains the result of mapping I_1. In other words, if new data is inserted in the database, then the elements of the mapping that are already computed are unaltered.

Definition 17 (Monotonicity) *A direct mapping $\mathcal{M}$ is monotone if for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$, and instances I_1, I_2 of $\mathbf{R}$ such that $I_1 \subseteq I_2$:*

$$\mathcal{M}(\mathbf{R}, \Sigma, I_1) \quad \subseteq \quad \mathcal{M}(\mathbf{R}, \Sigma, I_2)$$

3.3.2 Semantics preservation

A direct mapping is semantics preserving if the satisfaction of a set of PKs and FKs by a Relational Database is encoded in the translation process. Two forms of semantics preservation are defined: positive and negative. Given a relational schema $\mathbf{R}$, a set Σ of PKs and FKs over $\mathbf{R}$ and an instance I of $\mathbf{R}$, a positive semantics preserving mapping should generate from I a consistent RDF graph if $I \models \Sigma$. A

negative semantics preserving mapping should generate from I an inconsistent RDF graph if $I \not\models \Sigma$. In other words, if a database satisfies the integrity constraints, the result of applying a direct mapping should be a consistent RDF graph. However, if a database does not satisfy the integrity constraints, the result of applying a direct mapping should be an inconsistent RDF graph.

Definition 18 (Positive Semantics Preservation) *A direct mapping $\mathcal{M}$ is positive semantics preserving if for every relation schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and instance I of $\mathbf{R}$:*

$$\textit{If } I \models \Sigma \quad \textit{then} \quad \mathcal{M}(\mathbf{R}, \Sigma, I) \textit{ is consistent under the OWL semantics.}$$

Definition 19 (Negative Semantics Preservation) *A direct mapping $\mathcal{M}$ is positive semantics preserving if for every relation schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and instance I of $\mathbf{R}$:*

$$\textit{If } I \not\models \Sigma \quad \textit{then} \quad \mathcal{M}(\mathbf{R}, \Sigma, I) \textit{ is inconsistent under the OWL semantics.}$$

If a direct mapping is both positive and negative semantics preserving, it is then full semantics preserving.

Definition 20 (Full Semantics Preservation) *A direct mapping $\mathcal{M}$ is semantics preserving if for every relation schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and instance I of $\mathbf{R}$:*

$$I \models \Sigma \quad \textit{iff} \quad \mathcal{M}(\mathbf{R}, \Sigma, I) \textit{ is consistent under the OWL semantics.}$$

3.4 The Direct Mapping $\mathcal{DM}$

The direct mapping $\mathcal{DM}$ proposed in this research integrates and extends the functionalities of the direct mappings presented in our previous work [129]. $\mathcal{DM}$ is

defined as a set of Datalog rules[5] comprising the translation of relational schemas and relational instances.

Section 3.4.1 presents the predicates that are used to store a relational database and server as the input of $\mathcal{DM}$. Section 3.4.2 presents the predicates that are used to store an ontology and the Datalog rules used to generate an ontology from the relational schema and the set of PKs and FKs. Section 3.4.3 presents the Datalog rules used to generate the OWL vocabulary from the ontology that was derived from the relational schema and a set of PKs and FKs. Finally, Section 3.4.4 presents the Datalog rules that generates RDF triples from a relational instance.

3.4.1 Storing relational databases

Given that the direct mapping $\mathcal{DM}$ is specified by a set of Datalog rules, its input $(\mathbf{R}, \Sigma, I)$ has to be encoded as a set of relations. The following predicates are used to store a relational schema $\mathbf{R}$ and a set Σ of PKs and FKs over $\mathbf{R}$.

- $\textsc{Rel}(r)$: Indicates that r is a relation name in $\mathbf{R}$, e.g. $\textsc{Rel}(\texttt{"STUDENT"})$ indicates that `STUDENT` is a relation name.[6]

- $\textsc{Attr}(a, r)$: Indicates that a is an attribute in the relation r in $\mathbf{R}$, e.g. $\textsc{Attr}(\texttt{"NAME"}, \texttt{"STUDENT"})$ holds.

- $\mathrm{PK}_n(a_1, \ldots, a_n, r)$: Indicates that $r[a_1, \ldots, a_n]$ is a primary key in Σ, e.g. $\mathrm{PK}_1(\texttt{"SID"}, \texttt{"STUDENT"})$ holds.

- $\mathrm{FK}_n(a_1, \ldots, a_n, r, b_1, \ldots, b_n, s)$: Indicates that $r[a_1, \ldots, a_n] \subseteq_{\mathrm{FK}} s[b_1, \ldots, b_n]$ is a foreign key in Σ, e.g. $\mathrm{FK}_1(\texttt{"CODE"}, \texttt{"COURSE"}, \texttt{"DID"}, \texttt{"DEPT"})$ holds.

Moreover, the following predicate is used to store the tuples in a relational instance I of a relational schema $\mathbf{R}$.

[5] The reader is referred to [6] for the syntax and semantics of Datalog.
[6] As is customary, double quotes are used to delimit strings.

- 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 t_1 of table `STUDENT` such that t_1.`SID` = `"1"` and t_1.`NAME` = `NULL` is stored by using the facts VALUE(`"1"`, `"SID"`, `"id1"`, `"STUDENT"`) and VALUE(`NULL`, `"NAME"`, `"id1"`, `"STUDENT"`), assuming that `id1` is the identifier of tuple t_1.

3.4.2 Storing an ontology

The first step is to synthesize an ontology from the relational schema and the set of PKs and FKs given as input. We have found that when a relational schema has been created using standard data engineering methodology with supporting Computer-Aided Software Engineering (CASE) tools, the synthesized ontology can be quite good. Since the quality of a databases data modeling is rarely of that high quality, and the meaning of "*good ontology*" is subjective, we mitigate the controversy by calling the result collection of ontology declarations a *putative ontology* (PO).

Each relation name in the schema is classified as a class or a binary relation (which is used to represent a many-to-many relationship between entities in an ER/UML diagram). Foreign keys are represented as object properties and attributes of relations as data type properties. More specifically, the following predicates are used to store the extracted ontology:

- CLASS(c): Indicates that c is a class.
- OP$_n(p_1, \ldots, p_n, d, r)$: Indicates that $p_1, \ldots, p_n$ $(n \geq 1)$ form an object property with domain d and range r.
- DTP(p, d): Indicates that p is a data type property with domain d.

The above predicates are defined by the Datalog rules described in the following sections.

Identifying Binary Relations

Binary relations are a special type of relation because they represent a relationship between relations. Binary relations are defined by auxiliary predicates in order to facilitate identifying classes, object properties and data type properties. A relation R is a binary relation between two relations S and T if

1. both S and T are different from R,
2. R has exactly two attributes A and B, which form a primary key of R,
3. A is the attribute of a foreign key in R that points to S,
4. B is the attribute of a foreign key in R that points to T,
5. A is not the attribute of two distinct foreign keys in R,
6. B is not the attribute of two distinct foreign keys in R,
7. A and B are not the attributes of a composite foreign key in R, and
8. relation R does not have incoming foreign keys.

The corresponding Datalog rule is:

$$\begin{aligned}\textsc{BinRel}(R, A, B, S, C, T, D) \leftarrow \\ &\mathrm{PK}_2(A, B, R), \neg\textsc{ThreeAttr}(R), \\ &\mathrm{FK}_1(A, R, C, S), R \neq S, \mathrm{FK}_1(B, R, D, T), R \neq T, \\ &\neg\textsc{TwoFK}(A, R), \neg\textsc{TwoFK}(B, R), \\ &\neg\textsc{OneFK}(A, B, R), \neg\textsc{FKto}(R).\end{aligned} \tag{3.1}$$

In a Datalog rule, negation is represented with the symbol $\neg$ and upper case letters are used to denote variables. Thus, the previous rule states that the relation

R is a binary relation between two relations S and T if the following conditions are satisfied. (a) Expression $\mathrm{PK}_2(A, B, R)$ in (3.1) indicates that attributes A and B form a primary key of R. (b) Predicate ThreeAttr checks whether a relation has at least three attributes, and it is defined as follows from the base predicate Attr:

$$\begin{aligned} \textsc{ThreeAttr}(R) \quad \leftarrow \quad & \textsc{Attr}(X, R), \textsc{Attr}(Y, R), \\ & \textsc{Attr}(Z, R), X \neq Y, X \neq Z, Y \neq Z. \end{aligned}$$

Thus, expression $\neg$ThreeAttr(R) in (3.1) indicates that R has at least two attributes. Notice that by combining this expression with $\mathrm{PK}_2(A, B, R)$, it is concluded that A, B are exactly the attributes of R. (c) Expressions $\mathrm{FK}_1(A, R, C, S)$ and $\mathrm{FK}_1(B, R, D, T)$ in (3.1) indicate that A is the attribute of a foreign key in R that points to S and B is the attribute of a foreign key in R that points to T, respectively. (d) Expressions $R \neq S$ and $R \neq T$ in (3.1) indicate that both S and T are different from relation R. (e) Predicate TwoFK checks whether an attribute of a relation is the attribute of two distinct foreign keys in that relation, and it is defined as follows from the base predicate FK_1:

$$\begin{aligned} \textsc{TwoFK}(X, Y) \quad &\leftarrow \quad \mathrm{FK}_1(X, Y, U_1, V_1), \mathrm{FK}_1(X, Y, U_2, V_2), U_1 \neq U_2 \\ \textsc{TwoFK}(X, Y) \quad &\leftarrow \quad \mathrm{FK}_1(X, Y, U_1, V_1), \mathrm{FK}_1(X, Y, U_2, V_2), V_1 \neq V_2 \end{aligned}$$

Thus, expressions $\neg$TwoFK(A, R) and $\neg$TwoFK(B, R) in (3.1) indicate that attribute A is not the attribute of two distinct foreign keys in R and B is not the attribute of two distinct foreign keys in R, respectively. (f) Predicate OneFK checks whether a pair of attributes of a relation are the attributes of a composite

foreign key in that relation:

$$\begin{aligned} \textsc{OneFK}(X,Y,Z) &\leftarrow \text{FK}_2(X,Y,Z,U,V,W) \\ \textsc{OneFK}(X,Y,Z) &\leftarrow \text{FK}_2(Y,X,Z,U,V,W) \end{aligned}$$

Thus, expression $\neg\textsc{OneFK}(A,B,R)$ in (3.1) indicates that attributes A, B of R are not the attributes of a composite foreign key in R. (g) Finally, predicate FKTO checks whether a relation with two attributes has incoming foreign keys:

$$\begin{aligned} \textsc{FKto}(X) &\leftarrow \text{FK}_1(U_1,Y,V,X) \\ \textsc{FKto}(X) &\leftarrow \text{FK}_2(U_1,U_2,Y,V_1,V_2,X) \end{aligned}$$

Thus, expression $\neg\textsc{FKto}(R)$ in (3.1) indicates that relation R does not have incoming foreign keys.

For instance, BINREL("ENROLLED", "SID", "CID", "STUDENT", "SID", "COURSE", "CID") holds in the example because "ENROLLED" is a relation with only two attributes, "SID" and "CID", and "SID" is a foreign key to "STUDENT", and "CID" is a foreign key to "COURSE". Note that there is no condition in the rule (3.1) that requires S and T to be different, allowing binary relations that have their domain equal to their range. Also note that, for simplicity, it is assumed in the rule (3.1) that a binary relation R consists of only two attributes A and B. However, this rule can be easily extended to deal with binary relations generated from many-to-many relationships between entities in an ER/UML diagram that have more than two attributes.

Identifying Classes

In this context, a class is any relation that is not a binary relation because a binary relation represents a relationship between different relations. That is, predicate

Class is defined by the following Datalog rules:

$$\begin{aligned} \textsc{Class}(X) &\leftarrow \textsc{Rel}(X), \neg\textsc{IsBinRel}(X) \\ \textsc{IsBinRel}(X) &\leftarrow \textsc{BinRel}(X, A, B, S, C, T, D) \end{aligned}$$

In the example, Class("DEPT"), Class("STUDENT") and Class("COURSE") hold.

Identifying Object Properties

For every $n \geq 1$, the following rule is used for identifying object properties that are generated from foreign keys[7]:

$$\begin{aligned} \mathrm{OP}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T) \leftarrow \\ \mathrm{FK}_n(X_1, \ldots, X_n, S, Y_1, \ldots, Y_n, T), \neg\textsc{IsBinRel}(S) \end{aligned}$$

This rule states that a foreign key represents an object property from the entity containing the foreign key (domain) to the referenced entity (range). It should be noticed that this rule excludes the case of binary relations, as there is a special rule for this type of relations (see rule (3.1)). In the example, OP_2("CODE", "DID", "COURSE", "DEPT") holds as CODE is a foreign key in the table COURSE that references attribute DID in the table DEPT.

Identifying Data type Properties

Every attribute in a non-binary relation is mapped to a data type property:

$$\mathrm{DTP}(A, R) \leftarrow \textsc{Attr}(A, R), \neg\textsc{IsBinRel}(R)$$

[7]Notice that although an infinite number of rules is considered in the definition of $\mathcal{DM}$, for every concrete relational database only a finite number of these rules are needed.

For instance, DTP("NAME", "STUDENT") holds in the example, while DTP("SID", "ENROLLED") does not hold as ENROLLED is a binary relation.

3.4.3 Translating a relational schema into an OWL ontology

The following are the rules that translate a relational database schema into an OWL ontology.

Generating IRIs for Classes, Object Properties and Data type Properties

The following family of rules produce IRIs for classes, binary relations, object properties and data type properties identified by the mapping (which are stored in the predicates Class, BinRel, OP_n and DTP, respectively). The following assumptions are made. A base IRI base is given for the relational database to be translated (for example, "http://example.edu/db/"). Additionally a family of built-in predicates Concat_n ($n \geq 2$) such that Concat_n has $n+1$ arguments and $\text{Concat}_n(x_1, \ldots, x_n, y)$ holds if y is the concatenation of the strings x_1, ..., x_n. $\mathcal{DM}$ uses the following Datalog rules to produce IRIs for classes and data type properties:

$$\text{ClassIRI}(R, X) \leftarrow \text{Class}(R), \text{Concat}_2(\texttt{base}, R, X)$$
$$\text{DTP\_IRI}(A, R, X) \leftarrow \text{DTP}(A, R), \text{Concat}_4(\texttt{base}, R, \texttt{"\#"}, A, X)$$

For instance, http://example.edu/db/STUDENT is the IRI for the STUDENT relation in the example, and http://example.edu/db/STUDENT#NAME is the IRI for attribute NAME in the STUDENT relation (recall that DTP("NAME", "STUDENT") holds in the example). Moreover, $\mathcal{DM}$ uses the following family of Datalog rules to generate IRIs for object properties. First, for object properties generated from binary relations,

the following rule is used:

$$\text{OP\_IRI}_1(R, A, B, S, C, T, D, X) \leftarrow \text{BinRel}(R, A, B, S, C, T, D),$$
$$\text{Concat}_{10}(\texttt{base}, R, \texttt{"\#"}, A, \texttt{","}, B, \texttt{","}, C, \texttt{","}, D, X)$$

Thus, `http://example.edu/db/ENROLLED#SID,CID,SID,CID` is the IRI for binary relation `ENROLLED` in the example. Second, for object properties generated from a foreign key consisting of n attributes ($n \geq 1$), the following rule is used:

$$\begin{aligned}
&\text{OP_IRI}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T, X) \leftarrow \\
&\quad \text{OP}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T), \\
&\quad \text{Concat}_{4n+4}(\texttt{base}, S, \texttt{","}, T, \texttt{"\#"}, X_1, \texttt{","}, \ldots, X_{n-1}, \texttt{","}, \\
&\qquad X_n, \texttt{","}, Y_1, \texttt{","}, \ldots, Y_{n-1}, \texttt{","}, Y_n, X)
\end{aligned}$$

Thus, given that OP_2(`"CODE"`, `"DID"`, `"COURSE"`, `"DEPT"`) holds in the example, IRI `http://example.edu/db/COURSE,DEPT#CODE,DID` is generated to represent the fact that `CODE` is a foreign key in the table `COURSE` that references attribute `DID` in the table `DEPT`.

Translating Relational Schemas

The following Datalog rules are used to generate the RDF ontological triples. First, a rule is used to collect all the classes:

$$\text{Triple}(U, \texttt{"rdf:type"}, \texttt{"owl:Class"}) \leftarrow \text{Class}(R), \text{ClassIRI}(R, U)$$

The predicate Triple is used to collect all the triples of the RDF graph generated by the direct mapping $\mathcal{DM}$. Second, the following family of rules is used to collect

all the object properties ($n \geq 1$):

$$\textsc{Triple}(U, \texttt{"rdf:type"}, \texttt{"owl:ObjectProperty"}) \leftarrow \\ \mathrm{OP}_n(X_1, \ldots, X_n, S, T), \mathrm{OP\_IRI}_n(X_1, \ldots, X_n, S, T, U)$$

Third, the following rule is used to collect the domains of the object properties ($n \geq 1$):

$$\textsc{Triple}(U, \texttt{"rdfs:domain"}, W) \leftarrow \mathrm{OP}_n(X_1, \ldots, X_n, S, T), \\ \mathrm{OP\_IRI}_n(X_1, \ldots, X_n, S, T, U), \textsc{ClassIRI}(S, W)$$

Fourth, the following rule is used to collect the ranges of the object properties ($n \geq 1$):

$$\textsc{Triple}(U, \texttt{"rdfs:range"}, W) \leftarrow \mathrm{OP}_n(X_1, \ldots, X_n, S, T), \\ \mathrm{OP\_IRI}_n(X_1, \ldots, X_n, S, T, U), \textsc{ClassIRI}(T, W)$$

Fifth, the following rule is used to collect all the data type properties:

$$\textsc{Triple}(U, \texttt{"rdf:type"}, \texttt{"owl:DatatypeProperty"}) \leftarrow \\ \mathrm{DTP}(A, R), \mathrm{DTP\_IRI}(A, R, U)$$

Finally, the following rule is used to collect the domains of the data type properties:

$$\textsc{Triple}(U, \texttt{"rdfs:domain"}, W) \leftarrow \\ \mathrm{DTP}(A, R), \mathrm{DTP\_IRI}(A, R, U), \textsc{ClassIRI}(R, W)$$

3.4.4 Translating a database instance into RDF

The Datalog rules that map a relational database instance into RDF assertional triples consist of two parts. First, a set of rules for generating IRIs. Second, a set of rules that generate RDF.

Generating IRIs for tuples

A family of predicates are introduced that produce IRIs for the tuples being translated, where it is assumes that an IRI `base` is given for the relational database (for example, `"http://example.edu/db/"`). First, $\mathcal{DM}$ uses the following set of Datalog rules to produce IRIs for the tuples of the relations having a primary key:

$$\begin{aligned}\textsc{RowIRI}_n(V_1, V_2, \ldots, V_n, A_1, A_2, \ldots, A_n, T, R, X) \leftarrow\\ \mathrm{PK}_n(A_1, A_2, \ldots, A_n, R), \textsc{Value}(V_1, A_1, T, R),\\ \textsc{Value}(V_2, A_2, T, R), \ldots, \textsc{Value}(V_n, A_n, T, R),\\ \textsc{Concat}_{4n+2}(\texttt{base}, R, \texttt{"\#"}, A_1, \texttt{"="}, V_1, \texttt{","},\\ A_2, \texttt{"="}, V_2, \texttt{","}, \ldots, \texttt{","}, A_n, \texttt{"="}, V_n, X)\end{aligned}$$

Thus, given that the facts $\mathrm{PK}_1(\texttt{"SID"}, \texttt{"STUDENT"})$ and $\textsc{Value}(\texttt{"1"}, \texttt{"SID"}, \texttt{"id1"}, \texttt{"STUDENT"})$ hold in the example, the IRI `http://example.edu/db/STUDENT#SID=1` is the identifier for the tuple in table `STUDENT` with value `1` in the primary key. Moreover, $\mathcal{DM}$ uses the following rule to generate blank nodes for the tuples of the relations not having a primary key:

$$\textsc{BlankNode}(T, R, X) \leftarrow \textsc{Value}(V, A, T, R), \textsc{Concat}_3(\texttt{"\_:"}, R, T, X)$$

Translating Relational Instances

The direct mapping $\mathcal{DM}$ generates three types of triples when translating a relational instance: table triples, reference triples and literal triples. The following are Datalog rules for each one of these cases.

For table triples, $\mathcal{DM}$ produces for each tuple t in a relation R, a triple indicating that t is of type r. To construct these tuples, $\mathcal{DM}$ uses the following auxiliary rules:

$$\begin{aligned}
\textsc{TupleID}(T, R, X) \leftarrow & \textsc{Class}(R), \mathrm{PK}_n(A_1, \ldots, A_n, R), \\
& \textsc{Value}(V_1, A_1, T, R), \ldots, \textsc{Value}(V_n, A_n, T, R), \\
& \textsc{RowIRI}_n(V_1, \ldots, V_n, A_1, \ldots, A_n, T, R, X) \\
\textsc{TupleID}(T, R, X) \leftarrow & \textsc{Class}(R), \neg\textsc{HasPK}_n(R), \\
& \textsc{Value}(V, A, T, R), \textsc{BlankNode}(T, R, X)
\end{aligned}$$

That is, $\textsc{TupleID}(T, R, X)$ generates the identifier X of a tuple T of a relation R, which is an IRI if R has a primary key or a blank node otherwise. Notice that in the preceding rules, predicate $\textsc{HasPK}_n$ is used to check whether a table R with n attributes has a primary key (thus, $\neg\textsc{HasPK}_n(R)$ indicates that R does not have a primary key). Predicate $\textsc{HasPK}_n$ is defined by the following n rules:

$$\textsc{HasPK}_n(X) \leftarrow \mathrm{PK}_i(A_1, \ldots, A_i, X) \qquad i \in \{1, \ldots, n\}$$

The following rule generates the table triples:

$$\begin{aligned}
\textsc{Triple}(U, \texttt{"rdf:type"}, W) \leftarrow & \\
& \textsc{Value}(V, A, T, R), \textsc{TupleID}(T, R, U), \textsc{ClassIRI}(R, W)
\end{aligned}$$

For example, the following is a table triple In the example:

$$\begin{aligned}&\textsc{Triple}(\texttt{"http://example.edu/db/STUDENT\#SID=1"},\\&\qquad\texttt{"rdf:type"},\\&\qquad\texttt{"http://example.edu/db/STUDENT"})\end{aligned}$$

For reference triples, $\mathcal{DM}$ generates triples that store the references generated by binary relations and foreign keys. More precisely, the following Datalog rule is used to construct reference triples for object properties that are generated from binary relations:

$$\begin{aligned}\textsc{Triple}(U,V,W) \leftarrow\ & \textsc{BinRel}(R,A,B,S,C,T,D),\\ & \textsc{Value}(V_1,A,T_1,R), \textsc{Value}(V_1,C,T_2,S),\\ & \textsc{Value}(V_2,B,T_1,R), \textsc{Value}(V_2,D,T_3,T),\\ & \textsc{TupleID}(T_2,S,U),\\ & \text{OP\_IRI}_1(R,A,B,S,C,T,D,V),\\ & \textsc{TupleID}(T_3,T,W)\end{aligned}$$

Moreover, the following Datalog rule is used to construct reference triples for object

properties that are generated from foreign keys ($n \geq 1$):

$$\begin{aligned}\textsc{Triple}(U, V, W) \leftarrow\ & \\ & \mathrm{OP}_{2n}(A_1, \ldots, A_n, B_1, \ldots, B_n, S, T), \\ & \textsc{Value}(V_1, A_1, T_1, S), \ldots, \textsc{Value}(V_n, A_n, T_1, S), \\ & \textsc{Value}(V_1, B_1, T_2, T), \ldots, \textsc{Value}(V_n, B_n, T_2, T), \\ & \textsc{TupleID}(T_1, S, U), \textsc{TupleID}(T_2, T, W), \\ & \mathrm{OP\_IRI}_{2n}(A_1, \ldots, A_n, B_1, \ldots, B_n, S, T, V)\end{aligned}$$

Finally, $\mathcal{DM}$ produces for every tuple t in a relation R and for every attribute A of R, a triple storing the value of t in A, which is called a literal triple. The following Datalog rule is used to generate such triples:

$$\begin{aligned}\textsc{Triple}(U, V, W) \leftarrow\ & \mathrm{DTP}(A, R), \textsc{Value}(W, A, T, R), \\ & W \neq \texttt{NULL}, \textsc{TupleID}(T, R, U), \mathrm{DTP\_IRI}(A, R, V)\end{aligned}$$

Notice that in the above rule, the condition $W \neq \texttt{NULL}$ is used to check that the value of the attribute A in a tuple T in a relation R is not null. Thus, literal triples are generated only for non-null values. The following is an example of a literal triple:

$$\begin{aligned}\textsc{Triple}(&\texttt{"http://example.edu/db/STUDENT\#SID=1"}, \\ &\texttt{"http://example.edu/db/STUDENT\#NAME"}, \texttt{"John"})\end{aligned}$$

3.5 Properties of $\mathcal{DM}$

The direct mapping $\mathcal{DM}$ is now studied with respect to the two fundamental properties (information preservation and query preservation) and the two desirable properties (monotonicity and semantics preservation) as defined in Section 3.1.

3.5.1 Information preservation of $\mathcal{DM}$

First, $\mathcal{DM}$ does not lose any information in the relational instance being translated:

Theorem 1 *The direct mapping $\mathcal{DM}$ is information preserving.*

The proof of this theorem is straightforward. It involves providing a computable mapping $\mathcal{N} : \mathcal{G} \rightarrow \mathcal{I}$ that satisfies the condition in Definition 15. That is, a computable mapping $\mathcal{N}$ that can reconstruct the initial relational instance from the generated RDF graph. The reader is referred to the Appendix for the full proof.

3.5.2 Query preservation of $\mathcal{DM}$

The way $\mathcal{DM}$ maps relational data into RDF allows to translate a relational algebra query over a relational instance it into an equivalent SPARQL query over the generated RDF graph.

Theorem 2 *The direct mapping $\mathcal{DM}$ is query preserving.*

Angles and Gutierrez proved that SPARQL has the same expressive power as relational algebra [8]. Thus, one may be tempted to think that this result could be used to prove Theorem 2. However, the version of relational algebra considered in [8] does not include the `NULL` value, and hence cannot be used to prove this result. An outline of the proof of this theorem is presented. The reader is referred to the Appendix for the details.

Assume that a relational schema $\mathbf{R}$ and a set Σ of PKs and FKs over $\mathbf{R}$ is given. The proof follows by showing that for every relational algebra query Q over $\mathbf{R}$, there exists a SPARQL query $Q^{\star}$ such that for every instance I of $\mathbf{R}$ (possibly including null values) satisfying Σ:

$$tr([\![Q]\!]_I) \quad = \quad [\![Q^{\star}]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \tag{3.2}$$

Interestingly, the proof that the previous condition holds is by induction on the structure of Q, and thus it provides a constructive bottom-up algorithm for translating a relational algebra query Q into an equivalent SPARQL query $Q^\star$, that is, a query $Q^\star$ satisfying condition (3.2). Consider the database used as the running example and the relational algebra query $\sigma_{\texttt{Name=Juan}}(\texttt{STUDENT}) \bowtie \texttt{ENROLLED}$, which is used as a running example and translate it step by step to SPARQL, showing how the translation algorithm works.

For the sake of readability, the function ν is introduced which retrieves the IRI for a given relation R, denoted by $\nu(R)$, and the IRI for a given attribute A in a relation R, denoted by $\nu(A, R)$. The inductive proof starts by considering the two base relational algebra queries: the identity query R, where R is a relation name in the relational schema $\mathbf{R}$, and the query $\texttt{NULL}_A$. These two base queries give rise to the following three base cases for the inductive proof.

Non-binary relations: Assume that Q is the identity relational algebra query R, where $R \in \mathbf{R}$ is a non-binary relation (that is, IsBinRel(R) does not hold). Moreover, assume that $att(R) = \{A_1, \ldots, A_\ell\}$, with the corresponding IRIs $\nu(R) = r, \nu(A_1, R) = a_1, \ldots, \nu(A_\ell, R) = a_\ell$. Then a SPARQL query $Q^\star$ satisfying (3.2) is constructed as follows:

$$\begin{aligned}\text{SELECT } \{?A_1, \ldots, ?A_\ell\} \Big[\cdots \Big(\Big(\Big((?X, \texttt{"rdf:type"}, r) \\ \text{OPT } (?X, a_1, ?A_1) \Big) \text{ OPT } (?X, a_2, ?A_2) \Big) \\ \text{OPT } (?X, a_3, ?A_3) \Big) \cdots \text{ OPT } (?X, a_\ell, ?A_\ell) \Big].\end{aligned}$$

Notice that in order to not lose information, the operator OPT is used (instead of AND) because the direct mapping $\mathcal{DM}$ does not translate NULL values. In the example, the relation name STUDENT is a non-binary relation. Therefore the

following equivalent SPARQL query is generated with input `STUDENT`:

$$\text{SELECT } \{\texttt{?SID}, \texttt{?NAME}\} \left[\left((?X, \texttt{"rdf:type"}, \texttt{:STUDENT}) \right.\right.$$
$$\text{OPT } (?X, \texttt{:STUDENT\#SID}, \texttt{?SID})\Big)$$
$$\left.\text{OPT } (?X, \texttt{:STUDENT\#NAME}, \texttt{?NAME})\right]$$

It should be noticed that in the previous query, the symbol : has to be replaced by the base IRI used when generating IRIs for relations and attributes in a relation (see Section 3.4.3)[8].

Binary relations: Assume that Q is the identity relational algebra query R, where $R \in \mathbf{R}$ is a binary relation (that is, IsBinRel(R) holds). Moreover, assume that $att(R) = \{A_1, A_2\}$, where A_1 is a foreign key referencing the attribute B of a relation S, and A_2 is a foreign key referencing the attribute C of a relation T. Finally, assume that $\nu(R) = r$, $\nu(B, S) = b$ and $\nu(C, T) = c$, Then a SPARQL query $Q^\star$ satisfying (3.2) is defined as follows:

$$\text{SELECT } \{?A_1, ?A_2\} \ ((?T_1, r, ?T_2) \quad \text{AND } (?T_1, b, ?A_1) \ \text{AND } (?T_2, c, ?A_2)).$$

Given that a binary relation is mapped to an object property, the values of a binary relation can be retrieved by querying the datatype properties of the referenced attributes. In the example, the relational name `ENROLLED` is a binary relation. There-

[8] In SPARQL terminology, we have included the following prefix in the query: `@prefix : <http://example.edu/db/>`, if the base IRI is `<http://example.edu/db/>`.

fore the following equivalent SPARQL query is generated with input `ENROLLED`:

$$
\begin{aligned}
&\text{SELECT } \{\texttt{?SID}, \texttt{?CID}\}(\\
&\qquad (?T_1, \texttt{:ENROLLED\#SID,CID,SID,CID}, ?T_2) \ \text{ AND} \\
&\qquad (?T_1, \texttt{:STUDENT\#SID}, \texttt{?SID}) \ \text{ AND} \\
&\qquad (?T_2, \texttt{:COURSE\#CID}, \texttt{?CID})).
\end{aligned}
$$

Empty relation: Assume that $Q = \texttt{NULL}_A$, and define $Q^\star$ as the empty graph pattern { }. Then the condition (3.2) holds because of the definition of the function *tr*, which does not translate `NULL` values to mappings.

