Efficient Querying and Analytics of Semantic Web Data

Alexandra Roatis

Université Paris-Sud

École Doctorale 427 : Informatique Paris-Sud

Laboratoire de Recherche en Informatique (LRI)

Discipline : Informatique

Thèse de doctorat

Soutenue le **22 Septembre 2014** par

Alexandra ROATIŞ

Efficient Querying and Analytics of Semantic Web Data

Thèse dirigée par :

Mme. Ioana Manolescu	*Inria Saclay*
M. François Goasdoué	*Université Rennes 1*
M. Dario Colazzo	*Université Paris-Dauphine*

Rapporteurs :

M. Alon Halevy	*Google Research*
M. Frank van Harmelen	*Vrije Universiteit Amsterdam*

Examinateurs :

M. Serge Abiteboul	*Inria Saclay & ENS Cachan*
Mme. Christine Froidevaux	*Université Paris-Sud*
M. Philippe Rigaux	*CNAM Paris*

"Anyone who has lost track of time when using a computer
knows the propensity to dream,
the urge to make dreams come true
and the tendency to miss lunch."

Tim Berners-Lee

Résumé

L'utilité et la pertinence des données se trouvent dans l'information qui peut en être extraite. Le taux élevé de publication des données et leur complexité accrue, par exemple dans le cas des données du Web sémantique autodescriptives et hétérogènes, motivent l'intérêt de techniques efficaces pour la manipulation de données. Dans cette thèse, nous utilisons la technologie mature de gestion de données relationnelles pour l'interrogation des données du Web sémantique.

La première partie se concentre sur l'apport de réponse aux requêtes sur les données soumises à des contraintes RDFS, stockées dans un système de gestion de données relationnelles. L'information implicite, résultant du raisonnement RDF est nécessaire pour répondre correctement à ces requêtes. Nous introduisons le fragment des bases de données RDF, allant au-delà de l'expressivité des fragments étudiés précédemment. Nous élaborons de nouvelles techniques pour répondre aux requêtes dans ce fragment, en étendant deux approches connues de manipulation de données sémantiques RDF, notamment par saturation de graphes et reformulation de requêtes. En particulier, nous considérons les mises à jour de graphe au sein de chaque approche et proposerons un procédé incrémental de maintenance de saturation. Nous étudions expérimentalement les performances de nos techniques, pouvant être déployées au-dessus de tout moteur de gestion de données relationnelles.

La deuxième partie de cette thèse considère les nouvelles exigences pour les outils et méthodes d'analyse de données, issues de l'évolution du Web sémantique. Nous revisitons intégralement les concepts et les outils pour l'analyse de données, dans le contexte de RDF. Nous proposons le premier cadre formel pour l'analyse d'entrepôts RDF. Notamment, nous définissons des schémas analytiques adaptés aux graphes RDF hétérogènes à sémantique riche, des requêtes analytiques qui (au-delà de cubes relationnels) permettent l'interrogation flexible des données et schémas, ainsi que des opérations d'agrégation puissantes de type OLAP. Des expériences sur une plateforme entièrement implémentée démontrent l'intérêt pratique de notre approche.

Mots clés : RDF, réponse aux requêtes, raisonnement, entrepôt de données, OLAP

Abstract

The utility and relevance of data lie in the information that can be extracted from it. The high rate of data publication and its increased complexity, for instance the heterogeneous, self-describing Semantic Web data, motivate the interest in efficient techniques for data manipulation. In this thesis we leverage mature relational data management technology for querying Semantic Web data.

The first part focuses on query answering over data subject to RDFS constraints, stored in relational data management systems. The implicit information resulting from RDF reasoning is required to correctly answer such queries. We introduce the database fragment of RDF, going beyond the expressive power of previously studied fragments. We devise novel techniques for answering Basic Graph Pattern queries within this fragment, exploring the two established approaches for handling RDF semantics, namely graph saturation and query reformulation. In particular, we consider graph updates within each approach and propose a method for incrementally maintaining the saturation. We experimentally study the performance trade-offs of our techniques, which can be deployed on top of any relational data management engine.

The second part of this thesis considers the new requirements for data analytics tools and methods emerging from the development of the Semantic Web. We fully redesign, from the bottom up, core data analytics concepts and tools in the context of RDF data. We propose the first complete formal framework for warehouse-style RDF analytics. Notably, we define analytical schemas tailored to heterogeneous, semantic-rich RDF graphs, analytical queries which (beyond relational cubes) allow flexible querying of the data and the schema as well as powerful aggregation and OLAP-style operations. Experiments on a fully-implemented platform demonstrate the practical interest of our approach.

Keywords: RDF, query answering, reasoning, data warehouse, OLAP

Acknowledgements

I wish I could say that writing this section has allowed me to finally put into words my feelings of gratitude towards all the wonderful people who have shaped me into the person I am today. I am still struggling for the right way to say it and I guess that, as any thesis, this part is still full of future potential. While I cannot justly acknowledge now the rippling effect that their influence will have on my life, I feel the impact has been consequential and I strive here to express my deeply felt gratitude.

I wish to start by thanking my three thesis advisors for the parental care that they have invested in my education and development.

Dear Ioana, I honestly don't know how to thank you enough for giving me this opportunity, for teaching me so many things relating to research and to life in general, and for opening the door to countless future prospects. I would like you to know that the biggest compliment I have ever received is being compared to you and that I am striving every day to become worthy of it. I truly admire your uncanny ability to see beyond the spoken words, to understand people's intentions and the meaning behind their words, to make the right comment and ask the most pertinent question at the ideal time. I find your work ethic inspirational and I hope to learn to invest the same determination and discipline in my work as you do every day.

Dear François, I cannot express what your advice has meant to me. I have always been in awe of your opinion and every conversation we had during these three years has filled me with a passionate interest for science and learning. I now examine and write every formalism through your perspective, aiming at clarity and making sure to leave nothing undefined, and I am truly proud of my work only when it has gotten your stamp of approval. I profoundly appreciate the patience with which you have guided me through this learning process. I hope to have learned some of it as well and to apply the same composure in my advice to others.

Dear Dario, I cannot thank you enough for your kindness and trust. Starting from my first teaching assignment and till the very end of my PhD I have always felt your confidence in my abilities and it has motivated me to aim higher and higher. Your support has encouraged me to have faith in myself. Your belief in my choices has been of consequential importance to my future steps, and I deeply appreciate it.

I want to thank Alon Halevy and Frank van Harmelen for reviewing my PhD thesis, and Serge Abiteboul, Christine Froidevaux and Philippe Rigaux for accepting to be part of my PhD jury. I highly appreciate all of your comments, questions, advice and kind encouragements.

I also wish to express my gratitude towards my Bachelor's thesis advisor, Daniela Zaharie. Thank you for the knowledge you have imparted on me, for teaching me to love algorithms, and for putting me on the path of research.

I want to thank all the people that I had the privilege and pleasure to interact with on a professional and personal level during these three years. I warmly mention here the Inria OAK team members (formerly Leo and Gemo). I thank you for the great work environment and your friendship during these years.

To Nicole, I would like to say that I have always admired you. In my opinion, you are the definition of what a professor should be, irreproachable in appearance and extremely kind at heart. I hope to some day reach the high standard you have set.

Dear Melanie, I truly appreciate your friendship and I deeply admire your approach to life and your dedication to both work and family. I hope to one day be able to follow suite. Also, thank you for allowing me to act as teaching assistant for your database classes and to learn from the great example you set.

I thank Philippe Chatalic and Laurent Simon for trusting me as teaching assistant for their artificial intelligence course for the past two years. And I thank Camille and Jean-Baptiste for the collaboration on preparing the classes and assignments during the second year and the friendship that resulted from this collaboration.

I thank Xavier and Michèle for supervising my mission doctorale during the first year of PhD and I am grateful to all the people in the Autoportrait team for their collaboration and advice on that project.

For Alin, my former colleague and first friend in France, I have a question. How did you manage to tolerate all my questions? I am very grateful for all the help and advice you gave me when I started my first internship here. Your aid was invaluable in navigating the "misteries" of command lines and accessing servers. Thank you for all of it and for always offering your help with a smile!

To my first office mates, Stamatis and Francesca: it's been such a long time since we each did our first internship in the team. Sharing the office with you was a real pleasure. However the even bigger pleasure was getting to know you during these years. Stamatis, thank you for the friendly attitude you have always shown me and for inspiring me to lead a healthier life. I finally joined a gym! Dear Francesca, thank you for becoming the friend that I was missing ever since I left Romania, for listening to my troubles, for the advice you gave me and for filling my mailbox with postcards. It is a privilege to call you my friend!

To Asterios and Jesús, with whom I shared the office for the duration of my PhD: I thank you for welcoming me in your little clique, for always providing solutions to my questions and advice when I needed it. It has been a pleasure to briefly work with you and to learn from you. Even more, I thank you for all the chats and jokes we have shared and the smiles they have triggered. I greatly enjoyed spending each work day in your company.

To my newest office mates, Elham and Katerina: even though we have shared the office only for a little time, it has been a real pleasure so far. I am loving the friendly, one may even say girly, environment we have. Your lovely smiles offer a motivational boost to each day. Dear Elham, I also want to add that it has been a real pleasure to co-supervise your internship so far, and I want to thank you for becoming my friend in the process. You make even a tough work day pass by smoothly and end in a smile.

I thank Šejla and Damian for the work collaboration. I have learned a lot from both of you. Danai, thank you for being the friend in need. I really appreciate it.

Gianluca, Tushar and Benjamin, thank you for being not only my colleagues, but also good friends and for bringing an unconventional dose of fun to the work environment. Gianluca, I appreciate your help through the bureaucratic mess of renting an apartment. Tushar, thank you for collaborating with me on the WaRG demo and creating the beautiful visual interface for the analytics framework we developed. And Benjamin, I am very grateful for all the aid you have given me in writing proper French.

I wish to thank Maëva for the dedicated manner with which she has handled all the paperwork for missions and for her help through many bureaucratic procedures.

I share my deep gratitude towards everyone from Lannion, for making my stay there so enjoyable, that I look for every opportunity to visit them again.

I wish to thank all my friends from Romania, in particular Alina, Cipi, Andrei, Rosi, Codruţa, Anda and Oana for always welcoming me with open arms when I visit and for their continued support during these years. I miss you all dearly! Thank you Driss for being the first friend I made at a conference and for proving that such friendships are meant to last. Dear Fredrik, thank you for your friendship, support and for all the great times we spent together.

I thank Ştefania for bringing me the feeling of home and for becoming my close friend. I am very grateful to you for keeping me alive (with home cooked meals and pizza) and well caffeinated while I was writing my thesis. And I thank you for all the conversations and all the laughs we have had since we met. You are a real comfort and a valuable friend.

Dear Steven, you have been the best of friends ever since I have uprooted my life to this strange new country. Thank you for being the person I could always count on to listen to my troubles and to remind me that whatever happens and wherever I go, I should not forget my smile. And most of all, thank you for the music!

Dear Carmen, I am so proud of you! As the older sibling I should try to be the role model, but the truth is that you are my inspiration. Thank you for your honesty, care and love. I know that wherever life takes us, I can always count on you.

I finish by expressing my deep gratitude towards my parents. Thank you for the way you raised me, for all the sacrifices you have made along the way and for your unending love. I know how difficult it has been for you to see me move to a new country and I thank you for letting me go. I will always do my best to make you proud.

Finally, I want to thank serendipity for putting me in the path of all these wonderful people. I wish I could say that I knew it all along, that my decisions have been calculated and that every step I made has been carefully thought out to get me here. But the truth is that I had the fortune to base my decisions on a bit of good intuition and a lot of great advice. So, I conclude by thanking one more time the wise voices that convinced me to do a PhD and advised me during its duration.

Contents

List of Figures

List of Tables

Abbreviations

RDF	**R**esource **D**escription **F**ramework
BGP	**B**asic **G**raph **P**attern
RDBMS	**R**elational **D**ata**B**ase **M**anagement **S**ystem
OLAP	**O**n-**L**ine **A**nalytical **P**rocessing
AnS	**An**alytical **S**chema
AnQ	**An**alytical **Q**ueries

rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
rdf:type	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
rdfs:subClassOf	http://www.w3.org/2000/01/rdf-schema#subClassOf
rdfs:subPropertyOf	http://www.w3.org/2000/01/rdf-schema#subPropertyOf
rdfs:domain	http://www.w3.org/2000/01/rdf-schema#domain
rdfs:range	http://www.w3.org/2000/01/rdf-schema#range
rdf:Literal	http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal
rdfs:Class	http://www.w3.org/2000/01/rdf-schema#Class
rdf:Property	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
xsd:int	http://www.w3.org/2001/XMLSchema#int

Chapter 1

Introduction

"I can't recommend this book too highly." Such a statement can be followed by praise for the book or by criticism, elucidating the intended meaning. The context determines the book's properties. Regular speech is filled with ambiguous entities, e.g., "jaguar" can be an animal or a car, "duck" can be used as a verb or a noun, the combination of the two positive words "yeah" and "sure" results in a negative assertion. The human mind can easily make the necessary disambiguations given the context, but how do we teach context to a computer?

The initial vision behind the Semantic Web [Berners-Lee01] was to make Web pages as comprehensible to machines as they are to humans. The Resource Description Framework (RDF [W3Cb]) specification, first recommended by the World Wide Web Consortium (W3C) in February 1999, allows to uniquely identify entities and concepts through the use of uniform resource identifiers (or URIs, in short). Moreover, such resources can be interrelated, when a resource r_1 is stated to have a property whose name is given by the resource r_2 and whose value is a third resource r_3. Modeling such a statement as a directed link going from r_1 to r_3 and labeled r_2 leads to a graph representation of a set of interconnected resources, or in other terms, a linked data set. The semi-structured nature of this model is apparent: data can be heterogeneous (different resources may have very different sets of properties defined) and self-describing (the structure of the data is encoded in the data itself). These characteristics make it a very suitable format for Web-based data exchange.

"Linked Data is the Semantic Web done right" [Berners-Lee08] is Tim Berners-Lee's characterization to the W3C-promoted set of good practices for publishing and connecting information on the Web. Common resources are used to link datasets to each other building a Web of interconnected information. Efficient techniques for taking advantage of such semantic-rich, interconnected data bring data management (and computing, more generally) one step closer to the intelligent computers dreamed of by the Web's founders, capable of understanding and exploiting the meaning of content on the Web.

1.1 Data vs. Technology

"Information is the oil of the 21st century, and analytics is the combustion engine." The high importance of information in today's world has been aptly described by Peter Sondergaard in his statement at the Gartner Symposium [Pettey11]. Data is used in industry to increase profit, and in the fields of research to drive innovation. Its applications are countless, starting from curing diseases to winning public elections.

However, data is only as valuable as the information that can be extracted from it. The data volume being analyzed is growing exponentially while the technologies to manipulate it are lagging behind [Win, Gantz12].

Scientific data is obtained using increasingly performant tools, capable of extensive measurements and complex simulations. Efficient analysis of such experimental data is the bottleneck in today's scientific progress [Ailamaki10].

The volume of social data obtained by user interactions with public websites, software and sensors is coming close behind. Users have moved from being sole consumers of data, to publishing at an alarming rate. With this high amount of user data available, user personalized applications are an expectation.

Transactional (structured) data has been the subject of research for the past decades, resulting in a great variety of efficient relational data management tools. However, recent years have seen a growing interest in the use and analysis of unstructured data taking on different shapes and sizes, e.g., text, images, audio, video. This creates a big gap between the data being published and the ability of existing tools to analyze it. New tools and technologies have been developed for this purpose. However, oftentimes these tools are ad-hoc and application-specific systems which are leaving some of the database experts with the feeling of seeing attempts at reinventing the wheel [Mohan13].

On the other hand, the utility of semi-structured data models, such as W3C's Resource Description Framework [W3Cb] is unquestioned. RDF facilitates the integration of data from different sources and formats, making it easy for new data to be interrogated together with data stored in the old (legacy) systems. The use of metadata (data about data), the addition of semantic constraints, and the high emphasis on collaboration and sharing of on-line resources, in this already expansive volume of data makes traditional relational technologies even more difficult to apply. Moreover, since the general pattern is to continuously appended new data, multidimensional structures are naturally built, where time plays a key role.

This scientific horizon shows a world of massive unstructured and semantic rich data which cannot be easily ported to the technologies that have been perfected over the decades for storing and manipulating its structured counterpart.

1.2 Motivation: Leverage Mature RDBMS Technology for Querying Semantic Data

Given the high rate of data publication [Win, Gantz12] and its increased complexity, the goal of this thesis is twofold:

- First, to leverage existing technologies for efficient answering of queries over data subject to semantic constraints;
- Second, to formalize procedures for powerful analytics on such data.

We outline the respective research problems next.

1.2.1 Querying Data under Constraints

We start by looking at query answering over this complex and semantic rich data. The literature provides several scalable solutions for querying RDF graphs using relational data management systems (RDBMSs, in short) or RDBMS-style specialized engines [Abadi07, Neumann10b, Weiss08]. These works, however, ignore the essential RDF feature called *entailment*, which allows modeling *implicit information* within RDF. Taking entailment into account is crucial for answering SPARQL queries, as ignoring implicit information leads to incomplete answers [W3Cd]. Thus, to capitalize on (and benefit from) scalable RDBMS performance, SPARQL query answering can be split into a *reasoning* step which handles entailment outside the RDBMSs, and a *query evaluation* step delegated to RDBMSs.

A popular reasoning step is *graph saturation* (*closure*). This consists of pre-computing (making explicit) and adding to the RDF graph all implicit information. Answering queries using saturation amounts to evaluating the queries against the saturated graph. While saturation leads to efficient query processing, it requires time to be computed, space to be stored, and must be recomputed upon updates.