The inductive step in the proof of Theorem 2 is the following. Assume that the theorem holds for relational algebra queries Q_1 and Q_2. That is, there exists SPARQL queries $Q_1^\star$ and $Q_2^\star$ such that:

$$tr([\![Q_1]\!]_I) = [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}, \tag{3.3}$$

$$tr([\![Q_2]\!]_I) = [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \tag{3.4}$$

The proof continues by presenting equivalent SPARQL queries for the following relational algebra operators: selection (σ), projection (π), rename (δ), join ($\bowtie$), union ($\cup$) and difference ($\setminus$). It is important to notice that the operators left-outer join, right-outer join and full-outer join are all expressible with the previous operators, hence cases for these operators are not presented.

Selection: In order to define query $Q^\star$ satisfying condition (3.2), four cases are considered. In all these cases, the established equivalence in (3.3) is used.

1. If Q is $\sigma_{A_1=a}(Q_1)$, then

$$Q^\star = (Q_1^\star \text{ FILTER } (?A_1 = a)).$$

2. If Q is $\sigma_{A_1 \neq a}(Q_1)$, then

$$Q^\star \quad = \quad (Q_1^\star \text{ FILTER } (\neg(?A_1 = a) \wedge \text{bound}(?A_1))).$$

3. If Q is $\sigma_{\texttt{IsNull}(A_1)}(Q_1)$, then

$$Q^\star \quad = \quad (Q_1^\star \text{ FILTER } (\neg\, \text{bound}(?A_1))).$$

4. If Q is $\sigma_{\texttt{IsNotNull}(A_1)}(Q_1)$, then

$$Q^\star \quad = \quad (Q_1^\star \text{ FILTER } (\text{bound}(?A_1))).$$

These equivalences are straightforward. However, it is important to note the use of bound$(\cdot)$ in the second case; as the semantics of relational algebra states that if Q is the query $\sigma_{A_1 \neq a}(Q_1)$, then $[\![Q]\!]_I = \{t \in [\![Q_1]\!]_I \mid t.A_1 \neq \texttt{NULL}$ and $t.A_1 \neq a\}$, the variable $?A_1$ has to be bound because the values in the attribute A_1 in the answer to $\sigma_{A_1 \neq a}(Q_1)$ are different from NULL. Following the example, the following SPARQL query is generated with input $\sigma_{\texttt{Name=Juan}}(\texttt{STUDENT})$:

$$\Big(\text{SELECT } \{\texttt{?SID}, \texttt{?NAME}\} \Big[\Big((?X, \texttt{"rdf:type"}, \texttt{:STUDENT}) \text{ OPT } (?X, \texttt{:STUDENT\#SID}, \texttt{?SID})\Big) \text{ OPT } (?X, \texttt{:STUDENT\#NAME}, \texttt{?NAME})\Big]\Big) \text{ FILTER } (\texttt{?NAME} = \texttt{Juan})$$

Projection: Assume that $Q = \pi_{\{A_1,\ldots,A_\ell\}}(Q_1)$. Then query $Q^\star$ satisfying condition (3.2) is defined as (SELECT $\{?A_1, \ldots, ?A_\ell\}$ $Q_1^\star$). It is important to notice that the

nested SELECT queries is used to deal with projection, as well as in two of the base cases, which is a functionality specific to SPARQL 1.1 [73].

Rename: Assume that $Q = \delta_{A_1 \to B_1}(Q_1)$ and $att(Q) = \{A_1, \ldots, A_\ell\}$. Then query $Q^\star$ satisfying condition (3.2) is defined as (SELECT $\{?A_1$ AS $?B_1, ?A_2, \ldots, ?A_\ell\}$ $Q_1^\star$). Notice that this equivalence holds because the rename operator in relational algebra renames one attribute to another and projects all attributes of Q.

Join: Assume that $Q = (Q_1 \bowtie Q_2)$, where $(att(Q_1) \cap att(Q_2)) = \{A_1, \ldots, A_\ell\}$. Then query $Q^\star$ satisfying condition (3.2) is defined as follows:

$$\Big[\Big(Q_1^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell))\Big) \text{ AND } \Big(Q_2^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell))\Big)\Big].$$

Note the use of bound$(\cdot)$ which is necessary in the SPARQL query in order to guarantee that the variables that are being joined on are not null. Following the example, Figure 3.1 shows the SPARQL query generated with input $\sigma_{\texttt{Name=Juan}}(\texttt{STUDENT}) \bowtie \texttt{ENROLLED}$.

Union: Assume that $Q = (Q_1 \cup Q_2)$. Then query $Q^\star$ satisfying condition (3.2) is simply defined as ($Q_1^\star$ UNION $Q_2^\star$). Notice that in this case, the already established equivalences (3.3) and (3.4) are used.

Difference: Assume that $Q = (Q_1 \setminus Q_2)$. In this case, it is also possible to define a SPARQL query $Q^\star$ satisfying condition (3.2). The reader is referred to the appendix for the complete description of $Q^\star$.

3.5.3 Monotonicity and semantics preservation of $\mathcal{DM}$

Consider the two desirable properties identified in Section 3.3. First, it is straightforward to see that $\mathcal{DM}$ is monotone, because all the negative atoms in the Datalog rule defining $\mathcal{DM}$ refer to the schema, the PKs and the FKs of the database, and

$$\Big[\Big(\Big(\text{SELECT } \{\texttt{?SID}, \texttt{?NAME}\}\Big[$$
$$\Big((?X, \texttt{"rdf:type"}, \texttt{:STUDENT}) \text{ OPT } (?X, \texttt{:STUDENT\#SID}, \texttt{?SID})\Big) \text{ OPT}$$
$$(?X, \texttt{:STUDENT\#NAME}, \texttt{?NAME})\Big]\Big)$$
$$\text{FILTER } (\texttt{?NAME} = \texttt{Juan})\Big) \text{ FILTER } (\text{bound}(\texttt{?SID}))\Big]$$
$$\text{AND}$$
$$\Big[\Big(\text{SELECT } \{\texttt{?SID}, \texttt{?CID}\}$$
$$\Big((?T_1, \texttt{:ENROLLED\#SID,CID,SID,CID}, ?T_2) \text{ AND } (?T_1, \texttt{:STUDENT\#SID}, \texttt{?SID}) \text{ AND}$$
$$(?T_2, \texttt{:COURSE\#CID}, \texttt{?CID})\Big)\Big)$$
$$\text{FILTER } (\text{bound}(\texttt{?SID}))\Big]$$

Figure 3.1: SPARQL translation of the relational algebra query $\sigma_{\texttt{Name}=\texttt{Juan}}(\texttt{STUDENT}) \bowtie \texttt{ENROLLED}$.

these elements are kept fixed when checking monotonicity.

The situation is completely different for the case of semantics preservation. Each type of semantics preservation, positive and negative, needs to be analyzed independently. It is straightforward to see that $\mathcal{DM}$ is positive semantics preserving because the ontology derived from the relational database is not able to express contradictions, hence is always a consistent RDF graph.

Theorem 3 *The direct mapping $\mathcal{DM}$ is positive semantics preserving.*

The proof consists of going through each rule of $\mathcal{DM}$ and noting that the rules do not generate an inconsistency. Hence, the result of applying $\mathcal{DM}$ to a database will always be a consistent RDF graph.

The following example shows that the direct mapping $\mathcal{DM}$ is not negative semantics preserving.

Example 6 Assume that a relational schema contains a relation with name `STUDENT` and attributes `SID`, `NAME`, and assume that the attribute `SID` is the primary key. Moreover, assume that this relation has two tuples, t_1 and t_2 such that t_1.`SID` $= 1$, t_1.`NAME` $=$ John and t_2.`SID` $= 1$, t_2.`NAME` $=$ Peter. It is clear that the primary key is violated, therefore the database is inconsistent. The result of applying $\mathcal{DM}$ is the following RDF graph (the base URI is omitted):

Triple("STUDENT", "rdf:type", "owl:Class")

Triple("STUDENT#SID", "rdf:type", "owl:DatatypeProperty")

Triple("STUDENT#NAME", "rdf:type", "owl:DatatypeProperty")

Triple("STUDENT#SID=1", "rdf:type", "STUDENT")

Triple("STUDENT#SID=1", "STUDENT#NAME", "John")

Triple("STUDENT#SID=1", "STUDENT#NAME", "Peter")

Even though the relational database violates the foreign key constraint, there is nothing the makes the resulting RDF graph inconsistent.

In fact, the result in Example 6 can be generalized as it is possible to show that the direct mapping $\mathcal{DM}$ always generates a consistent RDF graph, hence, it cannot be full semantics preserving.

Proposition 1 *The direct mapping $\mathcal{DM}$ is not full semantics preserving.*

Does this mean that the proposed direct mapping $\mathcal{DM}$ is incorrect? What could be done to create a direct mapping that is full semantics preserving? These problems are studied in the following section.

3.6 Semantics Preservation of Direct Mappings

Generating a full semantics preserving direct mapping poses a specific challenge due to the different assumptions made by Relational Databases and the Semantic Web. Extending the direct mapping $\mathcal{DM}$ to deal with primary keys is simple (Section 3.6.1). Dealing with foreign keys is more difficult. Section 3.6.2 presents the result that any direct mapping satisfying the condition of being monotone cannot be full semantics preserving. Finally, two possible ways of overcoming this limitation are presented.

3.6.1 A full semantics preserving direct mapping for primary keys

Recall that a primary key can be violated if there are repeated values or null values. At a first glance, one would assume that `owl:hasKey` and the Unique Name Assumption (UNA) could be used to create a full semantics preserving direct mapping for primary keys. OWL's `owl:hasKey` states that each instance of a class is uniquely identified by a property (or a set of properties) and, if two instances of the

class coincide on values for each of key properties, then these two instances are the same. The Unique Name Assumption means that different names always refer to different entities in the world. OWL does not make the Unique Name Assumption but does provide the `owl:differentFrom` constructs to express whether two names are distinct.

Consider now a database without null values, then a violation of the primary key would generate an inconsistency with `owl:hasKey` and `owl:differentFrom`. However, if null values are considered, the situation is different. Consider a database where the primary key has null values. Then the properties of `owl:hasKey`, which uniquely identify an instance, does not have any values. Therefore it is not possible to trigger an inconsistency. A different approach must be considered.

Consider a new direct mapping $\mathcal{DM}_{pk}$ that extends $\mathcal{DM}$ with the idea of representing the violation of an integrity constraint as a new set of Datalog rules. A Datalog rule is used to determine if the value of a primary key attribute is repeated, and a family of Datalog rules are used to determine if there is a value `NULL` in a column corresponding to a primary key. If some of these violations are found, then an artificial triple is generated that would produce an inconsistency. For example, the following rules are used to map a primary key with two attributes:

$$\textsc{Triple}(a, \texttt{"owl:differentFrom"}, a) \leftarrow \mathrm{PK}_2(X_1, X_2, R),$$
$$\textsc{Value}(V_1, X_1, T_1, R), \textsc{Value}(V_1, X_1, T_2, R),$$
$$\textsc{Value}(V_2, X_2, T_1, R), \textsc{Value}(V_2, X_2, T_2, R), T_1 \neq T_2$$
$$\textsc{Triple}(a, \texttt{"owl:differentFrom"}, a) \leftarrow \mathrm{PK}_2(X_1, X_2, R),$$
$$\textsc{Value}(V, X_1, T, R), V = \texttt{NULL}$$
$$\textsc{Triple}(a, \texttt{"owl:differentFrom"}, a) \leftarrow \mathrm{PK}_2(X_1, X_2, R),$$
$$\textsc{Value}(V, X_2, T, R), V = \texttt{NULL}$$

In the previous rules, a is any valid IRI. If $\mathcal{DM}_{pk}$ is applied to the database of Example 6, it is straightforward to see that starting from an inconsistent relational database, an RDF graph is obtained that is also inconsistent. In fact, the following holds:

Proposition 2 *The direct mapping $\mathcal{DM}_{pk}$ is information preserving, query preserving, monotone, and full semantics preserving if only PKs are considered. That is, for every relational schema $\mathbf{R}$, set Σ of (only) PKs over $\mathbf{R}$ and instance I of $\mathbf{R}$: $I \models \Sigma$ iff $\mathcal{DM}_{pk}(\mathbf{R}, \Sigma, I)$ is consistent under OWL semantics.*

Information preservation, query preservation and monotonicity of $\mathcal{DM}_{pk}$ are corollaries of the fact that these properties hold for $\mathcal{DM}$, and of the fact that the Datalog rules introduced to handle primary keys are monotone.

A natural question at this point is whether $\mathcal{DM}_{pk}$ can also deal with foreign keys. Unfortunately, an easily constructed counter-example shows that this is not the case.

Example 7 Consider the following inconsistent relational database. A relational schema that contains a relation with name `COURSE` and attributes `CID`, `TITLE`, `CODE`

with attribute CID is the primary key. A relation with name DEPT and attributes DID, NAME with attribute DID as the primary key. Additionally, there is a foreign key from the attribute CODE of COURSE to the attribute DID of DEPT. Moreover, assume that the relation COURSE has a tuples t such that $t.\texttt{ID} = 1$, $t.\texttt{TITLE} = \text{CS101}$ and $t.\texttt{CODE} = 2$. The result of applying $\mathcal{DM}$ or $\mathcal{DM}_{pk}$ is the following RDF graph (the base URI is omitted):

Triple("COURSE", "rdf:type", "owl:Class")

Triple("COURSE#CID", "rdf:type", "owl:DatatypeProperty")

Triple("COURSE#TITLE", "rdf:type", "owl:DatatypeProperty")

Triple("COURSE#CODE", "rdf:type", "owl:DatatypeProperty")

Triple("DEPT", "rdf:type", "owl:Class")

Triple("DEPT#DID", "rdf:type", "owl:DatatypeProperty")

Triple("DEPT#NAME", "rdf:type", "owl:DatatypeProperty")

Triple("COURSE,DEPT#CODE,DID", "rdf:type", "owl:ObjectProperty")

Triple("COURSE,DEPT#CODE,DID", "rdfs:domain", "COURSE")

Triple("COURSE,DEPT#CODE,DID", "rdfs:range", "DEPT")

Triple("COURSE#CID=1", "rdf:type", "COURSE")

Triple("COURSE#CID=1", "COURSE#CID", "1")

Triple("COURSE#CID=1", "COURSE#TITLE", "CS101")

Triple("COURSE#CID=1", "COURSE#CODE", "2")

It is not difficult to see that the resulting RDF graph is consistent.

Both $\mathcal{DM}$ and $\mathcal{DM}_{pk}$ generate a consistent RDF graph, which is not what

is desired. Does this mean that it is not possible to have a direct mapping that is full semantics preserving and considers foreign keys?

3.6.2 Semantics preserving direct mappings for primary keys and foreign keys

The following theorem shows that the desirable condition of being monotone is an obstacle to obtain a full semantics preserving direct mapping.

Theorem 4 *No monotone direct mapping is full semantics preserving.*

The proof of this theorem is by contradiction by assuming first that a direct mapping $\mathcal{M}$ is monotone and full semantics preserving. The reader is referred to the Appendix for the full proof.

It is important to understand the reasons why a full semantics preserving direct mapping has not been able to be defined up to now. The issue is with two characteristics of OWL:

1. it adopts the Open World Assumption (OWA), where a statement cannot be inferred to be false on the basis of failing to prove it, and

2. it does not adopt the Unique Name Assumption (UNA), allowing two different names can identify the same thing.

On the other hand, a relational database adopts the Closed World Assumption (CWA), where a statement is inferred to be false if it is not known to be true, which is the opposite of OWA. In other words, what causes an inconsistency in a relational database, can cause an inference of new knowledge in OWL.

In order to preserve the semantics of the relational database, it needs to be ensured that whatever causes an inconsistency in a relational database, will also cause an inconsistency in OWL. Following this idea, consider now the following

non-monotone direct mapping, $\mathcal{DM}_{pk+fk}$. This direct mapping extends $\mathcal{DM}_{pk}$ by introducing rules for verifying beforehand if there is a violation of a foreign key constraint. If such a violation exists, then an artificial RDF triple is created which will generate an inconsistency with respect to the OWL semantics. More precisely, the following family of Datalog rules are used in $\mathcal{DM}_{pk+fk}$ to detect an inconsistency in a relational database:

$$\begin{aligned}\textsc{Violation}(S) \leftarrow\ & \\ & \mathrm{FK}_n(X_1, \ldots, X_n, S, Y_1, \ldots, Y_n, T), \\ & \textsc{Value}_n(V_1, X_1, T, S), \ldots, \textsc{Value}(V_n, X_n, T, S), \\ & V_1 \neq \texttt{NULL}, \ldots, V_n \neq \texttt{NULL}, \\ & \neg\textsc{IsValue}_n(V_1, \ldots, V_n, Y_1, \ldots, Y_n, T)\end{aligned}$$

In the preceding rule, the predicate $\textsc{IsValue}_n$ is used to check whether a tuple in a relation has values for some given attributes. The predicate $\textsc{IsValue}_n$ is defined by the following rule:

$$\begin{aligned}\textsc{IsValue}_n(V_1, \ldots, V_n, B_1, \ldots, B_n, S)\ \leftarrow\ & \\ & \textsc{Value}(V_1, B_1, T, S), \ldots, \textsc{Value}(V_n, B_n, T, S)\end{aligned}$$

Finally, the following Datalog rule is used to obtain an inconsistency in the generated RDF graph:

$$\textsc{Triple}(a, \texttt{"owl:differentFrom"}, a) \quad \leftarrow \quad \textsc{Violation}(S)$$

In the previous rule, a is any valid IRI. It should be noticed that $\mathcal{DM}_{pk+fk}$ is non-monotone because if new data is added to the database, which now satisfies the FK constraint, then the artificial RDF triple needs to be retracted.

Theorem 5 *The direct mapping $\mathcal{DM}_{pk+fk}$ is information preserving, query preserving and full semantics preserving.*

Information preservation and query preservation of $\mathcal{DM}_{pk+fk}$ are corollaries of the fact that these properties hold for $\mathcal{DM}$ and $\mathcal{DM}_{pk}$.

A direct mapping that satisfies the four properties can be obtained by considering an alternative semantics of OWL that expresses integrity constraints. Because OWL is based on Description Logic, a version of DL that supports integrity constraints is needed. This is not a new idea. Integrity constraints are epistemic in nature and are about "what the knowledge base knows" [112]. Extending DL with the epistemic operator **K** has been studied [37, 52, 53]. Grimm et al. proposed to extend the semantics of OWL to support the epistemic operator [67]. Motik et al. proposed to write integrity constraints as standard OWL axioms but interpreted with different semantics for data validation purposes [100]. Tao et al. showed that integrity constraint validation can be reduced to SPARQL query answering [127]. Mehdi et al. introduced a way to answer epistemic queries to restricted OWL ontologies [96]. Thus, it is possible to extend $\mathcal{DM}_{pk}$ to create an information preserving, query preserving and monotone direct mapping that is also semantics preserving, but it is based on a non-standard version of OWL including the epistemic operator **K**.

3.7 Summary

This chapter presented a direct mapping of Relational Databases to RDF and OWL. To the best of our knowledge this is the first direct mapping, that has been formalized and studied with respect to these fundamental and desirable properties. The contributions are:

- A direct mapping Relational Databases to RDF and OWL formalized in non

recursive Datalog.

- A monotone, information preserving, query preserving and positive semantics preserving direct mapping which considers databases that have null values.
- A proof that the combination of monotonicity with the OWL semantics is an obstacle to generating a full semantics preserving direct mapping.
- A non-monotone direct mapping that is full semantics preserving.

Chapter 4

Ultrawrap: SPARQL Execution on Relational Data

[1] The previous chapter presented a method which automatically maps a relational database as a Semantic Web data source, namely RDF and OWL. The natural next step is to consider the problem of how to execute SPARQL queries against a relational database, per the direct mapping presented in Chapter 3. The goal is to define a method to execute SPARQL queries which makes use of optimizations incorporated in relational databases over the past several decades. Related work has been ad-hoc and predicated on incorporating optimizing transforms as part of the SPARQL to SQL translation, and/or executing some of the query outside the underlying SQL environment [1, 2, 117].

The explored hypothesis is: *existing commercial relational databases already subsume the algorithms and optimizations needed to support effective SPARQL execution on existing relationally stored data.* This research's postulate is that by carefully constructing unmaterialized SQL views to represent the direct mapping, the existing algorithmic machinery in SQL optimizers is already sufficient to effectively execute SPARQL queries on the relational data.

The experiment is embodied in a system, Ultrawrap, which wraps a relational database and virtualizes it as a Semantic Web data source. The direct mapping is encoded using SQL views. SPARQL queries are syntactically translated to SQL queries in terms of the views. In the course of executing a SPARQL query, the SQL optimizer uses the SQL views that represent the direct mapping, and optimizes its execution.

Ultrawrap is evaluated using two benchmark suites for each of the three major relational database management systems. Empirical analysis reveals two existing relational query optimizations that, if applied to the SQL produced from a simple syntactic translations of SPARQL queries (with bound predicate arguments)

[1]Part of this chapter has been published as: Juan F. Sequeda and Daniel P. Miranker. 2013. Ultrawrap: SPARQL execution on relational data. Journal of Web Semantics. 22 (October 2013), 19-39. Daniel P. Miranker was an advisor for this work.

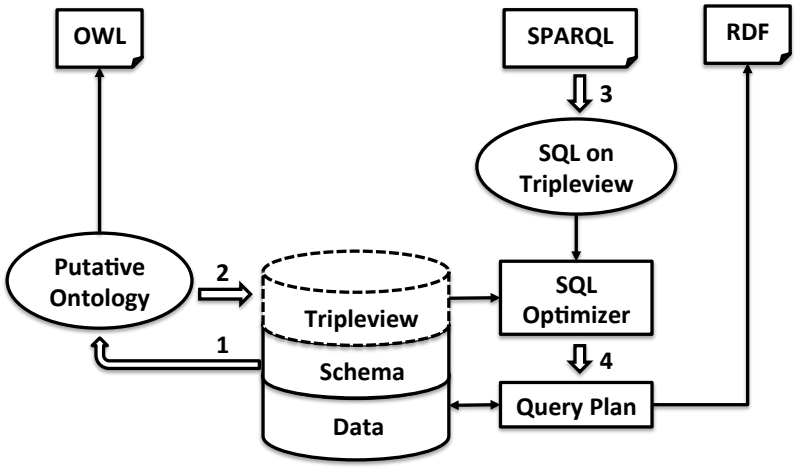


Figure 4.1: Architecture of Ultrawrap

to SQL, consistently yield query execution time that is comparable to that of SQL queries written directly for the relational representation of the data. The analysis further reveals the two optimizations are not uniquely required to achieve a successful wrapper system. The evidence suggests effective wrappers will be those that are designed to complement the optimizer of the target database.

4.1 Overview of Ultrawrap

Ultrawrap is comprised of four primary components as shown in Figure 4.1:

1. The translation of the relational schema to an OWL ontology, per the direct mapping (Chapter 2): the putative ontology (PO).

2. The creation of an intensional triple table in the database by augmenting the relational schema with one or more SQL Views: the Tripleview.

3. Translation of SPARQL queries to equivalent SQL queries operating on the Tripleview.

4. The native SQL query optimizer, which becomes responsible for rewriting triple based queries and effecting their execution on extensional relational data.

These four components can be seen as four different language compilers. As an ensemble, the first three provide for the logical mapping of schema, data and queries between the relational and Semantic Web languages. The fourth component, the SQL optimizer, is responsible for the evaluation of the data mappings and concomitant optimization of the query. The components of Ultrawrap may also be decomposed as a compilation phase and a runtime phase.

4.1.1 Compilation

The goal of the compilation phase is the creation of the Tripleview. The first step in the compilation phase is the translation of the relational schema to OWL.

Generating the Putative Ontology

To define the mapping of the relational data to RDF, the system first identifies the putative ontology of the relational schema. The direct mapping of Chapter 3 is implemented, which includes transformation rules for integrity constraints (foreign keys and primary keys). As a reminder, the direct mapping consists of the following: representing tables as ontological classes, foreign key attributes of a table as object properties and all other attributes as datatype properties. Tables that represent a many-to-many relationship (a.k.a. a join table) are translated to object properties. Each property has its respective domain and range. Both datatype and object properties have classes as domains. Datatype properties have a datatype as a range while object properties have a class as its range.

Creating the Tripleview

The putative ontology is the input to a second compilation step that creates a logical definition of the relational data as RDF and embeds it in a view definition. Per the Direct Mapping (Section 3), concatenating the table name with the primary key value or table name with attribute name creates unique identifiers for subject, predicate and objects. Subsequently, unique identifiers can be appended to a base URI. A SQL Triplequery is a SELECT-FROM-WHERE (SFW) statements which outputs triples. The WHERE clause filters attributes with null values (IS NOT NULL), given that null values are not expressible in RDF. The SQL Tripleview is comprised of a union of SQL Triplequeries.

Due to its simplicity, the starting point is the triple table approach. Even though, studies have shown that storing RDF with the triple table approach in a relational database is easily improved upon [4, 94], this issue is not relevant to Ultrawrap because the relational data is not being materialized in a triple table. Instead the relational data is virtually represented as a triple table through unmaterialized views. Even though the goal is to define a virtual triple table, the physical characteristics of the database and the capacity of the SQL optimizer to produce optimal physical plans need to be anticipated. Toward that end, two refinements to the Tripleview have been identified.

Refinement 1: The initial approach was to create a single Tripleview with 3 attributes: `(subject, predicate, object)`. The subject corresponds to concatenating the name of the table and the primary key value. The predicate is a constant value that corresponds to each attribute name of each table. There can be two types of object values: a value from the database or a concatenation of the name of a table with its primary key value. However, joins were slow because the optimizer was not exploiting the indexes on the primary keys. Therefore, the Tripleview was extended to consist of 5 attributes: `(subject, subject pk, predicate, object, object`

`pk)`. Separating the primary key in the Tripleview allows the query optimizer to exploit them because the joins are done on these values. If the object is a value, then a NULL is used as the primary key of the object. The subject and object are still kept as the concatenation of the table name with the primary key value because this is used to generate the final URI, which uniquely identifies each tuple in the database. For simplicity, composite keys were not considered in the Tripleview. Nevertheless, it is possible to augment the number of attributes in the Tripleview to include each separate key value.

Refinement 2: Refinement 1 represented the entire database in a single Tripleview. This meant that all values were cast to the same datatype (namely varchar). Even though all values were cast to varchar, it is observed throughout the experiments that the optimizer was still able to apply operators specific for other datatypes (i.e, $>$, $<$, etc). However, the size of the object field of the Tripleview is the size of the largest varchar which led to poor query performance. Due to this issue, it was beneficial to create a separate Tripleview for each datatype. For varchar, this includes each length declared in the schema. For example, datatypes with varchar(50) and varchar(200) are considered different. Using multiple Tripleviews requires less bookkeeping than one might anticipate. Each attribute is mapped to its corresponding Tripleview and stored in a hashtable. Then, given an attribute, the corresponding Tripleview can be retrieved.

Figure 4.2 shows an example relational database. Pseudo-code for creating the Tripleviews is shown in Algorithm 1, 2, 3 and 4. The complexity of these algorithms is linear with respect to the size of the putative ontology (PO). The logical contents of the Tripleviews are shown in Tables 4.1, 4.2, 4.3, 4.4. Figure 4.3 shows the CREATE VIEW statements for the Tripleviews.

Product				
id	label	pnum1	pnum2	prodFK
1	ABC	1	2	4
1	XYZ	3	3	5

Producer		
id	title	location
4	Foo	TX
5	Bar	CA

Figure 4.2: Example of Product and Producer tables of BSBM

$PO \longleftarrow$ Transform Schema to OWL;
$V \longleftarrow \emptyset$;
foreach *ontological object x of PO* **do**
  if *x is a OWL Class* **then**
    pk = getPrimaryKey(x) ;
    $S \longleftarrow$ `SELECT concat(x,pk) as s, pk as s_id, "rdf:type" as p, x as o, null as o_id FROM x` ;
    add S to V;
  end
end
return `createTripleview("Tripleview_type",` V`)`

Algorithm 1: Pseudo-code to create a Tripleview for types

$PO \longleftarrow$ Transform Schema to OWL;
$V \longleftarrow \emptyset$;
foreach *ontological object x of PO* **do**
  if *x is a OWL Datatype Property* **then**
    datatype = getDatatype(x);
    if *datatype == varchar* **then**
      domain = getDomain(x) ;
      pk = getPrimaryKey(domain) ;
      $S \longleftarrow$ `SELECT concat(domain,pk) as s, pk as s_id, "x" as p, x as o, null as o_id FROM domain WHERE x IS NOT NULL` ;
      add S to V;
    end
  end
end
return `createTripleview("Tripleview_varchar",` V`)`

Algorithm 2: Pseudo-code to create a Tripleview for Varchar Datatype Properties

```
PO ⟵ Transform Schema to OWL;
V ⟵ ∅;
foreach ontological object x of PO do
    if x is a OWL Object Property then
        domain = getDomain(x);
        d_pk = getPrimaryKey(domain);
        range = getRange(x);
        r_pk = getForeignKeyAttributeInDomain(range, domain);
        S ⟵ SELECT concat(domain, d_pk) as s, d_pk as s_id,
        concat(domain, #,range) as p, concat(range, r_pk) as
        o, r_pk as o_id FROM x ;
        add S to V;
    end
end
return createTripleview("Tripleview_object", V)
```

Algorithm 3: Pseudo-code to create a Tripleview for Object Properties

input : A name n of the tripleview and a list Q of Triple queries
output: The Tripleview

```
TripleView ⟵ 'CREATE VIEW n(s, s_id, p, o, o_id) AS' ;
foreach Triple query q in Q do
    if q is the last element in Q then TripleView ⟵ q;
    else TripleView ⟵ q + UNION ALL;
end
return TripleView
```

Algorithm 4: Pseudo-code of the createTripleview method

S	S_ID	P	O	O_ID
Product1	1	type	Product	NULL
Product2	2	type	Product	NULL
Producer4	4	type	Producer	NULL
Producer5	5	type	Producer	NULL

Table 4.1: Logical contents of Tripleview for types

```
CREATE VIEW Tripleview_type(s, s_id, p, o, o_id) AS
SELECT Product+id as s, id as s_id, type as p, Product as o, null as o_id
FROM Product
UNION ALL
SELECT Producer+id as s, id as s_id, type as p, Producer as o, null as o_id
FROM Producer;

CREATE VIEW Tripleview_varchar50(s, s_id, p, o, o_id) AS
SELECT Product+id as s, id as s_id, label as p, label as o, null as o_id
FROM Product WHERE label IS NOT NULL
UNION ALL
SELECT Producer+id as s, id as s_id, title as p, title as o, null as o_id
FROM Producer WHERE title IS NOT NULL
UNION ALL
SELECT Producer+id as s, id as s_id, location as p, location as o, null as o_id
FROM Producer WHERE location IS NOT NULL;

CREATE VIEW Tripleview_int(s, s_id, p, o, o_id) AS
SELECT Product+id as s, id as s_id, pNum1 as p, pNum1 as o, null as o_id
FROM Product WHERE pNum1 IS NOT NULL
UNION ALL
SELECT Product+id as s, id as s_id, pNum2 as p, pNum2 as o, null as o_id
FROM Product WHERE pNum2 IS NOT NULL;

CREATE VIEW Tripleview_object(s, s_id, p, o, o_id) AS
SELECT Product+id as s, id as s_id, Product#producer as p,
Producer+prodFk as o, prodFk as o_id
FROM Product
```

Figure 4.3: CREATE VIEW statements defining the Tripleviews

S	S_ID	P	O	O_ID
Product1	1	Product#label	ABC	NULL
Product2	1	Product#label	XYZ	NULL
Producer4	1	Producer#title	Foo	NULL
Producer4	1	Producer#location	TX	NULL
Producer5	1	Producer#title	Bar	NULL
Producer5	1	Producer#location	CA	NULL

Table 4.2: Logical contents of Tripleview for varchar

S	S_ID	P	O	O_ID
Product1	1	Product#pNum1	1	NULL
Product1	1	Product#pNum2	2	NULL
Product2	2	Product#pNum1	3	NULL
Product2	2	Product#pNum2	3	NULL

Table 4.3: Logical contents of Tripleview for int

S	S_ID	P	O	O_ID
Product1	1	Product#producer	Producer4	4
Product2	2	Product#producer	Producer5	5

Table 4.4: Logical contents of Tripleview for object properties

4.1.2 Runtime

Ultrawrap's runtime phase encompasses the translation of SPARQL queries to SQL queries on the Tripleviews and the maximal use of the SQL infrastructure to do the SPARQL query rewriting and execution.

SPARQL to SQL translation

SPARQL is a graph pattern matching query language that has the form:

```
SELECT ?var1 ? var2 . . .
WHERE {
   triple-pattern-1.
   triple-pattern-2.
   . . .
   triple-pattern-n.
}
```

where each triple-pattern consists of a subject, predicate, object and any of these can be a variable or a constant URI. Variables can occur in multiple triple-patterns implying a join. Consider the following SPARQL query as our running example:

```
SELECT ?lbl ?pn1
WHERE{
   ?x :label ?lbl.
   ?x :pnum1 ?pn1.
}
```

This SPARQL query binds the predicate of the first triple pattern to the constant `:label` and the predicate of the second triple-pattern to the constant `:pnum1`. The variable `?x` indicates that the results of triple-pattern-1 and triple-pattern-2 are to be joined and the final result is the projection of the binding to the variable `?lbl` and `?pn1`.

The translation of the SPARQL query to a SQL query on the Tripleviews follows a classic compiler structure. First, a parser converts the SPARQL query string to an Abstract Syntax Tree (AST). The AST is translated into an SPARQL algebra expression tree. The SQL translation is accomplished by traversing the expression tree and replacing each SPARQL operator. Each internal node of the expression tree represents a SPARQL binary algebra operator while the leaves represent a Basic Graph Patterns (BGP), which is a set of triple patterns. A SPARQL BGP is a set of triple patterns where each one maps to a Tripleview. A SPARQL Join maps to a SQL Inner Join, a SPARQL Union maps to the SQL Union, a SPARQL Optional maps to SQL Left-Outer Join. In the previous example, there is only one BGP with two triple patterns and a Join between the triple patterns. The resulting SQL query is:

```
SELECT t1.o AS lbl, t2.o AS pn1
FROM tripleview_varchar50 t1, tripleview_int t2
WHERE t1.p = 'label' AND t2.p = 'pnum1' AND t1.s_id = t2.s_id
```

Hereafter, this is called the Ultrawrap query. Note that the mapping mentioned in Refinement 2 was used in order to know which Tripleview to use. At the initial

setup of the runtime, a hash table with the contents of the mapping is created. Therefore given a property such as `:label` (key), the mapped Tripleview, in this case `tripleview_varchar50` (value) can be retrieved.

SQL engine is the Query Rewriter

Given the Ultrawrap query to be executed on the Tripleviews, the query is executed and it is observed how the SQL engine operates. These results are described in the following section. A main concern is if the SQL query can actually be parsed and executed on the Tripleviews, given the view is a very large union of a large amount of Triplequeries (SFW statements). In the evaluation, BSBM consisted of 10 tables with a total of 78 attributes and Barton consisted of 20 tables with a total of 61 attributes.

4.2 Two Important Optimizations

Upon succeeding in *ultrawrapping* different RDBMSs and reviewing query plans, two relational optimizations emerged as important for effective execution of SPARQL queries: 1) detection of unsatisfiable conditions and 2) self-join elimination. Perhaps, not by coincidence, these two optimizations are among semantic query optimization (SQO) methods introduced in the 1980s [39, 41, 121]. In SQO, the objective is to leverage the semantics, represented in integrity constraints, for query optimization. The basic idea is to use integrity constraints to rewrite a query into a semantically equivalent one. These techniques were initially designed for deductive databases and then integrated in commercial relational databases [41].

Figure 4.4 shows the logical query plan of the Ultrawrap SQL query from the running example. This section describes how the query plan evolves through these optimizations. Describing a general-purpose implementation of these optimizations is not in the scope of this paper. The interested reader may see [39, 41]. In this

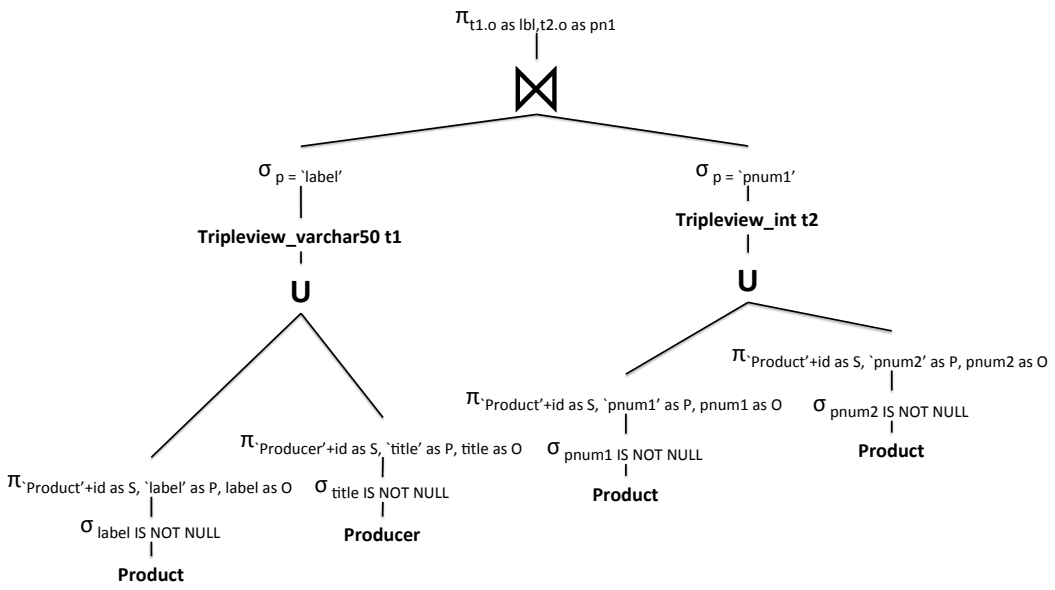


Figure 4.4: Initial query plan of the running example

query plan, for each of the triple patterns in the query, the Tripleview is accessed, which consists of a union of all the Triplequeries.

4.2.1 Detection of Unsatisfiable Conditions

The idea of this optimization is to determine that a query result is empty by determining, without executing the query. This happens, for example, when a pair of predicate constants are inconsistent [39]. The application of the following transformations eliminates columns from the plan that are not needed to evaluate the SPARQL query.

Elimination by Contradiction: Consider a query

```
SELECT * FROM R WHERE A = x AND A = y
```

such that `x` $\neq$ `y`. Then the result of that query is empty. For example, it is clear that the query

```
SELECT * FROM Product WHERE ProductID = 1 AND ProductID = 2
```

will never return results.

Unnecessary Union Sub-tree Pruning: Given a query that includes the UNION operator and where it has been determined that an argument of the UNION is empty; then the corresponding argument can be eliminated. For example:

```
UNION ALL ({}, S, T) = UNION ALL (S, T)
UNION ALL ({}, T)  = T
```

In Ultrawrap's Tripleview, the constant value in the predicate position acts as the integrity constraint. Consider the following Tripleview:

```
CREATE VIEW Tripleview_varchar50(s,s_id,p,o,o_id) AS
SELECT "Producer"+id as s, id as s_id, "title" as p, title as o, null as o_id
FROM Producer WHERE title IS NOT NULL
UNION ALL
SELECT "Product"+id as s, id as s_id, "label" as p, label as o, null as o_id
FROM Product WHERE label IS NOT NULL
```

Now consider the following Ultrawrap query return all labels:

```
SELECT o FROM Tripleview_varchar50 WHERE p = "label"
```

The first Triplequery from `Tripleview_varchar50` contains `"title" as p` which defines `p = "title"`. The query contains `p = "label"`. Both predicates cannot be satisfied simultaneously. Given the contradiction, the Triplequery containing `"title" as p` of `Tripleview_varchar50` can be replaced with the empty set. Since the Tripleview's definition includes all possible columns, any specific SPARQL query will only need a subset of the statements defined in the view. Application of elimination by contradiction enables removing, the unnecessary UNION ALL conditions. Thus the combination of the two transformations reduces the Tripleview to precisely

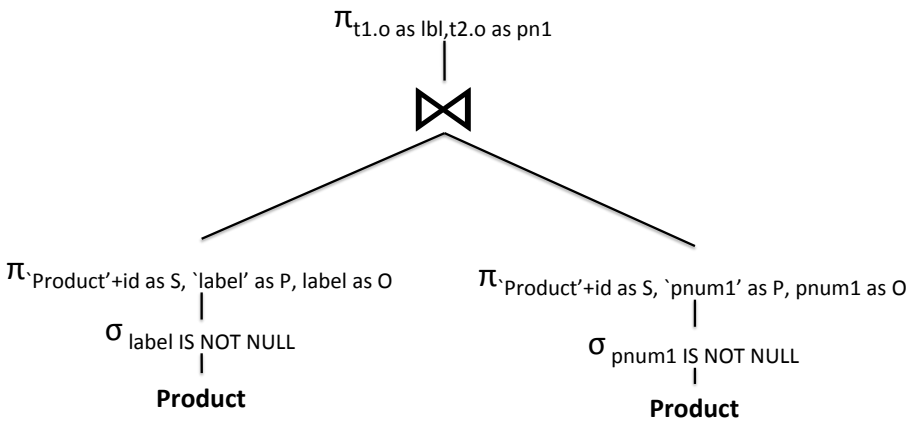


Figure 4.5: Query plan after application of Detection of Unsatisfiable Conditions optimization

the subset of referenced columns. The differences in the query plans in Figures 4.4 and 4.5 illustrate the impact of these optimizations.

Augmenting Ultrawrap

The Ultrawrap architecture is readily extended to include the detection of unsatisfiable conditions optimization. By creating such a version, Augmented Ultrawrap, the following controlled experiment was conducted. Instead of creating a mapping between each attribute in the database and its corresponding Tripleview, a mapping is created for each attribute to its corresponding Triplequery. For example, the property `:label` is mapped to the Triplequery: `SELECT "Product"+id as s, id as s_id, "label" as p, label as o, null as o_id FROM Product WHERE label IS NOT NULL`. At the initial setup of the runtime, a hash table with the contents of this mapping is generated. Therefore given a property such as `:label` (key), the mapped Triplequery (value) can be retrieved. The view definition nested in the SQL querys FROM clause is replaced with the Triplequery.

4.2.2 Self-join Elimination

Join elimination is one of the several SQO techniques, where integrity constraints are used to eliminate a literal clause in the query. This implies that a join could also be eliminated if the table that is being dropped does not contribute any attributes in the results [39]. The type of join elimination that is desired is the self-join elimination, where a join occurs between the same tables. Two different cases are observed: self-join elimination of projection and self-join elimination of selections.

Self-join elimination of projection: This occurs when attributes from the same table are projected individually and then joined together. For example, the following unoptimized query projects the attributes `label` and `pnum1` from the table `product` where `id = 1`. However each attribute projection is done separately and then joined:

```
SELECT p1.label, p2.pnum1
FROM product p1, product p2
WHERE p1.id = 1 and p1.id = p2.id
```

Given a self-join elimination optimization, the previous query may be rewritten as:

```
SELECT label, pnum1 FROM product WHERE id = 1
```

Self-join elimination of selection: This occurs when a selection on attributes from the same table are done individually and then joined together. For example, the following unoptimized query selects on `pnum1 > 100` and `pnum2 < 500` separately and then joined:

```
SELECT p1.id
FROM product p1, product p2
WHERE p1.pnum1 >100 and p2.pnum2 < 500 and p1.id = p2.id
```

Given a self-join elimination optimization, the previous query may be rewritten as:

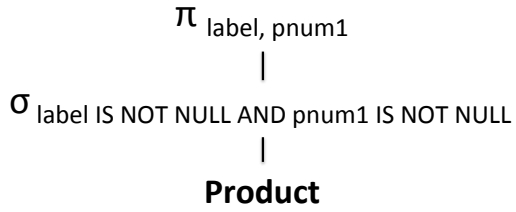


Figure 4.6: Query plan after self-join elimination optimization

```
SELECT id FROM product WHERE pnum1 > 100 and pnum2 < 500
```

Figure 4.6 shows the final query plan after the self-joins are removed.

4.3 Evaluation

The evaluation requires workloads where the SPARQL queries anticipated that the RDF data was derived from a relational database. Two existing benchmarks fulfill this requirement. The Berlin SPARQL Benchmark (BSBM) [23] imitates the query load of an e-commerce website. The Barton Benchmark [5] replicates faceted search of bibliographic data. For Barton, the readily available RDF data was derived from a dump of MITs Barton library catalog. The original relational data is not available. Similarly the only queries that are available are queries written in SQL against a triple table schema. A version of Barton on par with BSBM was created. This new version of Barton was packaged so the community may reuse it. In lieu of MITs library catalog a relational form of DBLP was used. The SPARQL queries and SQL queries that operate directly on the relational version of DBLP were deduced from the english specification of the queries. Details of the relational schemas and queries for the BSBM and Barton benchmark can be found in the Appendix.

The objective of this evaluation is to observe the behavior of commercial relational databases. Therefore the evaluation compares execution time, queries plans, and the optimizing transforms used between the Ultrawrap SQL queries and

the benchmark-provided SQL queries on the respective RDBMS. Other possible experiments include comparing Ultrawrap with other Wrapper systems (i.e. D2RQ). However this is not in the scope of this work. Nevertheless, as shown in the results, Ultrawrap query execution time is comparable with the execution time of benchmark-provided SQL queries. Such results have not been accomplished by any other Wrapper system [23, 66].

4.3.1 Platform

Ultrawrap was installed on Microsoft SQL Server 2008 R2 Developer Edition, IBM DB2 9.2 Express Edition and Oracle 11g Release 2 Enterprise Edition. Experiments were conducted on a Sun Fire X4150 with a four core Intel Xeon X7460 2.66 GHz processor and 16 GB of RAM running Microsoft Windows Server 2008 R2 Standard on top of VMWare ESX 4.0. SQL Server and Oracle had access to all cores and memory, while DB2 had only access to one core and 2 GB of RAM.

4.3.2 Workload

The BSBM dataset is equivalent to approximately 100 million RDF triples and requires approximately 11 GB of storage. For Barton, the DBLP dataset is equivalent to approximately 45 million RDF triples and requires approximately 4 GB of storage. Indexes were built on every foreign key and on attributes that were being selected on in the benchmark queries. The execution time was calculated by using the elapsed time returned from SQL Server's SET STATISTICS ON, DB2's db2batch and Oracle's SET TIMING ON option.

Note that the DB2 Express Edition limits itself to 2 GB of RAM. Otherwise, the available RAM is larger than the benchmark databases. To control for this, both cold and warm start experiments were run. Warm start experiments were done by loading the data, restarting the databases and executing variants of each

Selectivity	Inner Joins	Left-Outer Join	Predicate Variable
Low	BSBM 1, 3, 10, Barton 5, 7	-	BSBM 9, 11
High	BSBM 4, 5, 6, 12	BSBM 2, 7, 8	Barton 1, 2, 3, 4, 6

Table 4.5: Query characteristics of the BSBM and Barton queries

query twenty times. Cold start experiments were done by restarting the database after each execution of a query. The results of the cold start experiments are not qualitatively different than the warm start results and thus are omitted.

The benchmark queries consist of a wide variety of operators and characteristics: Basic Graph Patterns, UNION, FILTER, OPTIONAL, ORDER BY and unbounded predicates with high and low selectivity. Details about the queries for both BSBM and Barton benchmark can be found in the Appendix. Characteristics of the queries are shown in Table 4.5.

The initial assessment suggests observations be organized as four cases:

Case 1) Detection of Unsatisfiable Conditions and Self-join Elimination

If both optimizations are applied then the UNION ALLs of the Tripleviews should not appear in the query plans and redundant self-joins should be eliminated. The execution time and query plans of Ultrawrap queries should be comparable to the corresponding benchmark-provided SQL queries. This should be the case for all queries except the special-case of predicate variable queries, which form Case 4.

Case 2) Detection of Unsatisfiable Conditions without Self-join Elimination

If only the first optimization is applied, then the UNION ALLs of the Tripleviews do not appear in the query plans and the number of subqueries is equal to the number of triple patterns in the original SPARQL query. When the selectivity is high, the execution time of Ultrawrap queries should be comparable to benchmark-provided

SQL queries because the number of tuples that are self-joined is small. On the other hand, when selectivity is low the number of tuples joined is larger and the overhead is more evident. Note that the self-join elimination optimization can only be applied after the UNIONs have been eliminated; hence the complementary case does not occur.

Case 3) No optimizations

If no optimizations are applied then the UNION ALLs of the Tripleviews are not eliminated. In other words, the physical query plan is equal to the initial logical query plan (e.g. Figure 8). The Ultrawrap query execution time should not be comparable to the benchmark-provided SQL queries because every Triplequery in the Tripleviews must be executed.

Case 4) Predicate variable queries

Predicate variable queries are queries that have a variable in the predicate position of a triple pattern. Given a direct mapping, the predicate variable in a SPARQL query is a one-to-many mapping that ranges over all attributes in the database. These types of queries cannot use the mapping between the attributes and its corresponding Tripleview because the attribute is unknown. Further, because the attribute is unknown, detection of unsatisfiable conditions cannot be applied. For these queries, the Tripleview described in Refinement 1 is used.

In a paper on the use of views in data integration, Krishnamurthy et. al. [87] show that queries with variables ranging over attributes and table names are of higher order logic. Relational algebra languages, such as SQL, do not support higher order logic [87, 89]. Similarly, a SPARQL query with a predicate variable does not have a concise, semantically equivalent SQL query. By concise it is meant that the SQL query itself will avoid a union of queries over different tables or columns.

For the SPARQL predicate variable queries, when writing the benchmark SQL queries, a SQL developer has visibility on the SQL schema and has related domain knowledge. In most cases that developer will understand that only a few columns are of interest, and write a smaller SQL query than the corresponding SPARQL query. In other words, the SQL query will access certain individual columns, but the SPARQL query will expand to access all columns in the database. This occurs for all such queries for both benchmarks. Thus, it is arguable whether the tests comparing SPARQL queries that contain predicate variables, with the benchmark-provided SQL queries provides a semantically equivalent, apples-to-apples test. Nevertheless, the queries are executed and the data is included.

4.3.3 Results

Results of two experiments are reported. The first experiment, Ultrawrap Experiment, evaluates Ultrawrap implemented as presented. The second experiment, Augmented Ultrawrap Experiment, evaluates a version of Ultrawrap augmented with the detection of unsatisfiable conditions optimization.

DB2 implements both optimizations. SQL Server implements the detection of unsatisfiable conditions optimization. Oracle implements the self-join elimination optimization, but it fails to apply it if the detection of unsatisfiable conditions optimization is not applied. Neither optimization is applied on the predicate variables queries by any RDBMS. Table 4.6 summarizes the optimizations implemented by each RDBMS. The results of both experiments are presented in Figures 11-13. The Ultrawrap execution time is normalized w.r.t the benchmark-provided SQL query execution time for each respective RDBMS, i.e. benchmark-provided SQL query execution time is 1.

RDBMS	Detection of Unsatisfiable Conditions	Self-join Elimination
DB2	Yes	Yes
SQL Server	Yes	No
Oracle	No	Yes

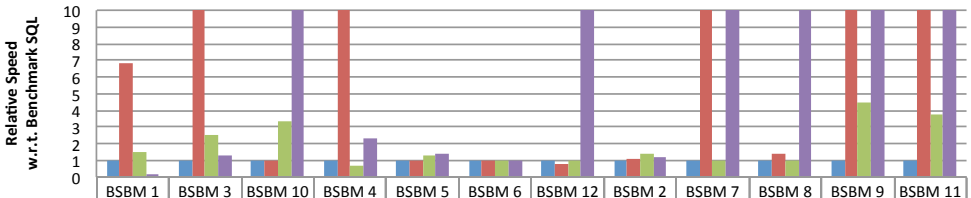
Table 4.6: Optimizations implemented by existing RDBMS

4.3.4 Ultrawrap Experiment

DB2 implements both optimizations. Therefore it is expected that it will execute Ultrawrap queries comparable to native SQL queries (Case 1). This is the case for 7 of the 12 SPARQL queries with bound predicates (BSBM 2, 5, 6, 8, 10, 12 and Barton 7). For the exceptions, BSBM 1, 3, 4 and Barton 5, the optimizer generated a query plan typical of the benchmark-provided SQL queries, but with a different join order. BSBM 7 has nested left-outer joins. For that query, the DB2 optimizer did not push the respective join predicates into the nested queries and corresponding index-based access paths are not exploited.

SQL Server implements the detection of unsatisfiable conditions optimizations but not self-join elimination. Thus, one would still expect that the high selectivity queries would perform comparable or better than the benchmark-provided SQL queries (Case 2). This is the case for all 7 such queries. For BSBM 4, the optimizer produced a different join order for the two versions of the query, but this time, the Ultrawrap query was better. For the low selectivity queries, review of the query plans reveals the discrepancy in performance is due precisely to the absence of the self-join elimination.

Although Oracle implements self-join elimination it does not apply it in this experiment, and thus does not apply either distinguished optimizations (Case 3). Nevertheless, on 7 of the 12 queries with bound predicates, the Ultrawrap execution is comparable or better than the benchmark-provided SQL query execution. Review of the query plans yields a third valuable optimization: join predicate push-down



	BSBM 1	BSBM 3	BSBM 10	BSBM 4	BSBM 5	BSBM 6	BSBM 12	BSBM 2	BSBM 7	BSBM 8	BSBM 9	BSBM 11
	Low Selectivity			High Selectivity				Left Outer Join w/ High Selectivity			Unbounded Predicate	
Benchmark SQL	1	1	1	1	1	1	1	1	1	1	1	1
UW DB2	6.86	14.53	1.02	14.74	1.01	1	0.76	1.08	1745	1.41	32011	67538
UW SQL Server	1.48	2.52	3.39	0.71	1.29	1.01	1	1.4	0.99	0.97	4.51	3.71
UW Oracle	0.15	1.28	475.69	2.29	1.37	0.99	15.36	1.21	74.48	2224.4	6710	10.89

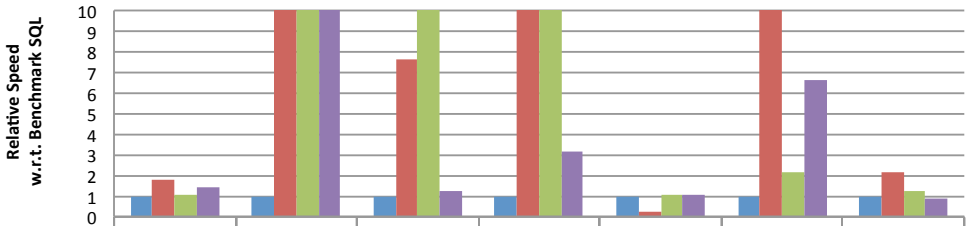
Figure 4.7: Ultrawrap Experiment results on BSBM

into each of the Triplequeries in the UNION ALL of the Tripleviews. Even though each Triplequery is executed, most do not contribute to the final result. By virtue of the additional predicate push-down the execution overhead is minimal.

It is expected that neither optimization be applied for predicate variable queries. This is the case for all three RDBMSs (Case 4). Nevertheless, there are some unanticipated results. The benchmark-provided SQL queries and Ultrawrap queries for Barton 1 and 6 have similar query plans hence the execution times are comparable. SQL Server outperforms the other systems on BSBM queries 9 and 11 while Oracle outperforms the other systems on Barton 3 and 4. For these cases, the optimizer pushed selects down.

4.3.5 Augmented Ultrawrap Experiment

Augmented Ultrawrap greedily applies the detection of unsatisfiable conditions optimization to the Ultrawrap SQL queries prior to passing the query to the RDBMS for evaluation. Note, that this optimization is not applicable to triple patterns with predicate variables. This should not, and did not impact the behavior of queries with predicate variables. For clarity and space, the corresponding data is omitted. Figure 4.9 contains the results for the Augmented Ultrawrap experiment.



	B1	B2	B3	B4	B6	B5	B7
	Unbounded Predicate					Low Selectivity	
Benchmark SQL	1	1	1	1	1	1	1
UW DB2	1.75	31.18	7.61	162.85	0.27	13.56	2.17
UW SQL Server	1.02	48.44	50.64	43.08	1.07	2.13	1.29
UW Oracle	1.43	116.64	1.21	3.12	1.04	6.67	0.92

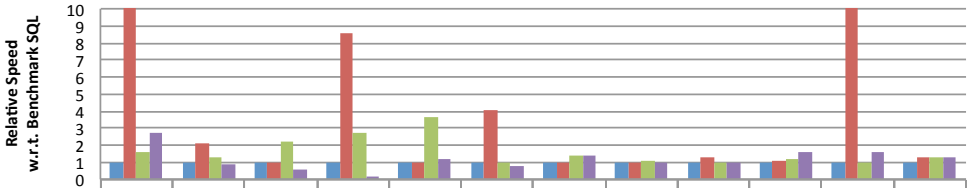
Figure 4.8: Ultrawrap Experiment results on Barton

In this experiment Cases 2 and 3 are eliminated. Of the three RDBMS only Oracle does not implement detection of unsatisfiable conditions. Thus, despite experimenting with closed proprietary systems, this experiment constitutes a controlled test of the value of this optimization.

Observe that Oracle now performs comparable or better on all bound predicate Ultrawrap queries than the comparable benchmark-provided SQL queries. Inspection of the plans reveals that the Oracle optimizer applies the self-join elimination optimization where it did not in the first experiment. Thus, in the second experiment, Oracle's plans include both distinguished optimizations (Case 1). For BSBM 1, 3, 4 and Barton 7, the Ultrawrap execution is better than the benchmark-provided SQL query execution because the optimizer produced an optimal join order for the Ultrawrap queries. To the best of our knowledge, the benchmark-provided SQL queries were tuned for better performance. Due to lack of Oracle DBA skills, benchmark-provided SQL queries BSBM 1, 3, 4 and Barton 7 were not tuned to the best performance possible.

SQL Server results are largely unchanged.

The only unanticipated results were changes for DB2 for the unsuccessful



	Barton 5	Barton 7	BSBM 1	BSBM 3	BSBM 10	BSBM 4	BSBM 5	BSBM 6	BSBM 12	BSBM 2	BSBM 7	BSBM 8
	Low Selectivity					High Selectivity				Left Outer Join w/ High Selectivity		
Benchmark SQL	1	1	1	1	1	1	1	1	1	1	1	1
UW DB2	12.33	2.08	1.02	8.58	1	4.03	1.02	1	1.28	1.04	303.07	1.29
UW SQL Server	1.62	1.32	2.19	2.69	3.69	0.99	1.36	1.1	1	1.24	0.94	1.27
UW Oracle	2.72	0.92	0.62	0.17	1.19	0.75	1.39	0.99	1	1.57	1.57	1.27

Figure 4.9: Augmented Ultrawrap Experiment results on BSBM and Barton bounded predicate queries.

bounded predicate queries from the Ultrawrap Experiment (BSBM 1, 3, 4, 7 and Barton 5). In all cases, performance improved. This was the result of changes in the join order, and choosing additional index-based access paths. But in only 1 of the 5 queries, BSBM 1, does the optimizer choose the same join-order as the benchmark-provided SQL query. A number of options were investigated to get better join orders and concomitant theories as to the search behavior of the DB2 optimizer. None of these options resolved the issues..

4.4 Discussion

The following points deserve elaboration:

Self-join elimination: The number of self-joins and their elimination is not, by itself, an indicator of poor performance. The impact of the self-join elimination optimization is a function of the selectivity and the number of properties in the SPARQL query that are co-located in a single table. The value of optimization is less as selectivity increases. Qualitatively, the result is predictable. The conclusion on quantitative results follows by comparing performance of low selectivity vs. high selectivity queries on SQL Server as shown in Figure 4.7 and 13. The number of

self-joins in the plan corresponds to the number of properties co-located in a table. The phenomenon is reminiscent of the debate concerning the use of row-stores vs. column stores started by Abadi et al [3, 4, 33, 122, 132]. Consideration of row-stores vs. column-stores is outside the scope of this dissertation. Nevertheless, these measurements may help ground that debate.

Join predicate push-down: The experiments with Oracle revealed that pushing join predicates can be as effective as the detection of unsatisfiable conditions optimization. For the case of BSBM 7 on Oracle, the optimizer did not push the join predicates down; hence the poor query execution time.

Left-Outer Joins: The experiments revealed that no commercial optimizer eliminates self left-outer joins and OPTIONALs appear in many of the queries where suboptimal join orders are determined. The experimental results are supportive of hearsay in the Semantic Web community that the endemic use of OPTIONAL in SPARQL queries, which compiles to a left-outer join, is outside the experience of the database community. The speculation is that these types of queries are not common in a relational setting, hence the lack of support in commercial systems.

Join Ordering: Join order is a major factor for poor query execution time, both on Ultrawrap and benchmark-provided SQL queries. Even though DB2 eliminated self-joins in the original Ultrawrap experiment, the optimizer often generated suboptimal join order for the Ultrawrap queries but did so less often for the Augmented Ultrawrap queries. A possible explanation is simply the size of the search space. For Ultrawrap queries the optimizer has to evaluate each query within the large union in the definition of the Tripleviews. The Augmented Ultrawrap eliminates unneeded UNION ALL elements, reducing the search space.

Counting NULLs: Each Triplequery in the Tripleview filters null values. Such a filter could produce an overhead, however a hypothesis is that the optimizer has statistics of null values and avoids the overhead.

4.5 Related Work

Per the taxonomy in Figure 1, systems that involve relational databases are RDBMS-backed Triplestores and Wrapper systems.

RDBMS-backed Triplestores store RDF in different database schemas. Many RDBMS-backed Triplestores use the triple table schema: a table with three attributes, containing one row for each triple [44]. Another approach is the property table: a table comprising of one column containing the subject plus one or more columns for predicates that are defined for the subject [133]. Abadi et al. introduced the vertical partitioned table: a table for every unique predicate in the data [4]. Three published research efforts concerning RDBMS-backed triplestores are described [4, 40, 55].