The alternative reasoning step is *query reformulation*. This consists in turning the query into a *reformulated query*, which, evaluated against the original graph, yields the exact answers to the original query. Since reformulation is made at query run-time, it is intrinsically robust to updates; reformulation is also typically very fast. However, reformulated queries are often syntactically more complex than the original ones, thus their evaluation may be costly.

Opinions with respect to which is the best option are split in the research community. Works generally focus on improving either one or the other technique. Motivated by this observation, we look into improving the state of the art for both approaches, choosing a set of semantic constraints that allows straightforward portability to any RDBMS, and endeavoring to make a thorough comparison of the two techniques in the same experimental setting.

1.2.2 Data Analysis

Having treated the problem of inference, we take the topic of query answering one step further, by considering data analytics. The standardization of the SPARQL query language now at v1.1 [W3C13] has lead to the emergence of many systems capable of storing, querying, and updating RDF, such as OWLIM [wwwf], RDF-3X [Neumann10a], Virtuoso [Erling07] etc. However, as more and more RDF datasets are made available, in particular Linked (Open) Data, application requirements also evolve, demanding advanced data analytics over semantic graphs.

Significant attempts have been made at translating RDF data to the realm of warehousing application and also proposing vocabularies for publishing such data. However, these works are mostly tailored for transforming the heterogeneous RDF data into structured tabular data. While such an approach enables the immediate use of performant data warehouse technologies, it takes away from the versatility of RDF.

Aiming at maintaining all the features that have given RDF its popularity, namely heterogeneity, rich semantics, ease of publication, we investigate data analytics in a context where the warehousing process is RDF specific. Moreover, since RDF datasets are rarely centered around a single set of facts, we look into a flexible choice of facts, dimensions and measure for data warehousing.

1.3 Contributions

This thesis aims at efficient query answering over RDF data. As such it addresses two main problems:

(*i*) query answering in dynamic RDF databases; and

(*ii*) RDF data warehouses analytics.

The overview below presents the thesis organization and main contributions.

Chapter 2 formalizes the RDF-related concepts used throughout the thesis.

Part I focuses on query answering in RDF databases that are subject to updates. In this context we make the following contributions:

Chapter 3 reviews the state of the art in RDF data management.

Chapter 4 presents the formalizations for our contributions:

- We identify the novel DB fragment of RDF, extending fragments previously studied [Arenas09, Goasdoué11, Kaoudi08, Urbani11] by the support of blank nodes.
- We propose novel BGP query answering techniques for this DB fragment, designed to work on top of on any standard conjunctive query processor (and in particular any off-the-shelf RDBMS). Specifically, we provide:

(i) an efficient novel *incremental RDF saturation maintenance algorithm*, based on which query answering reduces directly to query evaluation; and (ii) a *novel reformulation-based query answering algorithm*, required by the augmented expressive power of our DB fragment w.r.t. those in the literature.

Chapter 5 demonstrates the feasibility and efficiency of the above query answering techniques. We implemented and deployed our algorithms on top of the PostgreSQL [wwwe] RDBMS. The best choice among saturation- or reformulation-based query answering depends on the queries, the amount of implicit data and the frequency and volume of updates to the data and/or schema. Our experiments study these possible cases and show which strategy works best for each of them.

The **Concluding Remarks** summarize the placement of our work with respect to the state of the art.

Part II presents a full redesign, from the bottom up, of the core data analytics concepts and tools, leading to a complete formal framework for warehouse-style analytics on RDF data, particularly suited to heterogeneous, semantic-rich corpora of Linked Data.

Chapter 6 starts by reviewing that state of the art pertinent to the topic.

Chapter 7 lists our contributions:

- We devise a *full-RDF* warehousing approach, where the base data *and* the warehouse extent are RDF graphs.
- We introduce *RDF analytical schemas*, which are graphs of classes and properties themselves, having nodes (classes) connected by edges (properties) with *no single central concept (node)*. This contrasts with the typical relational data warehouse star or snowflake schemas. The core idea behind many-node analytical schemas is to define *each node (respectively edge) by an independent query* over the base data.
- We define *analytical queries* over our decentralized analytical schemas. Such queries are highly flexible in the choice of measures and classifiers, while supporting all the classical analytical cubes and operations (slice, dice etc.).

Chapter 8 presents experiments on our fully implemented operational prototype and empirically demonstrate its interest and performance.

The **Concluding Remarks** relate our contribution to the existing works in the state of the art.

Chapter 9 provides a thesis summary relating the two main topics. It also presents the multiple research opportunities that this work has inspired.

Chapter 2

A Brief Review of RDF Data Management

This chapter provides the background information required to follow the problems raised in this thesis and their proposed solutions. The concepts used throughout the work are illustrated and formalized as follows.

First, Section 2.1 describes the Resource Description Framework (RDF) [W3Cb], a graph-based data model recommended by the W3C for interchanging data on the Web.

The following Section 2.2 presents the W3C standard for querying RDF, namely the SPARQL Protocol and RDF Query Language (SPARQL) [W3Cd].

Finally, Section 2.3 puts RDF data storage in the context of relational database management systems (RDBMSs) [Codd70].

2.1 The RDF Data Model

Initially designed as a data model for metadata, RDF is now generally accepted as the W3C standard for Semantic Web applications. Using RDF, one can describe the properties of (Web) resources through simple statements (called triples). This basic syntax (detailed in Section 2.1.1) is both human-readable and machine-processable. Furthermore, RDF enables exploiting and sharing a mix of structured and semi-structured data.

Its heterogeneous generic nature makes RDF easily adaptable to diverse contexts, notably giving it a key role in publishing and connecting Web data. As such, RDF data can be expressed in multiple serialization formats, for instance RDF/XML [W3C14c], Turtle [W3C14e], N-Triples [W3C14a], RDFa [W3C14b], etc.

An ontology language can be used to enhance the description of RDF data. Section 2.1.2 shows how, using the RDF Schema (RDFS) language, one can express useful constraints on the resources and their properties used in RDF triples. The interpretation of such constraints highlights a powerful feature of RDF: its ability to express implicit information. Finally, the process of inferring implicit information from explicit RDF triples is detailed in Section 2.1.3.

Assertion	Triple	Relational notation
Class	s rdf:type o	o(s)
Property	s p o	p(s, o)

TABLE 2.1: RDF statements.

2.1.1 RDF Graphs

An *RDF graph* (or *graph*, in short) is a set of *triples* of the form s p o. A triple states that its *subject* s has the *property* p, and the value of that property is the *object* o. Given a set U of uniform resource identifiers (URIs), a set L of typed or un-typed literals (constants), and a set B of blank nodes (unknown URIs or literals), such that U, B and L are pairwise disjoint, a triple is *well-formed* whenever its subject belongs to $U \cup B$, its property belongs to U, and its object belongs to $U \cup B \cup L$. In what follows, only well-formed triples are considered.

Blank nodes are an essential feature of RDF enabling the support of *unknown URI/literal tokens*. For instance, one can use a blank node _:b_1 to state that the country of _:b_1 is *France* while the city of the same _:b_1 is *Paris*. Many such blank nodes can co-exist within a graph, e.g., one may also state that the country of _:b_2 is *Romania* while the city of _:b_2 is *Timişoara*; at the same time, the population of *Timişoara* can be said to be an unspecified value _:b_3.

Notations. In the following, s, p, o and _:b are used in triples (possibly with subscripts) as placeholders. Literals are shown as strings between quotes, e.g., "*string*". Finally, the set of values – URIs, blank nodes, literals – of an RDF graph G is denoted Val(G).

Table 2.1 shows how to use triples to describe resources, that is, to express class (unary relation) and property (binary relation) assertions. A resource URI is built of a label belonging to a namespace. The RDF standard [Recommendation04] provides a set of built-in classes and properties, as part of the rdf: and rdfs: pre-defined namespaces, e.g., rdf:type specifies the class(es) to which a resource belongs. All the standard namespaces and resources used in this thesis are detailed in *Abbreviations*. The namespaces for resources used as examples are replaced by the prefix ":" for readability.

A more intuitive representation of an RDF graph can be drawn from its triples where every (distinct) subject or object value is represented by a node labeled with this value. For every triple, there is a directed edge labeled with the property value from the subject node to the object node. Following the RDF standard [Recommendation04], Definition 2.1 formalizes this representation of an RDF graph. The notation $f_{|d}$ is used to denote the restriction of a function f to its sub-domain d.

Definition 2.1 (***Graph notation of an RDF graph***).
An RDF graph is a labeled directed graph $\mathtt{G} = \langle \mathcal{N}, \mathcal{E}, \lambda \rangle$ where:

- $\mathcal{N}$ is the set of *nodes*, $\mathcal{N}^0$ denotes the nodes in $\mathcal{N}$ having no outgoing edge, and $\mathcal{N}^{>0} = \mathcal{N} \setminus \mathcal{N}^0$;
- $\mathcal{E} \subseteq \mathcal{N}^{>0} \times \mathcal{N}$ is the set of *directed edges*;
- $\lambda : \mathcal{N} \cup \mathcal{E} \to U \cup B \cup L$ is a *labeling function* such that $\lambda_{|\mathcal{N}}$ is injective, with $\lambda_{|\mathcal{N}^0} : \mathcal{N}^0 \to U \cup B \cup L$ and $\lambda_{|\mathcal{N}^{>0}} : \mathcal{N}^{>0} \to U \cup B$, and $\lambda_{|\mathcal{E}} : \mathcal{E} \to U$.

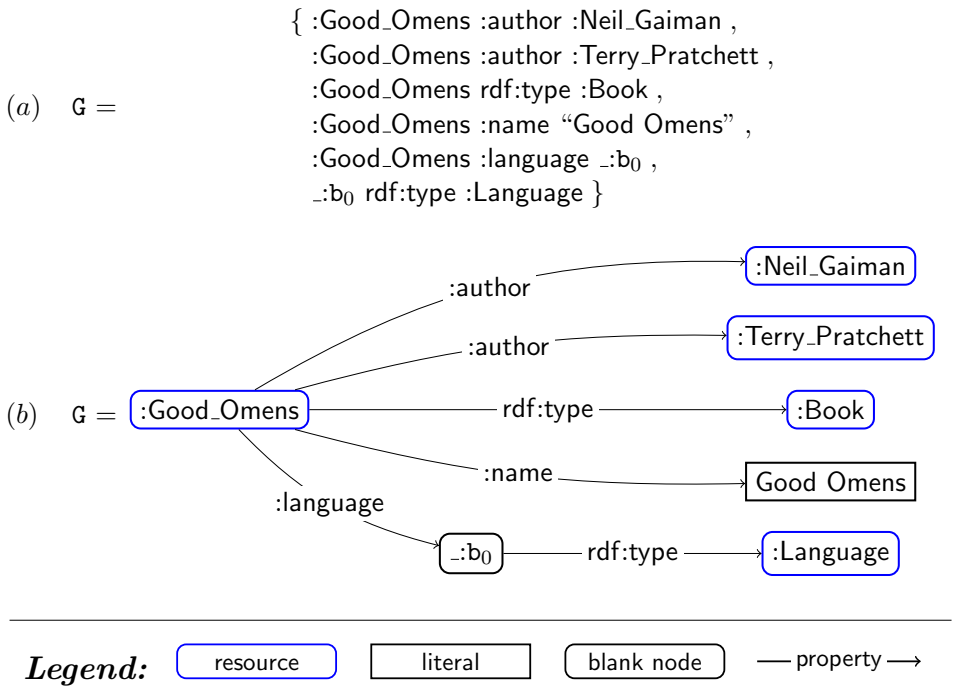


FIGURE 2.1: Sample RDF graph. Alternative representations.

Example 2.2 (***RDF graph***)**.**
Consider an RDF graph comprising information about books and authors. Figure 2.1 (a) shows the triples, while (b) depicts the dataset using its graph notation. The RDF graph features a resource :Good_Omens *whose name is* "Good Omens" *and whose type is* :Book*. It was written by two authors (*:author*), namely* :Neil_Gaiman *and* :Terry_Pratchett*, in a language* _:b0 *that is not known in this dataset.*

2.1.2 RDF Schema

RDF Schema (RDFS) is a valuable feature of RDF used for enhancing the descriptions in graphs. RDFS triples declare *semantic constraints* between the classes and the properties in these graphs, through the use of built-in properties modeling sub-class (rdfs:subClassOf) and sub-property relationships (rdfs:subPropertyOf), typing the first attribute (a.k.a. domain) of a property (rdfs:domain) and typing the second attribute (a.k.a. range) of a property (rdfs:range).

Table 2.2 shows the allowed constraints and how to express them, also relating these constraints to relational inclusion constraints under the open-world assumption.

Open-world interpretation of RDFS constraints. Traditionally, constraints can be interpreted in two ways [Abiteboul95]: under the closed-world assumption (CWA) or under the open-world assumption (OWA). Under CWA, any fact not present in the database is assumed not to hold. Under this assumption, if the set of database facts

Constraint	Triple	OWA interpretation
Sub-class	s rdfs:subClassOf o	$s \subseteq o$
Sub-property	s rdfs:subPropertyOf o	$s \subseteq o$
Domain typing	s rdfs:domain o	$\Pi_{\text{domain}}(s) \subseteq o$
Range typing	s rdfs:range o	$\Pi_{\text{range}}(s) \subseteq o$

TABLE 2.2: RDFS statements.

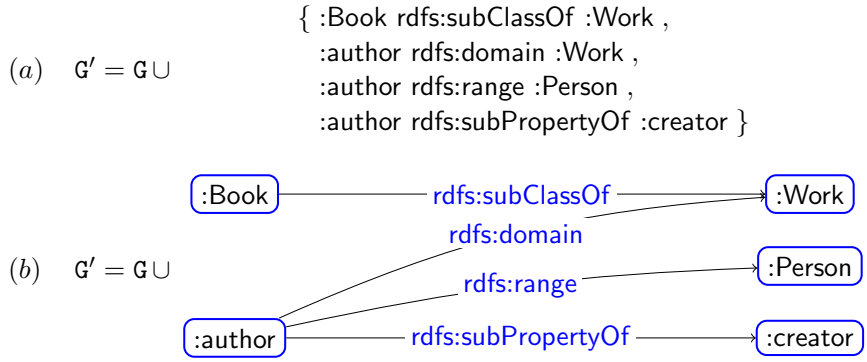


FIGURE 2.2: Sample RDF Schema triples. Alternative representations.

does not respect a constraint, then the database is *inconsistent*. For instance, the CWA interpretation of a constraint of the form $R_1 \subseteq R_2$ is: any tuple in the relation R_1 *must* also be in the relation R_2 *in the database*, otherwise the database is inconsistent. On the contrary, under OWA, some facts may hold even though they are *not in the database*. For instance, the OWA interpretation of the same example is: any tuple t in the relation R_1 *is considered as being also in the relation* R_2 (the inclusion constraint *propagates* t *to* R_2).

The RDF data model – and accordingly, the present work – is based on OWA, and this is how all the constraints in Table 2.2 are interpreted. For instance, if the following two triples hold in the graph: :author rdfs:domain :Work and :Good_Omens :author :Neil_Gaiman, then so does the triple :Good_Omens rdf:type :Work. The latter is due to the rdfs:domain constraint in Table 2.2.

Example 2.3 (***Schema for an RDF graph***)**.**
Consider next to the graph G *from Figure 2.1, the schema depicted in Figure 2.2. This schema expresses semantic (or ontological) constraints like a* :Book *is a* :Work*, the domain of* :author *is* :Work*, while its range is* :Person*, that being the author of something (*:author*) is one way of creating something (*:creator*).*

2.1.3 Entailment

The above discussion about OWA illustrated an important RDF feature: *implicit triples*, considered to be part of the graph even though they are not explicitly present in it. An example is :Good_Omens rdf:type :Work, which is implicit in the graph G′ of Figure 2.2.

The W3C names *RDF entailment* the mechanism through which, based on the set of explicit triples and some *entailment rules* (to be described soon), implicit RDF triples are derived. We denote by $\vdash^i_{\text{RDF}}$ *immediate entailment*, i.e., the process of deriving new triples through a single application of an entailment rule. More generally, (full) *RDF entailment* can be defined as follows: a triple $\mathtt{s\ p\ o}$ is entailed by a graph $\mathtt{G}$, denoted $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$, if and only if there is a sequence of applications of immediate entailment rules that leads from $\mathtt{G}$ to $\mathtt{s\ p\ o}$ (where at each step of the entailment sequence, the triples previously entailed are also taken into account). Table 2.3 shows multiple examples of immediate entailment rules directly linked to the RDFS constraints in Table 2.2.

Graph saturation. The immediate entailment rules allow defining the finite *saturation* (a.k.a. *closure*) of an RDF graph $\mathtt{G}$, which is the graph, denoted $\mathtt{G}^\infty$, defined as the fixed-point obtained by repeatedly applying $\vdash^i_{\text{RDF}}$ on $\mathtt{G}$:

- $\mathtt{G}^0 = \mathtt{G}$
- $\mathtt{G}^\alpha = \mathtt{G}^{\alpha-1} \cup \{\mathtt{s\ p\ o} \mid \mathtt{G}^{\alpha-1} \vdash^i_{\text{RDF}} \mathtt{s\ p\ o}\}$

The saturation of a graph is unique (up to blank node renaming), and does not contain any implicit triples (they have all been made explicit by saturation). An obvious connection holds between the triples entailed by a graph $\mathtt{G}$ and its saturation: $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$ if and only if $\mathtt{s\ p\ o} \in \mathtt{G}^\infty$.