Abadi et al argue for the use of column-store based relational systems as the basis of RDBMS-backed triplestores, as compared to more common row-stores. The paper does not address SPARQL to SQL translation. With respect to translation, the papers core contribution is the mapping of RDF to a relational schema comprising one table for each predicate value. The resulting tables each contain two columns, the subject and object. The organization is well suited for join processing on a column-store database.

Chebotko et al. present a translation of SPARQL to SQL, where the RDF is modeled as a triple table. They argue that their translation may be composed with additional mappings, enabling their translation to be applied to any relational model of RDF. They reported empirical results for a synthetic RDF test set of 1 million triples. The generated SQL resembles the relational algebra rules used to define the semantics of SPARQL, resulting in multiple coalesce functions in one projection, null-accepting predicates, and outer union implementations [40]. The translation is proven to be semantics preserving.

Elliot et al. improve upon Chebotko. Chebotko et al.'s methods exploit

nested SQL queries. Central to Ellliot et al.'s contribution are algorithms that produce flat SQL queries. Their evaluation was also on a triple table schema with datasets between 2-5 million RDF triples [55].

Related studies have compared native triplestores with Wrapper systems and native triplestores with relational database. In 2007, Svihla and Jelinek determined that Wrapper systems are faster than the Jena and Sesame triplestores [126]. In 2009, Schmidt et al compared Sesame triplestore with the triple table, vertical partitioned storage scheme and the native relational scheme on MonetDB, a column-store relational database. This study concluded that none of the RDF schemes was competitive to the native relational scheme [116]. In 2010, MahmoudiNasab and Sakr also compared the triple table, property table and vertical partitioned storage scheme with the native relational scheme on IBM DB2. They also concluded that none of the storage schemes compete with the native relational scheme [94]. In conclusion, benchmark-provided SQL queries on relationally stored data outperform any other approach.

4.6 Concluding Remarks

To date, wrapper systems have suffered problems in performance and scalability [23, 66]. Yet, enterprise class relational database systems do not suffer so. Ultrawrap, and the experiments in this paper move the focus to the relational systems. The primary result being that the application of two known semantic query optimizations may yield a query plan typical of a relational query plan.

The research commenced with a hypothesis that not only were such optimizing transforms already known, but they are already implemented in commercial software. To support the hypothesis, it is not necessary to demonstrate that a single system is universally good. Nor does the hypothesis stipulate that the optimizer will do the right thing every time. However, where and how a system failed to attain

an excellent query plan is as critical to the analysis as success. Although in some cases, as the target RDBMSs are proprietary systems we can only speculate to root causes.

For SPARQL queries with bound predicate arguments the experimental results support the hypothesis. Two key optimizing transformations do appear in commercial RDBMSs, and when applied render a SPARQL query plan comparable to the plan generated for benchmark provided SQL queries. These optimizations are not unique. Experiments reveal a third optimization, join predicate push-down, which pushes join predicates into a view containing unions, enables useful performance improvements across the workload, but does not rewrite the plan into one more typical of a comparable SQL query.

Although we stipulated that tuned SQL query plans for the benchmark provided SQL queries forms a good baseline, the existence of optimizations, perhaps not yet known in the literature, may provide further improvement. The third optimization, join predicate push-down underscores the problem is not closed.

Even if one is satisfied with this research's existence proposition, the empirical results still demonstrate there is work to be done. Analysis of incongruous performance between benchmark SQL queries and SPARQL queries revealed that relational optimizers do not always determine optimal join orders. This is not news, and even one of the benchmark SQL queries was not optimized correctly. However, this issue manifest more often for the SPARQL queries. It is not possible to examine the internals of these systems to determine if the complexity of the Ultrawrap queries is challenging the optimizers cost function, the search strategy or both. Recall Ultrawrap transforms a SPARQL query to a SQL query by naively substituting SPARQL operators in the SPARQL query plan with relational operators. Independent of the reason for the optimizers failing to determine optimal join orders, the mere fact that they are failing suggests there is opportunity for improvement by

means of less naive translations.

Even though Ultrawrap was not compared to other Wrapper systems, the results of the experiments show that SPARQL queries with bound predicates on Ultrawrap execute at nearly the same speed as semantically equivalent benchmark-provided SQL queries. These results have not been accomplished by any other Wrapper systems [23, 66].

The only point of controversy may be the distinction of SPARQL queries with predicate variables. In these queries, a mapping stipulates that the variable may be bound to the name of any column in the database. With these semantics, none of the commercial RDBMSs are able to eliminate any elements of the Tripleview union. However, developers familiar with the SQL schema of the RDBMS application are able to choose particular columns from specific tables.

Queries with predicate variables should not be dismissed as a special case. Queries of this form are intrinsic to faceted search, an increasingly common use case. Even so, two arguments that maintain support for our hypothesis include; one, per Krishnamurthy et al [87], predicate variables are a syntactic construct of higher-order logic, therefore the simple SQL queries expressed in the benchmark as equivalent to the SPARQL queries, produce the same answers on the test data, (they are operationally equivalent), but their formal semantics is not the same, and thus should not be used as a basis of comparison. The formally equivalent queries will contain a union [21] and bear comparable performance penalty. A second, more constructive argument is before writing the benchmark-provided SQL query, the SQL developers determined, a priori, which attributes were relevant to the query and which were inconsistent, and they themselves detected unsatisfiable conditions and simply did not code them. In any case, queries with unbound predicate variables remain an open problem.

4.7 Summary

To summarize, our findings include:

- A mapping of relational data to a Tripleview comprising three columns does not instigate the SQL optimizers to use indexes. The view was refined to reflect physical schema properties.

- Two known query optimizations, detection of unsatisfiable conditions and self-join elimination [39], when applied, not only result in comparable execution times between SPARQL and the benchmark-provided SQL queries with bound predicates, the optimizers will often produce identical query plans.

- In some cases, a third optimizing transform, join predicate push down, can be as effective as the detection of unsatisfiable conditions.

- SPARQL queries containing variables that bind to the predicate position remain troublesome. This problem is related to an already described problem concerning the use of views in the implementation of data integration systems.

- The impact of the self-join elimination optimization is a function of the selectivity and the number of properties in the SPARQL query that are co-located in a single table.

- No system, including those that eliminated self equi-joins, eliminated the self left outer joins. The SPARQL optional operator is, by definition, a left outer join.

By starting with a simple wrapper system and evaluating it with sophisticated SQL query optimizers we are able to identify existing, well understood optimization methods that enable wrappers. The results provide a foundation for identifying minimal requirements for effective wrapper systems.

Chapter 5

Ontology Based Data Access

[1] The Direct Mapping (Chapter 3) and Ultrawrap (Chapter 4) address the problem of automatically virtualizing a relational database as a Semantic Web data source. The focus of this chapter is to consider OWL ontologies that have been created independent of the relational database, as the target. The ontologies being considered have richer expressive features including subclass and transitivity. The problem is now the following: given a source relational database, a target OWL ontology with subclass and transitivity, and a mapping from the source database to the target ontology, how can SPARQL queries over the target ontology be answered by the relational database through use of the mappings? This paradigm is called Ontology-Based Data Access (OBDA).

Commonly, researchers have taken one of two approaches to developing OBDA systems: a materialization-based approach (forward chaining) or a rewriting-based approach (backward chaining). In the materialization-based approach, the input relational database D, target ontology $\mathcal{O}$ and mapping $\mathcal{M}$ (from D to $\mathcal{O}$) are used to derive new triples that are stored in an RDF database D_o, which is considered to be the materialization of the data in D given $\mathcal{M}$ and $\mathcal{O}$. The answer to a SPARQL query Q over the target ontology is computed by directly posing Q over D_o [12]. See Figure 1.8.

In the rewriting-based approach, three steps are executed. First, a SPARQL query Q over the ontology $\mathcal{O}$ is given. A new SPARQL query Q_o is generated from Q and $\mathcal{O}$. The query Q_o contains the knowledge of the ontology $\mathcal{O}$ and is called the rewriting of Q w.r.t to $\mathcal{O}$. The majority of the OBDA literature focuses on this step [103]. Second, the mapping, $\mathcal{M}$, is used to translate the SPARQL query Q_o to a SQL query Q_{sql} over D [106, 107]. Finally, Q_{sql} is evaluated on the database D, which gives us the answer to the initial query Q. See Figure 1.7.

[1]Part of this chapter has been published as: Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. 2014. OBDA: Query Rewriting or Materialization? In Practice, Both!. In Proceedings of the 13th International Semantic Web Conference (ISWC '14), Springer-Verlag New York, Inc., New York, NY, USA, 535-551. Marcelo Arenas and Daniel P. Miranker were advisors for this work.

This chapter presents Ultrawrap$^{\text{OBDA}}$, an extension to Ultrawrap (Chapter 4), which is an OBDA system comprising bidirectional evaluation. That is, a hybridization of query rewriting (backward chaining) and materialization (forward chaining).

The objective is to effect optimizations by pushing processing into the Relational Databases Management Systems (RDBMS) and closer to the stored data. Thus, maintaining a theme of making maximal use of existing SQL infrastructure. It is observed that by compiling the ontological entailments as mappings, implementing the mappings as SQL views and materializing a subset of the views, the underlying SQL optimizer is able to reduce the execution time of a SPARQL query by rewriting the query in terms of the views specified by the mappings. To the best of our knowledge, this is the first OBDA system supporting ontologies with transitivity by using SQL recursion.

5.1 Overview of Ultrawrap$^{\text{OBDA}}$

Similar to Ultrawrap (Chapter 4), two phases are distinguished: a compile phase and a runtime phase. In the compile phase, the following is given as input: a relational database D, an ontology $\mathcal{O}$ and a mapping $\mathcal{M}$ from D to $\mathcal{O}$. Consider the following relational database instance from the running example:

$$\{\texttt{EMP}(1, \texttt{Alice}, \texttt{CTO}, \texttt{NULL}),$$
$$\texttt{EMP}(2, \texttt{Bob}, \texttt{JavaProgrammer}, 1),$$
$$\texttt{EMP}(3, \texttt{John}, \texttt{SysAdmin}, 1)\}$$

and the following ontology:

$$\texttt{triple}(:\texttt{Programmer}, \texttt{subClass}, :\texttt{ITEmployee})$$

$$\texttt{triple}(:\texttt{SysAdmin}, \texttt{subClass}, :\texttt{ITEmployee})$$

The following is a mapping $\mathcal{M}$, from the relational database to the ontology. Section 5.2.1 describes the mapping in detail:

$$\texttt{EMP}(s, x_1, \texttt{Java}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Programmer} \rightarrow \texttt{triple}(s, p, o)$$

$$\texttt{EMP}(s, x_1, \texttt{SysAd}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:SysAdmin} \rightarrow \texttt{triple}(s, p, o)$$

The first step of this phase is to embed the ontological entailments of $\mathcal{O}$ in the mapping $\mathcal{M}$, which gives rise to a new mapping $\mathcal{M}^\star$. This new mapping, $\mathcal{M}^\star$, is called the *saturation* of $\mathcal{M}$ w.r.t. $\mathcal{O}$. The saturated mapping $\mathcal{M}^\star$ now includes the following, as described in Section 5.2.2 and 5.2.3 :

$$\texttt{EMP}(s, x_1, \texttt{Java}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:ITEmployee} \rightarrow \texttt{triple}(s, p, o)$$

$$\texttt{EMP}(s, x_1, \texttt{SysAd}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:ITEmployee} \rightarrow \texttt{triple}(s, p, o)$$

Subsequently, mapping $\mathcal{M}^\star$ is implemented using SQL views, following the approach of Ultrawrap (Chapter 4) and as further described in Section 5.2.4 .

```
CREATE VIEW ProgrammerView(S, P, O) AS
  SELECT SID as S, "rdf:type" as P, ":Programmer" as O
  FROM EMP WHERE JOB = "Java"

CREATE VIEW SysAdminView(S, P, O) AS
  SELECT SID as S, "rdf:type" as P, ":SysAdmin" as O
```

```
  FROM EMP WHERE JOB = "SysAdmin"

CREATE VIEW ITEmployeeView(S, P, O) AS
  SELECT SID as S, "rdf:type" as P, ":ITEmployee" as O
  FROM EMP WHERE JOB = "Java"
  UNION ALL
  SELECT SID as S, "rdf:type" as P, ":ITEmployee" as O
  FROM EMP WHERE JOB = "SysAdmin"
```

In order to improve query performance, an important issue is to decide which views should be materialized using the data in D, as discussed in Section 5.3. The hypothesis is that if a RDBMS rewrites queries in terms of materialized views, then optimal query performance is achieved by only materializing the views representing mappings to the leaf classes. In this case, the only views that need to be materialized are `ProgrammerView` and `SysAdminView`. This is the last step of the compilation phase.

In the runtime phase, the input is a query Q over the target ontology $\mathcal{O}$, which is written in SPARQL. The problem is to answer this query by rewriting it into a SQL query over the views. Consider the following SPARQL query which asks for all the IT Employees: `SELECT ?x WHERE {?x rdf:type :ITEmployee}`. The syntactic translation of the previous SPARQL query to a SQL query in terms of the views is: `SELECT t1.s AS x FROM ITEmployee t1`.

A key observation at this point is that existing SQL optimizers are able to perform rewritings in order to execute queries against materialized views. This means that the SQL optimizer is able to rewrite the original query, which is in terms of the `ITEmployee` un-materialized view into a new query in terms of the `ProgrammerView` and `SysAdminView` materialized views. Figure 5.1 illustrates the architecture of Ultrawrap$^{\text{OBDA}}$.

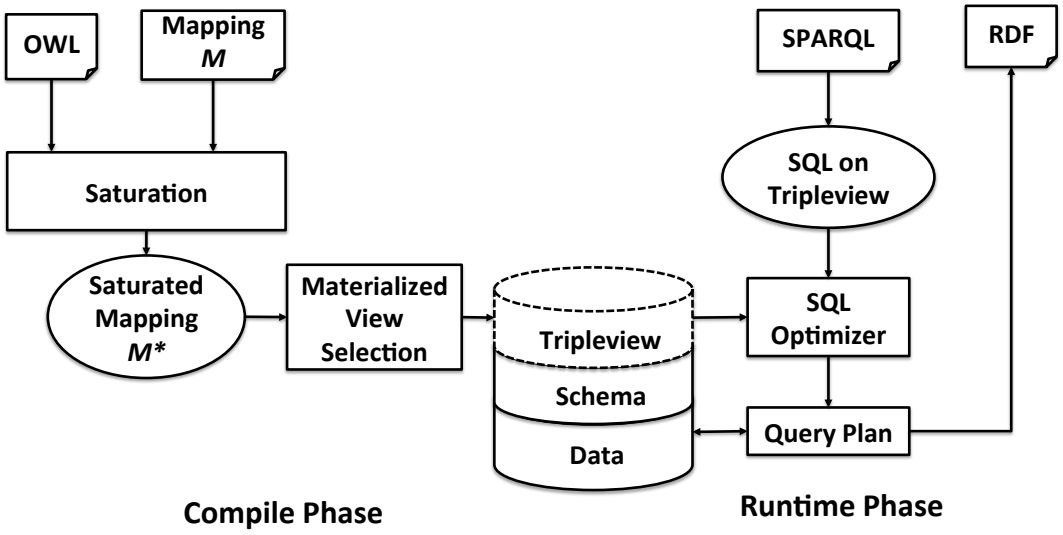


Figure 5.1: Ultrawrap$^{\text{OBDA}}$ Architecture

5.2 Mapping Relational Databases to RDF for OBDA

Mappings from relational databases to RDF are called RDB2RDF mappings. This section defines the notion of saturation of an RDB2RDF mapping w.r.t. an ontology, which plays a fundamental role in this approach. An efficient algorithm is defined to compute saturated mappings for ontologies not containing transitive predicates. The results are then extended to include transitive predicates. Finally, the implementation of RDB2RDF mappings as views in Ultrawrap$^{\text{OBDA}}$ is presented.

5.2.1 Relational databases to RDF mappings

Recall from the Preliminaries (Chapter 2) that **D** is a countably infinite domain of constants including the contents of a database. **I** is the infinite set of IRIs. **L** is the infinite set of literals. **O** is a set of reserved keywords used to define an ontology.

In an OBDA system, relational databases are mapped into RDF graphs, where every constant from **D** is transformed into either a IRI or a literal. This process is usually carried over by using some built-in transformation functions [13,

118]. For the sake of simplicity, assume that $\mathbf{D} = (\mathbf{I} \setminus \mathbf{O}) \cup \mathbf{L}$, which allows the use of the constants in a relational database directly as IRIs or literals in an RDF graph. Recall that `triple` is a ternary predicate that stores RDF graphs[2]. Notice that the keywords in $\mathbf{O}$ are not allowed in $\mathbf{D}$, as these are reserved for the specification of ontologies.

A mapping from a relation database to RDF is denoted as an RDB2RDF mapping. There are two related standards [13, 49]. Herein, an alternative approach is adopted which has been widely used in the data exchange [12] and data integration areas [90], and which is based on the use of first-order logic and its semantics to define mappings. Two types of first-order logic rules are relational database to RDF mappings are considered: for classes and predicates.

Definition 21 (Class RDB2RDF rule) *Given a relational schema* $\mathbf{R}$ *such that* $\texttt{triple} \notin \mathbf{R}$*, a* class RDB2RDF-rule ρ *over* $\mathbf{R}$ *is a first-order formula of the form:*

$$\forall s \forall p \forall o \forall \bar{x} \ \alpha(s, \bar{x}) \wedge p = \texttt{rdf:type} \wedge o = c \rightarrow \texttt{triple}(s, p, o), \tag{5.1}$$

where $\alpha(s, \bar{x})$ *is a first-order formula over* $\mathbf{R}$ *and* $c \in \mathbf{D}$*.*

Definition 22 (Predicate RDB2RDF rule) *Given a relational schema* $\mathbf{R}$ *such that* $\texttt{triple} \notin \mathbf{R}$*, a* predicate RDB2RDF-rule ρ *over* $\mathbf{R}$ *is a first-order formula of the form:*

$$\forall s \forall p \forall o \forall \bar{x} \ \beta(s, o, \bar{x}) \wedge p = c \rightarrow \texttt{triple}(s, p, o), \tag{5.2}$$

where $\beta(s, o, \bar{x})$ *is a first-order formula over* $\mathbf{R}$ *and* $c \in \mathbf{D}$*.*

Finally, an RDB2RDF-rule over $\mathbf{R}$ is either a class or a predicate RDB2RDF-rule over $\mathbf{R}$. In what follows, the universal quantifiers $\forall s \forall p \forall o \forall \bar{x}$ are omitted from

[2]See Section 2.2.1 for more details.

RDB2RDF rules. It is implicitly assumed that these variables are universally quantified.

Example 8 Consider the relation `EMP(EID,NAME,TYPE,MAN)`, the ontology class `:CTO` and the ontology property `foaf:name`. The following class RDB2RDF rule maps all the instances of the `EMP` table where `TYPE = "CTO"` as instances of the `:Executive` class:

$$\texttt{EMP}(s, x_1, \texttt{CTO}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Executive} \rightarrow \texttt{triple}(s, p, o) \tag{5.3}$$

The following predicate RDB2RDF rule maps all the instances of the `EMP` table and the corresponding value of the `NAME` attribute to the `foaf:name` property:

$$\texttt{EMP}(s, o, x_2, x_3) \wedge p = \texttt{foaf:name} \rightarrow \texttt{triple}(s, p, o) \tag{5.4}$$

Let $\mathbf{R}$ be a relational schema. An RDB2RDF mapping $\mathcal{M}$ over $\mathbf{R}$ is a finite set of RDB2RDF rules over $\mathbf{R}$. Given an RDB2RDF mapping $\mathcal{M}$ and an instance I over $\mathbf{R}$, the result of applying $\mathcal{M}$ over I, denoted by $[\![\mathcal{M}]\!]_I$, is an instance over the schema $\{\texttt{triple}\}$ that is defined as the result of the following process. For every RDB2RDF rule of the form (5.1) and value $c_1 \in \mathbf{D}$, if there exists a tuple of values $\bar{d}$ from $\mathbf{D}$ such that $I \models \alpha(c_1, \bar{d})$, then $\texttt{triple}(c_1, \texttt{rdf:type}, c)$ is included as a fact of $[\![\mathcal{M}]\!]_I$, and likewise for every RDB2RDF rule of the form (5.2). Notice that $[\![\mathcal{M}]\!]_I$ represents an RDF graph and, thus, mapping $\mathcal{M}$ is a mapping from relational databases into RDF graphs.

Example 9 Consider the relational database from the running example, and let $\mathcal{M}$ be an RDB2RDF mapping consisting of the rules (5.3) and (5.4) from Example 8.

Assume the following instance of the example schema:

$$I = \{\texttt{EMP}(\texttt{1}, \texttt{Alice}, \texttt{CTO}, \texttt{NULL}),$$
$$\texttt{EMP}(\texttt{2}, \texttt{Bob}, \texttt{JavaProgrammer}, \texttt{1}),$$
$$\texttt{EMP}(\texttt{3}, \texttt{John}, \texttt{SysAdmin}, \texttt{1})\}$$

then $[\![\mathcal{M}]\!]_I$ consists of the following RDF triples:

`triple(1, rdf:type, :Executive)`

`triple(1, foaf:name, "Alice")`

`triple(2, foaf:name, "Bob")`

`triple(3, foaf:name, "John")`

5.2.2 Saturation of RDB2RDF mappings

Extending an RDB2RDF mapping to embed a given ontology is a fundamental step in this approach. This process is formalized by the notion of a saturated mapping[3]. The intuition is the following.

Consider an instance I of a relational schema R, and ontology $\mathcal{O}$ and a mapping $\mathcal{M}$. $\mathcal{M}^\star$ is considered the saturated mapping of $\mathcal{M}$ if the ontology $\mathcal{O}$ is embedded in $\mathcal{M}^\star$. This means that if the saturated mapping $\mathcal{M}^\star$ is applied over the relational instance I, denoted as $[\![\mathcal{M}^\star]\!]_I$, then the resulting RDF graph would be the same as if the mapping $\mathcal{M}$ is applied over I, denoted as $[\![\mathcal{M}]\!]_I$ and then deriving new RDF triples using the ontology $\mathcal{O}$ in a forward chaining approach. The formal definition of saturated mapping is the following.

Definition 23 (Saturated mapping) *Let $\mathcal{M}$ and $\mathcal{M}^\star$ be RDB2RDF mappings over a relational schema $\mathbf{R}$ and $\mathcal{O}$ an ontology and where $\Sigma_\mathcal{O}$ is a set of first-order*

[3]The term "saturated" is inspired by the saturation step of the RQR algorithm from Perez-Urbina et. al [105].

formulae which encodes $\mathcal{O}$. Then $\mathcal{M}^\star$ is a saturation of $\mathcal{M}$ w.r.t. $\mathcal{O}$ if for every instance I over $\mathbf{R}$ and assertional RDF-triple (a, b, c):

$$[\![\mathcal{M}]\!]_I \cup \Sigma_\mathcal{O} \models \texttt{triple}(a, b, c) \quad \textit{iff} \quad \texttt{triple}(a, b, c) \in [\![\mathcal{M}^\star]\!]_I.$$

This section studies the problem of computing a saturated mapping from a given mapping and ontology. In particular, the focus is on non-transitive ontologies, i.e. ontologies not mentioning any triple of the form $(a, \texttt{rdf:type}, \texttt{transProp})$. In Section 5.2.3, these results are extended to the case of ontologies with transitivity.

In order to generate a saturated mapping, the idea is to create inference rules that generate new RDB2RDF rules from the existing ones in the input mapping and the input ontology. Table 5.1 presents inference rules for eight ontological constructs: subclass (`subClass`), subproperty (`subProp`), domain (`dom`), range (`range`), equivalent class (`equivClass`), equivalent property (`equivProp`), inverse property (`inverse`) and symettric property (`symProp`). The saturation step is performed by exhaustively applying the inference rules in Table 5.1.

Given an inference rule $t{:}\frac{\rho_1}{\rho_2}$ from Table 5.1, where t is a triple and ρ_1, ρ_2 are RDB2RDF rules, and given an RDB2RDF mapping $\mathcal{M}$ and an ontology $\mathcal{O}$, the following needs to be done to apply $t{:}\frac{\rho_1}{\rho_2}$ over $\mathcal{M}$ and $\mathcal{O}$. First, replace the letters `A` and `B` in t with actual URIs, say $a \in \mathbf{I}$ and $b \in \mathbf{I}$, respectively.[4] Second, check whether the triple obtained from t by replacing `A` by a and `B` by b belongs to $\mathcal{O}$, and whether the RDB2RDF rule obtained from ρ_1 by replacing `A` by a belongs to $\mathcal{M}$. If both conditions hold, then the inference rule can be applied, and the result is an RDB2RDF mapping $\mathcal{M}'$ consisting of the rules in $\mathcal{M}$ and the rule obtained from ρ_2 by replacing `A` by a and `B` by b.

Example 10 Consider the RDB2RDF rule (5.3) from Example 8 which maps all

[4] If $t = (\texttt{A}, \texttt{rdf:type}, \texttt{symProp})$, then only replace `A` by a.

$$(\texttt{A}, \texttt{subClass}, \texttt{B}) \quad : \quad \frac{\alpha(s,\bar{x}) \wedge p = \texttt{rdf:type} \wedge o = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\alpha(s,\bar{x}) \wedge p = \texttt{rdf:type} \wedge o = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$(\texttt{A}, \texttt{subProp}, \texttt{B}) \quad : \quad \frac{\beta(s,o,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\beta(s,o,\bar{x}) \wedge p = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$(\texttt{A}, \texttt{dom}, \texttt{B}) \quad : \quad \frac{\beta(s,o,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\beta(s,y,\bar{x}) \wedge p = \texttt{rdf:type} \wedge o = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$(\texttt{A}, \texttt{range}, \texttt{B}) \quad : \quad \frac{\beta(s,o,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\beta(y,s,\bar{x}) \wedge p = \texttt{rdf:type} \wedge o = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$\begin{array}{c}(\texttt{A}, \texttt{equivClass}, \texttt{B}) \\ \text{or } (\texttt{B}, \texttt{equivClass}, \texttt{A})\end{array} \quad : \quad \frac{\alpha(s,\bar{x}) \wedge p = \texttt{rdf:type} \wedge o = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\alpha(s,\bar{x}) \wedge p = \texttt{rdf:type} \wedge o = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$\begin{array}{c}(\texttt{A}, \texttt{equivProp}, \texttt{B}) \\ \text{or } (\texttt{B}, \texttt{equivProp}, \texttt{A})\end{array} \quad : \quad \frac{\beta(s,o,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\beta(s,o,\bar{x}) \wedge p = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$\begin{array}{c}(\texttt{A}, \texttt{inverse}, \texttt{B}) \\ \text{or } (\texttt{B}, \texttt{inverse}, \texttt{A})\end{array} \quad : \quad \frac{\beta(s,o,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\beta(o,s,\bar{x}) \wedge p = \texttt{B} \rightarrow \texttt{triple}(s,p,o)}$$

$$(\texttt{A}, \texttt{rdf:type}, \texttt{symProp}) \quad : \quad \frac{\beta(s,o,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}{\beta(o,s,\bar{x}) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}$$

Table 5.1: Inference rules to compute saturated mappings.

the instances of the EMP table where TYPE = "CTO" as instances of the :Executive class:

$$\texttt{EMP}(s, x_1, \texttt{CTO}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Executive} \rightarrow \texttt{triple}(s,p,o)$$

and assume the following ontology that states that all Executives are also Employees:

$$\mathcal{O} = \{\texttt{triple}(\texttt{:Executive}, \texttt{subClass}, \texttt{:Employee})\}$$

Then by applying the first inference rule in Table 5.1:

$$(\texttt{:Executive}, \texttt{subClass}, \texttt{:Employee}) \quad : \quad \frac{\texttt{EMP}(s, x_1, \texttt{CTO}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Executive} \rightarrow \texttt{triple}(s,p,o)}{\texttt{EMP}(s, x_1, \texttt{CTO}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Employee} \rightarrow \texttt{triple}(s,p,o)}$$

the following RDB2RDF rule is inferred:

$$\texttt{EMP}(s, x_1, \texttt{CTO}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Employee} \rightarrow \texttt{triple}(s, p, o)$$

which maps all the instances of the `EMP` table where `TYPE = "CTO"` as instances of the `:Employee` class.

Given an RDB2RDF mapping $\mathcal{M}$ and an ontology $\mathcal{O}$, $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ denotes the RDB2RDF mapping obtained from $\mathcal{M}$ and $\mathcal{O}$ by successively applying the inference rules in Table 5.1 until the mapping does not change. The following theorem shows that $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ is a saturation of $\mathcal{M}$ w.r.t. $\mathcal{O}$, which justifies its use in our system.

Theorem 6 *For every RDB2RDF mapping $\mathcal{M}$ and ontology $\mathcal{O}$ in RDFS, it holds that $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ is a saturation of $\mathcal{M}$ w.r.t. $\mathcal{O}$.*

Theorem 6 is a corollary of the fact that the first six rules in Table 5.1 encode the rules to infer assertional triples from an inference system for RDFS given by Munoz et. al. [101]. A natural question at this point is whether $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ can be computed efficiently. In this setting, the approach based on exhaustively applying the inference rules in Table 5.1 can be easily transformed into a polynomial time algorithm for this problem. However, if this transformation is done naïvely, then the resulting algorithm is not efficient. For this reason, an algorithm is presented to compute $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ that is linear in the size of the input RDB2RDF mapping $\mathcal{M}$ and ontology $\mathcal{O}$, which are denoted by $\|\mathcal{M}\|$ and $\|\mathcal{O}\|$, respectively.

Theorem 7 *There exists an algorithm that, given an RDB2RDF mapping $\mathcal{M}$ and a non-transitive ontology $\mathcal{O}$, computes $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ in time $O(\|\mathcal{M}\| \cdot \|\mathcal{O}\|)$.*

The following are the main ingredients of the algorithm in Theorem 7. Fix a mapping $\mathcal{M}$ and an ontology $\mathcal{O}$. In the first part, the algorithm transforms $\mathcal{O}$ into an instance $I_{\mathcal{O}}$ over the relational schema $\{\texttt{triple}\}$, which satisfies that for

every $(a, b, c) \in \mathcal{O}$: (1) if $b \in \{\texttt{subClass}, \texttt{subProp}, \texttt{dom}, \texttt{range}, \texttt{rdf:type}\}$, then $\texttt{triple}(a, b, c) \in I_{\mathcal{O}}$; (2) if $b = \texttt{equivClass}$, then $\texttt{triple}(a, \texttt{subClass}, c) \in I_{\mathcal{O}}$ and $\texttt{triple}(c, \texttt{subClass}, a) \in I_{\mathcal{O}}$; (3) if $b = \texttt{equivProp}$, then $\texttt{triple}(a, \texttt{subProp}, c) \in I_{\mathcal{O}}$ and $\texttt{triple}(c, \texttt{subProp}, a) \in I_{\mathcal{O}}$; and (4) if $b = \texttt{inverse}$, then $\texttt{triple}(a, \texttt{inverse}, c) \in I_{\mathcal{O}}$ and $\texttt{triple}(c, \texttt{inverse}, a) \in I_{\mathcal{O}}$. Obviously, $I_{\mathcal{O}}$ can be computed in time $O(\|\mathcal{O}\|)$.