RDF entailment is part of the RDF standard itself; in particular, *the answers of a query posed on* $\mathtt{G}$ *must take into account all triples in* $\mathtt{G}^\infty$, since *the semantics of an RDF graph is its saturation*, that is: any graph $\mathtt{G}$ is equivalent to, and models, its saturation $\mathtt{G}^\infty$. In Sesame [wwwg], Jena [wwwc], OWLIM [wwwf] etc., RDF entailment is supported through *saturation*.

Immediate entailment rules. We give here an overview of the different kinds of immediate entailment rules upon which RDF entailment relies.

A first kind of rule generalizes triples using blank nodes. For instance, if $\mathtt{s}$ is an instance of $\mathtt{o}$, then there exists an instance of $\mathtt{o}$, namely the blank node _:b:

$$\mathtt{s}\ \text{rdf:type}\ \mathtt{o} \vdash_{\text{RDF}} \mathtt{\_:b}\ \text{rdf:type}\ \mathtt{o}$$

A second kind of rule derives entailed triples from the semantics of built-in classes and properties. For example, RDF provides rdfs:Class, whose semantics is the set of all built-in and user-defined classes. Thus, if a resource is of type $\mathtt{o}$, then $\mathtt{o}$ is a class:

$$\mathtt{s}\ \text{rdf:type}\ \mathtt{o} \vdash_{\text{RDF}} \mathtt{o}\ \text{rdf:type}\ \text{rdfs:Class}$$

Finally, the third kind of rule derives entailed triples from the constraints modeled in an RDFS. Some rules derive entailed RDFS statements, through the transitivity of class and property inclusions, and from inheritance of domain and range typing. Using a tabular notation, with the entailed (consequence) triple shown at the bottom, some examples are:

TABLE 2.3: Entailment rules [Recommendation04].

(a) Schema-level entailment from a single instance-level triple.

Entailment pattern	Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
RDFS axioms + rdfs10	s rdf:type o	o rdfs:subClassOf o
rdfD2 + rdfs6	s p o	p rdfs:subPropertyOf p

(b) Schema-level entailment from a single schema-level triple.

Entailment pattern	Triple	Entailed triple ($\vdash^i_{\text{RDF}}$)
RDFS axioms + rdfs10	s_1 rdfs:subClassOf s_2	s_1 rdfs:subClassOf s_1
RDFS axioms + rdfs10	s_1 rdfs:subClassOf s_2	s_2 rdfs:subClassOf s_2
RDFS axioms + rdfs6	p_1 rdfs:subPropertyOf p_2	p_1 rdfs:subPropertyOf p_1
RDFS axioms + rdfs6	p_1 rdfs:subPropertyOf p_2	p_2 rdfs:subPropertyOf p_2
RDFS axioms + rdfs6	p rdfs:domain s	p rdfs:subPropertyOf p
RDFS axioms + rdfs10	p rdfs:domain s	s rdfs:subClassOf s
RDFS axioms + rdfs6	p rdfs:domain rdf:Literal	p rdfs:subPropertyOf p
RDFS axioms + rdfs6	p rdfs:range s	p rdfs:subPropertyOf p
RDFS axioms + rdfs10	p rdfs:range s	s rdfs:subClassOf s
RDFS axioms + rdfs6	p rdfs:range rdf:Literal	p rdfs:subPropertyOf p

(c) Schema-level entailment from two schema triples.

Entailment pattern	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
rdfs11	s rdfs:subClassOf s_1, s_1 rdfs:subClassOf s_2	s rdfs:subClassOf s_2
rdfs5	p rdfs:subPropertyOf p_1, p_1 rdfs:subPropertyOf p_2	p rdfs:subPropertyOf p_2
ext1	p rdfs:domain s_1, s_1 rdfs:subClassOf s	p rdfs:domain s
ext2	p rdfs:range s_1, s_1 rdfs:subClassOf s	p rdfs:range s
ext3	p rdfs:subPropertyOf p_1, p_1 rdfs:domain s	p rdfs:domain s
ext4	p rdfs:subPropertyOf p_1, p_1 rdfs:range s	p rdfs:range s

(d) Instance-level entailment from combining schema and instance triples.

Entailment pattern	Triples	Entailed triple ($\vdash^i_{\text{RDF}}$)
rdfs9	s_1 rdfs:subClassOf s_2, s rdf:type s_1	s rdf:type s_2
rdfs7	p_1 rdfs:subPropertyOf p_2, s p_1 o	s p_2 o
rdfs2	p rdfs:domain s, s_1 p o_1	s_1 rdf:type s
rdfs3	p rdfs:range s, s_1 p o_1	o_1 rdf:type s

$$\frac{s \text{ rdfs:subClassOf } o_1 \quad o_1 \text{ rdfs:subClassOf } o_2}{s \text{ rdfs:subClassOf } o_2} \qquad \frac{p_1 \text{ rdfs:subPropertyOf } p_2 \quad p_2 \text{ rdfs:domain } o}{p_1 \text{ rdfs:domain } o}$$

Some other rules derive entailed RDF statements, through the propagation of values (URIs, blank nodes, and literals) from sub-classes and sub-properties to their super-classes and super-properties, and from properties to classes typing their domains and ranges. Within our running example:

$$\frac{\text{:author rdfs:subPropertyOf :creator} \quad \text{:Good\_Omens :author :Neil\_Gaiman}}{\text{:Good\_Omens :creator :Neil\_Gaiman}}$$

Restricted rule sets. While these families of rules are part of the W3C specification [W3Cb], they are not all of equal interest. For instance, consider the generalization to blank nodes: it is probably more interesting to know that s rdf:type o than to know that *some unknown* _:b has the type o. Similarly, the fact that rdfs:Class is an instance of itself is of limited interest. Other rules, such as those stating that, e.g., :Book is a subclass of :Work are comparatively much more useful.

Clearly defining the set of entailment rules is absolutely essential, because query answers are defined based on the saturated graph (see Section 2.2). Formally identified RDF fragments [Arenas09, Goasdoué11, Kaoudi08, Urbani11] each consider only a useful subset of the existing rules. This thesis is also based on such an approach, considering only the entailment rules detailed in Table 2.3.

2.2 Querying RDF Graphs

The SPARQL query language is typically used for RDF graph pattern matching with the purpose of extracting either tabular information about the resources in a graph or constructing new RDF graphs.

The querying capabilities of the latest SPARQL 1.1 version [W3C13] allow a wide range of features, like conjunctions and/or disjunctions of required and/or optional graph patterns, use of aggregation functions, subqueries, negation, etc. Moreover it permits choosing among different sets of entailment rules through the use of entailment regimes [W3Ca]. The query semantics considered in this thesis (shown in Table 2.3) correspond to the *RDFS Entailment Regime* [W3Ca].

The contributions of this thesis are based on a subset of SPARQL, namely its conjunctive query fragment introduced in Section 2.2.1. The following Section 2.2.2 introduces concepts useful for data analysis, presented as types of queries.

2.2.1 BGP Queries

This work considers the well-known subset of SPARQL consisting of (unions of) *basic graph pattern* (BGP) queries, also known as SPARQL conjunctive queries.

A BGP is a *set of triple patterns*, or triples in short. Each triple has a subject, property and object. Subjects and properties can be URIs, blank nodes or variables; objects can also be literals.

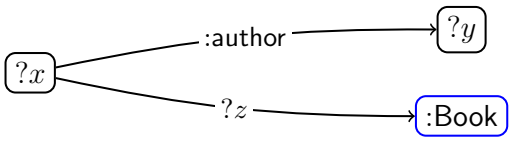
The focus is set on the boolean BGP queries of the form `ASK WHERE` $\{t_1, \ldots, t_\alpha\}$, and on the non-boolean BGP queries of the form `SELECT` $\bar{x}$ `WHERE` $\{t_1, \ldots, t_\alpha\}$, where $\{t_1, \ldots, t_\alpha\}$ is a BGP, i.e., a set of triples modeling their conjunction; the variables $\bar{x}$ in the head of the query are called *distinguished variables*, and are a subset of the variables occurring in $t_1, \ldots, t_\alpha$.

Notations. Without loss of generality, in the following the conjunctive query notation $q(\bar{x})$:- $t_1, \ldots, t_\alpha$, where $\{t_1, \ldots, t_\alpha\}$ is a BGP, is used for both `ASK` and `SELECT` queries (for boolean queries, $\bar{x}$ is empty). The head of q denoted $\texttt{head}(q)$ is $q(\bar{x})$, and the body of q denoted $\texttt{body}(q)$ is $t_1, \ldots, t_\alpha$. Variables in queries are denoted by a question mark before the variable name, e.g., $?x$. Further, $\texttt{VarBl}(q)$ represents the set of variables *and* blank nodes occurring in the query q. The set of values (URIs, blank nodes, literals) of a graph `G` is denoted $\texttt{Val}(\texttt{G})$.

BGP query graph. Each triple atom in the body of a BGP query can be seen as a *generalized RDF triple*, where, beyond URIs, blank nodes and literals, *variables* may appear in any of the subject, predicate and object positions. This leads to a *graph notation for BGP queries*, which can be seen as a corresponding generalization of the RDF graph representation in Definition 2.1. For instance, the body of the query:

$$q(?x, ?y, ?z) \text{ :- } ?x \text{ :author } ?y,\ ?x\ ?z \text{ :Book}$$

is represented by the graph:



Query evaluation. Given a query q and an RDF graph `G`, the *evaluation of q against* `G` is:

$$q(\texttt{G}) = \{\bar{x}_\mu \mid \mu : \texttt{VarBl}(q) \to \texttt{Val}(\texttt{G}) \text{ is a total assignment such that } t_1^\mu \in \texttt{G}, t_2^\mu \in \texttt{G}, \ldots, t_\alpha^\mu \in \texttt{G}\}$$

where for a given triple (or triple set) t, we denote by t^μ the result of replacing every occurrence of a variable or blank node $e \in \texttt{VarBl}(q)$ in t, by the value $\mu(e) \in \texttt{Val}(\texttt{G})$. If q is boolean, the empty answer set encodes false, while the non-empty answer set made of the empty tuple $\emptyset_\mu = \langle\rangle$ encodes true.

Notice that (normative) query evaluation *treats the blank nodes in a query as non-distinguished variables* [Abiteboul11]. That is, one could consider equivalently queries

without blank nodes or queries without non-distinguished variables. Thus, in the sequel, without loss of generality, we consider queries where all blank nodes have been replaced by distinct (new) non-distinguished variable symbols.

Query answering. It is important to keep in mind the distinction between query *evaluation* and query *answering*. The evaluation of q against $\mathtt{G}$ only uses $\mathtt{G}$'s explicit triples, thus may lead to an incomplete answer set. The (complete) *answer set* of q against $\mathtt{G}$ is obtained by the evaluation of q against $\mathtt{G}^{\infty}$, denoted by $q(\mathtt{G}^{\infty})$.

Example 2.4 (***BGP query answering***)**.**
The following query asks for the names of works somehow related to Neil Gaiman*:*

$$\begin{aligned} q(?x) \text{ :- } & ?y \text{ :name } ?x, \\ & ?y \text{ rdf:type :Work}, \\ & ?y\ ?z \text{ :Neil\_Gaiman} \end{aligned}$$

The complete answer set to the query q on the RDF graph $\mathtt{G}'$ from Figure 2.2 is:

$$q(\mathtt{G}'^{\infty}) = \{\ \langle \text{“Good Omens”} \rangle\ \}.$$

The answer results from $\mathtt{G}' \vdash_{\text{RDF}}$:Good_Omens rdf:type :Work *and the assignment:*

$$\mu = \{\ ?x \rightarrow \text{“Good Omens”},\ ?y \rightarrow \text{:Good\_Omens},\ ?z \rightarrow \text{:author}\ \}.$$

Note that evaluating q against $\mathtt{G}'$ leads to the incomplete (empty) answer set $q(\mathtt{G}') = \{\langle\rangle\}$.

2.2.2 Queries for Data Analysis

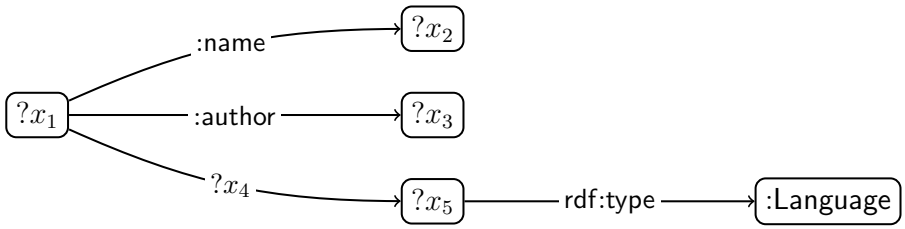
Rooted queries. Data analysis typically allows investigating particular sets of facts according to relevant criteria (a.k.a. *dimensions*) and measurable or countable attributes (a.k.a. *measures*) [Jensen10]. In this thesis, *rooted* BGP queries play a central role as they are used to specify the set of facts to analyze, as well as the dimensions and the measures to be used (Section 7.3).

Definition 2.5 (***Rooted query***)**.**
Let q be a BGP query, $\mathtt{G} = \langle \mathcal{N}, \mathcal{E}, \lambda \rangle$ its graph representation and $n \in \mathcal{N}$ a node whose label is a variable in q. The query q is *rooted in* n iff $\mathtt{G}$ is a connected graph and any other node $n' \in \mathcal{N}$ is reachable from n following the directed edges in $\mathcal{E}$.

Example 2.6 (***Rooted query***)**.**
The query q described below, asking for the names of works, their authors and related language, is a rooted BGP query, with $?x_1$ as root node.

$$\begin{aligned} q(?x_2, ?x_3, ?x_5) \text{ :- } & ?x_1 \text{ :name } ?x_2, \\ & ?x_1 \text{ :author } ?x_3, \\ & ?x_1\ ?x_4\ ?x_5, \\ & ?x_5 \text{ rdf:type :Language} \end{aligned}$$

Its graph representation below shows that every node is reachable from the root $?x_1$.



Join queries. Another useful concept is that of *join query*, which joins BGP queries on their *distinguished variables* and projects out some of these variables. Join queries will be used when defining data warehouse analyses (Section 7.4).

Definition 2.7 (***Join query***)**.**
Let $q_1, \ldots, q_n$ be BGP queries whose non-distinguished variables are pairwise disjoint. We say $q(\bar{x}) \text{ :- } q_1(\bar{x}_1) \wedge \cdots \wedge q_n(\bar{x}_n)$, where $\bar{x} \subseteq \bar{x}_1 \cup \cdots \cup \bar{x}_n$, is a *join query* q of $q_1, \ldots, q_n$. The answer set to $q(\bar{x})$ is defined to be that of the BGP query $q^{\bowtie}$:

$$q^{\bowtie}(\bar{x}) \text{ :- } \mathsf{body}(q_1(\bar{x}_1)), \cdots, \mathsf{body}(q_n(\bar{x}_n))$$

In the above, $q_1, q_2, \ldots, q_n$ do not share *non-distinguished variables* (variables not present in the query head). This assumption is made *without loss of generality*, as one can easily rename non-distinguished variables in $q_1, q_2, \ldots, q_n$ in order to meet the condition. In the sequel, it is assumed that such renaming has been applied in join queries.

Example 2.8 (***Join query***)**.**
Consider the BGP queries q_1*, asking for the works having a name and their author, and* q_2*, asking for works and their language:*

$$q_1(?x_1, ?x_3) \text{ :- } ?x_1 \text{ :name } ?x_2,\ ?x_1 \text{ :author } ?x_3$$

$$q_2(?x_1, ?x_4) \text{ :- } ?x_1 \text{ :language } ?x_4,\ ?x_4 \text{ rdf:type :Language}$$

The join query $q^{\bowtie}_{1,2}(x_1, x_3) \text{ :- } q_1(x_1, x_3) \wedge q_2(x_1, x_4)$ *asks for the works and their author, for those works having a name and a language, i.e.,*

$$q^{\bowtie}_{1,2}(x_1, x_3) \text{ :- } ?x_1 \text{ :name } ?x_2,\ ?x_1 \text{ :author } ?x_3,\ ?x_1 \text{ :language } ?x_4,\ ?x_4 \text{ rdf:type :Language}$$

Other join queries can be obtained from q_1 *and* q_2 *by returning a different subset of the head variables* x_1, x_2, x_3*, and/or by changing their order in the query head etc.*

2.3 RDF vs. Relational Data Management

The RDF research community has shown considerable interest in capitalizing on (and benefiting from) the scalable performance of relational data management systems (RDBMSs, in short). The literature provides several scalable solutions for querying RDF graphs using RDBMSs or RDBMS-style specialized engines [Abadi07, Neumann10b, Weiss08], while other works have invested effort into translating large fragments of SPARQL to SQL [Chebotko09].

RDF graphs turn out to be a special case of incomplete relational databases based on *V-tables* [Abiteboul95, Imielinski84], which allow using variables in their tuples. Note that using a variable multiple times in a V-table allows expressing joins on unknown values.

An important result on V-table querying is that the standard relational evaluation (which sees variables in V-tables as constants) computes the exact answer set of any conjunctive query [Abiteboul95, Imielinski84]. From a practical viewpoint, this provides a possible way of answering conjunctive queries against V-tables using standard relational database management systems. We use the same observation to obtain complete answer sets to BGP queries using RDBMS evaluation, as follows.

Given a graph $\mathtt{G}$, we encode it into the V-table $\mathtt{Triple}(\mathtt{s}, \mathtt{p}, \mathtt{o})$ storing the triples of $\mathtt{G}$ as tuples, in which blank nodes become variables. Then, given a BGP query $q(\bar{x}) \text{ :- } \mathtt{s}_1\ \mathtt{p}_1\ \mathtt{o}_1, \ldots, \mathtt{s}_n\ \mathtt{p}_n\ \mathtt{o}_n$, in which blank nodes have been equivalently replaced by fresh non-distinguished variables, the SPARQL evaluation $q(\mathtt{G})$ of q against $\mathtt{G}$ is obtained by the relational evaluation of the conjunctive query $Q(\bar{x}) \text{ :- } \bigwedge_{i=1}^{n} \mathtt{Triple}(\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)$ against the $\mathtt{Triple}$ table. Indeed, *SPARQL and relational evaluations coincide with the above encoding*, as relational evaluation amounts to finding all the total assignments from the variables of the query to the values (constants and variables) in the $\mathtt{Triple}$ table, so that the query becomes a subset of that $\mathtt{Triple}$ table.

It follows that evaluating $Q(\bar{x}) \text{ :- } \bigwedge_{i=1}^{n} \mathtt{Triple}(\mathtt{s}_i, \mathtt{p}_i, \mathtt{o}_i)$ against the $\mathtt{Triple}$ table containing the saturation of $\mathtt{G}$, instead of $\mathtt{G}$ itself, computes the answer set of q against $\mathtt{G}$. In other words, BGP query answering boils down to conjunctive query evaluation on a saturated database.

Conceptually, the above observation is the reason why the simple RDF query answering approach taken in previous works such as [Abadi07, Weiss08, Neumann09, Sidirourgos08, Neumann08] is sound. Completeness, on the other hand, requires query answering to go beyond basic query evaluation by also returning the implicit answers. The novel saturation-based query answering algorithm described in Section 4.2 also immediately follows from the above observation and is complete w.r.t. the discussed fragment semantics. In contrast, the reformulation-based query answering technique introduced in Section 4.3 requires a quite subtler approach.

Example 2.9 (***Answering queries on V-tables***)**.**
Provided that the saturation of the graph $\mathtt{G}'$ *from Figure 2.2 is encoded into a V-table* $\mathtt{Triple}(\mathtt{s}, \mathtt{p}, \mathtt{o})$ *as described above, the answer set of the BGP query from Example 2.4 against* $\mathtt{G}'$ *is the same as the result of evaluating the relational query* $Q(x)$ *against the V-table* $\mathtt{Triple}$*:*

$$Q(x) \text{ :- } \texttt{Triple}(\, y,\, \text{:name},\, x\,) \wedge \\ \texttt{Triple}(\, y,\, \text{rdf:type},\, \text{:Work}\,) \wedge \\ \texttt{Triple}(\, y,\, z,\, \text{:Neil\_Gaiman}\,).$$

2.4 Outlook

The following chapters present the contributions of this thesis, organized into two main topics.

First, we discuss efficient query answering in RDF databases. Essential to this part are the notion of schema and entailment presented in Sections 2.1.2 and 2.1.3, the difference between query evaluation and query answering discussed in Section 2.2.1 and the use of RDBMSs for storing RDF described in Section 2.3.

In the second part we formalize analytical querying of RDF data warehouses. The RDF graph definition shown in Section 2.1.1 together with the specialized queries presented in Section 2.2.2 are key notions necessary for following the formalizations pertaining to this topic.

Part I

Efficient Query Answering against Dynamic RDF Databases

Chapter 3

RDF Database Management Overview

This chapter presents the main problems raised by the efficient and expressive management of large volumes of Semantic Web (and in particular RDF) data. We present these problems, the main techniques and algorithms proposed towards addressing them within the Databases, Semantic Web and Knowledge Management communities, and finally characterize the techniques implemented within the major existing tools, commercial systems and research prototypes.

The central problem we consider is query answering, that is: given an RDF database and a query asked against this database, compute the answer to the query against the database. Going beyond the classical database problem of query evaluation (which focuses on identifying and ordering a set of operations that compute the results out of the data explicitly present in the database), query answering also requires reasoning mechanisms in order to take into account, when computing query answers, not only the explicit data but also the data implicitly present there. Implicit information in a Semantic Web (RDF) database is due to the presence of semantic (schema) rules which state that certain premises (found in the schema and/or in the data) entail certain consequences (or implicit facts). In order to compute correct query answers, explicit and implicit data must be taken into account.

Our discussion of query answering models, algorithms and techniques is divided into two sections.

First, Section 3.1 presents the main languages used to query RDF databases, the most popular schema languages used in conjunction with RDF databases, and the main query answering techniques.

Second, Section 3.2 outlines storage, indexing and query processing techniques put forward for the task of RDF query evaluation. We detail the advantages and technical difficulties raised by each approach, followed by a look into the expressive power supported by and algorithms implemented within the existing platforms.

Based on this description, Section 3.3 characterizes the models and languages implemented within a comprehensive set of RDF data management tools, issued from industry and research.

Finally, in Section 3.4, we examine the open issues related to RDF reasoning and data management, and conclude.

3.1 Query Answering

Ontologies are used to encode and organize information about specific domains, commonly natural and formal scientific fields or business intelligence. Formal ontology languages are vocabularies for knowledge representation, denoting types and relationships between the concepts of a domain. Such languages are often based on first-order logic (FOL) [Russell10] and include some reasoning rules for inference.

Inference can be applied with multiple purposes [Abiteboul11]. It can facilitate data integration by detecting and resolving inconsistencies between data sources. Or it can be used to support search or optimize query evaluation.

Deductive database research [Ramakrishnan95] was motivated by a desire to blend logic programming with relational databases, in order to build systems capable of efficiently handling large datasets while also enabling powerful reasoning capabilities. As such, it proposes two main approaches to reasoning. One option is to apply inference in a data-driven fashion. This strategy is called forward chaining [Russell10] and consists of the repeated application of *modus ponens* starting from a set of known facts and a set of inference rules. The inference process stops once the goal is reached or no new facts are deduced. The second option is to apply a goal-driven inference. This strategy is known as backward chaining [Russell10] and starts from a set of hypotheses working backwards to find the facts that prove them.

Among the efforts focused on devising the query language for RDF [Haase04], SPARQL was the one to became an W3C standard. Now at version 1.1, SPARQL supports aggregates, negation etc. Aiming at profiting from the legacy RDBMS optimizations, some works have looked into translating SPARQL to other query languages, like SQL [Chebotko09] and Datalog [Polleres07].

In the following, Section 3.1.1 outlines the main RDF and SPARQL fragments previously considered in RDF data management works. Sections 3.1.2 and 3.1.3 discuss each of the two established techniques for handling the crucial problem of reasoning for the purpose of answering queries over semantically rich Web data.

3.1.1 Fragments of RDF/SPARQL

As explained above, FOL is the basis of most ontology languages; however, inference in FOL is in general undecidable. Therefore, research has focused on isolating semantically interesting fragments for which inference is feasible.

A family of knowledge representation languages known as Description Logic (DL) [Baader03] provides the foundations of the Web Ontology language (OWL) [W3Ce] recommended by W3C, and consequently the RDF/RDFS ontology languages. A DL knowledge base (KB) is a first order theory made of a *Tbox* (the schema-level) and an *Abox* (the instance-level). In such a KB, the notion of logical consequence plays a role similar to that of entailment in RDF: implicit schema-level and instance-level statements can be exhibited.

Different from the RDF/RDFS specification, DL KBs rule out the possibility to express incomplete information through blank nodes. Also, it is not possible to use a class or a property as a constant in the instance-level of a DL KB (a KB being a first order theory, the sets of relations and of constants are disjoint).

The query language considered in the DL fragment of RDF is that of (union of) relational conjunctive queries, in which atoms are either of the form ClassName(s) or PropertyName(s, o), with ClassName a class and PropertyName a property. This corresponds to BGP queries whose triples are only of the form s rdf:type ClassName or s PropertyName o, not allowing the use of variables in place of classes and properties for expressing unspecified relations.

Significant *fragments* (dialects) of RDF and SPARQL study RDF query processing focusing on efficient query answering over data subject to expressive semantic constraints. The tractability of RDF/S and OWL fragments has been discussed in [terHorst05, Pichler08]. Frequently these fragments are correlated with the relational conjunctive SPARQL subset [Adjiman07, Calvanese07, Gottlob11], and its extensions [Arenas09, Goasdoué11, Kaoudi08, Urbani11].

An important factor in measuring the query answering efficiency is the choice of reasoning time. Two main approaches have been established in the literature.

First, one can opt for static reasoning in a forward chaining [Russell10] fashion. Such reasoning is data driven and applied independent of the query being asked. After establishing the considered fragment (rule set), the saturation (or closure) of a dataset is computed offline and used for answering user queries. We detail this technique in Section 3.1.2.

The alternative approach is to apply reasoning dynamically at query evaluation time. Inference here is based on the backward chaining [Russell10] and focused on the query demands. It requires reformulating the query using the chosen fragment constraints, and then evaluating the reformulated query over the fact base. More details on this approach are given in Section 3.1.3.

The literature has also considered other approaches that combine forward and backward chaining, e.g., [Christophides03, Matono05, Stuckenschmidt05, Kaoudi08, Urbani11].

3.1.2 Data Saturation

Data saturation is the static approach to reasoning. It consists of exhaustive forward inferencing by computing all the implicit triples and explicitly storing them. Each

newly inferred triple may participate in the inference of other triples, until a fixed-point is reached, meaning that no new facts can be inferred. After this, query answering is reduced to simple query evaluation over this saturated dataset.

The technique is widely used in commercial systems, due to its principal advantages:

- after the database saturation is materialized and stored, query answering can benefit from the standard optimized evaluation of such systems;
- since inference is applied off-line, it does not hinder query evaluation, making it as efficient as it can be.

The core advantage of saturation is the efficient support it provides to query processing. However, this efficiency comes at a price paid in terms of:

- the time necessary for computing the saturation;
- the storage space necessary to save the inferred data;
- the effort required to maintain (or recompute) the closure after updates. In particular, updates to the schema may lead to an important recomputation effort, as schema triples are likely to participate to many inference chains.

Data saturation is a well studied subject. Correct and complete saturation algorithms for generic inference rule sets can be found e.g., in textbooks such as [Abiteboul11]. In the particular context of inference for RDF, [Broekstra03a] proves the utility and feasibility of the exhaustive forward inferencing approach. The work also highlights the benefits of taking into account the characteristics of the entailment rules and ordering them for improved efficiency.

Particular interest has been given to diminishing the drawbacks of saturation, in particular those related to updates. The main problem considered is identifying the triples which no longer hold in the saturation as a consequence of deleting some of the explicit database triples. Therefore, incremental saturation algorithms were proposed [Broekstra03a, wwwf, Bishop11, Urbani12, Urbani13]. Such algorithms seek to maintain the saturation when the explicit schema and data triples change, as opposed to recomputing it from scratch.

The technique proposed in [Broekstra03a] relies on the storage and management of *justifications* (reasons of entailment) for implicit triples. While efficient on graphs with few entailed triples, this truth maintenance technique was not tailored for handling updates on large datasets. Handling these entailed triple justifications becomes a bottleneck as the augmented data size makes their number grow. Consequently, the work aptly points out the efficiency issues raised by saturation when there are deletions in the underlying database. To maximize efficiency, [Bishop11] proposes to compute only the relevant justifications for the entailed triples affected by an update, at maintenance time. From the expressive power viewpoint, this work focuses on the DL fragment of RDF. While efficient for instance-level updates, the authors mention that schema updates still pose a problem w.r.t. maintenance time.

The works above focus on propagating the effects of deleted explicit database triples. A distinct problem is considered in [Gutierrez11] which tackles the problem of updating an RDF graph so that a given triple is guaranteed no longer to hold in the updated graph; the authors provide a deterministic and feasible algorithm for the task.

Parallelization has also been proposed as an optimization strategy for saturation algorithms [Weaver09, Urbani10, Urbani12, Urbani13]. In [Weaver09] the authors propose a parallel approach to RDFS entailment implemented using the well-known Message Passing Interface. The work underlines the high potential for parallelization of the RDFS rules of [w3cf] given a proper ordering of the rules. The work [Urbani10] expands the ontology expressive power with the OWL Horst rules [terHorst05]. MapReduce is exploited in [Urbani12] to compute the saturation of a graph stored in a distributed file system. The work in [Urbani13] treats the problem of maintaining the materialized data closure upon addition and removal of data. It employs parallelism for optimizing data addition and derivation counts for data deletions. Instead of using a Hadoop distributed architecture such as [Urbani12], [Urbani13] makes use of the parallelism offered by multi-core hardware. In [Weaver09, Urbani12] and [Urbani13] the data is split between processes, while every process can use the full schema for reasoning in parallel.

In [Stuckenschmidt05] the authors propose an RDF fragment requiring the materialization of only a small part of the closure, and the use of query rewritings to infer all the implicit triples at run-time.

3.1.3 Query Reformulation

When certain restrictions or preferences prevent us from changing the data, the alternative approach to reasoning is to change the query in a backward chaining fashion. This implies transforming the query based on the schema constraints into a *reformulated query*, which returns the correct answers when evaluated over the explicit RDF graph.

Query reformulation has the benefit of being intrinsically robust w.r.t. data updates, since it is done at query run time. Also, even in the case of large ontologies, the reformulation process is typically swift, since it applies the entailment rules directly to the query. Therefore, in general, the reformulation process can be executed in memory.

The main drawback of reformulation lies in the fact that reformulated queries tend to be syntactically intricate. This typically increases significantly the evaluation cost.

Despite the many query reformulation algorithms proposed in the literature [Adjiman07, Arenas09, Calvanese07, Goasdoué11, Gottlob11, Kaoudi08, Urbani11], large volumes of data still pose significant difficulties for the efficient evaluation of reformulated queries, due to the query size and complexity. For example, depending on the used schema, even queries that start by having a moderate-size, can be reformulated into large unions of hundreds (even thousands) of queries. The evaluation of such unions is challenging even for the highly efficient off-the-shelf RDBMSs.

Reformulation-based query answering has been investigated in RDF fragments ranging from the Description Logic (DL) [Baader03] one [Adjiman07, Calvanese07, Gottlob11],

i.e., modeling simple DL knowledge bases, to a slight extension thereof allowing values to be used both as constants and classes/properties [Arenas09, Goasdoué11, Kaoudi08, Urbani11]. The fragments in the works mentioned above pose restrictions on triples (no blank nodes) and on entailment (only the RDFS entailment rules are considered).

Reformulation-based query answering in the DL fragment of RDF has been investigated for relational conjunctive queries [Adjiman07, Calvanese07, Gottlob11], while the slight extension thereof considered in [Arenas09, Goasdoué11, Kaoudi08, Urbani11] has been investigated for one-triple BGP queries [Kaoudi08, Urbani11], BGP queries [Goasdoué11], and SPARQL queries [Arenas09]. Because relational conjunctive queries are rule out the possibility of having variables in place of classes or properties, they are less expressive than BGP queries which allows this option.

In [Kaoudi08, Urbani11], one-triple BGP queries are reformulated using a standard backward-chaining algorithm [Russell10] on first order encodings of the entailment rules dedicated to RDFS.

In [Arenas09], SPARQL queries are reformulated into *nested* SPARQL, i.e., an extension of SPARQL in which properties in triples can be nested regular expressions. While such nested reformulated queries are more compact, the queries we produce are more practical, since their evaluation can be directly delegated to *any* off-the-shelf RDBMS, or to an RDF engine such as RDF-3X [Neumann10b] even if it is unaware of reasoning.

Among the well-established RDF data management systems, only Virtuoso [wwwh] (supporting only the rdfs:subClassOf and rdfs:subPropertyOf RDFS rules) and AllegroGraph [wwwa] (allowing more RDFS rules, but providing incomplete reasoning in some cases) support reasoning at query time.

Datalog has also been considered as a reformulation language, e.g., [Rosati10, Giacomo12] reformulate queries in a DL-Lite setting into non-recursive Datalog programs.

3.2 Storage and Indexing

An RDF dataset can be seen either as a set of triples (leading to a tabular notation) or as a labeled directed graph. Consequently, the research done in data storage has focused on efficiently representing one of these two structures.

Aiming to benefit from scalable commercial system performance, RDF data storage is often delegated to relational back ends like Jena [wwwc, Wilkinson03], OWLIM [wwwf, Bishop11], Sesame [wwwg, Broekstra02], Virtuoso [wwwh, Erling12], Oracle's Semantic Graphs [ora, Chong05] etc., or to graph databases, e.g., AllegroGraph [wwwa], Neo4J [wwwd]. New specialized engines based on relational algebra, and dedicated to RDF data storage also provide scalable solutions for querying RDF graphs [Abadi07, Neumann10b, Weiss08].

Given that a triple set can be seen as a three-attribute relation, many works investigated relational approaches to storing them. The aim of such works is to benefit from the reliability and performance of mature RDBMS technology, developed extensively over the last decades.

Triple tables. One option is storing RDF in a three-attribute relation, one for each part of the triple. This approach was adopted by Sesame [Broekstra02], Hexastore [Weiss08] and RDF-3x [Neumann08]. Such tabular storage requires computing many self joins in order to evaluate graph pattern queries. Therefore, diverse indexing and compression schemes [Neumann09, Neumann11] have been proposed to optimize the space-time performance. Notably RDF-3x [Neumann08] proposed dictionary encoding the strings denoting URIs and literals using integer keys. Header-Dictionary-Triples (HDT) [Fernández13] is a dictionary encoding submitted to the W3C enabling high compression of the stored RDF data. Effort was also invested into optimizing query evaluation by devising appropriate ways to store and index RDF graphs, e.g., [Neumann08, Neumann09, Neumann10b, Udrea07, Weiss08].

Property tables. The study [Duan11a] of frequently used RDF datasets shows that usually the number of distinct properties in a dataset is a few orders of magnitude smaller than the number of subjects and objects. Given this sparsity of properties w.r.t. the other two triple components, another relational approach is to group the data by property. As such, triples are stored into property tables, which are relations of two or more columns. The first column is dedicated to triple subject values, while the other columns in a tuple store the object values of triples with that subject and the property for which the table was built. Property tables were first considered in Jena [wwwc, Wilkinson03]. Due to the heterogeneous nature of RDF, this type of storage frequently results in sparse tables, i.e., some subjects may be connected by the same property to a variable number of objects. Clustering and partitioning were proposed for improving this storage [Levandoski09]. An alternative property table implementation is to create a two column table for storing the triple subject and object, for each property of the graph [Abadi07, Sidirourgos08].