In the second part, the algorithm transforms $\mathcal{M}$ as follows: into an instance $I_{\mathcal{M}}$ over a relational schema consisting of binary predicates $\texttt{F}_{\texttt{class}}$ $\texttt{F}_{\texttt{pred}}$, $\texttt{Ch}$, $\texttt{R}_{\texttt{s}}$ and $\texttt{R}_{\texttt{o}}$. First, for every class RDB2RDF-rule in $\mathcal{M}$ of the form (5.1), a fresh natural number m is assigned as an identifier of formula $\alpha(s, \bar{x})$, and then fact $\texttt{F}_{\texttt{class}}(m, c)$ is included in $I_{\mathcal{M}}$ (thus, $\texttt{F}_{\texttt{class}}$ is used to store the class RDB2RDF-rules in $\mathcal{M}$). Second, for every predicate RDB2RDF-rule in $\mathcal{M}$ of the form (5.2), a fresh natural number n is assigned as an identifier of formula $\beta(s, o, \bar{x})$, and then fact $\texttt{F}_{\texttt{pred}}(n, c)$ is included in $I_{\mathcal{M}}$ (thus, $\texttt{F}_{\texttt{pred}}$ is used to store the predicate RDB2RDF-rules in $\mathcal{M}$). Moreover, in this case fresh natural numbers k_1, k_2 and k_3 are assigned as identifiers of formulae $\beta(o, s, \bar{x})$, $\beta(s, y, \bar{x})$ and $\beta(y, s, \bar{x})$ (where y is a fresh variable), respectively, and then the facts $\texttt{Ch}(n, k_1)$, $\texttt{Ch}(k_1, n)$, $\texttt{R}_{\texttt{s}}(n, k_2)$ and $\texttt{R}_{\texttt{o}}(n, k_3)$ are included in $I_{\mathcal{M}}$ (thus, these predicates are used to store some syntactic modifications of formulae that are needed in the inference rules in Table 5.1). Finally, in this case fresh natural numbers ℓ_1 are ℓ_2 are assigned as identifiers of formulae $\beta(o, z, \bar{x})$ and $\beta(z, o, \bar{x})$ (where z is a fresh variable), respectively, and then the facts $\texttt{R}_{\texttt{s}}(k_1, \ell_1)$ and $\texttt{R}_{\texttt{o}}(k_1, \ell_2)$ are included in $I_{\mathcal{M}}$. It is easy to see that $I_{\mathcal{M}}$ can be computed in time $O(\|\mathcal{M}\|)$.

With all the previous terminology, the problem of computing $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ can be reduced to the problem of computing the minimal model of a Datalog program $\Pi_{\mathcal{M},\mathcal{O}}$, which consists of the facts in $(I_{\mathcal{O}} \cup I_{\mathcal{M}})$ together with the following set Δ of

rules representing the inference rules in Table 5.1:

$$\begin{aligned}
\texttt{triple}(X, \texttt{subClass}, Y), \texttt{F}_{\texttt{class}}(U, X) &\rightarrow \texttt{F}_{\texttt{class}}(U, Y) \\
\texttt{triple}(X, \texttt{subProp}, Y), \texttt{F}_{\texttt{pred}}(U, X) &\rightarrow \texttt{F}_{\texttt{pred}}(U, Y) \\
\texttt{triple}(X, \texttt{dom}, Y), \texttt{F}_{\texttt{pred}}(U, X), \texttt{R}_{\texttt{s}}(U, V) &\rightarrow \texttt{F}_{\texttt{class}}(V, Y) \\
\texttt{triple}(X, \texttt{range}, Y), \texttt{F}_{\texttt{pred}}(U, X), \texttt{R}_{\texttt{o}}(U, V) &\rightarrow \texttt{F}_{\texttt{class}}(V, Y) \\
\texttt{triple}(X, \texttt{inverse}, Y), \texttt{F}_{\texttt{pred}}(U, X), \texttt{Ch}(U, V) &\rightarrow \texttt{F}_{\texttt{pred}}(V, Y) \\
\texttt{triple}(X, \texttt{rdf:type}, \texttt{symProp}), \texttt{F}_{\texttt{pred}}(U, X), \texttt{Ch}(U, V) &\rightarrow \texttt{F}_{\texttt{pred}}(V, X),
\end{aligned}$$

where X, Y, U and V are variables. Notice that Δ is a fixed set of rules (it depends neither on $\mathcal{M}$ nor on $\mathcal{O}$), and also that Δ does not include rules for the keywords `equivClass` and `equivProp`, as these are represented in $I_{\mathcal{O}}$ by using the keywords `subClass` and `subProp`, respectively.

In order to compute the minimal model of $\Pi_{\mathcal{M},\mathcal{O}}$, the variables in the above rules are instantiated to generate a ground Datalog program $\Pi'_{\mathcal{M},\mathcal{O}}$ having the same minimal model as $\Pi_{\mathcal{M},\mathcal{O}}$. The key observation here is that $\Pi'_{\mathcal{M},\mathcal{O}}$ can be computed in time $O(\|\mathcal{M}\| \cdot \|\mathcal{O}\|)$, which proves Theorem 7 as the minimal model of a ground Datalog program can be computed in linear time [48] and the time needed to compute $(I_{\mathcal{O}} \cup I_{\mathcal{M}})$ is $O(\|\mathcal{M}\| + \|\mathcal{O}\|)$. More precisely, $\Pi'_{\mathcal{M},\mathcal{O}}$ is defined as $(I_{\mathcal{O}} \cup I_{\mathcal{M}}) \cup \Delta'$, where Δ' is generated from Δ as follows. For every fact $\texttt{triple}(a, b, c) \in I_{\mathcal{O}}$, look for the only rule in Δ where this fact can be applied, and replace this rule by a new one where X is replaced by a and Y is replaced by c (or just X is replaced by a if $b = \texttt{rdf:type}$ and $c = \texttt{symProp}$). For example, consider a triple $\texttt{triple}(a, \texttt{subClass}, c)$, then generate the rule $\texttt{triple}(a, \texttt{subClass}, b), \texttt{F}_{\texttt{class}}(U, a) \rightarrow \texttt{F}_{\texttt{class}}(U, b)$. Let Δ_1 be the result of this process. Given that the set of rules Δ is fixed, then Δ_1 can be computed in time $O(\|\mathcal{O}\|)$. Now for every rule ρ in Δ_1, the following is done to transform ρ into a ground rule. First replace the variable U in ρ by a value n in $I_{\mathcal{M}}$. If ρ

$$(\texttt{A}, \texttt{rdf:type}, \texttt{transProp}) \quad : \quad \frac{\{\beta_i(s,o,\bar{x}_i) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)\}_{i=1}^{k}}{\mathrm{TC}_{[\bigvee_{i=1}^{k} \exists \bar{x}_i \beta_i]}(s,o) \wedge p = \texttt{A} \rightarrow \texttt{triple}(s,p,o)}.$$

Table 5.2: Inference rule to compute saturated mappings for transitivity

also contains a variable V, then notice that there exists at most one value m in $I_{\mathcal{M}}$ for which the antecedent of the rule could hold, as there exists at most one value m such that $\texttt{R}_\texttt{s}(n,m)$ holds, and likewise for predicates $\texttt{R}_\texttt{o}$ and $\texttt{Ch}$. Thus, in this case replace variable V in ρ for such a value m to generate a ground Datalog rule. We conclude that the resulting set Δ' of ground Datalog rules is computed in time $O(\|\mathcal{M}\| \cdot \|\mathcal{O}\|)$ (given that the size of Δ_1 is $O(\|\mathcal{O}\|)$). This concludes the sketch of the proof of Theorem 7.

5.2.3 Dealing with transitive predicates

This section presents how the approach presented in the previous section can be extended with recursive predicates. This functionality is of particular interest as the current work on OBDA does not consider transitivity, mainly because the query language considered in that work is SQL without recursion [35]. From now on, given a first-order formula $\varphi(x,y)$, $\mathrm{TC}_{\varphi}(x,y)$ is used to denote the transitive closure of $\varphi(x,y)$. This formula can be written in many different formalisms. For example, if $\varphi(x,y)$ is a conjunction of relational atoms, then $\mathrm{TC}_{\varphi}(x,y)$ can be written as follows in Datalog:

$$\varphi(x,y) \rightarrow \mathrm{TC}_{\varphi}(x,y), \qquad \varphi(x,z), \mathrm{TC}_{\varphi}(z,y) \rightarrow \mathrm{TC}_{\varphi}(x,y).$$

In Ultrawrap$^{\mathrm{OBDA}}$, $\mathrm{TC}_{\varphi}(x,y)$ is written as an SQL query with recursion. Then to deal with an ontology $\mathcal{O}$ containing transitive predicates, the set of inference rules in Table 5.1 is extended with the inference rule presented in Table 5.2.

The rule in Table 5.2 states that given a transitive predicate $\mathtt{A}$, any number k of RDB2RDF rules of the form $\beta_i(s, o, \bar{x}_i) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s, p, o)$ can be considered for this predicate. Subsequently, a new RDB2RDF rule can be generated for $\mathtt{A}$ by putting together the conditions $\beta_i(s, o, \bar{x}_i)$ in a formula $\gamma(s, o) = \bigvee_i \exists \bar{x}_i \beta_i(s, o, \bar{x}_i)$, and then using the transitive closure $\mathrm{TC}_\gamma(s, o)$ of γ in an RDB2RDF rule $\mathrm{TC}_\gamma(s, o) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s, p, o)$. In order for this approach to work, notice that the syntax of RDB2RDF rules (5.1) and (5.2) needs to be extended, so that formulae α and β in them can be arbitrary formulae in a more expressive formalism such as (recursive) Datalog.

Example 11 Consider the following RDB2RDF rule:

$$\mathtt{EMP}(s, x_1, x_2, o) \wedge p = \mathtt{:hasSuperior} \rightarrow \mathtt{triple}(s, p, o)$$

and assume the following ontology:

$$\mathcal{O} = \{\mathtt{triple}(\mathtt{:hasSuperior}, \mathtt{rdf:type}, \mathtt{transProp})\}$$

Then by applying the transitivity inference rule in Table 5.2

$$(\mathtt{:hasSuperior}, \mathtt{rdf:type}, \mathtt{transProp}) \quad : \quad \frac{\mathtt{EMP}(s, x_1, x_2, o) \wedge p = \mathtt{:hasSuperior} \rightarrow \mathtt{triple}(s, p, o)}{\mathrm{TC}_{\mathtt{EMP}}(s, x_1, x_2, o) \wedge p = \mathtt{:hasSuperior} \rightarrow \mathtt{triple}(s, p, o)}$$

the following RDB2RDF rule is inferred:

$$\mathrm{TC}_{\mathtt{EMP}}(s, x_1, x_2, o) \wedge p = \mathtt{:hasSuperior} \rightarrow \mathtt{triple}(s, p, o) \tag{5.5}$$

5.2.4 Implementing RDB2RDF mappings as views

Inspired by our previous work on Ultrawrap (Chapter 4), RDB2RDF mappings are represented as SQL views. A first step is to represent each RDB2RDF rule as a SQL query, called the Triplequery. Finally, a Tripleview is the union of Triplequeries.

Definition 24 (Triplequery) *Given a* RDB2RDF-rule ρ*, a Triplequery implements* ρ *as a SQL query which outputs triples.*

Example 12 Consider the following RDB2RDF rules:

$$\texttt{EMP}(s, x_1, \texttt{CTO}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Employee} \rightarrow \texttt{triple}(s, p, o)$$
$$\texttt{EMP}(s, x_1, \texttt{SysAdmin}, x_3) \wedge p = \texttt{rdf:type} \wedge o = \texttt{:Employee} \rightarrow \texttt{triple}(s, p, o)$$

The corresponding Triplequeries are the following:

```
SELECT SID as S, "rdf:type" as P, ":Employee" as O
FROM EMP WHERE JOB = "CTO"
SELECT SID as S, "rdf:type" as P, ":Employee" as O
FROM EMP WHERE JOB = "SysAdmin"
```

Example 13 Consider the RDB2RDF rule (5.5) of Example 11:

$$\mathrm{TC}_{\texttt{EMP}}(s, x_1, x_2, o) \wedge p = \texttt{:hasSuperior} \rightarrow \texttt{triple}(s, p, o)$$

The corresponding Triplequery using SQL recursion is the following:

```
SELECT X as S, ":hasSuperior" as P, Y as O FROM
(WITH TCEMP(X, Y) AS (
  SELECT EID, MAN FROM EMP
  UNION ALL
```

```
  SELECT EMP.EID, TCEMP.Y FROM EMP, TCEMP WHERE EMP.MAN = TCEMP.X)
SELECT X, Y FROM TCEMP)
```

In practice, the Triplequeries include additional projections in order to support indexes, URI templates, datatypes and languages. However, for readability, this simple version of these queries is considered (the reader is referred to Chapter 4 for specific details).

To implement an RDB2RDF mapping, all the class (resp. predicate) RDB2RDF-rules for the same class (resp. predicate) are grouped together to generate a SQL view called the Tripleview.

Definition 25 (Tripleview) *A Tripleview is a SQL view comprising the union of the Triplequeries for the same class (resp. predicate).*

Example 14 The Tripleview for the class `:Employee` using the Triplequeries of Example 12 is the following:

```
CREATE VIEW EmployeeView(S, P, O) AS
  SELECT SID as S, "rdf:type" as P, ":Employee" as O
  FROM EMP WHERE JOB = "CTO"
  UNION ALL
  SELECT SID as S, "rdf:type" as P, ":Employee" as O
  FROM EMP WHERE JOB = "SysAdmin"
```

Example 15 The Tripleview for the transitive predicate `:hasSuperior` using the Triplequery of Example 13 is the following:

```
CREATE VIEW HasSuperiorView(S, P, O) AS
  SELECT X as S, ":hasSuperior" as P, Y as O FROM
   (WITH TCEMP(X, Y) AS (
```

```
    SELECT EID, MAN FROM EMP
    UNION ALL
    SELECT EMP.EID, TCEMP.Y FROM EMP, TCEMP WHERE EMP.MAN = TCEMP.X)
SELECT X, Y FROM TCEMP)
```

5.3 Executing SPARQL Queries

This section describes how SPARQL queries are executed and optimized over the RDBMS through a cost model that determines which views should be materialized.

5.3.1 SPARQL rewriting

The runtime phase executes SPARQL queries on the RDBMS. Ultrawrap's approach is reused. SPARQL queries are translated to SQL queries in terms of the views defined for every class and property, which are denoted as Tripleviews (see Section 5.2.4).

Continuing with the example in Section 5.2.4, consider now a SPARQL query which asks for all the Employees: `SELECT ?x WHERE {?x rdf:type :Employee}`. It is clear that this query needs to be rewritten to ask for the CTO and SysAdmin. The `EmployeeView` Tripleview in Section 5.2.4 implements the mappings to the Employee class which consists of two Triplequeries, one each for CTO and SysAdmin. Therefore, it is sufficient to generate a SQL query in terms of the `EmployeeView`. Given that a Tripleview models a table with three columns, a SPARQL query is syntactically translated to a SQL query in terms of the Tripleview. The resulting SQL query is `SELECT t1.s AS x FROM EmployeeView t1`.

A natural question at this point is whether every SPARQL query has an equivalent SQL query in this context, where RDB2RDF mappings play a fundamental role. In what follows a positive answer is given to this question.

First some terminology is presented. Recall that $[\![P]\!]_G$ denotes the answer to a SPARQL query P over an RDF graph G, which consists of a set of solution mappings, that is, a set of functions that assign a value to each selected variable in P. For example, if P is the SPARQL query `SELECT ?x WHERE {?x rdf:type ?y}`, and G is an RDF graph consisting of the triples (`1, rdf:type, :Employee`) and (`2, rdf:type, :Employee`), then $[\![P]\!]_G = \{\mu_1, \mu_2\}$, where μ_1 and μ_2 are functions with domain {?x} such that μ_1(?x) = `1` and μ_2(?x) = `2`. Moreover, given a SQL query Q (that may use recursion) over a relational schema $\mathbf{R}$ and an instance I of $\mathbf{R}$, the notation $[\![Q]\!]_I$ is used to represent the answer of Q over I, which consists of a set of tuples in this case. Finally, to compare the answer of a SQL query with the answer of a SPARQL query, the function *tr* is used to transform a tuple into a solution mapping (this function is defined in Section 3.2.2).

Given an RDB2RDF mapping $\mathcal{M}$ over a relational schema $\mathbf{R}$ and a SPARQL query P, an SQL query Q over $\mathbf{R}$ is said to be a SQL-rewriting of P under $\mathcal{M}$ if for every instance I of $\mathbf{R}$, it holds that $[\![P]\!]_{[\![\mathcal{M}]\!]_I} = tr([\![Q]\!]_I)$. Moreover, P is said to be SQL-rewritable under $\mathcal{M}$ if there exists a rewriting of P under $\mathcal{M}$.

Theorem 8 *Given an RDB2RDF mapping $\mathcal{M}$, every SPARQL query is SQL-rewritable under $\mathcal{M}$.*

The proof that the previous condition holds is by induction on the structure of a SPARQL query P and, thus, presents a (naïve) bottom-up algorithm for translating P into an equivalent SQL query Q (given the mapping $\mathcal{M}$). The base case is a single a triple pattern t = {`s p o`}, where each one of its component is either a IRI or a literal or a variable. This triple pattern is first translated into a SPARQL query P_t, where each position in t storing a IRI or a literal is replaced by a fresh variable. Then a filter condition is added to ensure that these fresh variables are assigned the corresponding IRIs or literals. Finally, a SELECT clause is added to ensure that the output variables of t and P_t are the same. For example, if

$t = \{?\texttt{x}\ \texttt{rdf:type}\ \texttt{Employee}\}$, then P_t is the following SPARQL query:

$\texttt{SELECT}\ ?\texttt{x}\ \ \texttt{WHERE}\ \ \{?\texttt{x}\ ?\texttt{y}\ ?\texttt{z}\}\ \texttt{FILTER}\ (?\texttt{y} = \texttt{rdf:type}\ \&\&\ ?\texttt{z} = \texttt{:Employee}).$

Then a SQL-rewriting of P_t under $\mathcal{M}$ is computed just by replacing a triple pattern of the form $\{?\texttt{s}\ ?\texttt{p}\ ?\texttt{o}\}$ by a union of all the Triplequeries representing the RDB2RDF rules in $\mathcal{M}$, and also replacing the SPARQL filter condition in P_t by a filter condition in SQL.

In the inductive step, assume that the theorem holds for two SPARQL queries P_1 and P_2. The proof then continues by presenting rewritings for the SPARQL queries constructed by combining P_1 and P_2 through the operators SELECT, AND (or '.' operator), OPTIONAL, FILTER and UNION, which is done by using existing approaches to translate SPARQL to SQL [9, 40].

5.3.2 Cost Model for View Materialization

A common approach for query optimization is to use materialized views [70]. Given that RDB2RDF mappings are implemented as views, it is a natural to pursue this option. There are three implementation alternatives:

1. **Materialize all the views**: This approach gives the best query response time. However, it consumes the most space.
2. **Materialize nothing**: In this approach, every query needs to read the base tables. However, no extra space is needed.
3. **Materialize a subset of the views**: Try to find a trade-off between the best query response time and the amount of space required.

This section presents a cost model for evaluating these three models[5]. First some terminology. The ontologies being considered consist of a hierarchy of classes

[5]Addressing updates to the database is future work.

which form a tree. A root class of an ontology is a class that has no superclasses. A leaf class of an ontology is a class that has no subclasses. The depth of a class is the number of subclass relationships from the class to the root class (notice that there is a unique path from a class to the root class). Moreover, the maximum depth of an ontology is maximum depth of all class present in the ontology.

First, assume that the cost of answering a query Q is equal to the number of rows present in the relation used to construct Q. For example, if a relation R has 100 rows, then the cost of the query `SELECT * FROM R` is 100. Second, assume a single relation R and that mappings are from a query on the relation R with a selection on an attribute A, to a class in the ontology. In Example 9, the relation R is `EMP`, the attribute A is `JOB` and the mapping is to the class `Executive`. Finally, consider a query workload of queries asking for the instances of a class in the ontology, i.e. `SELECT ?x WHERE {?x rdf:type A}`, which can be translated into the Tripleview implementing the mapping to the class `A`. We will build up to the cost model through a set of scenarios and observations.

Scenario 1: Consider a relation R with 100 rows and an attribute X which has two possible distinct values: A' or B'. The ontology consists of (`A`, `subClass`, `C`) and (`B`, `subClass`, `C`). Assuming uniformity, 50 rows of R are A and the other 50 rows are B. The Tripleviews are the following:

```
CREATE VIEW Aview SELECT id as S, "rdf:type" as P, "A" as O
FROM R WHERE X = 'A'

CREATE VIEW Bview SELECT id as S, "rdf:type" as P, "B" as O
FROM R WHERE X = 'B'
```

Consider the query Q = `SELECT ?x WHERE {?x rdf:type A}`. If the views are not materialized and there is no index on attribute X, then the cost of Q is equivalent to

scanning the entire relation R, which is 100 (recall that R has 100 rows). However, if the Tripleview is materialized, then the Tripleview will have 50 rows, hence the cost of Q is 50. The same for the Tripleview representing the mapping to the class `B`. The `A` and `B` class are the minimal classes in the ontology, i.e. these classes do not have children. Therefore the sum of all the rows in the `Aview` and `Bview` is 100, which is the same number of rows in the original R relation.

Observation 1: Given a set of Tripleviews, representing mappings from a relation to each minimal class of an ontology, the sum of all the rows in the set of Tripleviews is equivalent to the number of rows in the relation.

Scenario 2: Consider a relation R with 100 rows and an attribute X which has four possible distinct values: A', B', C' and D'. The ontology consists of (`A`, `subClass`, `E`), (`B`, `subClass`, `E`), (`C`, `subClass`, `F`), (`D`, `subClass`, `F`), (`E`, `subClass`, `G`), (`F`, `subClass`, `G`) (`G` is the root). Assuming uniformity, 25 rows of R are A, 25 rows are B, and so on. The Tripleviews representing the mappings to classes `A`, `B`, `C` and `D` are equivalent to the Tripleview in Scenario 1. The additional Tripleviews are:

```
CREATE VIEW Eview
SELECT id as S, "rdf:type" as P, "E" as O FROM R WHERE X = "A"
UNION ALL
SELECT id as S, "rdf:type" as P, "E" as O FROM R WHERE X = "B"

CREATE VIEW Fview
SELECT id as S, "rdf:type" as P, "F" as O FROM R WHERE X = "C"
UNION ALL
SELECT id as S, "rdf:type" as P, "F" as O FROM R WHERE X = "D"
```

It is observed that for each depth of the ontology, the sum of all the rows of each node in that depth is equal to the number of rows in the relation. In other words, classes `E` and `F` are in depth 1. `Eview` consists of a union of a the rows where X is A' or B', for a total of 50 rows (25 rows in A' and 25 rows in B'). `Fview` consists of a union of the rows where X is C' or D', for another total of 50. The grand total is 100, the same number of rows of R.

Observation 2: The sum of all the rows of each Tripleview representing the mapping to classes in depth d of an ontology, is equivalent to the number of rows of the relation.

Scenario 3: Consider the Tripleview `Eview` which consists of a union of two Triplequeries. Each of these Triplequeries represents the mapping rule for the child classes of `E`, namely the mappings to the classes `A` and `B`. Additionally, `Eview` could be rewritten in terms of the child Tripleviews:

```
CREATE VIEW Eview
SELECT s, p, o FROM Aview UNION ALL
SELECT s, p, o FROM Bview
```

Observation 3: A Tripleview representing a mapping to a class, can be rewritten into the union of Tripleviews representing the mapping to the child classes.

Scenario 4: Consider the Tripleview representing the mapping for the class `G`:

```
CREATE VIEW GView
SELECT s, p, o FROM EView UNION ALL
SELECT s, p, o FROM Fview
```

Observe that `EView` and `Fview` can also be rewritten in terms of their children, hence:

```
CREATE VIEW GView
SELECT s, p, o FROM Aview UNION ALL
SELECT s, p, o FROM Bview UNION ALL
SELECT s, p, o FROM Cview UNION ALL
SELECT s, p, o FROM Dview
```

Observation 4 A Tripleview representing the mapping to any class in the ontology can be rewritten into a union of Tripleviews representing the mappings to the minimal classes of an ontology

Scenario 5: Consider the case when there is no materialization and a query Q asking for the instances of a class C in an ontology. The query Q is equivalent to the Tripleview representing the mapping to the class C. Per Observation 4, Q can be rewritten into a union of the Tripleviews representing the mappings to the minimal classes underneath C. Each view must scan all the rows in R. Hence the cost of $Q = n * N_R$ where n is the number of minimal classes underneath C. However, what happens when the view representing the mapping of C is materialized? How many rows are going to be in that view? In Scenario 2, the attribute X had 4 distinct values: A', B', C' and D'. Assuming uniformity, the selectivity is 25%. That means that each Tripleview for `A`, `B`, `C` and `D` has 25% of the total number of rows of R. Consider now the case when all the views are materialized: The cost of executing a query Q is equivalent to the number of rows in the Tripleview which is equal to the sum of all the rows for all leaf nodes that are underneath Q. Therefore the cost of $Q = n * N_R * S(A, R)$ where $S(A, R)$ denotes the selectivity of the attribute A in the relation R.

The Cost Model

The cost model is the following: If all the views implementing mappings are materialized, the query cost is $n \times N_R \times S(A, R)$ where n is the number of leaf classes underneath the class that is being queried for, N_R is the number of tuples of the relation R in the mapping, and $S(A, R)$ is the selectivity of the attribute A of the relation R in the mapping. The space cost is $N_R + (N_R \times d)$ where d is the maximum depth of the ontology. The reason for this cost is the number of rows in a materialized view depends on the selectivity of the attribute and the number of leaf classes. Additionally, the sum of all the rows of each Tripleview representing the mapping to classes in a particular depth d of an ontology, is equivalent to the number of rows of the relation at most. If no views are materialized, then the query cost is $n \times N_R$, assuming there are no indices. The space cost is simply N_R. The reason for this cost is because to answer a query, the entire relation needs to be accessed n times because there are no indices[6].

The question now is: Can we achieve the query cost of materializing all the views while keeping space to a minimum? The hypothesis is the following: If a RDBMS rewrites queries in terms of materialized views, then by only materializing the views representing mappings to the leaf classes, the query cost is $n \times N_R \times S(A, R)$, the same as if all the views were materialized, and the space cost would only be $2 \times N_R$. The rationale is the following: A Tripleview representing a mapping to a class, can be rewritten into the union of Tripleviews representing the mapping to the child classes. Subsequently, a Tripleview representing the mapping to any class in the ontology can be rewritten into a union of Tripleviews representing the mappings to leaf classes of an ontology. Finally, given a set of Tripleviews representing mappings from a relation to each leaf class of an ontology, the sum of all the rows in the set of Tripleviews is equivalent to the number of rows in the relation.

[6] In the evaluation, we consider the case when indices are present.

Given the extensive research of answering queries using views [71] and the fact that Oracle implements query rewriting on materialized views[7], we strongly suspect that our hypothesis will hold. The following evaluation section supports our hypothesis.

5.4 Evaluation

5.4.1 Benchmarks

The evaluation requires benchmarks consisting of a relational database schema and data, ontologies, mappings from the database to ontologies and a query workload. Thus, we created a synthetic benchmark, the *Texas Benchmark*, inspired by the Wisconsin Benchmark [51]. Additionally, the Berlin SPARQL Benchmark (BSBM) [23] was extended.

Texas Benchmark

The Texas Benchmark is composed of a single relation with 1 million rows, reminiscent of the Wisconsin Benchmark[51] and ontologies with subclasses. The relation has a first attribute which serves as a primary key, a set of additional filler attributes in order to take up space and then a set of six different integer-valued attributes which are non-unique. The main purpose of these attributes is to provide a systematic way to model a wide range of selectivity factors. Each attribute is named after the range of values the attribute assumes: `TWO`, `FIVE`, `TEN`, `TWENTY`, `FIFTHY` and `HUNDRED`. For example, the attribute `FIVE` assumes a range of values from 1 to 5. Thus, the selection `FIVE` = `1` will have a 20% selectivity. In addition to the data, we created five different ontologies, consisting of a maximum depth between 2-5. The branching factor is uniform and the number of leaves is 100 for each ontology.

[7] http://docs.oracle.com/cd/B28359_01/server.111/b28313/qrbasic.htm

The query workload consists of asking for an instance of a class at each depth of the ontology.

Extension of BSBM

BSBM replicates the query workload of an e-commerce website. Products have a type that is part of a ProductType hierarchy. In the original benchmark, the ProductType hierarchy is implicit in the data through a parent-child relationship. The extension of BSBM adds a transitive property to the ontology which is mapped to the ProductType's parent-child relationship. Additionally, the ProductType hierarchy was made explicit as an ontology with subclasses. Every product is mapped to one leaf class of the ProductType ontology. In the experiments, a dataset consisting of 1 million products was created with the benchmark driver, hence the product table has 1 million rows. The resulting ProductType ontology has a maximum depth of 4 and consists of 3949 classes from which 3072 are leaf-level classes. The selectivity of the attribute in the mappings to ProductTypes is approximately 0.1%. To be consistent with the results of the Texas Benchmark, the query workload also consists of asking for an instances of a class at each depth of the ontology. The query workload for the transitivity part consists of asking for Products of a particular ProductType including the label and a numeric property of the Products, therefore including joins. More details about the benchmarks can be found in the Appendix.