The alternative to relational storage is to interpret the sets of triples as graphs.

Key-Value stores. RDF datasets can also be seen as directed graphs (recall Section 2.1.1). With this representation in mind, RDF data can be stored in graph data management systems. Such systems are typically Key-Value stores, relating each node (key) to the adjacent ones (values), e.g., gStore [Zou11], Neo4J [wwwd].

RDF cubes. Other works [Matono06, Atre10, Virgilio12] propose storing dictionary encoded triples as points in a 3D space, called RDF cubes.

Distributed and cloud-based RDF stores. To handle large datasets, the literature has proposed to take advantage of several distributed storage schemes, like those relying on DHTs [Kaoudi08], MapReduce [Urbani09, Urbani11, Huang11] or older parallel frameworks, e.g., C/MPI [Weaver09]. More recently, many systems have been devised with the goal of handling very large volumes of RDF data in a cloud environment; a recent survey dedicated to such systems is [Kaoudi14].

RDF query optimization and statistics. In order to make the evaluation of RDF queries more efficient, RDF statistics have been introduced for instance in [Maduko07, Maduko08, Neumann11]. Beyond the usage of indexing, cost-based RDF query optimization is studied in [Neumann09], while an heuristic-based approach is described in [Tsialiamanis12].

Materialized views for RDF. Materialized views are a well-known and very effective technique for improving the performance of query evaluation. Accordingly, proposals for automatically recommending RDF materialized views have been proposed e.g., in [Dritsou11, Goasdoué11].

3.3 Systems

Below, we outline the main features of query answering in the best-known RDF storage and management platforms.

AllegroGraph [wwwa] AllegroGraph's *RDFS++* performs run-time reasoning, sometimes incomplete, based on backward chaining. It supports all the RDFS predicates and some of OWL's. It is not complete, but it has predictable and fast performance.

Jena [wwwc] relies on saturation-based query answering.

Oracle Spatial and Graph [ora] The *RDF Semantic Graph* features of [ora] provide persistent inferencing based on forward-chaining, that supports RDFS [w3cf], OWL 2 [W3Ce], SKOS [W3Cc], and user-defined rules. While the system does not currently support query rewriting, the topic has been considered in [Wu12].

OWLIM [wwwf] Focusing on a DL fragment of RDF, it implements a forward-chaining approach for materializing all implicit information before query processing/ It then employs both inferencing techniques to compute only the relevant justifications w.r.t. an update, at maintenance time [Bishop11].

RDF data management prototypes such as Hexastore [Weiss08] or RDF-3X [Neumann08, Neumann09] assume the data is already saturated, and thus focus exclusively on query evaluation and transactions [Neumann10b]. In the latter case, the interplay between updates and semantics is not considered.

Sesame implements the justification technique [Broekstra03a] to handle entailed RDF triples.

Virtuoso [wwwh, Erling09, Erling12] Virtuoso's SPARQL compiler uses a backward chaining implementation for inferring triples that are not physically stored, meaning queries return the complete answer set without having all the implied triples materialized. Its reasoning supports some of the RDFS and OWL predicates. Virtuoso also provides explicit means to saturate the database.

WebPie [Urbani12] uses MapReduce to compute the saturation of an RDF graph.

3.4 Summary

While data saturation can be considered a mature field of research, the state of the art at the beginning of this work in 2011 showed a need for efficient incremental maintenance

algorithms. Such algorithms, are required so that saturation-based query answering remains a reasonable option in the setting of updates, notably in the case of schema updates. The work of [Broekstra03a] has shown that handling justifications for the entailed triples becomes cumbersome as the data size increases. Next [Bishop11] combines forward and backward chaining to compute only the necessary entailment justifications at maintenance time, but points out that schema updates are still problematic. In [Gutierrez11] the authors prove the feasibility of their saturation maintenance algorithms, but consider the orthogonal problem of finding which triples to delete so that entailed triples no longer hold.

Query reformulation on the other hand still presents a lot of paths worth investigating. The works exploring query answering through reformulation for the DL fragment and its extensions [Adjiman07, Calvanese07, Gottlob11, Arenas09, Goasdoué11, Kaoudi08, Urbani11] do not consider the use of blank nodes for modeling incomplete or existential information. Furthermore, the relational conjunctive queries in [Adjiman07, Calvanese07, Gottlob11] disallow the use of variables instead of classes or properties.

Keeping in mind the advantages of evaluating queries in the highly optimized existing RDBMSs, the query language considered by [Arenas09] is too general to be easily translated and plugged on top of a standard RDBMS or a specialized RDF engine, e.g., RDF-3X [Neumann10b].

Finally, the works mentioned above are focused on either improving the static, or the dynamic (run-time) approach to reasoning. Therefore, the literature shows a crucial need for a practical comparison of the two techniques in a common setting, considering both query answering and data updates.

In the following chapters we address these open problems.

Chapter 4

Query Answering in RDF Databases

A promising method for efficiently querying RDF data consists of translating SPARQL queries into efficient RDBMS-style operations. However, answering SPARQL queries requires handling RDF reasoning, which must be implemented outside the relational engines.

In Section 4.1, we introduce the database (DB) fragment of RDF, going beyond the expressive power of previously studied RDF fragments.

We devise novel sound and complete techniques for answering BGP queries within the DB fragment of RDF. These techniques are designed to be deployed on top of any RDBMS engine and explore the two established approaches for handling RDF semantics, namely data saturation (Section 4.2) and query reformulation (Section 4.3).

In particular, we focus on handling database updates within each approach, which raise specific difficulties due to the rich RDF semantics. Consequently, we propose a method for incrementally maintaining the database saturation.

This work has led to several publications, an article [Goasdoué12c] in the French conference Reconnaissance des Formes et l'Intelligence Artificielle (RFIA 2012)*, a poster [Goasdoué12b] in the* Proceedings of the 21st World Wide Web Conference (WWW 2012) *and the* INRIA research report RR-8018 *[Goasdoué12a]. The main results of this work have been published in the* Proceedings of the 16th International Conference on Extending Database Technology (EDBT 2013) *[Goasdoué13].*

4.1 The Database Fragment of RDF

We define a restriction of RDF, dubbed the *database (DB) fragment*, aiming simultaneously at an expressive fragment, and at one for which saturation- and reformulation-based query *answering* can be efficiently implemented on top of any conjunctive query *evaluation* engine, be it an RDBMS or a reasoning-agnostic RDF engine. This DB fragment is obtained by:

- *restricting RDF entailment to the rules dedicated to RDF Schema* only (a.k.a. RDFS entailment). These rules are shown in Table 2.3;
- *not restricting graphs in any way.* In other words, any triple allowed by the RDF specification is also allowed in the DB fragment.

The goal in identifying this fragment is to clearly separate an *instance level* made of assertions (or facts, or "data") and a *schema level* comprising the semantics (or constraints, or "schema"). This separation is the cornerstone of translating efficient data management techniques to the realm of RDF [Gottlob11], while remaining faithful as much as possible to the RDF specification. Making this separation in a well-principled manner is a delicate task, due to the way facts, classes, constraints, and type statements are all modeled at the same level in RDF (Section 2.1).

An RDF graph belonging to this DB fragment will hence forth be referred to as a *database*. A database $\mathtt{db}$ is a pair $\langle \mathtt{S}, \mathtt{D} \rangle$, where $\mathtt{S}$ and $\mathtt{D}$ are two disjoint sets of triples. $\mathtt{S}$ triples can only be RDFS statements such as those shown in Table 2.2. We call these triples the *schema-level* of $\mathtt{db}$. The other triples, of the forms listed in Table 2.1, where $\mathtt{p}$ is different from the four RDFS-specific properties above, belong to $\mathtt{D}$, and are called the *instance-level* of $\mathtt{db}$. Observe that $\mathtt{S}$ and $\mathtt{D}$ provide a way to partition any RDF graph (any triple $t \in \mathtt{db}$ belongs to exactly one of them, a.k.a. $(t \in \mathtt{S} \wedge t \notin \mathtt{D}) \vee (t \notin \mathtt{S} \wedge t \in \mathtt{D})$).

The DB fragment is delimited from the general RDF model by restricting the immediate entailment rules to *only* those listed in Table 2.3. The saturation of a database $\mathtt{db}$ with this aforementioned entailment rule set is denoted $\mathtt{db}^{\propto}$, thus $\mathtt{db}^{\propto} \subseteq \mathtt{db}^{\infty}$.

Example 4.1 (***RDF database – running example***)**.**
The RDF database $\mathtt{db} = \langle \mathtt{S}, \mathtt{D} \rangle$*, whose triples are shown below, is used as a running example throughout this chapter.*

The instance of this database describes the resource $:\mathsf{doi}_1$ *that belongs to an unknown class, whose title is* "RDF Analytics: Lenses over Semantic Graphs"*, whose author is* "Alexandra Roatis" *and having an unknown contact author. This paper is in the proceedings of an unknown resource whose name is* "WWW'14"*. Lastly, the URI* :edbt2013 *is a conference and* :name*, the property associating names to resources, was created by* "John Doe"*.*

$$
\begin{aligned}
\mathtt{D} = \{\ & :\mathsf{doi}_1\ \mathsf{rdf{:}type}\ \_{:}\mathsf{b}_0\ , \\
& :\mathsf{doi}_1\ :\mathsf{title}\ \text{"RDF Analytics: Lenses over Semantic Graphs"}\ , \\
& :\mathsf{doi}_1\ :\mathsf{author}\ \text{"Alexandra Roatis"}\ , \\
& :\mathsf{doi}_1\ :\mathsf{contactAuthor}\ \_{:}\mathsf{b}_1\ , \\
& :\mathsf{doi}_1\ :\mathsf{inProceedingsOf}\ \_{:}\mathsf{b}_2\ , \\
& :\mathsf{name}\ :\mathsf{createdBy}\ \text{"John Doe"}\ , \\
& :\mathsf{edbt2013}\ \mathsf{rdf{:}type}\ :\mathsf{Conference}\ , \\
& \_{:}\mathsf{b}_2\ :\mathsf{name}\ \text{"WWW'14"}\ \}
\end{aligned}
$$

Next to this instance comes a schema stating that poster papers together with the unknown class $\_{:}\mathsf{b}_0$ *of which* $:\mathsf{doi}_1$ *is an instance, are sub-classes of conference papers, which are scientific papers. Moreover, titles, authors, and contact authors (themselves*

a particular case of authors) are used to describe papers, which are connected to the conferences in whose proceedings they appear. Finally, names describe conferences, and creators describe resources.

$$
\begin{aligned}
\mathtt{S} = \{ & \text{:PosterConferencePaper rdfs:subClassOf :ConferencePaper ,} \\
& \text{\_:b}_0 \text{ rdfs:subClassOf :ConferencePaper ,} \\
& \text{:ConferencePaper rdfs:subClassOf :Paper ,} \\
& \text{:title rdfs:domain :Paper ,} \\
& \text{:title rdfs:range rdf:Literal ,} \\
& \text{:author rdfs:domain :Paper ,} \\
& \text{:author rdfs:range rdf:Literal ,} \\
& \text{:contactAuthor rdfs:subPropertyOf :author ,} \\
& \text{:inProceedingsOf rdfs:domain :ConferencePaper ,} \\
& \text{:inProceedingsOf rdfs:range :Conference ,} \\
& \text{:name rdfs:domain :Conference ,} \\
& \text{:name rdfs:range rdf:Literal ,} \\
& \text{:createdBy rdfs:range rdf:Literal } \}
\end{aligned}
$$

Figure 4.1 depicts graphically the database db.

4.1.1 Query Evaluation on the DB Fragment

The *evaluation of a BGP query q against a database* db is exactly the evaluation of q against the graph db, i.e., $q(\mathtt{db})$, and the *answer set of q against* db is $q(\mathtt{db}^\infty)$, thus $q(\mathtt{db}^\infty) \subseteq q(\mathtt{db}^\infty)$.

User queries may traverse both the schema- and instance-level of the database. In our running example, one can ask for the ranges of properties describing conference papers:

$$\text{ClassRelatedToConfPaper}(?t) \text{ :- } ?x \text{ rdf:type :ConferencePaper, } ?x\ ?y\ ?z,\ ?y \text{ rdfs:range } ?t$$

However, the separation between schema and data, corresponds to many users' intuitive comprehension of the database. In some settings, users may want to specify that their queries be evaluated *only against the instance-level* or *only against the schema-level* database.

From a database perspective, queries whose evaluation is asked against the (saturated) instance-level database only are the most familiar. While schema triples are not returned by such queries, they do impact their answer, because the saturation of the instance-level database (necessary in order to return complete answers) relies on the schema-level triples. For instance, an instance-level query returning the city of EDBT 2013 is:

$$\text{EDBTCity}(?y) \text{ :- } ?x \text{ :name “EDBT'13”, } ?x \text{ :city } ?y$$

Instance-level queries can also return classes and properties associated to specific values. For instance, one can ask for the classes to which a given resource :res belongs:

$$\text{ClassFinding}(?x) \text{ :- :res rdf:type } ?x$$

From a knowledge representation perspective, a class of interesting queries can be evaluated over the schema-level database alone. Such queries offer a convenient means to *explore the relationships between the classes and properties of a schema, including the implied relationships*. For instance, one can ask whether a given class is a sub-class of another:

$$\mathsf{SubClassChecking}() \text{ :- :PosterConferencePaper rdfs:subClassOf :Paper}$$

or, what are the classes typing the domain of a given property:

$$\mathsf{DomainFinding}(?x) \text{ :- :inProceedingsOf rdfs:domain } ?x$$

Another example of an exploration query is:

$$\mathsf{AllTriples}(?x, ?y, ?z) \text{ :- } ?x\ ?y\ ?z$$

returning all the triples of the database. By restricting the query to only the schema-level database, the user retrieves all direct or entailed relationships among classes and properties.

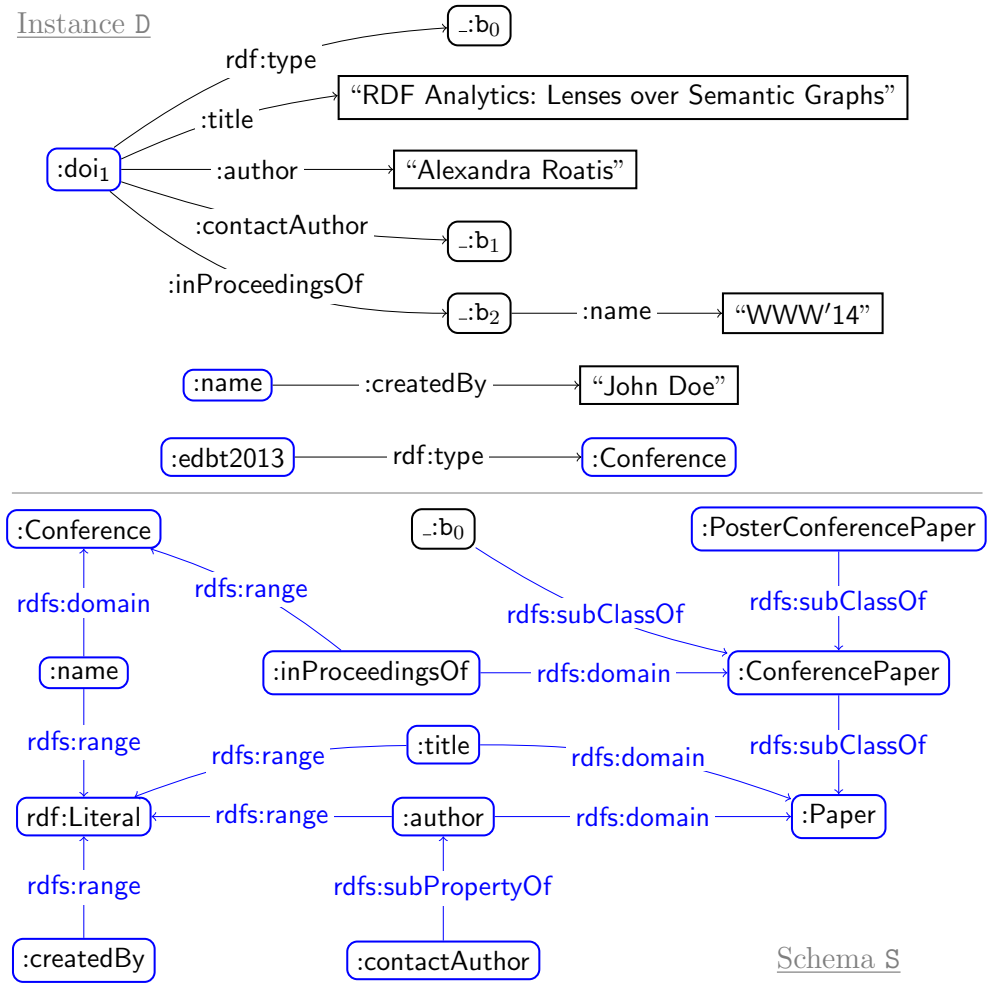


FIGURE 4.1: Running example: RDF database – graph representation.

The above discussion shows that our setting is general enough to integrate both database-style instance-level querying and knowledge representation-style schema-level querying, while also allowing a smooth integration of both levels through queries on both database components.

In this chapter, given the overwhelming practical impact of querying only the instance-level (implicit and explicit) data, the focus is set on efficient query answering algorithms for this problem.