5.4.2 Measurements and Scenarios

The objective of the experiments is to observe the behavior of a commercial relational databases, namely Oracle, and its capabilities of supporting subclass and transitivity reasoning under the proposed approach. Therefore, the evaluation compares execution time and queries plans of SPARQL queries. With the Texas Benchmark, a comparison was made w.r.t. two dimensions: depth of an ontology and selectivity

	ontop	or index	or leaves	union index	union leaves	all mat
D6_100	7.31	7.37	7.44	5.28	5.42	4.80
D6_50	7.44	7.55	7.54	5.14	5.76	4.92
D6_20	9.76	9.88	9.88	6.28	5.71	5.26
D6_10	10.51	10.74	10.43	7.20	6.90	6.37
D6_5	17.24	17.26	10.97	19.75	13.42	9.09
D6_2	22.54	22.88	20.36	23.21	21.17	20.18
D5_100	5.38	5.39	5.42	4.97	4.38	3.85
D5_50	5.56	5.84	5.61	4.92	4.52	4.02
D5_20	9.01	8.52	8.41	6.16	5.47	5.05
D5_10	9.86	9.30	9.04	7.20	6.52	6.13
D5_5	17.32	17.80	9.71	20.24	10.98	9.32
D5_2	21.80	25.28	19.33	23.27	19.67	20.13
D4_100	5.63	5.88	5.81	4.92	4.59	3.96
D4_50	5.83	6.08	6.25	5.10	4.82	4.18
D4_20	6.28	6.56	6.46	5.51	5.34	5.00
D4_10	7.43	7.76	7.52	6.47	6.10	5.80
D4_5	17.54	17.60	10.84	20.96	11.89	9.74
D4_2	22.33	23.30	20.25	23.40	19.96	19.88
D3_100	6.21	7.36	6.48	3.80	3.92	3.53
D3_50	6.40	6.64	10.15	4.09	3.76	3.54
D3_20	6.93	7.15	6.94	4.41	4.25	3.90
D3_10	8.30	8.47	8.08	5.91	5.38	5.14
D3_5	16.55	16.93	7.85	16.80	7.61	7.58
D3_2	23.61	23.43	19.91	22.78	20.09	19.98
D2_100	1.13	1.28	1.24	1.23	1.10	1.05
D2_50	1.41	1.50	1.48	1.45	1.28	1.30
D2_20	2.23	2.40	2.20	2.23	1.99	1.99
D2_10	4.22	4.42	3.99	4.26	3.91	3.81
D2_5	16.39	16.98	10.47	16.77	7.67	8.14
D2_2	23.43	23.19	20.24	22.66	19.59	20.18
Average	10.85	11.16	9.34	10.21	8.11	7.59

Figure 5.2: Results of Texas Benchmark (sec)

of the attribute that is being mapped. In BSBM, given the fixed 1 million product dataset, the depth of the hierarchy and selectivity is also fixed. Six scenarios were considered in the evaluation:

- **all-mat**: all the views are materialized
- **union-leaves**: only views representing mappings to the leaf classes are materialized, implemented with UNION
- **or-leaves**: same as in the previous scenario but with the views implemented with OR instead of UNION
- **union-index**: none of the views, implemented with UNION, are materialized, instead an index on the respective attributes have been added
- **or-index**: same as in the previous scenario but with the views implemented with OR
- **ontop**: a comparison against Ontop, a state of the art OBDA system [114].

The only competing system was Ontop because to the best of our knowledge, this is the only OBDA system that supports RDB2RDF mappings and SPARQL.

The experiments were conducted on Oracle 11g R2 EE installed on a Sun Fire X4150 with a four core Intel Xeon X7460 2.66 GHz processor and 16 GB of RAM, running Microsoft Windows Server 2008 R2 Standard on top of VMWare ESX 4.0.

5.4.3 Results

An initial assessment supports the following four expected observations:

1. The fastest execution time is *all-mat*

2. The hypothesis holds, meaning that the execution time of *union-leaves* should be comparable, if not equal, to the execution time of *all-mat*;

3. Given that the Ontop system generates SQL queries with OR instead of UNION [114], the execution time of *ontop* and *or-index* should be comparable if not equal

4. With transitivity, the fastest execution time is when the views are materialized.

Figure 5.2 shows the results of the Texas Benchmark in a form of a heat map, which evaluates subclass reasoning. The darker colors corresponds to the fastest query execution time. The x-axis consists of the six scenarios. In the y-axis, D6_100 means Depth 6 on Selectivity of 100. The values are the average execution time of the query workload. Notice that the expected observations (1), (2) and (3) hold. The fastest execution time corresponds to *all-mat*. The execution time of *union-leaves* is comparable, if not equal, to the execution time of *all-mat*, because Oracle was able to rewrite queries in terms of the materialized views as shown in physical query plan in Figure 5.5. The number of rows examined is equivalent to the number of rows in the views where everything was materialized. This result provides evidence supporting the hypothesis and validates the proposed cost model. Finally the execution time of *ontop* and *or-index* are comparable.

Figure 5.3 shows the results of the BSBM Benchmark for subclass reasoning. The expected observations also hold in this case. Note that results are not reported for Ontop because the setup of the SPARQL endpoint timed-out after 2 hours[8]. Given that the selectivity is much lower compared to the selectivities in the Texas Benchmark, it is observed that for queries asking for instances of classes that are in depth 1 (child of the root Class), the or-index outperforms union-leaves. A hypothesis is that there is a slight overhead when rewriting queries over a large

[8]The issue was reported to the Ontop developers.

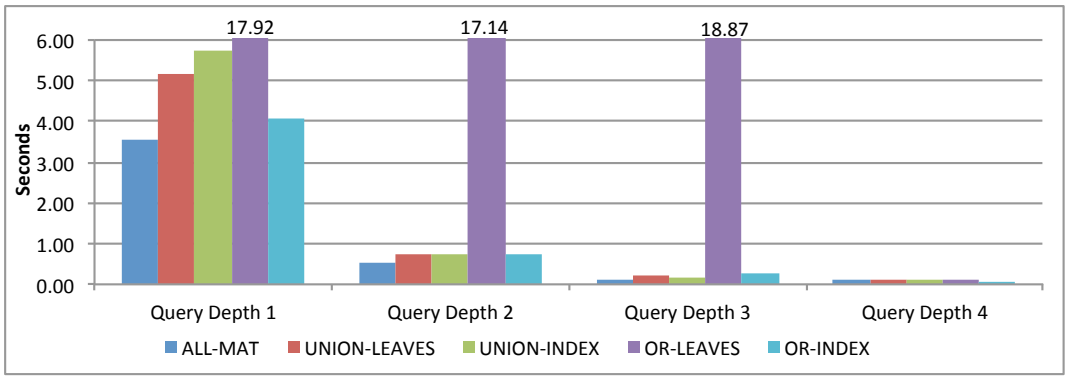


Figure 5.3: Results of Subclass reasoning on BSBM

number of views. However, for the rest of the queries, the overhead diminishes. It is observed that the execution time of or-leaves is the worst because the database is not able to rewrite the query into the materialized views when the views are implemented with OR. Finally, throughout both benchmarks, it is observed that *or-index* is competitive w.r.t *union-leaves*.

Figure 5.4 shows the results of the transitivity experiments on the BSBM Benchmark. Notice that the expected observations (4) holds. Given that Ontop does not support transitivity, it is not possible to compare with them. Therefore the only comparison is between materialized and non-materialized views. The Simple query requests all the ancestors of the given ProductType. The Join query requests all ancestors of the given ProductType and its corresponding Products. Therefore there is a join between ProductType and Product. The More Join query is similar to Join query, however it requests the name and a numeric property of the products, hence there are more joins. It is clear that materializing the view outperforms the non-materialized view for the following reasons: when the view is materialized, the size of the view is known beforehand and the optimizer is able to do a range scan with the index, as shown in the physical query plan in Figure 5.6. However, when the view is not materialized, the size is not known therefore the optimizer does a

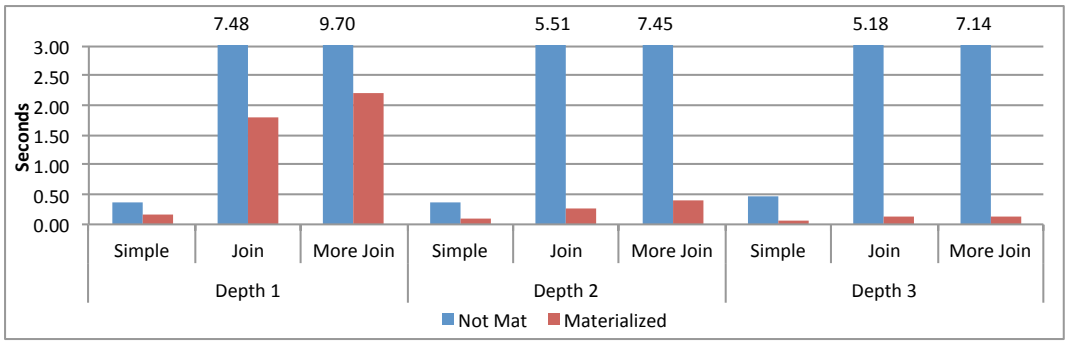


Figure 5.4: Results of Transitivity reasoning on BSBM

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			23
VIEW			23
UNION-ALL			
MAT_VIEW REWRITE ACCESS	PRODUCTTYPE2267	FULL	6
MAT_VIEW REWRITE ACCESS	PRODUCTTYPE2269	FULL	5
MAT_VIEW REWRITE ACCESS	PRODUCTTYPE2266	FULL	6
MAT_VIEW REWRITE ACCESS	PRODUCTTYPE2268	FULL	6

Figure 5.5: Oracle Query Plan for Rewriting using Materialized Views

full scan of the table, as shown in the physical query plan in Figure 5.7.

5.5 Related Work

This research builds upon the work of Rodriguez-Muro et. al. [113, 114] and our previous work on Ultrawrap (Chapter 4). Rodriguez-Muro et. al. introduced the idea of compiling ontological entailments as mappings, which they named $\mathcal{T}$-Mappings. There are three key differences between Rodriguez-Muro et. al. and our work in this paper:

1. extension of the work of Rodriguez-Muro et. al. to support more than hierarchy of classes and properties, including transitivity;

2. introduction of an efficient algorithm that generates saturated mappings while

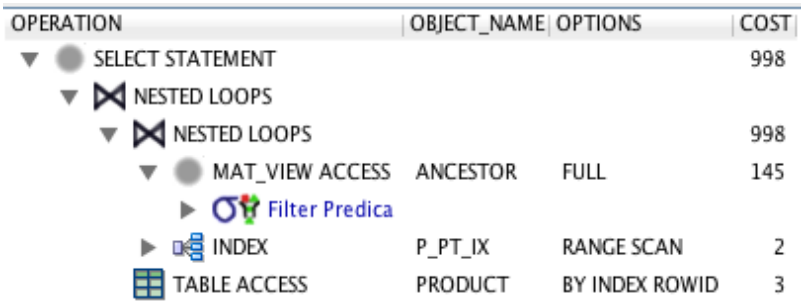


Figure 5.6: Oracle Query Plan for recursion with materialized view

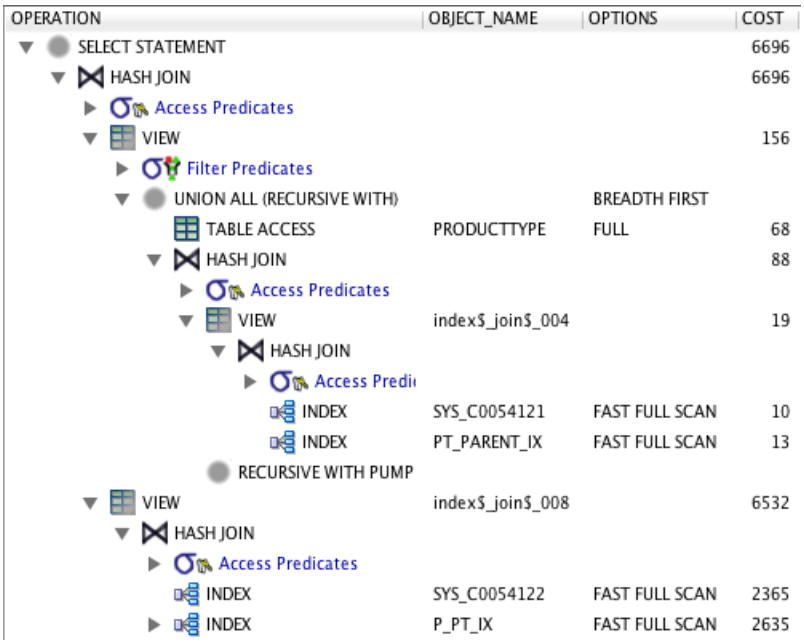


Figure 5.7: Oracle Query Plan for recursion with a non-materialized view

Rodriguez-Muro et. al. has not presented an algorithm before; and

3. representing the mappings as SQL views and study when the views should be materialized.

Ultrawrap is a system that encodes a fixed mapping, the direct mapping [13, 118], of the database as RDF. These mappings are implemented using unmaterialized SQL views. The approach presented in this paper extends Ultrawrap in three important aspects:

1. supports a customized mapping language
2. supports reasoning through saturated mappings
3. considers materializing views for query optimization

Another related work is the combined approach [92], which materializes entailments as data, without considering mappings, and uses a limited form of query rewriting. The main objective of this approach is to deal with the case of infinite materialization which occurs when the ontology expresses cycles and has existential quantifiers in the head. In this research such ontologies are not considered.

5.6 Summary

This chapter presented Ultrawrap$^{\text{OBDA}}$, which to the best of our knowledge, is the first OBDA system supporting ontologies with transitivity by using SQL recursion. Ultrawrap$^{\text{OBDA}}$ is able to push processing into the RDBMS by implementing mappings using materialized views and taking advantage of existing query rewriting techniques. The contributions are:

- A linear algorithm to compile ontological entailments as mappings.

- A proof that every SPARQL query can be rewritten into a SQL query in the context of mappings. It is important to mention that such a result is a minimal requirement for a query-rewriting OBDA system relying on relational database technology.
- A cost model to determine which views to materialize to attain the fastest execution time.
- An empirical evaluation comparing with a state-of-the-art OBDA system, which validates the cost model and demonstrates favorable execution times

Per related work, existing OBDA approaches only exploit the relational algebra capabilities of RDBMS [35]. The experimental results provide evidence that existing advanced capabilities implemented in RDBMS, such as recursion and query rewriting using materialized views, can be utilized for OBDA. This does not mean that OBDA systems should rely exclusively on RDBMS technology to do the heavy lifting. Systems such as MySQL lack these advanced optimizations. However, these results suggest that the OBDA community should exploit more of the advanced optimizations in existing RDBMS.

Chapter 6

Conclusions

In this thesis, an answer to the general question was presented: *How and to what extent can Relational Databases be integrated with the Semantic Web?*. The answer comes in three parts:

- **Relational Databases can be directly mapped to RDF and OWL**: Relational Databases can be automatically mapped to the Semantic Web. An OWL ontology can be generated from the relational schema and the relational data can be represented as an RDF graph. This mapping does not loose information, preserves queries, is monotone and is positive semantics preserving. Additionally, it is not possible to have a monotone and full semantics preserving direct mapping.

- **Relational Databases can evaluate and optimize SPARQL queries**: Relational Databases are able to efficiently evaluate SPARQL queries. By implementing the direct mapping using SQL views, relational optimizer exploit two important semantic query optimizations: detection of unsatisfiable conditions and self join elimination.

- **Relational Databases can act as reasoners**: Given a Relational Database, an OWL ontology with inheritance and transitivity (in the OWL-SQL profile), and a mapping between the two, Relational Databases are able to act reasoner. This is possible by implementing the mappings as SQL views and including SQL recursion, materializing a subset of the views based on a cost model, and exploiting existing optimizations such as query rewriting using materialized views.

The results of this research is embodied in a system called Ultrawrap.

6.1 Summary of Results

In this dissertation, answers to two specific research questions were provided:

1. Given a Relational Database, how can the Relational Database be automatically virtualized as a Semantic Web data source?

2. Given a Relational Database, an OWL ontology with inheritance and transitivity, and a mapping between the two, how can a Relational Database act as a reasoner?

The answer to these questions comes as three contributions.

6.1.1 Contribution 1: Direct Mapping

- A monotone, information preserving, query preserving and positive semantics preserving direct mapping which generates an OWL ontology from the relational schema, coined by us as the *putative ontology* and RDF from the relational data. Databases that have null values are considered.

- The semantics of this direct mapping is defined using Datalog.

- A proof that the combination of monotonicity with the OWL semantics is an obstacle to generating a full semantics preserving direct mapping.

- A non-monotone direct mapping that is full semantics preserving.

To the best of our knowledge this is the first direct mapping, that has been formalized and studied with respect to these fundamental (information and query preserving) and desirable (monotone and semantics preserving) properties. Subsequently, it served as the foundation of a W3C standard [13].

6.1.2 Contribution 2: Ultrawrap

- A method capable of evaluating SPARQL queries against the Relational Database, per the direct mapping.

- A system, Ultrawrap, which is organized as a set of four compilers that 1) extracts a putative ontology from the relational schema, 2) encodes a logical representation of the database w.r.t the ontology using SQL views, 3) syntactically translates SPARQL queries to SQL queries in terms of the views and 4) delegates optimizations to the SQL engine.

- Identification of two known query optimizations, detection of unsatisfiable conditions and self-join elimination, used for the evaluation of SPARQL queries.

- An empirical analysis which consistently yields that the performance of SPARQL query execution on Ultrawrap is comparable to the performance of SQL queries written directly for the relational representation of the data. Such empirical results have not been achieved elsewhere in the literature.

To the best of our knowledge, this is the first system that reuses existing relational database optimizations to evaluate SPARQL.

6.1.3 Contribution 3: Ontology-Based Data Access

- A method for Relational Databases to support inheritance and transitivity by 1) compiling the ontology as mappings, 2) implementing the mappings as SQL views and 3) optimizing by virtue of materializing a subset of the views.

- A linear algorithm to compile ontological entailments as mappings.

- A proof that every SPARQL query can be rewritten into a SQL query in the context of mappings.

- A cost model to determine which views to materialize to attain the fastest execution time.

- An empirical analysis which validates the cost model and demonstrates favorable execution times. The analysis reveals that by materializing a subset of

views and exploiting the capabilities beyond relational algebra, such as query rewriting using materialized views and SQL recursion, relational database are able to effectively act as reasoners.

To the best of our knowledge, this is the first OBDA system supporting ontologies with transitivity by using SQL recursion and reusing existing relational database optimizations in order to act as a reasoner.

6.2 Lessons Learned

The Semantic Web is a relatively new technology, which promises integration and reuse of data by intelligent agents at a web-scale through use of a graph data model, a graph query language and ontologies. However, this vision is not unique to the Semantic Web. The vision of combining logic and data in order to create intelligent agents has been a *holy grail* of Computer Science. Therefore, it is important to understand the relationship between new technologies, such as the Semantic Web, with past and existing work.

To the best of our knowledge, this dissertation is the first to present a methodological study on the relationships between enterprise class Relational Databases and Semantic Web. However, what other past and existing work can and should be studied w.r.t the Semantic Web, in order to not repeat history and reinvent the wheel? Two research areas come to mind: Deductive Databases and Objected Oriented Databases.

Deductive databases are database systems that can make deductions; conclude additional facts from a set of given rules and facts stored in the database. Datalog is the typical language to specify the rules, facts and queries. Integrating Relational Databases with the Semantic Web effectively makes a deductive database. In other words, Ultrawrap$^{\text{OBDA}}$is a deductive database. However, what approaches

to implement deductive databases have been researched in the past that we should learn from and reuse, in order to not repeat and reinvent the wheel? The survey book "Logic Programming and Databases" by Ceri, Gottlob and Tanca [38] provides a systematic overview of research combining logic programming and relational databases, up to 1990.

Object Oriented Databases (OODB) are database systems where information is represented in forms as objects per object-oriented programming [84]. There seems to be an auspicious relationship between OODB and Semantic Web. For example, in OODB, any real world entity is modeled as an object, which belongs to a class, and is associated with a unique identifier. In the Semantic Web, real world entities are instances of an ontological class and also have unique identifiers: URIs. Additionally, subclass is a property shared in both technologies. It seems that our layer cake methodology can be applied to understand the relationship between these two technologies.

6.3 Future Work

Semantic Web technology provides the following features. OWL Ontologies enable reasoning **(reasoning)**. SPARQL queries with variables in the predicate position reveal metadata. This is useful because it enables exploration of the data in case the schema is not known beforehand. Additionally, queries of this form are intrinsic to faceted search **(variable predicate)**. Given the graph model of RDF, the latest version of SPARQL, SPARQL 1.1, increased the expressivity and now provides constructs to navigate the graph **(graph traversal)**. Another virtue of dealing with graphs is that insertion of data is reduced to adding an edge with a node to the graph. There are no physical requirements to conform to a schema **(dynamic schema)**. Finally, data can be easily integrated be simply adding edges between nodes of different graphs **(data integration)**.

A goal of this dissertation is to understand *up to what extent* can Relational Databases be integrated with the Semantic Web. The extent of this dissertation has focused on mappings and reasoning. A remaining question is: can that extent be expanded? And up to where? We call this the *Tipping Point problem.*

Assume the starting point are legacy relational databases and we want to take advantage of these five features of the Semantic Web (reasoning, variable predicate, graph traversal, dynamic schema, data integration)? How much can be subsumed by Relational Database technology before the balance is tipped over and we end up using native Semantic Web technology? What is the tipping point (or points)?

- **Reasoning**: This dissertation proposed to represent ontological entailments as mappings and implement them as views. Subsequently, a subset of these views are materialized. Open questions remain. What is the state of the art of other RDBMS's optimizers in order to support this approach? How does this approach respond to complex query workloads? The model assumed a read-only database, therefore, what is the cost of maintaining views when the underlying data is updated? Evidence is provided that Relational Databases can act as reasoners for the OWL-SQL profile. Can the expressivity of OWL-SQL be increased while maintaining efficient computation by the RDBMS optimizer? What is the trade-off between reasoning over relational databases with mappings and using native RDF databases which supports reasoning?

- **Variable Predicate**: For queries with variables in the predicate position, the mapping stipulates that the variable may be bound to the name of any column in the database. These queries are a syntactic construct of higher order logic. Ultrawrap translates these queries into a SQL query consisting of a union for each attribute in the database. This query ends up reading the entire database and suffers a performance penalty. What optimizations can be implemented in order to overcome this issue? What hints can be provided in a query?

- **Graph Traversal**: Regular Path Queries and SPARQL 1.1 property path queries enable pattern-based reachability queries. These types of queries enable the traversal and navigation of the graph. A natural question is how much of SQL recursion can be used to implement these types of queries?

- **Dynamic schema**: Relational Databases have a fixed schema. Insertion of data needs to adhere to the schema. A schema needs to be altered in case new data is inserted which does not adhere to the schema. Can a Relational Database become hybrid graph/relational database? What effect does the sparsity of data have? What is the best storage manager (column vs row store)?

- **Data Integration**: When it comes to integrate disparate databases, one approach is to extract the relational data, transform it physically to RDF and then load it into a RDF database (ETL). Another approach is to federate queries. In other words, legacy data continues to reside in the relational databases and queries are sent to each source (Federation). Which approach is practical? Depending on what? Can hybrid system be efficient?

An overarching theme is the need to create systematic and real-world benchmarks in order to evaluate different solutions for these features.

These open questions provide a roadmap to further expand the extent that Relational Databases can be integrated with the Semantic Web.

Chapter 7

Appendix

7.1 Additional Operators in Relational algebra

It is important to notice that the operators left-outer join, right-outer join and full-outer join are all expressible with the previous operators. For example, assume that R and S are relation names such that $att(R) \cap att(S) = \{A_1, A_2, \ldots, A_k\}$ and $att(S) \setminus att(R) = \{B_1, B_2, \ldots, B_\ell\}$, then the left-outer join for R and S is defined by the following expression:

$$\Big[R \bowtie S\Big] \cup \Big[\sigma_{\texttt{IsNull}(A_1)}(R) \cup \sigma_{\texttt{IsNull}(A_2)}(R) \cup \cdots \cup \sigma_{\texttt{IsNull}(A_k)}(R) \cup R \bowtie \Big(\sigma_{\texttt{IsNotNull}(A_1)}(\sigma_{\texttt{IsNotNull}(A_2)}(\cdots \sigma_{\texttt{IsNotNull}(A_k)}(\pi_{\{A_1,A_2,\ldots,A_k\}}(R))\cdots)) \setminus \pi_{\{A_1,A_2,\ldots,A_k\}}(S)\Big)\Big] \bowtie \Big[\texttt{NULL}_{B_1} \bowtie \texttt{NULL}_{B_2} \bowtie \cdots \bowtie \texttt{NULL}_{B_\ell}\Big].$$

Similar expressions can be used to express the right-outer join and the full-outer join.

7.2 Semantics of SPARQL

Let P be a SPARQL graph pattern. In the rest of the paper, we use var(P) to denote the set of variables occurring in P. In particular, if t is a triple pattern, then var(t) denotes the set of variables occurring in the components of t. Similarly, for a built-in condition R, we use var(R) to denote the set of variables occurring in R.

In what follows, we present the semantics of graph patterns for a fragment of SPARQL for which the semantics of nested SELECT queries is well understood [73, 10, 11]. More specifically, in what follows we focus on the class of graph patterns P satisfying the following condition: P is said to be *non-parametric* if for every sub-pattern $P_1 = (\text{SELECT } \{?A_1 \text{ AS } ?B_1, \ldots, ?A_m \text{ AS } ?B_m, ?C_1, \ldots, ?C_n\}\ P_2)$ of P and every variable $?X$ occurring in P, if $?X \in (\text{var}(P_2) \setminus \{?A_1, \ldots, ?A_m, ?C_1, \ldots, ?C_n\})$, then $?X$ does not occur in P outside P_1.

To define the semantics of SPARQL graph pattern expressions, we need to

introduce some terminology. A mapping μ is a partial function $\mu : \mathbf{V} \to (\mathbf{I} \cup \mathbf{L})$. Abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ. The domain of μ, denoted by $\mathrm{dom}(\mu)$, 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 $x \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping. The mapping with empty domain is denoted by $\mu_\emptyset$ (notice that this mapping is compatible with any other mapping). 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 $\mathrm{dom}(\mu_{|_W}) = \mathrm{dom}(\mu) \cap W$ and $\mu_{|_W}(?X) = \mu(?X)$ for every $?X \in \mathrm{dom}(\mu) \cap W$. Finally, given a mapping μ and a sequence $?A_1$, ..., $?A_m$, $?B_1$, ..., $?B_m$ of pairwise distinct elements from $\mathbf{V}$ such that $\mathrm{dom}(\mu) \cap \{?B_1, \ldots, ?B_m\} = \emptyset$, define $\rho_{\{?A_1 \to ?B_1, \ldots, ?A_m \to ?B_m\}}(\mu)$ as a mapping such that:

$$\begin{aligned}\mathrm{dom}(\rho_{\{?A_1 \to ?B_1, \ldots, ?A_m \to ?B_m\}}(\mu)) \; &= (\mathrm{dom}(\mu) \setminus \{?A_1, \ldots, ?A_m\}) \cup \\ &\quad \{?B_i \mid i \in \{1, \ldots, m\} \text{ and } ?A_i \in \mathrm{dom}(\mu)\},\end{aligned}$$

and for every $x \in \mathrm{dom}(\rho_{\{?A_1 \to ?B_1, \ldots, ?A_m \to ?B_m\}}(\mu))$:

$$\rho_{\{?A_1 \to ?B_1, \ldots, ?A_m \to ?B_m\}}(\mu)(x) = \begin{cases} \mu(?A_i) & x = ?B_i \text{ for some } i \in \{1, \ldots, m\} \\ \mu(x) & \text{otherwise} \end{cases}$$

We have all the necessary ingredients to define the semantics of graph pattern expressions. As in [104], we define this semantics as a function $[\![\cdot]\!]_G$ that takes a graph pattern expression and returns a set of mappings. For the sake of readability, the semantics of filter expressions is presented separately.

The evaluation of a graph pattern P over an RDF graph G, denoted by $[\![P]\!]_G$, is defined recursively as follows.

1. If P is { } and G is nonempty, then $[\![P]\!]_G = \{\mu_\emptyset\}$. If P is { } and $G = \emptyset$, then

$[\![P]\!]_G = \emptyset$.

2. If P is a triple pattern t, then $[\![P]\!]_G = \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\}$.
3. If P is (P_1 AND P_2), then $[\![P]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]_G,\ \mu_2 \in [\![P_2]\!]_G \text{ and } \mu_1 \sim \mu_2\}$.
4. If P is (P_1 OPT P_2), then $[\![P]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![P_1]\!]_G,\ \mu_2 \in [\![P_2]\!]_G \text{ and } \mu_1 \sim \mu_2\} \cup \{\mu \in [\![P_1]\!]_G \mid \text{ for every } \mu' \in [\![P_2]\!]_G : \mu \not\sim \mu'\}$.
5. If P is (P_1 UNION P_2), then $[\![P]\!]_G = \{\mu \mid \mu \in [\![P_1]\!]_G \text{ or } \mu \in [\![P_2]\!]_G\}$.
6. If P is (P_1 MINUS P_2), then $[\![P]\!]_G = \{\mu \in [\![P_1]\!]_G \mid \text{ for every } \mu' \in [\![P_2]\!]_G : \mu \not\sim \mu' \text{ or } \mathrm{dom}(\mu) \cap \mathrm{dom}(\mu') = \emptyset\}$.
7. If P is (SELECT $\{?A_1$ AS $?B_1, \ldots, ?A_m$ AS $?B_m, ?C_1, \ldots, ?C_n\}$ P_1), then:

$$[\![P]\!]_G \quad = \quad \{\rho_{\{?A_1 \to ?B_1, \ldots, ?A_m \to ?B_m\}}(\mu_{|_{\{?A_1, \ldots, ?A_m, ?C_1, \ldots, ?C_n\}}}) \mid \mu \in [\![P_1]\!]_G\}.$$

The semantics of filter expressions goes as follows. Given a mapping μ and a built-in condition R, we say that μ satisfies R, denoted by $\mu \models R$, if:

1. R is bound($?X$) and $?X \in \mathrm{dom}(\mu)$;
2. R is $?X = c$, $?X \in \mathrm{dom}(\mu)$ and $\mu(?X) = c$;
3. R is $?X = ?Y$, $?X \in \mathrm{dom}(\mu)$, $?Y \in \mathrm{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
4. R is $(\neg R_1)$, R_1 is a built-in condition, and it is not the case that $\mu \models R_1$;
5. R is $(R_1 \vee R_2)$, R_1 and R_2 are built-in conditions, and $\mu \models R_1$ or $\mu \models R_2$;
6. R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions, $\mu \models R_1$ and $\mu \models R_2$.

Then given an RDF graph G and a filter expression (P FILTER R):

$$[\![(P \text{ FILTER } R)]\!]_G \quad = \quad \{\mu \in [\![P]\!]_G \mid \mu \models R\}.$$

7.3 Proofs

7.3.1 Proof of Theorem 1

We show that $\mathcal{DM}$ is information preserving by providing a computable mapping $\mathcal{N} : \mathcal{G} \to \mathcal{I}$ that satisfies the condition in Definition 15. More precisely, given a relational schema $\mathbf{R}$, a set Σ of PKs and FKs and an instance I of $\mathbf{R}$ satisfying Σ, next we should how $\mathcal{N}(G)$ is defined for $\mathcal{DM}(\mathbf{R}, \Sigma, I) = G$.

- **Step 1:** Identify all the ontological class triples (i.e Triple(r, `"rdf:type"`, `"owl:Class"`)). The IRI r identifies an ontological class R'. For every R' that was retrieved from G, map it to a relation name R.

- **Step 2:** Identify all the datatype triples of a given class (i.e Triple(a, `"rdf:type"`, `"owl:DatatypeProperty"`), Triple(a, `"rdfs:domain"`, r_i)). The IRI a identifies the datatype property A' and the IRI r identifies the ontological class R' that is the domain of A'. Every datatype property A' with domain R' is mapped to an attribute A of relation name R.