4.1.2 Query Answering on the DB Fragment

This thesis investigates two query answering techniques against RDF databases, namely *saturation-* and *reformulation-based*. Each technique performs a specific *pre-processing* step, either on the database or on the queries, to deal with entailed triples, after which query answering is reduced to query evaluation.

Saturation-based query answering is rather straightforward. The saturation of the database is computed using the allowed entailment rules. Then, the answer set of every query against the (original) database is obtained by query evaluation against the saturation. The advantage of this approach is that it is easy to implement. Notably, query evaluation can be delegated to a standard RDBMS as illustrated in Section 2.3. Its disadvantages are that database saturation needs time to be computed and space to store all the entailed triples. Moreover, the saturation must be somehow recomputed upon every database update.

Reformulation-based query answering reformulates a query q w.r.t. a database $\mathtt{db}$ into another query q' (using the immediate entailment rules), so that the evaluation of q' against the (original) database $\mathtt{db}$, denoted $q'(\mathtt{db})$, is exactly the answer set of q against $\mathtt{db}$ (i.e., $q(\mathtt{db}^{\infty})$). The advantage of reformulation is that the database saturation does not need to be (re)computed. The disadvantage is that every incoming query must be reformulated, which often results in a more complex query.

The following sections focus on saturation- and reformulation-based query answering only for *instance-level queries*. Theorem 4.2 shows that to answer such queries, among the DB fragment's rules shown in Table 2.3, it suffices to consider only the entailment rules in Table 2.3(d).

Theorem 4.2. *Let* $\mathtt{db}$ *be a database,* t_1 *be a triple of the form* $\mathtt{s}$ $\mathtt{rdf{:}type}$ $\mathtt{o}$*, and* t_2 *be a triple of the form* $\mathtt{s}$ $\mathtt{p}$ $\mathtt{o}$*.* $t_1 \in \mathtt{db}^{\infty}$ *(respectively,* $t_2 \in \mathtt{db}^{\infty}$*) iff there exists a sequence of application of the rules in Table 2.3(d) leading from* $\mathtt{db}$ *to* t_1 *(respectively* t_2*), assuming that each entailment step relies on* $\mathtt{db}$ *and all triples previously entailed.*

Appendix A.1 reports the proof for Theorem 4.2.

4.2 The Saturation-based Approach

The first and simplest query answering technique resembles those previously discussed in the literature [Abadi07, Weiss08, Neumann09, Sidirourgos08, Neumann08]: computing

$$\frac{\{\mathbf{c}_1 \text{ rdfs:subClassOf } \mathbf{c}_2, \mathtt{s} \text{ rdf:type } \mathbf{c}_1\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s} \text{ rdf:type } \mathbf{c}_2\}} \tag{4.1}$$

$$\frac{\{\mathbf{p} \text{ rdfs:domain } \mathbf{c}, \mathtt{s}\ \mathbf{p}\ \mathtt{o}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s} \text{ rdf:type } \mathbf{c}\}} \tag{4.2}$$

$$\frac{\{\mathbf{p} \text{ rdfs:range } \mathbf{c}, \mathtt{s}\ \mathbf{p}\ \mathtt{o}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{o} \text{ rdf:type } \mathbf{c}\}} \tag{4.3}$$

$$\frac{\{\mathbf{p}_1 \text{ rdfs:subPropertyOf } \mathbf{p}_2, \mathtt{s}\ \mathbf{p}_1\ \mathtt{o}\} \subseteq \mathtt{db}}{\mathtt{db} = \mathtt{db} \cup \{\mathtt{s}\ \mathbf{p}_2\ \mathtt{o}\}} \tag{4.4}$$

FIGURE 4.2: Saturation rules for an RDF database db.

the instance-level saturation of a given database, then evaluating the original query against it (Section 4.2.1). This thesis' main contribution in the area of saturation-based query answering is showing how to efficiently handle *database changes at the instance- or schema-level*. Section 4.2.2 provides a novel incremental algorithm for saturation maintenance, while Section 4.2.3 formally establishes the correctness of our saturation-based query answering technique.

4.2.1 Database Saturation

The `Saturate` algorithm relies on the saturation rules in Figure 4.2, which are a direct implementation of the entailment rules in Table 2.3(d). In Figure 4.2 and in the sequel, the bold symbols (possibly with subscripts) $\mathbf{c}$ for a class, and $\mathbf{p}$ for a property, denote some unspecified values.

The rules in Figure 4.2 define a set of database transformations of the form $\frac{input}{output}$, where the input consists of a database satisfying a boolean condition and the output is a new database for which the entailed triple was made explicit. Intuitively, given a database `db`, `Saturate(db)` applies exhaustively the rules in Figure 4.2, on `db` plus all the gradually generated triples.

The output of `Saturate(db)` is defined as the fixed-point $\texttt{Saturate}^{\infty}(\texttt{db})$, where:

$$\begin{aligned}
&\texttt{Saturate}^0(\texttt{db}) = \texttt{db}\\
&\texttt{Saturate}^{k+1}(\texttt{db}) = \texttt{Saturate}^k(\texttt{db}) \cup \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ s.t.}\\
&\qquad \text{applying rule } (i) \text{ on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^k(\texttt{db})\\
&\qquad \text{produces } t_3, \text{ where } t_2 \notin \texttt{Saturate}^{k-1}(\texttt{db}) \text{ when defined } \}
\end{aligned}$$

Example 4.3 (***Saturation of a database***).
The saturation of the `db` *described in Figure 4.1 is shown below.*

$$
\begin{aligned}
\texttt{Saturate}^0(\texttt{db}) &= \texttt{db} \\
\\
\texttt{Saturate}^1(\texttt{db}) &= \texttt{Saturate}^0(\texttt{db}) \cup \\
&\quad \{\ :\text{doi}_1\ \text{rdf:type}\ :\text{ConferencePaper}\ , \\
&\quad\ \ :\text{doi}_1\ \text{rdf:type}\ :\text{Paper}\ , \\
&\quad\ \ :\text{doi}_1\ :\text{author}\ \_:\text{b}_1\ , \\
&\quad\ \ \_:\text{b}_2\ \text{rdf:type}\ :\text{Conference}\ \} \\
\\
\texttt{Saturate}^2(\texttt{db}) &= \texttt{Saturate}^1(\texttt{db})
\end{aligned}
$$

Theorem 4.4 shows that the `Saturate` algorithm terminates. It also provides upper bounds for the size of the output saturation and its computation time.

Theorem 4.4. *Given a database* `db`*, the size (number of triples) of the output of* `Saturate(db)` *is in* $O(\#\texttt{db}^2)$ *and the time to compute it is in* $O(\#\texttt{db}^3)$*, with* $\#\texttt{db}$ *the size (number of triples) of* `db`*.*

Appendix A.2 reports the proof for Theorem 4.4.

Our experiments (Table 5.1 in Chapter 5) show that in practical cases, `Saturate(db)` has a more moderate size, but it can still be significantly larger than `db`; moreover, in practical databases, the theoretical time complexity is far from being reached (Figure 5.2).

4.2.2 Saturation Maintenance upon Updates

Saturation-based query answering is efficient at query time, since one only has to evaluate the original query. However, the saturation must be somehow recomputed to reflect the impact of updates.

This section studies the problem of efficiently maintaining the database saturation upon two kinds of updates: triple *insertion* and *deletion*. Taking inspiration from the rich literature on incremental view maintenance in databases [Gupta99], the aim is to devise *incremental* algorithms, which do not re-compute the saturation, but just modify it to reflect the update.

An important issue is to keep track of the *multiple ways in which a triple was entailed (i.e., derived)*. This is significant when considering both implicit data and updates: for a given update, we must decide whether this adds/removes one *reason why a triple belongs to the saturation*. When this count reaches 0, the implied triple should be removed. A naive implementation would record the inference paths of each implied triple, that is: all sequences of reasoning rules that have lead to that triple being present in the saturation. However, as shown in [Broekstra03b], the volume of such justification grows very fast and thus the approach does not scale. Instead, we chose to keep track of the *number of reasons* why a triple has been inferred, and provide maintenance algorithms which rely only on this (much more compact) information.

We extend the previous notion of database saturation, so that it becomes a *multiset* in which a triple appears as many times as it can be entailed. Formally, given a database `db`, the saturation is now defined as the fixed-point $\texttt{Saturate}_+^\infty(\texttt{db})$ obtained from the following $\texttt{Saturate}_+$ algorithm, where $\uplus$ is the union operator for multisets.

$$\begin{aligned}
&\texttt{Saturate}_+^0(\texttt{db}) = \texttt{db}\\
&\texttt{Saturate}_+^{k+1}(\texttt{db}) = \texttt{Saturate}_+^k(\texttt{db}) \uplus \{t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that}\\
&\qquad \text{applying rule } (i) \text{ on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^k(\texttt{db})\\
&\qquad \text{produces } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<k}(\texttt{db}) \text{ when defined}\}
\end{aligned}$$

Proposition 4.5 expresses the obvious relationship between the set-based saturation and the multiset-based saturation of a database; $\texttt{set}(\cdot)$ returns the set of (distinct) elements occurring in a given multiset.

Proposition 4.5. *For any RDF database* db

$$\texttt{Saturate}(\texttt{db}) = \texttt{set}(\texttt{Saturate}_+(\texttt{db})) \textit{ holds.}$$

Appendix A.3 reports the proof for Proposition 4.5.

Example 4.6 (***Multiset saturation of a database***)**.**
Consider again the previously introduced database db *described in Figure 4.1. Its multiset-based saturation is shown below.*

$$\begin{aligned}
\texttt{Saturate}_+^0(\texttt{db}) \;=\;& \texttt{db}\\
\texttt{Saturate}_+^1(\texttt{db}) \;=\;& \texttt{Saturate}_+^0(\texttt{db}) \uplus\\
&\{\ :\text{doi}_1\ \text{rdf:type}\ :\text{ConferencePaper}\ ,\\
&\ \ :\text{doi}_1\ \text{rdf:type}\ :\text{Paper}\ ,\\
&\ \ :\text{doi}_1\ \text{rdf:type}\ :\text{Paper}\ ,\\
&\ \ :\text{doi}_1\ :\text{author}\ _:\text{b}_1\ ,\\
&\ \ :\text{doi}_1\ \text{rdf:type}\ :\text{ConferencePaper}\ ,\\
&\ \ _:\text{b}_2\ \text{rdf:type}\ :\text{Conference}\ ,\\
&\ \ _:\text{b}_2\ \text{rdf:type}\ :\text{Conference}\ \}\\
\texttt{Saturate}_+^2(\texttt{db}) \;=\;& \texttt{Saturate}_+^1(\texttt{db}) \uplus\\
&\{\ :\text{doi}_1\ \text{rdf:type}\ :\text{Paper}\ ,\\
&\ \ :\text{doi}_1\ \text{rdf:type}\ :\text{Paper}\ ,\\
&\ \ :\text{doi}_1\ \text{rdf:type}\ :\text{Paper}\ \}\\
\texttt{Saturate}_+^3(\texttt{db}) \;=\;& \texttt{Saturate}_+^2(\texttt{db})
\end{aligned}$$

With the multiset-based saturation and Proposition 4.5 in place, Theorem 4.7 shows how saturation can be *incrementally* maintained upon update; $\uplus$ is again multiset union, while $\setminus_+$ is multiset difference.

Theorem 4.7. *Let* $\texttt{db} = \langle \texttt{S}, \texttt{D} \rangle$ *be a database.*

Insertion: $\texttt{Saturate}_+(\texttt{db} \cup \{t\}) \;=$

- $\texttt{Saturate}_+(\texttt{db})$ *if* $t \in \texttt{db}$. *Otherwise,* $t \notin \texttt{db}$ *and:*

- $\texttt{Saturate}_+(\texttt{db}) \uplus [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]$ *if* t *is an instance-level triple;*
- $\texttt{Saturate}_+(\texttt{db}) \uplus \{t\} \uplus \biguplus_{t' \in \texttt{D}'} [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t'\}\rangle) \setminus_+ \texttt{S}\ ]$, *where the multiset* $\texttt{D}'$ *is* $\{t_3 \mid \exists\, i \in [4.1, 4.4]$ *such that applying rule (i) on* $\{t, t_2\}$ *with* $t_2 \in \texttt{Saturate}_+(\texttt{db})$ *yields* $t_3\}$, *if* t *is a schema-level triple.*

Deletion: $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) \ =$

- $\texttt{Saturate}_+(\texttt{db})$ *if* $t \notin \texttt{db}$. *Otherwise,* $t \in \texttt{db}$ *and:*
- $\texttt{Saturate}_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]$ *if* t *is an instance-level triple;*
- $\texttt{Saturate}_+(\texttt{db}) \setminus_+ \{t\} \setminus_+ \biguplus_{t' \in \texttt{D}'} [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t'\}\rangle) \setminus_+ \texttt{S}\ ]$ *where the multiset* $\texttt{D}'$ *is* $\{t_3 \mid \exists\, i \in [4.1, 4.4]$ *such that applying rule (i) on* $\{t, t_2\}$ *with* $t_2 \in \texttt{Saturate}_+(\texttt{db})$ *yields* $t_3\}$, *if* t *is a schema-level triple.*

Appendix A.4 reports the proof for Theorem 4.7.

Theorem 4.7 reads as follows. Inserting a triple already in the database, or deleting a triple that is not in the database, does not require any work. Otherwise, inserting (deleting) a given instance or schema triple also adds to (removes from) the current saturation all instance-level triples whose derivation uses this given triple.

Optimized implementation. From a practical viewpoint, the multiset $\texttt{Saturate}_+(\texttt{db})$ can be compactly stored (Example 4.8) as the set $\texttt{Saturate}(\texttt{db})$, for which every triple is tagged with:

(*i*) a boolean indicating whether it *belongs to* db (T) or is *only entailed by* db (F), and

(*ii*) an integer indicating how many times it appears in $\texttt{Saturate}_+(\texttt{db})$.

The above provides a single lightweight representation for db, $\texttt{Saturate}(\texttt{db})$, and $\texttt{Saturate}_+(\texttt{db})$.

Example 4.8 (***Compact storage of the multiset saturation of a database***). *Given* $\texttt{Saturate}_+(\texttt{db})$ *illustrated in Example 4.6, its compact representation is:*

$$\begin{aligned}\texttt{Saturate}_+(\texttt{db}) = \texttt{S} \uplus \{\ &(\ :\text{doi}_1\ \text{rdf:type}\ \_{:}\text{b}_0\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ :\text{doi}_1\ \text{:title}\ \text{“RDF Analytics: ...”}\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ :\text{doi}_1\ \text{:author}\ \text{“Alexandra Roatis”}\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ :\text{doi}_1\ \text{:contactAuthor}\ \_{:}\text{b}_1\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ :\text{doi}_1\ \text{:inProceedingsOf}\ \_{:}\text{b}_2\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ \text{:name :createdBy “John Doe”}\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ \text{:edbt2013 rdf:type :Conference}\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ \_{:}\text{b}_2\ \text{:name “WWW’14”}\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ :\text{doi}_1\ \text{rdf:type :ConferencePaper}\ ,\ \text{false}\ ,\ 2\ ),\\ &(\ :\text{doi}_1\ \text{rdf:type :Paper}\ ,\ \text{false}\ ,\ 5\ ),\\ &(\ :\text{doi}_1\ \text{:author}\ \_{:}\text{b}_1\ ,\ \text{false}\ ,\ 1\ ),\\ &(\ \_{:}\text{b}_2\ \text{rdf:type :Conference}\ ,\ \text{false}\ ,\ 2\ )\ \}\end{aligned}$$

The example below shows the saturation maintenance process for an instance insertion and a schema deletion:

(*i*) *First we illustrate maintaining the saturation upon the insertion of a triple* $t =$ $:\text{doi}_2$:inProceedingsOf :edbt2013. *The triple is saturated using the schema* $\mathtt{S}$, *resulting in a set of entailed triples represented as* $\mathtt{Saturate}_+(\langle \mathtt{S}, \{t\}\rangle) \setminus_+ \mathtt{S}$ *in Theorem 4.7. This set contains three new triples, one of which already exists explicitly in the saturation. The updates made on* db *are shown below, where the first triple is updated and the other three are inserted:*

$$\begin{aligned}\mathtt{Saturate}_+(\mathtt{db}) = \{\ \ldots\ ,\ &(\ :\text{edbt2013 rdf:type :Conference}\ ,\ \text{true}\ ,\ \mathbf{2}\ ),\\ \ldots\ ,\ &(\ :\text{doi}_2\ :\text{inProceedingsOf :edbt2013}\ ,\ \text{true}\ ,\ 1\ ),\\ &(\ :\text{doi}_2\ \text{rdf:type :ConferencePaper}\ ,\ \text{false}\ ,\ 1\ ),\\ &(\ :\text{doi}_2\ \text{rdf:type :Paper}\ ,\ \text{false}\ ,\ 1\ )\ \}\end{aligned}$$

(*ii*) *Deleting the schema triple* $t =$:contactAuthor rdfs:subPropertyOf :author *causes its removal from* $\mathtt{Saturate}_+(\mathtt{db})$, *together with all the instance triples entailed by it and the database instance:*

$$\mathtt{D}' = \{\ :\text{doi}_1\ :\text{author}\ \_{:}\text{b}_1\ \},$$

and the instance triples entailed by $\mathtt{D}'$ *and the schema:*

$$\biguplus_{t' \in \mathtt{D}'} [\ \mathtt{Saturate}_+(\langle \mathtt{S}, \{t'\}\rangle) \setminus_+ \mathtt{S}\ ] = \{\ :\text{doi}_1\ \text{rdf:type :Paper}\ \}.$$

Notice that only one instance of $:\text{doi}_1$ rdf:type :Paper *is removed (the count is decreased to* $\boldsymbol{4}$*), as it still is entailed by other facts.*

Handling cyclic hierarchies. The RDF Schema specification [w3cf] allows cyclic sub-class and sub-property hierarchies. Such relations can be used to model equivalent classes or properties. For example, given two classes $\mathtt{c}_1$ and $\mathtt{c}_2$, declaring the triples $\mathtt{c}_1$ rdfs:subClassOf $\mathtt{c}_2$ and $\mathtt{c}_2$ rdfs:subClassOf $\mathtt{c}_1$ amounts to asserting that the two classes are equivalent, i.e., all instances of $\mathtt{c}_1$ are also instances of $\mathtt{c}_2$ and all instances of $\mathtt{c}_2$ are also instances of $\mathtt{c}_1$.