- **Step 3:** For each class R' and the datatype properties $A'_1 \dots A'_n$ that have domain R', we can recover the instances of relation R with the following SPARQL query:

$$
\begin{aligned}
Q_1 \;=\; \text{SELECT } \{?A_1, \dots, ?A_n\} \Big[\cdots \Big(\Big(\Big((?X, \texttt{"rdf:type"}, r_i) \\
\text{OPT } (?X, a_1, ?A_1) \Big) \text{ OPT } (?X, a_2, ?A_2) \Big) \text{ OPT } (?X, a_3, ?A_3) \Big) \\
\cdots \text{ OPT } (?X, a_n, ?A_n) \Big].
\end{aligned}
$$

- **Step 4:** Identify all the object property triples (i.e. TRIPLE(r, `"rdf:type"`, `"owl:ObjectProperty"`)). The IRI r that only has one element left of the # sign means that r identifies the object property R' in the ontology that was originally mapped from a binary relation. This object property R' is mapped back to a binary relation name R. The two elements following the # sign identify the attributes of the relation R. From the triples TRIPLE(r, `"rdfs:domain"`, s) and TRIPLE(r, `"rdfs:range"`, t), the IRI s identifies the ontological class S' which is mapped to the relation S and the IRI t identifies the ontological class T' which is mapped to the relation T. Additionally, the elements in the third and fourth position after the # identify the attributes which are being referenced from relations S and T respectively. For sake of simplicity, assume that the relation R references the attribute B of relation S which is mapped to a datatype property B' with domain S' and IRI b. Additionally, the relation R references the attribute C of relation T, which is mapped to a datatype property C' with domain T' and IRI c.

  We can now recover the instances of the relation R with the following SPARQL query:

$$\begin{aligned} Q^{\star} \quad = \quad & (\text{SELECT } \{?A_1, ?A_2\} \\ & ((?T_1, r, ?T_2) \text{ AND } (?T_1, b, ?A_1) \text{ AND } (?T_2, c, ?A_2))). \end{aligned}$$

- **Step 5:** Given that the result of a SPARQL query is a set Ω of solution mapping μ, we need to translate each solution mapping $\mu \in \Omega$ into a tuple t. We define a function tr^{-1} as the inverse of function tr, that is, for each solution mapping μ and variable $?A$ in the domain of μ, tr^{-1} assigns the value of $\mu(?A)$ to $t.A$. Then the mapping function $\mathcal{N}$ over G is defined as the following relational instance. For every non-binary relation name identified in

Steps 1, 2, 3, define $R^{\mathcal{N}(G)}$ as $tr^{-1}([\![Q_1]\!]_G)$, and for every binary relation R identified in Step 4, define $R^{\mathcal{N}(G)}$ as $tr^{-1}([\![Q_2]\!]_G)$.

It is straightforward to prove that for every relational schema $\mathbf{R}$, set Σ of PKs and FKs and an instance I of $\mathbf{R}$ satisfying Σ, it holds that $\mathcal{N}(\mathcal{M}(\mathbf{R}, \Sigma, I)) = I$. This concludes the proof of the theorem.

7.3.2 Proof of Theorem 2

We need to prove that for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and relational algebra query Q over $\mathbf{R}$, there exists a SPARQL query $Q^\star$ such that for every instance I of $\mathbf{R}$ including null values:

$$tr([\![Q]\!]_I) \quad = \quad [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}.$$

In what follows, assume that $\mathbf{R}$ is a relational schema, Σ is a set of PKs and FKs over $\mathbf{R}$, and I is an instance of $\mathbf{R}$ satisfying Σ. The following lemma is used in the proof of the theorem.

Lemma 1 *Let Q_1 be a relational algebra query over $\mathbf{R}$ such that $att(Q_1) = \{A_1, \ldots, A_\ell\}$, and assume that $Q_1^\star$ is a SPARQL graph pattern such that:*

$$tr([\![Q_1]\!]_I) \quad = \quad [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}.$$

Then we have that:

$$tr([\![Q_1]\!]_I) \quad = \quad [\![(\text{SELECT } \{?A_1, \ldots, ?A_\ell\} \; Q_1^\star)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}.$$

First, we prove that $tr([\![Q_1]\!]_I) \subseteq [\![(\text{SELECT } \{?A_1, \ldots, ?A_\ell\} \; Q_1^\star)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q_1]\!]_I)$. Then there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu$. Thus, given that $att(Q_1) = \{A_1, \ldots, A_\ell\}$, we conclude that $\text{dom}(\mu) \subseteq$

$\{?A_1, \ldots, ?A_\ell\}$. Given that $tr([\![Q_1]\!]_I) = [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we have that $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Hence, from the fact that $\text{dom}(\mu) \subseteq \{?A_1, \ldots, ?A_\ell\}$, we conclude that $\mu \in [\![(\text{SELECT } \{?A_1, \ldots, ?A_\ell\}\ Q_1^\star)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we prove that $[\![(\text{SELECT } \{?A_1, \ldots, ?A_\ell\}\ Q_1^\star)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q_1]\!]_I)$. Assume that $\mu \in [\![(\text{SELECT } \{?A_1, \ldots, ?A_\ell\}\ Q_1^\star)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then there exists a mapping $\mu' \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $\mu = \mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}$. From the fact that $tr([\![Q_1]\!]_I) = [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we conclude that $\mu' \in tr([\![Q_1]\!]_I)$. Thus, there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu'$. But then given that $att(Q_1) = \{A_1, \ldots, A_\ell\}$, we conclude by definition of tr that $\text{dom}(\mu') \subseteq \{?A_1, \ldots, ?A_\ell\}$. Therefore, given that $\mu = \mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}$, we have that $\mu = \mu'$ and, hence, $\mu \in tr([\![Q_1]\!]_I)$ since $\mu' \in tr([\![Q_1]\!]_I)$. We now prove the theorem by induction on the structure of relational algebra query Q.

Base Case: For the sake of readability, we introduce a function ν that retrieves the IRI for a given relation R, denoted by $\nu(R)$, and the IRI for a given attribute A in a relation R, denoted by $\nu(A, R)$. In this part of the proof, we need to consider three cases.

- **Non-binary relations:** Assume that Q is the identity relational algebra query R, where R is a non-binary relation according to the definition given in Section 3.4.2. Moreover, assume that $att(R) = \{A_1, \ldots, A_\ell\}$, with the corresponding IRIs $\nu(R) = r, \nu(A_1, R) = a_1, \ldots, \nu(A_\ell, R) = a_\ell$. Finally, let $Q^\star$ be the following SPARQL query:

$$
\begin{aligned}
Q^\star \ = \ \text{SELECT } \{?A_1, \ldots, ?A_\ell\} \ \Big[\cdots \\
\Big(\Big(\Big((?X, \texttt{"rdf:type"}, r) \text{ OPT } (?X, a_1, ?A_1) \Big) \text{ OPT } (?X, a_2, ?A_2) \Big) \\
\text{OPT } (?X, a_3, ?A_3) \Big) \cdots \text{ OPT } (?X, a_\ell, ?A_\ell) \Big].
\end{aligned}
$$

Next we prove that $tr(\llbracket Q \rrbracket_I) = \llbracket Q^\star \rrbracket_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

First, we show that $tr(\llbracket Q \rrbracket_I) \subseteq \llbracket Q^\star \rrbracket_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr(\llbracket Q \rrbracket_I)$. Then there exists a tuple $t \in \llbracket Q \rrbracket_I$ such that $tr(t) = \mu$ and, hence, $t \in R^I$. Without loss of generality, assume that there exists $k \in \{0, \ldots, \ell\}$ such that (1) $t.A_i \neq \texttt{NULL}$ for every $i \in \{1, \ldots k\}$, and (2) $t.A_j = \texttt{NULL}$ for every $j \in \{k+1, \ldots, \ell\}$. By definition of tr, we have that $t.A_i = \mu(?A_i)$ for every $i \in \{1, \ldots, k\}$, and that $\text{dom}(\mu) = \{?A_1, \ldots, ?A_k\}$. Given the definition of $\mathcal{DM}$, we have that the following holds: $\textsc{Class}(R)$ and $\text{DTP}(A_i, R)$ for every $i \in \{1, \ldots, \ell\}$. Hence, given that R is not a binary relation (that is, $\textsc{IsBinRel}(R)$ does not hold), we have that the following triples are included in $\mathcal{DM}(\mathbf{R}, \Sigma, I)$:

- $(r_{id}, \texttt{"rdf:type"}, r)$, where r_{id} is the tuple id for the tuple t, and
- (r_{id}, a_i, v_i), where $i \in \{1, \ldots, k\}$ and v_i is the value of attribute A_i in the tuple t, that is, $t.A_i = v_i$.

Thus, given that no triple of the form (r_{id}, a_j, v_j) is included in $\mathcal{DM}(\mathbf{R}, \Sigma, I)$, for $j \in \{k+1, \ldots, \ell\}$, we conclude that $\mu \in \llbracket Q^\star \rrbracket_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by definition of $Q^\star$ and the fact that $\mu = tr(t)$.

Second, we show that $\llbracket Q^\star \rrbracket_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr(\llbracket Q \rrbracket_I)$. Assume that $\mu \in \llbracket Q^\star \rrbracket_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Without loss of generality, assume that $\text{dom}(\mu) = \{?A_1, \ldots, ?A_k\}$, where $0 \leq k \leq \ell$. Then by definition of $Q^\star$, we have that there exists an IRI r_{id} such that $\mathcal{DM}(\mathbf{R}, \Sigma, I)$ contains triples $(r_{id}, \texttt{"rdf:type"}, r)$ and $(r_{id}, a_i, \mu(?A_i))$, for every $i \in \{1, \ldots, k\}$, and it does not contain a triple of the form (r_{id}, a_j, v_j), for every $j \in \{k+1, \ldots, \ell\}$. Given the definition of $\mathcal{DM}(\mathbf{R}, \Sigma, I)$ and the fact that $\textsc{IsBinRel}(R)$ does not hold, we conclude that there exists a tuple $t \in R^I$ such that: (1) the IRI assigned by $\mathcal{DM}$ to t is r_{id}, (2) $t.A_i = \mu(?A_i)$ for every $i \in \{1, \ldots, k\}$, and (3) $t.A_j = \texttt{NULL}$ for every $j \in \{k+1, \ldots, \ell\}$. Thus, given that $tr(t) = \mu$ and $t \in R^I$, we conclude that $\mu \in tr(\llbracket Q \rrbracket_I)$ (recall that

$[\![Q]\!]_I = R^I$).

- **Binary relation:** Assume that Q is the identity relational algebra query R, where R is a binary relation according to the definition given in Section 3.4.2. Moreover, assume that $att(R) = \{A_1, A_2\}$, where A_1 is a foreign key referencing the attribute B of a relation S, and A_2 is a foreign key referencing the attribute C of a relation T. Finally, assume that $\nu(R) = r$, $\nu(B, S) = b$ and $\nu(C, T) = c$, and define $Q^\star$ as the following SPARQL 1.1 query:

$$\begin{aligned} Q^\star \quad = \quad & (\text{SELECT } \{?A_1, ?A_2\} \\ & ((?T_1, r, ?T_2) \text{ AND } (?T_1, b, ?A_1) \text{ AND } (?T_2, c, ?A_2))). \end{aligned}$$

  Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

  First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$ and, hence, $t \in R^I$. Given the definition of mapping $\mathcal{DM}$, we have that all the following hold: $\textsc{BinRel}(R, A_1, A_2, S, B, T, C)$, $\text{PK}(A_1, A_2, R)$, $\text{FK}_1(A_1, R, B, S)$, $\text{FK}_1(A_2, R, C, T)$, $\textsc{Class}(S)$, $\text{DTP}(B, S)$, $\textsc{Class}(T)$, $\text{DTP}(C, T)$, $\textsc{Rel}(S)$, $\textsc{Attr}(B, S)$, $\textsc{Rel}(T)$ and $\textsc{Attr}(C, T)$. From this, we conclude that there exist tuples $t_1 \in S^I$, $t_2 \in T^I$ such that $t.A_1 = t_1.B \neq \texttt{NULL}$ and $t.A_2 = t_2.C \neq \texttt{NULL}$, and we also conclude that the following triples are included in $\mathcal{DM}(\mathbf{R}, \Sigma, I)$:

  - (s_{id}, r, t_{id}) where s_{id} is the tuple id for tuple t_1 and t_{id} is the tuple id for tuple t_2,
  - (s_{id}, b, v_1), where v_1 is the value of attribute B in the tuple t_1, that is, $t_1.B = v_1$,
  - (t_{id}, c, v_2), where v_2 is the value of attribute C in the tuple t_2, that is, $t_2.C = v_2$.

Given that $t.A_1 = t_1.B = v_1$, $t.A_2 = t_2.C = v_2$ and $tr(t) = \mu$, we conclude by definition of $Q^\star$ that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, which implies that $\mathrm{dom}(\mu) = \{?A_1, ?A_2\}$. By definition of $Q^\star$, we have that there exist IRIs s_{id}, t_{id} such that the following triples are in $\mathcal{DM}(\mathbf{R}, \Sigma, I)$: (s_{id}, r, t_{id}), $(s_{id}, b, \mu(?A_1))$ and $(t_{id}, c, \mu(?A_2))$. Hence, by definition of $\mathcal{DM}$, we have that there exist tuples $t_1 \in S^I$, $t_2 \in T^I$ such that: (1) s_{id} is the IRI assigned to t_1 by $\mathcal{DM}$, (2) $t_1.B = \mu(?A_1)$, (3) t_{id} is the IRI assigned to t_2 by $\mathcal{DM}$, and (4) $t_2.C = \mu(?A_2)$. Moreover, we also have by definition of $\mathcal{DM}$ that the following holds: $\textsc{BinRel}(R, A_1, A_2, S, B, T, C)$, $\mathrm{FK}_1(A_1, R, B, S)$ and $\mathrm{FK}_1(A_2, R, C, T)$. Hence, there exists tuple $t \in R^I$ such that $t.A_1 = t_1.B = \mu(?A_1)$ and $t.A_2 = t_2.C = \mu(?A_2)$. Therefore, given that $\mu = tr(t)$ (since $att(R) = \{A_1, A_2\}$ and $\mathrm{dom}(\mu) = \{?A_1, ?A_2\}$) and $t \in [\![Q]\!]_I$ (since $[\![Q]\!]_I = R^I$), we conclude that $\mu \in tr([\![Q]\!]_I)$.

- Third, assume that $Q = \mathtt{NULL}_A$, and let $Q^\star$ be the SPARQL query { }. We have that $[\![Q]\!]_I = \{t\}$, where t is a tuple with domain $\{A\}$ such that $t.A = \mathtt{NULL}$. Moreover, we have that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} = \{\mu_\emptyset\}$ since $\mathcal{DM}(\mathbf{R}, \Sigma, I)$ is a nonempty RDF graph. Thus, given that $tr(t) = \mu_\emptyset$, we conclude that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Inductive Step: Assume that the theorem holds for relational algebra queries Q_1 and Q_2. That is, there exists SPARQL queries $Q_1^\star$ and $Q_2^\star$ such that:

$$\begin{aligned} tr([\![Q_1]\!]_I) &= [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}, \\ tr([\![Q_2]\!]_I) &= [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \end{aligned}$$

To continue with the proof, we need to consider the following operators: selection (σ), projection (π), rename (δ), join ($\bowtie$), union ($\cup$) and difference ($\setminus$).

- **Selection:** We need to consider four cases.

  - **Case 1**. Assume that $Q = \sigma_{A_1=a}(Q_1)$, and $Q^\star = (Q_1^\star$ FILTER $(?A_1 = a))$. Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

    First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Thus, we have that $t \in [\![Q_1]\!]_I$ and $t.A_1 = a$. By definition of tr, we know that $t.A_1 = \mu(?A_1)$, from which we conclude that $\mu(?A_1) = a$ given that $t.A_1 = a$. Therefore, $\mu \models (?A_1 = a)$, from which we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ since $\mu = tr(t)$ and $tr(t) \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by induction hypothesis.

    Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \models (?A_1 = a)$, that is, $\mu(?A_1) = a$. By induction hypothesis, we have that $\mu \in tr([\![Q_1]\!]_I)$, and, hence, there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu$. By definition of tr, we know that $t.A_1 = \mu(?A_1)$, from which we conclude that $t.A_1 = a$ given that $\mu(?A_1) = a$. Given that $t \in [\![Q_1]\!]_I$ and $t.A_1 = a$, we have that $t \in [\![Q]\!]_I$. Therefore, we conclude that $\mu \in tr([\![Q]\!]_I)$ since $tr(t) = \mu$.

  - **Case 2**. Assume that $Q = \sigma_{A_1\neq a}(Q_1)$, and $Q^\star = (Q_1^\star$ FILTER $(\neg(?A_1 = a) \wedge \text{bound}(?A_1)))$. Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

    First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Given that $t \in [\![Q]\!]_I$, we have by the definition of the semantics of relational algebra that $t \in [\![Q_1]\!]_I$, $t.A_1 \neq a$ and $t.A_1 \neq \texttt{NULL}$. Thus, by definition of tr we have that $t.A_1 = \mu(?A_1)$ and $\mu(?A_1) \neq a$. Hence, we have that $\mu \models (\neg(?A_1 = a) \wedge \text{bound}(?A_1))$, from which we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ since $\mu = tr(t)$ and $tr(t) \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by induction hypothesis.

    Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in$

$[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \models (\neg(?A_1 = a) \wedge \text{bound}(?A_1))$, that is, $?A_1 \in \text{dom}(\mu)$ and $\mu(?A_1) \neq a$. By induction hypothesis we have that $\mu \in tr([\![Q_1]\!]_I)$ and, hence, there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu$. Given that $?A_1 \in \text{dom}(\mu)$ and $\mu(?A_1) \neq a$, it holds that $t.A_1 \neq \texttt{NULL}$ and $t.A_1 \neq a$. Thus, we have that $t \in [\![Q]\!]_I$, from which we conclude that $\mu \in tr([\![Q]\!]_I)$ since $\mu = tr(t)$.

– **Case 3**. Assume that $Q = \sigma_{\texttt{IsNull}(A_1)}(Q_1)$, and $Q^\star = (Q_1^\star \text{ FILTER } (\neg\,\text{bound}(?A_1)))$. Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Given that $t \in [\![Q]\!]_I$, we have that $t \in [\![Q_1]\!]_I$ and $t.A_1 = \texttt{NULL}$. Thus, we conclude by definition of tr that $?A_1 \notin \text{dom}(\mu)$ and, hence, $\mu \models \neg\,\text{bound}(?A_1)$. Therefore, we have that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ given that $\mu = tr(t)$ and $tr(t) \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by induction hypothesis.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \models (\neg\,\text{bound}(?A_1))$, that is, $?A_1 \notin \text{dom}(\mu)$. By induction hypothesis we have that $\mu \in tr([\![Q_1]\!]_I)$, from which we conclude that there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu$. By definition of tr and given that $?A_1 \notin \text{dom}(\mu)$, we have that $t.A_1 = \texttt{NULL}$ and, hence, $t \in [\![Q]\!]_I$. Therefore, we conclude that $\mu \in tr([\![Q]\!]_I)$ since $\mu = tr(t)$.

– **Case 4**. Assume that $Q = \sigma_{\texttt{IsNotNull}(A_1)}(Q_1)$, and $Q^\star = (Q_1^\star \text{ FILTER } (\text{bound}(?A_1)))$. Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Given that

$t \in [\![Q]\!]_I$, we have that $t \in [\![Q_1]\!]_I$ and $t.A_1 \neq \texttt{NULL}$. Thus, by definition of tr we have that $?A_1 \in \text{dom}(\mu)$ and $\mu(?A_1) = t.A_1$ and, hence, $\mu \models \text{bound}(?A_1)$. Therefore, we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ given that $\mu = tr(t)$ and $tr(t) \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \models \text{bound}(?A_1)$, that is, $?A_1 \in \text{dom}(\mu)$. By induction hypothesis we have that there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu$. Thus, by definition of tr we have that $t.A_1 = \mu(?A_1)$, which implies that $t.A_1 \neq \texttt{NULL}$. Therefore, we have that $t \in [\![Q]\!]_I$ and, hence, $\mu \in tr([\![Q]\!]_I)$ since $\mu = tr(t)$.

- **Projection:** Assume that $Q = \pi_{\{A_1,\ldots,A_\ell\}}(Q_1)$, and $Q^\star = (\text{SELECT } \{?A_1, \ldots, ?A_\ell\}\ Q_1^\star)$. Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

  First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Given that $t \in [\![Q]\!]_I$, there exists a tuple $t' \in [\![Q_1]\!]_I$ such that for every $A \in att(Q) : t.A = t'.A$. Without loss of generality, assume that: (1) $att(Q) = \{A_1, \ldots, A_k, A_{k+1}, \ldots, A_\ell\}$, (2) $t.A_i \neq \texttt{NULL}$ for every $i \in \{1, \ldots, k\}$, and (3) $t.A_j = \texttt{NULL}$ for every $j \in \{k+1, \ldots, \ell\}$. By definition of tr, we have that $t.A_i = \mu(?A_i)$ for every $i \in \{1, \ldots, k\}$, and that $\text{dom}(\mu) = \{?A_1, \ldots, ?A_k\}$. Given that $t' \in [\![Q_1]\!]_I$, we have for $\mu' = tr(t')$ that: (1) $\mu' \in tr([\![Q_1]\!]_I)$, (2) $\text{dom}(\mu) \subseteq \text{dom}(\mu')$, (3) $\text{dom}(\mu) = (\{?A_1, \ldots, ?A_\ell\} \cap \text{dom}(\mu'))$, and (4) $t.A_i = t'.A_i = \mu(?A_i) = \mu'(?A_i)$ for every $i \in \{1, \ldots, k\}$. Thus, we have in particular that:

$$\mu \ = \ \mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}. \qquad (\dagger)$$

By induction hypothesis we have that $\mu' \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, from which we conclude that $\mu'_{|_{\{?A_1,\ldots,?A_\ell\}}} \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Thus, we conclude from (†) that

$\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then there exists a mapping $\mu' \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $\mu = \mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}$. By induction hypothesis, we have that $\mu' \in tr([\![Q_1]\!]_I)$, from we conclude that there exists a tuple $t' \in [\![Q_1]\!]_I$ such that $tr(t') = \mu'$. Let t be a tuple with domain $\{A_1, \ldots, A_\ell\}$ such that $t.A_i = t'.A_i$ for every $i \in \{1, \ldots, \ell\}$. Then, given that $t' \in [\![Q_1]\!]_I$, we have that $t \in [\![Q]\!]_I$, and given that $\mu' = tr(t')$ and $\mu = \mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}$, we have that $\mu = tr(t)$. Therefore, we conclude that $\mu \in tr([\![Q]\!]_I)$.

- **Rename:** Assume that $att(Q) = \{A_1, \ldots, A_\ell\}$ and $Q = \delta_{A_1 \to B_1}(Q_1)$, and let $Q^\star = (\text{SELECT } \{?A_1 \text{ AS } ?B_1, ?A_2, \ldots, ?A_\ell\}\ Q_1^\star)$. Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

  First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Given that $t \in [\![Q]\!]_I$, there exists a tuple $t' \in [\![Q_1]\!]_I$ such that $t.B_1 = t'.A_1$ and $t.A_i = t'.A_i$ for every $i \in \{2, \ldots, \ell\}$. Without loss of generality, assume that there exists $k \in \{1, \ldots, \ell\}$ such that: (1) $t.A_i \neq \texttt{NULL}$ for every $i \in \{2, \ldots, k\}$, and (2) $t.A_j = \texttt{NULL}$ for every $j \in \{k+1, \ldots, \ell\}$. To finish the proof, we consider two cases.

  - Assume that $t.B_1 \neq \texttt{NULL}$. Then it follows from conditions (1), (2) and definition of tr that $\mu(?A_1) = t.B_1 = t'.A_1$, $\mu(?A_i) = t.A_i = t'.A_i$ for every $i \in \{2, \ldots, k\}$ and $\text{dom}(\mu) = \{?A_1, ?A_2, \ldots, ?A_k\}$. Let $\mu' = tr(t')$. Then by definition of tr, we have that $\rho_{\{?A_1 \to ?B_1\}}(\mu') = \mu$. Moreover, given that $\mu' = tr(t')$ and $t' \in [\![Q_1]\!]_I$, we conclude that $\mu' \in tr([\![Q_1]\!]_I)$ and, hence, $\mu' \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by induction hypothesis. Thus, we have that $\rho_{\{?A_1 \to ?B_1\}}(\mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}) \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, from which we conclude

that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ since $\mu'_{|_{\{A_1,\ldots,A_\ell\}}} = \mu'$ and $\rho_{\{?A_1 \to ?B_1\}}(\mu') = \mu$.

– Assume that $t.B_1 = \texttt{NULL}$. Then it follows from conditions (1), (2) and definition of tr that $\mu(?A_i) = t.A_i = t'.A_i$ for every $i \in \{2,\ldots,k\}$ and $\text{dom}(\mu) = \{?A_2, ?A_2, \ldots, ?A_k\}$. Let $\mu' = tr(t')$. Then by definition of tr, we have that $\rho_{\{?A_1 \to ?B_1\}}(\mu') = \mu$. Moreover, given that $\mu' = tr(t')$ and $t' \in [\![Q_1]\!]_I$, we conclude that $\mu' \in tr([\![Q_1]\!]_I)$ and, hence, $\mu' \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by induction hypothesis. Thus, we have that $\rho_{\{?A_1 \to ?B_1\}}(\mu'_{|_{\{?A_1,\ldots,?A_\ell\}}}) \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, from which we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ since $\mu'_{|_{\{A_1,\ldots,A_\ell\}}} = \mu'$ and $\rho_{\{?A_1 \to ?B_1\}}(\mu') = \mu$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then there exists a mapping $\mu' \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $\mu = \rho_{\{?A_1 \to ?B_1\}}(\mu'_{|_{\{?A_1,\ldots,?A_\ell\}}})$. By induction hypothesis, we have that $\mu' \in tr([\![Q_1]\!]_I)$, from which we conclude that there exists a tuple $t' \in [\![Q_1]\!]_I$ such that $tr(t') = \mu'$. Let t be a tuple with domain $\{B_1, A_2, \ldots, A_\ell\}$ such that $t.B_1 = t'.A_1$ and $t.A_i = t'.A_i$ for every $i \in \{2,\ldots,\ell\}$. Then we have that $t \in [\![Q]\!]_I$. Given that $\mu' = tr(t')$ and $\mu = \rho_{\{?A_1 \to ?B_1\}}(\mu'_{|_{\{?A_1,\ldots,?A_\ell\}}})$, we have that $\mu = tr(t)$. Therefore, we conclude that $\mu \in tr([\![Q]\!]_I)$.

- **Join:** Assume that $Q = (Q_1 \bowtie Q_2)$, where $(att(Q_1) \cap att(Q_2)) = \{A_1,\ldots,A_\ell\}$, and let

$$Q^\star \;=\; \Big(\Big(Q_1^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell))\Big) \text{ AND } \Big(Q_2^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell))\Big)\Big).$$

Next we prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple t such that $\mu = tr(t)$ and $t \in [\![Q]\!]_I$. Thus, we have

that there exist tuples $t_1 \in [\![Q_1]\!]_I$ and $t_2 \in [\![Q_2]\!]_I$ such that: (1) $t.A_i = t_1.A_i = t_2.A_i \neq \texttt{NULL}$ for every $i \in \{1, \ldots, \ell\}$, (2) $t.A = t_1.A$ for every $A \in (att(Q_1) \setminus att(Q_2))$, and (3) $t.A = t_2.A$ for every $A \in (att(Q_2) \setminus att(Q_1))$. Let $\mu_1 = tr(t_1)$ and $\mu_2 = tr(t_2)$. By induction hypothesis and given that $\mu_1 \in tr([\![Q_1]\!]_I)$ and $\mu_2 \in tr([\![Q_2]\!]_I)$, we have that $\mu_1 \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu_2 \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Hence, from condition (1) and definition of tr, we conclude that:

$$\begin{aligned} \mu_1 &\in [\![(Q_1^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell)))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}, \\ \mu_2 &\in [\![(Q_2^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell)))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \end{aligned}$$

Thus, given that $\mu = \mu_1 \cup \mu_2$ by conditions (1), (2), (3) and definition of tr, we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then there exist mappings μ_1, μ_2 such that: (1) $\mu = \mu_1 \cup \mu_2$, (2) $\mu_1 \in [\![(Q_1^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \cdots \wedge \text{bound}(?A_\ell)))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, and (3) $\mu_2 \in [\![(Q_2^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell)))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. By induction hypothesis, we have that $\mu_1 \in tr([\![Q_1]\!]_I)$ and $\mu_2 \in tr([\![Q_2]\!]_I)$. Thus, there exist tuples $t_1 \in [\![Q_1]\!]_I$, $t_2 \in [\![Q_2]\!]_I$ such that $\mu_1 = tr(t_1)$ and $\mu_2 = tr(t_2)$. From conditions (1), (2), (3) and definition of tr, we have that $t_1.A_i = t_2.A_i = \mu(?A_i) \neq \texttt{NULL}$ for every $i \in \{1, \ldots, \ell\}$. Thus, given that $(att(Q_1) \cap att(Q_2)) = \{A_1, \ldots, A_\ell\}$, we have that $t \in [\![Q]\!]_I$, where $t : (att(Q_1) \cup att(Q_2)) \to (\mathbf{D} \cup \{\texttt{NULL}\})$ such that: (4) $t.A_i = t_1.A_i = t_2.A_i$ for every $i \in \{1, \ldots, \ell\}$, (5) $t.A = t_1.A$ for every $A \in (att(Q_1) \setminus att(Q_2))$, and (6) $t.A = t_2.A$ for every $A \in (att(Q_2) \setminus att(Q_1))$. Hence, we conclude that $\mu \in tr([\![Q]\!]_I)$, given that $\mu = tr(t)$ by definition of t, definition of tr and conditions (1), (2) and (3).

- **Union:** Assume that $Q = (Q_1 \cup Q_2)$ and $Q^\star = (Q_1^\star \text{ UNION } Q_2^\star)$. Next we

prove that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Thus, we have that $t \in [\![Q_1]\!]_I$ or $t \in [\![Q_2]\!]_I$. Without loss of generality, assume that $t \in [\![Q_1]\!]_I$. Then we have that $tr(t) \in tr([\![Q_1]\!]_I)$ and, hence, $tr(t) \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ by induction hypothesis. Therefore, $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ since $tr(t) = \mu$, from which we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ or $\mu \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Without loss of generality, assume that $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then, by induction hypothesis, we have that $\mu \in tr([\![Q_1]\!]_I)$, and, hence, there exists a tuple $t \in [\![Q_1]\!]_I$ such that $tr(t) = \mu$. Therefore, we conclude that $t \in [\![(Q_1 \cup Q_2)]\!]_I$, from which we deduce that $\mu \in tr([\![Q]\!]_I)$.