From an implementation view-point, such cycles may create issues when computing the saturation. Due to the set semantics of the Saturate algorithm new inferred triples are considered for subsequent iterations only if they are not already present in the saturation. On the other hand when moving from sets to multisets as in the case of the $\mathtt{Saturate}_+$ algorithm, such cyclic hierarchies may lead to an infinite multiset as shown in Example 4.9.

Example 4.9 (***Infinite multiset saturation of a database***)**.**
Consider the database $\mathtt{db}_c$ *made of the following three triples declaring a cyclic hierarchy between the classes* $\mathtt{c}_1$ *and* $\mathtt{c}_2$, *and that the resource* :res *belongs to the class* $\mathtt{c}_1$.

$$\begin{aligned}&\mathtt{c}_1\ \text{rdfs:subClassOf}\ \mathtt{c}_2\\ &\mathtt{c}_2\ \text{rdfs:subClassOf}\ \mathtt{c}_1\\ &:\text{res rdf:type}\ \mathtt{c}_1\end{aligned}$$

Its multiset-based saturation is shown below.

$$\texttt{Saturate}_+^0(\texttt{db}) = \texttt{db} \quad (0)$$

$$\texttt{Saturate}_+^1(\texttt{db}) = \texttt{Saturate}_+^0(\texttt{db}) \uplus \{ \text{:res rdf:type } c_2 \} \quad (1)$$

$$\texttt{Saturate}_+^2(\texttt{db}) = \texttt{Saturate}_+^1(\texttt{db}) \uplus \{ \text{:res rdf:type } c_1 \} \quad (2)$$

$$\texttt{Saturate}_+^3(\texttt{db}) = \texttt{Saturate}_+^2(\texttt{db}) \uplus \{ \text{:res rdf:type } c_2 \} \quad (3)$$

$$\texttt{Saturate}_+^4(\texttt{db}) = \texttt{Saturate}_+^3(\texttt{db}) \uplus \{ \text{:res rdf:type } c_1 \} \quad (4)$$

$\ldots$

Notice that at each odd number iteration a new triple of the form :res rdf:type c_2 *is added to the saturation multiset, which leads to the addition of a triple of the form* :res rdf:type c_1 *at each even number iteration, resulting into an infinite loop.*

There are two classical ways of handling the problem of cycles [Russell10].

I. *Static cycle analysis* can be applied before running the $\texttt{Saturate}_+$ algorithm to find the cycles in the schema. From these cycles we obtain the sets of equivalent classes/properties. Then we view each set as a single class/property and do not propagate values among them during saturation.

II. The second approach is to *dynamically control cycles* during saturation. For this we use a history of the classes/properties in which values are propagated through rdfs:subClassOf/rdfs:subPropertyOf, and set a blocking condition to prevent us from propagating a value to the same class/property more than once.

The implementation used for evaluating the algorithms in Chapter 5 is based on the former approach. This variant was chosen with the aim of reducing the saturation run-time memory consumption. Also, common-use RDF datasets do not frequently present such cycles. In particular, the datasets used for experiments in Chapter 5 (and frequently considered in the literature) are cycle free. Due to this diminished frequency, taking cycles into account only when they are a certainty is preferable and a natural optimization.

4.2.3 Saturation-based Query Answering

Based on the above notion of saturation, Theorem 4.10 shows the saturation-based query answering technique.

Theorem 4.10. *Given a BGP query q and a database* db*, the following holds:*

$$q(\mathtt{db}^{\infty}) = q(\mathtt{Saturate}(\mathtt{db})) = q(\mathtt{set}(\mathtt{Saturate}_{+}(\mathtt{db}))).$$

The proof for Theorem 4.10 trivially follows from Theorem 4.2, the definition of the `Saturate` algorithm (Section 4.2.1), and Proposition 4.5.

From a practical viewpoint, and based on the observations from Section 2.3, saturation-based query answering can be delegated to an RDBMS by:

(*i*) storing either `Saturate(db)` or the aforementioned compact representation of $\mathtt{Saturate}_{+}(\mathtt{db})$ in the `Triple` table, and

(*ii*) evaluating queries using the RDBMS engine.

Example 4.11 (***Saturation-based query answering***)**.**
Consider the query q asking for all resources and the classes to which they belong:

$$q(?x, ?y) \text{ :- } ?x \text{ rdf:type } ?y$$

The answer set of q against the previous database `db` *(Figure 4.1) is:*

$$q(\mathtt{Saturate}(\mathtt{db})) = q(\mathtt{set}(\mathtt{Saturate}_{+}(\mathtt{db}))) = \{ \langle :\mathsf{doi}_1, \_:\mathsf{b}_0 \rangle , \langle :\mathsf{doi}_1, :\mathsf{ConferencePaper} \rangle , \langle :\mathsf{edbt2013}, :\mathsf{Conference} \rangle , \langle :\mathsf{doi}_1, :\mathsf{Paper} \rangle , \langle \_:\mathsf{b}_2, :\mathsf{Conference} \rangle \}.$$

4.3 The Reformulation-based Approach

Given a query q and a database `db`, Sections 4.3.1 and 4.3.2 introduces an algorithm for reformulating queries in the DB fragment. The expressive power of the RDF DB fragment, however, widens the gap between query answering and conjunctive query evaluation: in this setting, simply evaluating the queries resulting from reformulation does not suffice to compute the correct result. To bridge this gap, Section 4.3.3 introduces a novel *non-standard query evaluation*, which, applied on reformulated queries, computes (still relying on an RDF-agnostic conjunctive query processor, e.g., an RDBMS) the sound and complete answer sets in our expressive DB fragment.

4.3.1 Query Reformulation

The `Reformulate` algorithm exhaustively applies the set of rules shown in Figure 4.3, starting from a query q and a database `db`. Each rule defines a transformation of the form $\frac{input}{output}$, where the input is of the form $\langle$ *logical condition on* `db` , *logical condition on* q $\rangle$

$$\frac{\langle \mathsf{s}\ ?y\ \mathsf{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?y \to \mathsf{rdf:type}\}}} \tag{4.5}$$

$$\frac{\langle \mathsf{s}_1\ \mathbf{p}\ \mathsf{o}_1 \in \mathsf{db}, \mathsf{s}\ ?y\ \mathsf{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?y \to \mathbf{p}\}}} \tag{4.6}$$

$$\frac{\langle \mathsf{s}_1\ \mathsf{rdfs:subPropertyOf}\ \mathbf{p} \in \mathsf{db}, \mathsf{s}\ ?y\ \mathsf{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?y \to \mathbf{p}\}}} \tag{4.7}$$

$$\frac{\langle \mathbf{p}\ \mathsf{rdfs:subPropertyOf}\ \mathsf{o}_1 \in \mathsf{db}, \mathsf{s}\ ?y\ \mathsf{o} \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?y \to \mathbf{p}\}}} \tag{4.8}$$

$$\frac{\langle \mathsf{s}_1\ \mathsf{rdf:type}\ \mathbf{c} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ ?z \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?z \to \mathbf{c}\}}} \tag{4.9}$$

$$\frac{\langle \mathsf{s}_1\ \mathsf{rdfs:subClassOf}\ \mathbf{c} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ ?z \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?z \to \mathbf{c}\}}} \tag{4.10}$$

$$\frac{\langle \mathbf{c}\ \mathsf{rdfs:subClassOf}\ \mathsf{o} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ ?z \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?z \to \mathbf{c}\}}} \tag{4.11}$$

$$\frac{\langle \mathsf{s}_1\ \mathsf{rdfs:domain}\ \mathbf{c} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ ?z \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?z \to \mathbf{c}\}}} \tag{4.12}$$

$$\frac{\langle \mathsf{s}_1\ \mathsf{rdfs:range}\ \mathbf{c} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ ?z \in q_\sigma \rangle}{q_{\sigma \cup \nu = \{?z \to \mathbf{c}\}}} \tag{4.13}$$

$$\frac{\langle \mathbf{c}_1\ \mathsf{rdfs:subClassOf}\ \mathbf{c}_2 \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c}_2 \in q_\sigma \rangle}{q_{\sigma[\mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c}_2 / \mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c}_1]}} \tag{4.14}$$

$$\frac{\langle \mathbf{p}\ \mathsf{rdfs:domain}\ \mathbf{c} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c} \in q_\sigma \rangle}{q_{\sigma[\mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c} / \mathsf{s}\ \mathbf{p}\ ?y]}} \tag{4.15}$$

$$\frac{\langle \mathbf{p}\ \mathsf{rdfs:range}\ \mathbf{c} \in \mathsf{db}, \mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c} \in q_\sigma \rangle}{q_{\sigma[\mathsf{s}\ \mathsf{rdf:type}\ \mathbf{c} / ?y\ \mathbf{p}\ \mathsf{s}]}} \tag{4.16}$$

$$\frac{\langle \mathbf{p}_1\ \mathsf{rdfs:subPropertyOf}\ \mathbf{p}_2 \in \mathsf{db}, \mathsf{s}\ \mathbf{p}_2\ \mathsf{o} \in q_\sigma \rangle}{q_{\sigma[\mathsf{s}\ \mathbf{p}_1\ \mathsf{o} / \mathsf{s}\ \mathbf{p}_2\ \mathsf{o}]}} \tag{4.17}$$

FIGURE 4.3: Reformulation rules for a partially instantiated query q_σ w.r.t. a database db.

and the output is a query q'. Each, but not both of the conditions in the input may be unspecified. Intuitively, each rule produces a new query when the rule's input conditions are satisfied, one by the database db, and the other by some query (either the original query q or a query q' produced by a previous application of a rule). The set of all queries produced by applying the rules is the result of the reformulation of q w.r.t. db.

A key concept for our reformulation-based query answering are:

Definition 4.12 (***Partially instantiated queries***).
Let $q(\bar{x})$:- $t_1, \ldots, t_\alpha$ be a query and σ be a mapping from a subset of q's variables and blank nodes, to some values (URIs, blank nodes, or literals).

The *partially instantiated query* q_σ is a query $q_\sigma(\bar{x}_\sigma) \text{ :- } (t_1, \ldots, t_\alpha)_\sigma$ where σ has been applied both on q's head variables $\bar{x}$ and on q's body. In a non-standard fashion, some distinguished (head) variables of q_σ can be bound. If $\sigma = \emptyset$, then q_σ coincides with the original (non-instantiated) query q.

By allowing constants in the head, partially instantiated queries go outside the reach of the defined BGP queries. Accordingly, a slight extension is required to the notions of BGP query evaluation and answer set, introduced in Section 2.2.1 for graphs and in Section 4.1.1 for databases, as follows.

Given a database $\mathtt{db}$ whose set of values (URIs, blank nodes, literals) is $\mathtt{Val}(\mathtt{db})$ and a query $q_\sigma(\bar{x}_\sigma) \text{ :- } (t_1, \ldots, t_\alpha)_\sigma$ whose set of variables and blank nodes is $\mathtt{VarBl}(q_\sigma)$, the *evaluation* of q_σ against $\mathtt{db}$ is:

$$q_\sigma(\mathtt{db}) = \{\ (\bar{x}_\sigma)_\mu \mid \mu : \mathtt{VarBl}(q_\sigma) \to \mathtt{Val}(\mathtt{db}) \text{ is a total assignment such that } ((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq \mathtt{db}\ \}$$

The *answer set* of q_σ against $\mathtt{db}$ is the evaluation of q_σ against $\mathtt{db}^\infty$, denoted $q_\sigma(\mathtt{db}^\infty)$.

4.3.2 Reformulation Rules and Algorithm

The rules (4.5)–(4.13) reformulate queries by binding one of their variables, either to the built-in property rdf:type or to a class or property name picked in the database. The other rules (4.14)–(4.17) replace some query triple with another, based on schema-level triples.

Consider for instance rule (4.5). The rule says: if a triple of the form $\mathtt{s}\ ?y\ \mathtt{o}$, i.e., having any kind of subject or object, but having a variable in the property position, appears in q_σ, then create the new query $q_{\sigma\cup\nu}$, which binds $?y$ to the built-in property rdf:type. Observe that if $?y$ was a distinguished variable in q_σ, a head variable in $q_{\sigma\cup\nu}$ will be bound after the rule application. Now consider rule (4.6) on some query q_σ. If q_σ contains a triple of the same form $\mathtt{s}\ ?y\ \mathtt{o}$, and the database $\mathtt{db}$ contains a triple with any $\mathbf{p}$ in the property position, the rule creates the new query $q_{\sigma\cup\nu}$ where $?y$ is bound to $\mathbf{p}$. Rules (4.7) and (4.8) instantiate query variables appearing in the property position, to values appearing in a rdfs:subPropertyOf statement of $\mathtt{db}$. The intuition is that both the subject and the object of a rdfs:subPropertyOf statements are properties, therefore they can be used to instantiate the property variable $?y$.

Rules (4.9)–(4.13) instantiate the variable $?z$ in a query triple of the form $\mathtt{s}$ rdf:type $?z$. The RDF meta-model specifies that the values of the rdf:type property are classes. Therefore, the rules bind $?z$ to $\mathtt{db}$ values that can be deemed as classes by means of entailment, i.e., those appearing in specific positions in schema-level triples. For instance, if $\mathbf{s}_1$ rdf:type $\mathbf{c} \in \mathtt{db}$, then $\mathbf{c}$ is a class and $?z$ in rule (4.9) can be instantiated to $\mathbf{c}$. Similarly, the subject and object of a rdfs:subClassOf statements are used in rules (4.10) and (4.11). Finally, rules (4.14)–(4.17) use schema triples to replace (denoted *old triple / new triple*) a triple in the input query with a new triple. Rule (4.14) exploits

rdfs:subClassOf statements: if the query q_σ asks for instances of class $\mathbf{c}_2$ and $\mathbf{c}_1$ is a subclass of $\mathbf{c}_2$, then instances of $\mathbf{c}_1$ should also be returned, and this is what the output query of this rule does. The last three rules are similar.

Example 4.13 (***Using the rules in Figure 4.3 to reformulate queries***)**.** *Consider the previously introduced database* db *(Figure 4.1) and the query q asking for all the triples of* db *(including the entailed ones).*

$$q(?x, ?y, ?z) \text{ :- } ?x\ ?y\ ?z$$

We show how some of the above rules can be used to reformulate q w.r.t. db*.*

(*i*) *Using q as input for rule (4.5) produces the query:* $q_{\{?y \to \text{rdf:type}\}}$*, i.e.,* $q(?x, \text{rdf:type}, ?z)$:- $?x$ rdf:type $?z$.

(*ii*) *Using* $q_{\{?y \to \text{rdf:type}\}}$ *as input for rule (4.11) can lead to:* $q_{\{?y \to \text{rdf:type}, ?z \to \text{:ConferencePaper}\}}$*, i.e.,* $q(?x, \text{rdf:type}, \text{:ConferencePaper})$:- $?x$ rdf:type :ConferencePaper.

(*iii*) *Finally, using* $q_{\{?y \to \text{rdf:type}, ?z \to \text{:ConferencePaper}\}}$ *as input for rule (4.14) can lead to:* $q(?x, \text{rdf:type}, \text{:ConferencePaper})$:- $?x$ rdf:type $\_\text{:b}_0$.

Query reformulation algorithm. For a query q and a database db, the output of $\texttt{Reformulate}(q, \texttt{db})$ is defined as the fixed-point $\texttt{Reformulate}^\infty(q, \texttt{db})$, where:

$$\begin{aligned}
&\texttt{Reformulate}^0(q, \texttt{db}) = \{q\}\\
&\texttt{Reformulate}^{k+1}(q, \texttt{db}) = \texttt{Reformulate}^k(q, \texttt{db}) \cup\\
&\quad \{\, q''_{\sigma''} \mid \exists i \in [4.5, \ldots, 4.17] \text{ such that applying rule } (i) \text{ on db and}\\
&\quad\ \text{some query } q'_{\sigma'} \in \texttt{Reformulate}^k(q, \texttt{db}) \text{ yields the query } q''_{\sigma''} \,\}
\end{aligned}$$

Theorem 4.14 shows that our reformulation algorithm terminates and provides an upper bound for the size of its output.

Theorem 4.14. *Given a BGP query q and a database* db*, the size (number of queries) of the output of* $\texttt{Reformulate}(q, \texttt{db})$ *is in* $O((6 * \#\texttt{db}^2)^{\#q})$*, with* #db *and* #$q$ *the sizes (number of triples) of* db *and q respectively.*

Appendix A.5 reports the proof for Theorem 4.14.

In practice, the size of a reformulated query is much smaller than the theoretical upper bound, but it may still be in the hundreds, depending on the query and the schema-level triples. The queries in the union obtained after reformulation have many common atoms, therefore important performance benefits can be achieved by evaluating each sub-expression common to several such queries, only once. In our experiments (see Chapter 5), the off-the-shelf PostgreSQL optimizer was able to recognize some such cases and handle them fairly well, but significant optimizations are still possible. Notably,

inspired by this observation, we are currently looking into ways to further optimize such reformulated queries. Chapter 9 briefly describes this ongoing work.