- **Difference:** Assume that $Q = (Q_1 \setminus Q_2)$, and that $att(Q_1) = att(Q_2) = \{A_1, \ldots, A_\ell\}$. Then for every (not necessarily nonempty) set $\mathcal{X} = \{i_1, i_2, \ldots, i_p\}$ such that $1 \leq i_1 < i_2 < \ldots < i_p \leq \ell$, define $R_\mathcal{X}$ as the following filter condition:

$$\Big(\text{bound}(?A_{i_1}) \wedge \text{bound}(?A_{i_2}) \wedge \cdots \wedge \text{bound}(?A_{i_p}) \wedge \neg\,\text{bound}(?A_{j_1}) \wedge \neg\,\text{bound}(?A_{j_2}) \wedge \cdots \wedge \neg\,\text{bound}(?A_{j_q})\Big),$$

where $1 \leq j_1 < j_2 < \cdots < j_q \leq \ell$ and $\{j_1, j_2, \cdots, j_q\} = (\{1, \ldots, \ell\} \setminus \{i_1, i_2, \ldots, i_p\})$. That is, condition $R_\mathcal{X}$ indicates that every variables $?A_i$ with $i \in \mathcal{X}$ is bound, while every variable $?A_j$ with $j \in (\{1, \ldots, \ell\} \setminus \mathcal{X})$ is not bound. Moreover, for every $\mathcal{X} \neq \emptyset$ define SPARQL graph pattern $P_\mathcal{X}$ as follows:

$$P_\mathcal{X} \;=\; ((Q_1^\star \text{ FILTER } R_\mathcal{X}) \text{ MINUS } (Q_2^\star \text{ FILTER } R_\mathcal{X})).$$

Notice that there are $2^\ell - 1$ possible graph patterns $P_{\mathcal{X}}$ with $\mathcal{X} \neq \emptyset$. Let P_1, P_2, ..., $P_{2^\ell - 1}$ be an enumeration of these graph patterns. Moreover, assuming that $?X$, $?Y$, $?Z$ are fresh variables, let $P_\emptyset$ be the following query:

$$\Big[\Big[\Big(Q_1^\star \text{ FILTER } R_\emptyset\Big) \text{ OPT } \Big(\Big(Q_2^\star \text{ FILTER } R_\emptyset\Big) \text{ AND } (?X, ?Y, ?Z)\Big)\Big] \text{ FILTER } (\neg \text{bound}(?X))\Big].$$

Then graph pattern $Q^\star$ is defined as follows:

$$Q^\star \quad = \quad (P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_{2^\ell - 1} \text{ UNION } P_\emptyset).$$

Next we show that $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. In this proof, we assume, by considering Lemma 1, that for every mapping μ such that $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ or $\mu \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, it holds that $\text{dom}(\mu) \subseteq \{?A_1, \ldots, ?A_\ell\}$.

First, we show that $tr([\![Q]\!]_I) \subseteq [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Assume that $\mu \in tr([\![Q]\!]_I)$. Then there exists a tuple $t \in [\![Q]\!]_I$ such that $tr(t) = \mu$. Thus, we have that $t \in [\![Q_1]\!]_I$ and $t \notin [\![Q_2]\!]_I$, from which we conclude by considering the induction hypothesis that $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \notin [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. We consider two cases to show that this implies that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

- Assume that $\text{dom}(\mu) \neq \emptyset$, and let $\mathcal{X} = \{i \in \{1, \ldots, \ell\} \mid ?A_i \in \text{dom}(\mu)\}$. Given that $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we have that $\text{dom}(\mu) \subseteq \{?A_1, \ldots, ?A_\ell\}$ and, hence, $\mathcal{X} \neq \emptyset$. Furthermore, we have that $\mu \models R_{\mathcal{X}}$ and, hence, $\mu \in [\![(Q_1^\star \text{ FILTER } R_{\mathcal{X}})]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. From this and the fact that $\mu \notin [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we conclude that:

  $$\mu \in [\![((Q_1^\star \text{ FILTER } R_{\mathcal{X}}) \text{ MINUS } (Q_2^\star \text{ FILTER } R_{\mathcal{X}}))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \quad (\ddagger)$$

  To see why this is the case, assume that ($\ddagger$) does not hold. Then

given that $\mu \in [\![(Q_1^\star \text{ FILTER } R_{\mathcal{X}})]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we conclude by definition of the operator MINUS that there exists a mapping $\mu' \in [\![(Q_2^\star \text{ FILTER } R_{\mathcal{X}})]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $\mu \sim \mu'$ and $(\text{dom}(\mu) \cap \text{dom}(\mu')) \neq \emptyset$. Given that $\mu' \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we have that $\text{dom}(\mu') \subseteq \{?A_1, \ldots, ?A_\ell\}$. Thus, given that $\mu' \models R_{\mathcal{X}}$ and $\text{dom}(\mu) \subseteq \{?A_1, \ldots, ?A_\ell\}$, we conclude that $\text{dom}(\mu) = \text{dom}(\mu')$. Therefore, given that $\mu \sim \mu'$, we have that $\mu = \mu'$, from which we conclude that $\mu \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, leading to a contradiction.

From (‡) and definition of $Q^\star$, we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ since $((Q_1^\star \text{ FILTER } R_{\mathcal{X}}) \text{ MINUS } (Q_2^\star \text{ FILTER } R_{\mathcal{X}})) = P_i$ for some $i \in \{1, \ldots, 2^\ell - 1\}$ (recall that $\mathcal{X} \neq \emptyset$).

– Assume that $\text{dom}(\mu) = \emptyset$. Then we have that $\mu \models R_\emptyset$ and, hence, $\mu \in [\![(Q_1^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. From this and the fact that $\mu \notin [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we conclude that:

$$\mu \in [\![\Big[\Big[\Big(Q_1^\star \text{ FILTER } R_\emptyset\Big) \text{ OPT } \Big(\Big(Q_2^\star \text{ FILTER } R_\emptyset\Big) \text{ AND } (?X, ?Y, ?Z)\Big)\Big] \quad (*)$$
$$\text{FILTER } (\neg\, \text{bound}(?X))\Big]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \quad (7.1)$$

To see why this is the case, assume that $(*)$ does not hold. Then given that $\mu \in [\![(Q_1^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\text{dom}(\mu) = \emptyset$, we have that there exists a mapping $\mu' \in [\![((Q_2^\star \text{ FILTER } R_\emptyset) \text{ AND } (?X, ?Y, ?Z))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $?X \in \text{dom}(\mu')$. Thus, there exist mappings $\mu_1 \in [\![(Q_2^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu_2 \in [\![(?X, ?Y, ?Z)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $\mu' = \mu_1 \cup \mu_2$. Given that $\mu_1 \in [\![(Q_2^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, we have that $\mu_1 \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu_1 \models R_\emptyset$. Thus, we have that $\text{dom}(\mu_1) \subseteq \{?A_1, \ldots, ?A_\ell\}$, from

which we conclude that $\text{dom}(\mu_1) = \emptyset$ (since $\mu_1 \models R_\emptyset$). Therefore, we have that $\mu = \mu_1$, which implies that $\mu \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and leads to a contradiction.

From $(*)$ and definition of $Q^\star$, we conclude that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$.

Second, we show that $[\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} \subseteq tr([\![Q]\!]_I)$. Assume that $\mu \in [\![Q^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then we consider two cases to prove that $\mu \in tr([\![Q]\!]_I)$.

- Assume that there exists $i \in \{1, \ldots, \ell\}$ such that $\mu \in [\![P_i]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then there exists $\mathcal{X} \neq \emptyset$ such that
  $\mu \in [\![((Q_1^\star \text{ FILTER } R_\mathcal{X}) \text{ MINUS } (Q_2^\star \text{ FILTER } R_\mathcal{X}))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Thus, we have that $\mu \in [\![Q_1]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \models R_\mathcal{X}$, from which we conclude that $\emptyset \subsetneq \text{dom}(\mu) \subseteq \{?A_1, \ldots, ?A_\ell\}$. From this fact and definition of the MINUS operator, we obtain that $\mu \notin [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Hence, by induction hypothesis, we conclude that $\mu \in tr([\![Q_1]\!]_I)$ and $\mu \notin tr([\![Q_2]\!]_I)$. That is, there exists a tuple t such that $tr(t) = \mu$, $t \in [\![Q_1]\!]_I$ and $t \notin [\![Q_2]\!]_I$, from which we conclude that $\mu \in tr([\![Q]\!]_I)$.
- Assume that $(*)$ holds. First we show that $[\![(Q_2^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} = \emptyset$. For the sake of contradiction, assume that there exists a mapping $\mu' \in [\![(Q_2^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Then given that $\mu' \in [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \models R_\emptyset$, we conclude that $\text{dom}(\mu') = \emptyset$. Given that $\mathcal{DM}(\mathbf{R}, \Sigma, I)$ is a nonempty RDF graph and $\text{dom}(\mu') = \emptyset$, we conclude that there exists a mapping $\mu'' \in [\![((Q_2^\star \text{ FILTER } R_\emptyset) \text{ AND } (?X, ?Y, ?Z))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ such that $\text{dom}(\mu'') = \{?X, ?Y, ?Z\}$. Thus, given that variables $?X$, $?Y$, $?Z$ are not mentioned in $(Q_1^\star \text{ FILTER } R_\emptyset)$, we conclude that μ'' is compatible with every mapping in $[\![(Q_1^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Thus, by definition of the OPT operator, we conclude that $?X$ belongs to the domain of every mapping in

$[\![((Q_1^\star \text{ FILTER } R_\emptyset) \text{ OPT } ((Q_2^\star \text{ FILTER } R_\emptyset) \text{ AND } (?X, ?Y, ?Z)))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$, which implies that

$[\![(((Q_1^\star \text{ FILTER } R_\emptyset) \text{ OPT } ((Q_2^\star \text{ FILTER } R_\emptyset) \text{ AND } (?X, ?Y, ?Z))) \text{ FILTER } (\neg \text{bound}(?X)))]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} = \emptyset$. But this leads to a contradiction, as we assume that $(*)$ holds.

Given that $(*)$ holds and $[\![(Q_2^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)} = \emptyset$, we conclude that $\mu \in [\![(Q_1^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \notin [\![(Q_2^\star \text{ FILTER } R_\emptyset)]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$. Hence, we have that $\mu \in [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and $\mu \notin [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}$ and, therefore, we conclude by induction hypothesis that $\mu \in tr([\![Q_1]\!]_I)$ and $\mu \notin tr([\![Q_2]\!]_I)$. That is, there exists a tuple t such that $tr(t) = \mu$, $t \in [\![Q_1]\!]_I$ and $t \notin [\![Q_2]\!]_I$, from which we conclude that $\mu \in tr([\![Q]\!]_I)$.

7.3.3 Proof of Proposition 1

Assume that we have a relational schema containing a relation with name `STUDENT` and attributes `SID`, `NAME`, and assume that the attribute `SID` is the primary key. Moreover, assume that this relation has two tuples, t_1 and t_2 such that t_1.`SID` $= 1$, t_1.`NAME` $=$ John and t_2.`SID` $= 1$, t_2.`NAME` $=$ Peter. It is clear that the primary key is violated, therefore the database is inconsistent. If $\mathcal{DM}$ would be semantics preserving, then the resulting RDF graph would be inconsistent under OWL semantics. However, the result of applying $\mathcal{DM}$, returns the following consistent RDF graph (assuming given a base IRI `"http://example.edu/db/"` for the mapping):

TRIPLE("http://example.edu/db/STUDENT", "rdf:type", "owl:Class")

TRIPLE("http://example.edu/db/STUDENT#NAME", "rdf:type", "owl:DatatypeProperty")

TRIPLE("http://example.edu/db/STUDENT#NAME", "rdfs:domain",
"http://example.edu/db/STUDENT")

TRIPLE("http://example.edu/db/STUDENT#SID", "rdf:type", "owl:DatatypeProperty")

TRIPLE("http://example.edu/db/STUDENT#SID", "rdfs:domain",
"http://example.edu/db/STUDENT")

TRIPLE("http://example.edu/db/STUDENT#SID=1",
"http://example.edu/db/STUDENT#NAME", "John")

TRIPLE("http://example.edu/db/STUDENT#SID=1",
"http://example.edu/db/STUDENT#NAME", "Peter")

TRIPLE("http://example.edu/db/STUDENT#SID=1",
"http://example.edu/db/STUDENT#SID", "1")

Therefore, $\mathcal{DM}$ is not semantics preserving.

7.3.4 Proof of Proposition 2

It is straightforward to see that given a relational schema $\mathbf{R}$, set Σ of (only) PKs over $\mathbf{R}$ and instance I of $\mathbf{R}$ such that $I \models \Sigma$, it holds that $\mathcal{DM}_{pk}(\mathbf{R}, \Sigma, I)$ is consistent under the OWL semantics. Likewise, if $I \not\models \Sigma$, then by definition of $\mathcal{DM}_{pk}$, the resulting RDF graph will have an inconsistent triple TRIPLE(a, "owl:differentFrom", a), which would generate an inconsistency under the OWL semantics.

7.3.5 Proof of Theorem 4

For the sake of contradiction, assume that $\mathcal{M}$ is a monotone and semantics preserving direct mapping. Then consider a schema $\mathbf{R}$ containing at least two distinct

relation names R_1, R_2, and consider a set Σ of PKs and FKs over $\mathbf{R}$ containing at least one foreign key from R_1 to R_2. Then we have that there exist instances I_1, I_2 of $\mathbf{R}$ such that $I_1 \subseteq I_2$, I_1 does not satisfy Σ and I_2 does satisfy Σ. Given that $\mathcal{M}$ is semantics preserving, we know that $\mathcal{M}(\mathbf{R}, \Sigma, I_2)$ is consistent under the OWL semantics, while $\mathcal{M}(\mathbf{R}, \Sigma, I_1)$ is not. Given that $\mathcal{M}$ is monotone, we have that $\mathcal{M}(\mathbf{R}, \Sigma, I_1) \subseteq \mathcal{M}(\mathbf{R}, \Sigma, I_2)$. But then we conclude that $\mathcal{M}(\mathbf{R}, \Sigma, I_1)$ is also consistent under the OWL semantics, given that $\mathcal{M}(\mathbf{R}, \Sigma, I_2)$ is consistent and $\mathcal{M}(\mathbf{R}, \Sigma, I_1) \subseteq \mathcal{M}(\mathbf{R}, \Sigma, I_2)$, which leads to a contradiction.

7.3.6 Proof of Theorem 5

It is straightforward to see that given a relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and instance I of $\mathbf{R}$ such that $I \models \Sigma$, it holds that $\mathcal{DM}_{pk+fk}(\mathbf{R}, \Sigma, I)$ is consistent under the OWL semantics. Likewise, if $I \not\models \Sigma$, then by definition of $\mathcal{DM}_{pk+fk}$, the resulting RDF graph will contain an inconsistent triple Triple(a, `"owl:differentFrom"`, a), which would generate an inconsistency under the OWL semantics.

7.4 Benchmark for Ultrawrap

Details of the BSBM and DBLP benchmark used to evaluate Ultrawrap can be found `http://ribs.csres.utexas.edu/ultrawrap/benchmark`.

7.5 Benchmark for Ultrawrap$^{\mathbf{OBDA}}$

Details of the Texas Benchmark and the extension to the BSBM benchmark used to evaluate Ultrawrap$^{\text{OBDA}}$can be found `http://www.obda-benchmark.org/`.

Bibliography

[1] D2RQ accessing relational databases as virtual rdf graphs. `http://d2rq.org/`.

[2] Virtuoso rdf views. `http://virtuoso.openlinksw.com/whitepapers/relational\%20rdf\%20views\%20mapping.html`.

[3] D. J. Abadi, S. Madden, and N. Hachem. Column-stores vs. row-stores: how different are they really? In *SIGMOD Conference*, pages 967–980, 2008.

[4] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 411–422, 2007.

[5] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Using the barton libraries dataset as an rdf benchmark. Technical Report MIT-CSAIL-TR-2007-036, MIT, 2007.

[6] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[7] D. Allemang and J. A. Hendler. *Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL, Second Edition*. Morgan Kaufmann, 2011.

[8] R. Angles and C. Gutierrez. The expressive power of sparql. In *International Semantic Web Conference*, pages 114–129, 2008.

[9] R. Angles and C. Gutierrez. The expressive power of sparql. In *ISWC*, pages 114–129, 2008.

[10] R. Angles and C. Gutierrez. SQL nested queries in SPARQL. In *AMW*, 2010.

[11] R. Angles and C. Gutierrez. Subqueries in SPARQL. In *AMW*, 2011.

[12] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

[13] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. Direct mapping of relational data to RDF. W3C Recomendation 27 September 2012, `http://www.w3.org/TR/rdb-direct-mapping/`.

[14] F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. In *IJCAI*, 2005.

[15] F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors. *Reasoning about Structured Objects: Knowledge Representation Meets Databases, Proceedings of 1st Workshop KRDB'94, Saarbrücken, Germany, September 20-22, 1994*, volume 1 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1994.

[16] F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors. *KRDB-95: Reasoning about Structured Objects: Knowledge Representation Meets Databases, Proceedings of the 2nd Workshop KRDB'95, Bielefeld, Germany, Septtember 11-12, 1995*, volume 2 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1995.

[17] F. Baader, M. Buchheit, M. A. Jeusfeld, and W. Nutt, editors. *Knowledge Representation Meets Databases, Proceedings of the 3rd Workshop KRDB'96,*

Budapest, Hungary, August 13, 1996, volume 4 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1996.

[18] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[19] F. Baader and B. Hollunder. KRIS: knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[20] F. Baader, M. A. Jeusfeld, and W. Nutt, editors. *Intelligent Access to Heterogeneous Information, Proceedings of the 4th Workshop KRDB-97, Athens, Greece, August 30, 1997*, volume 8 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1997.

[21] F. Barbançon and D. P. Miranker. Implementing federated databases systems by compiling schemasql. In *IDEAS*, pages 192–201, 2002.

[22] R. G. Bello, K. Dias, A. Downing, J. J. F. Jr., J. L. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in oracle. In *VLDB*, pages 659–664, 1998.

[23] C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[24] A. Borgida, D. Calvanese, L. Cholvy, and M.-C. Rousset, editors. *Proceedings of the 9th International Workshop on Knowledge Representation meets Databases (KRDB 2002), Toulouse France, April 21, 2002*, volume 54 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2002.

[25] A. Borgida, V. K. Chaudhri, and M. Staudt. Krdb '98: The 5th international workshop on knowledge representation meets databases. *SIGMOD Record*, 27(3):10–15, 1998.

[26] A. Borgida, V. K. Chaudhri, and M. Staudt, editors. *Proceedings of the 5th International Workshop on Knowledge Represenation Meets Databases (KRDB '98): Innovative Application Programming and Query Interfaces, Seattle, Washington, USA, May 31, 1998*, volume 10 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

[27] M. Bouzeghoub, M. Klusch, W. Nutt, and U. Sattler, editors. *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB 2000), Berlin, Germany, August 21, 2000*, volume 29 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2000.

[28] R. J. Brachman and H. J. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1985.

[29] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, L. A. Resnick, L. A. Resnick, and A. Borgida. Living with classic: When and how to use a kl-one-like language. In *Principles of Semantic Networks*, pages 401–456, 1991.

[30] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[31] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema, W3C recommendation, February 2004.

[32] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *ISWC*, pages 54–68, 2002.

[33] N. Bruno. Teaching an old elephant new tricks. In *CIDR*, 2009.

[34] F. Bry, C. Lutz, U. Sattler, and M. Schoop, editors. *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases*

(KRDB 2003), Hamburg, Germany, September 15-16, 2003, volume 79 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[35] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artif. Intell.*, 195:335–360, 2013.

[36] D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. Autom. Reason.*, 39(3):385–429, Oct. 2007.

[37] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Eql-lite: Effective first-order query processing in description logics. In *IJCAI*, pages 274–279, 2007.

[38] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer, 1990.

[39] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15(2):162–207, June 1990.

[40] A. Chebotko, S. Lu, and F. Fotouhi. Semantics preserving sparql-to-sql translation. *Data Knowl. Eng.*, 68(10):973–1000, 2009.

[41] Q. Cheng, J. Gryz, F. Koo, T. Y. C. Leung, L. Liu, X. Qian, and K. B. Schiefer. Implementation of two semantic query optimization techniques in db2 universal database. In *VLDB*, pages 687–698, 1999.

[42] R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3):216–237, Nov. 2002.

[43] J. Chomicki, editor. *Proceedings of the Workshop on Deductive Databases held in conjunction with the North American Conference on Logic Programming, Austin, Texas, USA, November 1, 1990*, volume TR-CS-90-14 of *Technical Report*. Kansas State University, 1990.

[44] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient sql-based rdf querying scheme. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 1216–1227, 2005.

[45] A. Chortaras, D. Trivela, and G. B. Stamou. Optimized query rewriting for owl 2 ql. In *CADE*, 2011.

[46] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[47] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. T. Groth, A. Haque, A. Harth, F. L. Keppmann, D. P. Miranker, J. Sequeda, and M. Wylot. Nosql databases for rdf: An empirical evaluation. In *International Semantic Web Conference*, pages 310–325, 2013.

[48] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[49] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF mapping language. W3C Recomendation 27 September 2012, `http://www.w3.org/TR/r2rml/`.

[50] H. Decker, U. Geske, A. C. Kakas, C. Sakama, D. Seipel, and T. Urpí, editors. *Deductive Databases and Logic Programming, Abduction in Deductive Databases and Knowledge-Based Systems, Proceedings of the ICLP'95 Joint Workshop, Shonan Village Center, Japan, June 17, 1995*, volume 266

of *GMD-Studien*. Gesellschaft für Mathematik und Datenverarbeitung MbH, 1995.

[51] D. J. DeWitt. The wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook*, pages 119–165. 1991.

[52] F. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.

[53] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. Comput. Log.*, 3(2):177–225, 2002.

[54] T. Eiter, M. Ortiz, M. Simkus, T.-K. Tran, and G. Xiao. Query rewriting for horn-shiq plus rules. In *AAAI*, 2012.

[55] 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, 2009.

[56] E. Franconi and M. Kifer, editors. *Proceedings of the 6th International Workshop on Knowledge Representation meets Databases (KRDB'99), Linköping, Sweden, July 29-30, 1999*, volume 21 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1999.

[57] C. Franke, S. Morin, A. Chebotko, J. Abraham, and P. Brazier. Distributed semantic web data management in hbase and mysql cluster. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, pages 105–112, 2011.

[58] P. Fraternali, U. Geske, C. Ruiz, and D. Seipel, editors. *Proceedings of the 6th International Workshop on Deductive Databases and Logic Programming (DDLP'98). In Conjunction with JICSLP'98*, volume 22 of *GMD Report*, 1998.

[59] H. Gallaire and J. Minker, editors. *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, 1977*, Advances in Data Base Theory, New York, 1978. Plemum Press.

[60] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.

[61] U. Geske, C. Ruiz, and D. Seipel, editors. *Proceedings of the Fifth International Workshop on Deductive Databases and Logic Programming, DDLP'97, Leuven, Belgium, July 1997. Accepted Papers*. GMD - German National Research Center for Information Technology, 1997.

[62] U. Geske and D. Seipel, editors. *Proceedings of the Workshop on Deductive Databases and Logic Programming, Second ICLP-Workshop on Deductive Databases, Santa Marherita Ligure, Italy, June 17, 1994*, volume 231 of *GMD-Studien*. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.

[63] B. Glimm, A. Hogan, M. Krotzsch, and A. Polleres. OWL-LD. `http://semanticweb.org/OWLLD/`.

[64] J. Goldstein and P.-Å. Larson. Optimizing queries using materialized views: A practical, scalable solution. In *SIGMOD*, pages 331–342, 2001.

[65] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, 2011.

[66] A. J. Gray, N. Gray, and I. Ounis. Can rdb2rdf tools feasibily expose large science archives for data integration? In *ESWC*, pages 491–505, 2009.

[67] S. Grimm and B. Motik. Closed world reasoning in the semantic web through epistemic operators. In *OWLED*, 2005.

[68] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003.

[69] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[70] A. Gupta and I. S. Mumick. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, 1999.

[71] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[72] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.

[73] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C Recommendation 21 March 2013, `http://www.w3.org/TR/sparql11-query/`.

[74] P. J. Hayes. The logic of frames. *Frame Conceptions and Text Understandings*, pages 46–61, 1979.

[75] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50:94–101, May 2007.

[76] J. Hendler. RDFS 3.0. In *W3C Workshop - RDF Next Steps*, 2010.

[77] W. D. Hillis. *The Connection Machine*. MIT Press, 1989.

[78] I. Horrocks. Using an expressive description logic: Fact or fiction? In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pages 636–649, 1998.

[79] J. Huang, D. J. Abadi, and K. Ren. Scalable sparql querying of large rdf graphs. *PVLDB*, 4(11):1123–1134, 2011.

[80] H. Itoh. Research and development on knowledge bases systems at icot. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 437–445, 1986.

[81] L. Kerschberg. Expert database systems (workshop rewiew). In *SIGMOD Conference*, pages 414–417, 1985.

[82] L. Kerschberg, editor. *Expert Database Systems, Proceedings From the First international Workshop, Kiawah Island, South Carolina, USA, October 24-27, 1984*. Benjamin/Cummings, 1986.

[83] L. Kerschberg, editor. *Expert Database Systems, Proceedings From the First International Conference, Charleston, South Carolina, USA, April 1-4, 1986*. Benjamin/Cummings, 1987.

[84] W. Kim. Object-oriented databases: Definition and research directions. *IEEE Trans. on Knowl. and Data Eng.*, 2(3):327–341, Sept. 1990.

[85] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in dl-lite. In *KR*, 2010.

[86] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to ontology-based data access. In *IJCAI*, pages 2656–2661, 2011.

[87] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *SIGMOD Conference*, pages 40–49, 1991.

[88] G. Ladwig and A. Harth. Cumulusrdf: Linked data management on nested key-value stores. In *7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*, 2011.

[89] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. Schemasql - a language for interoperability in relational multi-database systems. In *VLDB*, pages 239–250, 1996.

[90] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[91] M. Lenzerini, D. Nardi, W. Nutt, and D. Suciu, editors. *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases (KRDB 2001), Rome, Italy, September 15, 2001*, volume 45 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[92] C. Lutz, I. Seylan, D. Toman, and F. Wolter. The combined approach to obda: Taming role hierarchies using filters. In *ISWC*, pages 314–330, 2013.

[93] R. M. MacGregor. Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.

[94] H. MahmoudiNasab and S. Sakr. An experimental evaluation of relational rdf storage and querying techniques. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications*, pages 215–226, 2010.

[95] I. Mami and Z. Bellahsene. A survey of view selection methods. *SIGMOD Rec.*, 41(1):20–29, Apr. 2012.

[96] A. Mehdi, S. Rudolph, and S. Grimm. Epistemic querying of owl knowledge bases. In *ESWC (1)*, pages 397–409, 2011.

[97] M. Minsky. A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[98] K. Morfonios, S. Konakas, Y. Ioannidis, and N. Kotsis. Rolap implementations of the data cube. *ACM Comput. Surv.*, 39(4), Nov. 2007.

[99] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, and A. F. amd Carsten Lutz. Owl 2 web ontology language profiles (second edition), W3C recommendation, December 2012.

[100] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between owl and relational databases. *J. Web Sem.*, 7(2):74–89, 2009.

[101] S. Muñoz, J. Pérez, and C. Gutierrez. Simple and efficient minimal rdfs. *J. Web Sem.*, 7(3):220–234, 2009.

[102] T. Neumann and G. Weikum. The rdf-3x engine for scalable management of rdf data. *VLDB J.*, 19(1):91–113, 2010.

[103] M. Ortiz and M. Simkus. Reasoning and query answering in description logics. In *Reasoning Web*, pages 1–53, 2012.

[104] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[105] H. Pérez-Urbina, I. Horrocks, and B. Motik. Efficient query answering for owl 2. In *ISWC*, 2009.

[106] F. D. Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *EDBT*, 2013.

[107] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[108] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, `http://www.w3.org/TR/rdf-sparql-query/`.

[109] R. M. Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430, 1967.

[110] R. Ramakrishnan, editor. *Proceedings of the Workshop on Programming with Logic Databases. In Conjunction with ILPS, Vancouver, B.C., October 30, 1993*, volume #1183 of *Technical Report*. University of Wisconsin, 1993.

[111] K. Ramamohanarao, J. Harland, and G. Dong, editors. *Proceedings of the Workshop on Deductive Databases held in conjunction with the Joint International Conference and Symposium on Logic Programming, Washington, D.C., USA, Saturday, November 14, 1992*, volume CITRI/TR-92-65 of *Technical Report*. Department of Computer Science, University of Melbourne, 1992.

[112] R. Reiter. On integrity constraints. In *TARK*, pages 97–111, 1988.

[113] M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *AMW*, 2011.

[114] M. Rodríguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *ISWC*, pages 558–573, 2013.

[115] R. Rosati and A. Almatelli. Improving query answering over dl-lite ontologies. In *KR*, 2010.

[116] M. Schmidt, T. Hornung, N. Küchlin, G. Lausen, and C. Pinkel. An experimental comparison of rdf data management approaches in a sparql benchmark scenario. In *International Semantic Web Conference*, pages 82–97, 2008.

[117] A. Seaborne, D. Steer, and S. Williams. Sql-rdf. http://www.w3.org/2007/03/RdfRDB/papers/seaborne.html.

[118] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to rdf and owl. In *WWW*, pages 649–658, 2012.

[119] J. F. Sequeda, S. H. Tirmizi, O. Corcho, and D. P. Miranker. Survey of directly mapping sql databases to the semantic web. *Knowledge Eng. Review*, 26(4):445–486, 2011.

[120] J. F. Sequeda, S. H. Tirmizi, and D. P. Miranker. SQL databases are a moving target. In *W3C Workshop on RDF Access to Relational Databases*, 2007.

[121] S. T. Shenoy and Z. M. Ozsoyoglu. A system for semantic query optimization. In *SIGMOD*, pages 181–195, 1987.

[122] L. Sidirourgos, R. Goncalves, M. L. Kersten, N. Nes, and S. Manegold. Column-store support for rdf data management: not all swans are white. *PVLDB*, 1(2):1553–1563, 2008.

[123] S. J. Stolfo and D. P. Miranker. The dado production system machine. *J. Parallel Distrib. Comput.*, 3(2):269–296, June 1986.

[124] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in data base systems. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990.*, pages 281–290, 1990.

[125] M. Stonebraker, J. Woodfill, and E. Andersen. Implementation of rules in relational data base systems. *IEEE Database Eng. Bull.*, 6(4):65–74, 1983.

[126] M. Svihla and I. Jelínek. Benchmarking rdf production tools. In *DEXA*, pages 700–709, 2007.

[127] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in owl. In *AAAI*, 2010.

[128] S. H. Tirmizi, S. Aitken, D. A. Moreira, C. Mungall, J. Sequeda, N. H. Shah, and D. P. Miranker. Mapping between the obo and owl ontology languages. *J. Biomedical Semantics*, 2(S-1):S3, 2011.

[129] S. H. Tirmizi, J. Sequeda, and D. P. Miranker. Translating SQL Applications to the Semantic Web. In *DEXA*, pages 450–464, 2008.

[130] T. Venetis, G. Stoilos, and G. B. Stamou. Incremental query rewriting for owl 2 ql. In *Description Logics*, 2012.

[131] J. Weaver and J. A. Hendler. Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *International Semantic Web Conference*, pages 682–697, 2009.

[132] C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1(1):1008–1019, Aug. 2008.

[133] K. Wilkinson. Jena property table implementation. Technical Report HPL-2006-140, HP Laboratories, 2006.

[134] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex sql queries using automatic summary tables. In *SIGMOD*, pages 105–116, 2000.

Vita

Juan Sequeda was born in San Jose, California. He started to pursue a Computer Science degree at the Universidad del Valle in Cali, Colombia in 2003. He transfer to the University of Texas at Austin in 2006 and obtained a Bachelor of Science in Computer Science in 2008. He enrolled in the PhD program at the University of Texas at Austin in 2008.

Permanent Address: juanfederico@gmail.com

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.