Clearly, improving the conjunctive query processor's capability to recognize and factorize common sub-expressions may further speed up the evaluation of reformulated queries.

Example 4.15 (***Reformulation of a query w.r.t. a database***)**.**
The reformulation of the query $q(?x, ?y)$:- $?x$ rdf:type $?y$ *w.r.t.* db *(described in Figure 4.1), asking for all resources and the classes to which they belong is shown below.*

$$\texttt{Reformulate}^0(q, \texttt{db}) \;=\; \{\; q(?x, ?y) \text{ :- } ?x \text{ rdf:type } ?y \;\}$$

$$\begin{aligned}\texttt{Reformulate}^1(q, \texttt{db}) \;=\; &\texttt{Reformulate}^0(q, \texttt{db}) \cup \\ &\{\; q(?x, \text{:ConferencePaper}) \text{ :- } ?x \text{ rdf:type :ConferencePaper} , \\ &\;\; q(?x, \text{:PosterConferencePaper}) \text{ :- } ?x \text{ rdf:type :PosterConferencePaper} , \\ &\;\; q(?x, \_\text{:b}_0) \text{ :- } ?x \text{ rdf:type } \_\text{:b}_0 , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ rdf:type :Paper} , \\ &\;\; q(?x, \text{:Conference}) \text{ :- } ?x \text{ rdf:type :Conference} \;\}\end{aligned}$$

$$\begin{aligned}\texttt{Reformulate}^2(q, \texttt{db}) \;=\; &\texttt{Reformulate}^1(q, \texttt{db}) \cup \\ &\{\; q(?x, \text{:ConferencePaper}) \text{ :- } ?x \text{ rdf:type :PosterConferencePaper} , \\ &\;\; q(?x, \text{:ConferencePaper}) \text{ :- } ?x \text{ rdf:type } \_\text{:b}_0 , \\ &\;\; q(?x, \text{:ConferencePaper}) \text{ :- } ?x \text{ :inProceedingsOf } ?z , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ rdf:type :ConferencePaper} , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ :title } ?z , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ :author } ?z , \\ &\;\; q(?x, \text{:Conference}) \text{ :- } ?z \text{ :inProceedingsOf } ?x , \\ &\;\; q(?x, \text{:Conference}) \text{ :- } ?x \text{ :name } ?z \;\}\end{aligned}$$

$$\begin{aligned}\texttt{Reformulate}^3(q, \texttt{db}) \;=\; &\texttt{Reformulate}^2(q, \texttt{db}) \cup \\ &\{\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ rdf:type :PosterConferencePaper} , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ rdf:type } \_\text{:b}_0 , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ :inProceedingsOf } ?z , \\ &\;\; q(?x, \text{:Paper}) \text{ :- } ?x \text{ :contactAuthor } ?z \;\}\end{aligned}$$

$$\texttt{Reformulate}^4(q, \texttt{db}) \;=\; \texttt{Reformulate}^3(q, \texttt{db})$$

4.3.3 Reformulation-based Query Answering

It turns out that by handing the result of reformulating a query as explained above, directly to a conjunctive query processor for evaluation, may introduce erroneous answers:

Example 4.16 (***Erroneous reformulation-based query answering***)**.**
Consider again the database db *(Figure 4.1) and the query* $q(?x, y)$:- $?x$ rdf:type y. *The queries in* $\texttt{Reformulate}(q, \texttt{db})$ *are shown in Example 4.15.*

Evaluating this union of queries, in particular

$$q(?x, \text{:ConferencePaper}) \text{ :- } ?x \text{ rdf:type } \_:b_0$$

in $\texttt{Reformulate}^2(q, \texttt{db})$, *with the assignment*

$$\mu = \{ ?x \to \text{:edbt2013}, \_:b_0 \to \text{:Conference} \},$$

leads to the answer tuple $\langle$:edbt2013, :ConferencePaper$\rangle$. *This tuple does not belong to the correct answer (presented in Example 4.11). Thus, the tuple is an erroneous answer.*

As the above example suggests, the issue is due to blank nodes. The semantics of blank nodes in BGP queries does not match the purpose for which they are brought into query reformulation by the `Reformulate` algorithm. Remember that the semantics of a blank node in a BGP query against an RDF graph or database is that of a non-distinguished variable. However, when our `Reformulate` algorithm brings a blank node in a query through a variable binding or a triple replacement, it refers precisely to *that particular blank node* in the database (as opposed to: any value which matches an existential variable). In the above example, during the reformulation of q, when we use the rule (4.14) to reformulate $q(?x, \text{:ConferencePaper})$:- $?x$ rdf:type :ConferencePaper using the schema-level triple: $\_:b_0$ rdfs:subClassOf :ConferencePaper $\in$ db into $q(?x, \text{:ConferencePaper})$:- $?x$ rdf:type $\_:b_0$ the goal is indeed to find conference paper values for $?x$ from the anonymous (blank-node) subclass $\_:b_0$ of :ConferencePaper.

Non-standard evaluation and answer set of a query against a database. To overcome the above issue, we introduce *alternate notions of evaluation and of answer set of a partially instantiated query* against a database. The difference between the standard definitions from Section 4.3.1 and the non-standard ones concerns blank nodes. Standard evaluation is based on binding $\texttt{VarBl}(q)$, all the query variables and blank nodes, to database values. In contrast, the non-standard definition only seeks bindings for the query variables; *blank nodes are left untouched*, just like URIs and literals.

Formally, given a database db whose set of values (URIs, blank nodes, literals) is $\texttt{Val}(\texttt{db})$ and a query $q_\sigma(\bar{x}_\sigma)$:- $(t_1, \ldots, t_\alpha)_\sigma$ whose set of variables (no blank nodes) is $\texttt{Var}(q_\sigma)$, the *non-standard evaluation* of q_σ against db is defined as:

$$\tilde{q}_\sigma(\texttt{db}) = \{ (\bar{x}_\sigma)_\mu \mid \mu : \texttt{Var}(q_\sigma) \to \texttt{Val}(\texttt{db}) \text{ is a total assignment such that } ((t_1, \ldots, t_\alpha)_\sigma)_\mu \subseteq \texttt{db} \}$$

The *non-standard answer set* of q_σ against db is obtained by the non-standard evaluation of q_σ against $\texttt{db}^\infty$, which using the notation in this thesis is denoted $\tilde{q}_\sigma(\texttt{db}^\infty)$.

The next property shows how standard and non-standard definitions of query evaluation and answer set are related. It follows directly from the fact that the assignments μ involved in non-standard evaluations, defined on $\texttt{Var}(q)$ only, are a subset of those allowed in standard evaluations, defined on $\texttt{VarBl}(q)$, as non-standard evaluations implicitly assign any URI, blank node, or literal to itself.

Property 4.17. *Let* db *be a database and* q_σ *a (partially instantiated) query against* db.

1. $\tilde{q}_\sigma(\texttt{db}) \subseteq q_\sigma(\texttt{db})$ *and* $\tilde{q}_\sigma(\texttt{db}^\infty) \subseteq q_\sigma(\texttt{db}^\infty)$ *hold.*

2. *If q_σ does not contain blank nodes then $\tilde{q}_\sigma(\mathtt{db}) = q_\sigma(\mathtt{db})$ and $\tilde{q}_\sigma(\mathtt{db}^\infty) = q_\sigma(\mathtt{db}^\infty)$.*

With the above notion of non-standard evaluation in place, our reformulation-based query answering technique is specified by the following theorem.

Theorem 4.18. *Given a BGP query q without blank nodes and a database* db*, the following holds:*

$$q(\mathtt{db}^\infty) = \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db}).$$

Appendix A.6 reports the proof for Theorem 4.18.

Note that Theorem 4.18 considers queries without blank nodes. This assumption is made *without loss of generality*, since blank nodes *from the original query* can be immediately replaced with non-distinguished variables in BGP queries, without impacting the answer set in any way (as explained in Section 2.2.1). We only make the assumption in order to prevent confusion between the *original blank nodes* (i.e., non-distinguished variables) and those introduced by the reformulation steps, and which required the introduction of non-standard evaluation in order to avoid erroneous answers.

Implementing (non-)standard evaluation. While our alternate definitions are *non-standard* from an RDF perspective, they are just as easy to implement using e.g. an RDBMS, as the "standard" definitions. For "standard" RDF evaluation of a conjunctive BGP query q, translate q into SQL, taking care to replace each blank node with the respective relation attribute name; for "non-standard" evaluation, translate q into SQL by enclosing the blank nodes within quotes, so that the RDBMS treats each as a constant, to be matched only by the exact same value in the database.

From a practical perspective, Theorem 4.18 states: to answer a query q against a database db, it suffices to

(i) reformulate q w.r.t. db and

(ii) evaluate each reformulated query on the *original* database db, using the *non-standard* evaluation.

In other words, query reformulation (based on db) followed by non-standard evaluation of partially instantiated queries computes the exact answer set, and does not require saturating the database. Moreover (and importantly), these steps only require standard conjunctive query evaluation capabilities from the underlying system.

Importantly, based on Section 2.3, reformulation-based query answering can be delegated to *any* RDBMS by storing db in a `Triple` table, and then by evaluating queries using relational evaluation, *without replacing blank nodes by fresh non-distinguished variables.*

Example 4.19 (***Reformulation-based query answering***)**.**
Consider again the query $q(?x, ?y)$:- $?x$ rdf:type $?y$. The answer set of q against the previous database db *(Figure 4.1) is:*

$$\bigcup_{q'_{\sigma'} \in \texttt{Reformulate}(q,\texttt{db})} \tilde{q}'_{\sigma'}(\texttt{db}) = \{ \langle :\text{doi}_1, \_:\text{b}_0 \rangle , \langle :\text{doi}_1, :\text{ConferencePaper} \rangle , \langle :\text{edbt2013}, :\text{Conference} \rangle , \langle :\text{doi}_1, :\text{Paper} \rangle , \langle \_:\text{b}_2, :\text{Conference} \rangle \}.$$

Note that this answer set coincides with the one obtained by saturation-based query answering in Example 4.11.

4.4 Summary

In this chapter we have shown the theoretical interest of extending the state of the art in query answering over RDF databases.

We extended the considered RDF fragments for which efficient query answering methods are proposed, notably by the inclusion of blank nodes (Section 4.1). We propose novel query answering algorithms on the above fragment and prove their soundness and completeness. In particular, we proposed a novel saturation algorithm which allows incremental maintenance of the database (Section 4.2.2). Also, we show how to correctly answer reformulated queries given the extended fragment semantics (Section 4.3.3).

The following chapter will present an empirical evaluation of the proposed algorithms and prove their practicality and efficiency. Moreover, we compare the two algorithms in the same experimental settings and propose means of choosing among them the one best suited to the RDF database characteristics.

Chapter 5

RDF Query Answering: A Practical Assessment

The saturation and reformulation-based query answering techniques presented in Chapter 4 are designed to be deployed on top of any RDBMS(-style) engine. This chapter describes the experiments we performed to assess the practical utility of our query answering strategies within the DB fragment of RDF.

We start by describing the platform we used for implementing and testing our algorithms. Section 5.1 presents information on data storage, indexing scheme and choice of encodings.

The following sections present an empiric study of the performance of the two query answering techniques and their trade-offs. We discuss our results on data saturation in Section 5.2, query answering over the database instance in Sections 5.3 and 5.4, instance and schema updates in Section 5.5, and compare the two techniques from the perspective of both query answering and updates in Section 5.6.

5.1 Settings

This section opens the chapter by describing the settings of our experimental study.

Software and hardware. The `Saturate` and `Reformulate` algorithms described previously were implemented in Java 6. They were tested as add-ons to the already existing **RDFViewS** (standing for *RDF View Selection*) system [Goasdoué10], which given a conjunctive SPARQL query workload and a set of RDFS statements returns a set of recommended views for materialization, in order to optimize query processing times. Our `Reformulate` algorithm, in particular, is derived from the one implemented in [Goasdoué10] for view rewritings. It uses the appropriate (non-standard) semantics described in Section 4.3.3 for query answering over the DB fragment, also handling the use of blank nodes.

The system relies on a relational database back-end for storing data. The experiments in this thesis were deployed on top of the PostgreSQL (version 8.5 using standard default settings) back-end, on an 8-core DELL server at 2.13 GHz with 16 GB of RAM, running Linux 2.6.31.14. All times we report are averaged over five executions. PostgreSQL is an efficient open source RDBMS, frequently used in the literature [Abadi07, Sidirourgos08, Weiss08, Neumann08, Neumann09, Goasdoué10]. Given the generic nature of the proposed algorithms any other platform supporting conjunctive query evaluation can be used instead.

Data storage. *Instance-level* triples are stored in a `Triple`(`s`, `p`, `o`) table, the set-based saturation in a `Sat`(`s`, `p`, `o`) table, while the multiset-based saturation (required for incrementally maintaining the saturation) is compactly stored, as explained in Section 4.2.2, in a table `SatM`(`s`, `p`, `o`, `isExplicit`, `count`).

We do this to delegate RDF query evaluation to the relational server. Indeed, relational query evaluation coincides either with the standard RDF query evaluation when the query has no blank nodes (as is the case of our queries, Theorem 4.18) or with the non-standard one when the query has blank nodes (as is the case of our reformulations, Theorem 4.18).

Indexing. Each table is indexed by all permutations of the (`s`, `p`, `o`) columns, leading to a total of 6 indexes; the `spo` index is clustering. We adopted this indexing choice (inspired by [Neumann10a]) to give PostgreSQL efficient query evaluation opportunities. *Schema-level* triples are kept in memory. *All measured times are averaged over* 10 *executions.*

Dictionary encoded triples. Previous works, for example [Neumann08], have used *dictionary encodings* when storing an RDF database. In order to avoid storing and joining string-encoded RDF attributes, a dictionary is built associating an integer to each distinct URI or blank node. Queries are *encoded* by replacing constants with the respective integers, evaluated on the integer-encoded data, and their results *decoded* at the end of execution. We stored the encoding dictionary in a separate table, which we indexed by both the encoded string value and the integer encoding. We tested our experiments with and without a dictionary; we present results *with* this dictionary encoding, due to their increased performance.

Datasets. Our evaluation is based on the well-established DBLP [DBL], DBpedia [Lehmann14] and Barton [wwwb] RDF datasets and also three datasets of diverse sizes from the LUBM benchmark [Guo05]. The main characteristics of the considered datasets are summarized in Table 5.1. In [Duan11b], a detailed analysis of such datasets was performed, providing to the interested reader a deeper knowledge of the dataset, such as number of subjects, predicates, objects, types, instances per type, etc.

5.2 Performance of the Saturation Algorithms

We denote by $\mathbf{t_{sat}}$ the time to compute the saturation of a given database by the `Saturate` algorithm (described in Section 4.2.1), and by $\mathbf{t_{sat+}}$ the time to saturate the database by the `Saturate`$_+$ algorithm (introduced in Section 4.2.2).

	Schema	*DBLP*	*DBpedia*	*Barton*	*LUBM*		
a)	**number of triples**	41	5,666	101	84		

	Schema	*DBLP*	*DBpedia*	*Barton*	*LUBM*		
a)	**number of triples** ($\times\mathbf{10^6}$)	8.4	26.9	34.9	1	9.7	93.3

	Schema	*DBLP*	*DBpedia*	*Barton*	*LUBM*		
a)	**number of triples** ($\times\mathbf{10^6}$)	11.8	29.8	38.9	1.2	11.9	114.3
b)	**size increase** (%)	41.05	10.65	14.91	22.60	22.54	22.54
c)	**computation time** $\mathbf{t_{sat}}$ (s)	748	2,742	4,294	59	558	5,211

	Schema	*DBLP*	*DBpedia*	*Barton*	*LUBM*		
a)	**number of triples** ($\times\mathbf{10^6}$)	18.6	66	73.5	2.6	25.7	245.7
b)	**size increase** (%)	121.97	227.37	116.89	163.25	163.40	163.33
c)	**computation time** $\mathbf{t_{sat+}}$ (s)	799	2,977	4,586	60	595	5,728

TABLE 5.1: Graph characteristics and saturation times.

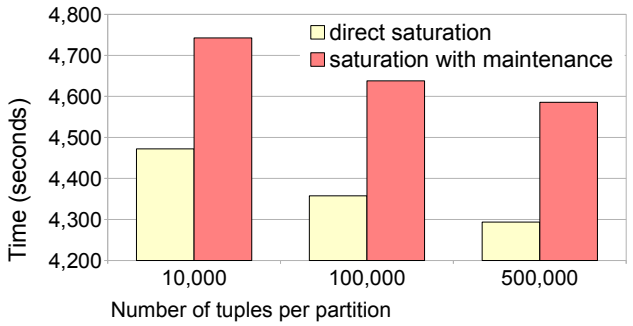
As Table 5.1 shows, saturation added between 10% and 41% to the database size. The values in row *a)* showing the number of triples for the **Multiset saturation**, are obtained by adding to the number of explicit triples every entailed triple, *as many times as it is derived*. As the next row *b)* shows, this more than doubles (even triples in the case of DBpedia) the size of each graph as entailed triples can be derived in several ways. Table 5.1 also shows the saturation times $\mathbf{t_{sat}}$ and $\mathbf{t_{sat+}}$ for each graph. As expected, `Saturate` is (slightly) faster than `Saturate`$_+$. However, if the graph is updated, one can maintain the saturation only if `Saturate`$_+$ was used, as explained in Section 4.2.2.

The size of the graphs makes it difficult for a Java program to process the whole data in memory in one pass. Therefore, we use a partition-based saturation method, which reads the graphs in partitions of a specified number of triples at a time. The positive logic nature of RDFS makes implementing incremental updates close to trivial [Gutierrez06]. Our partition-based approach can be seen as successive incremental updates made on the saturation table.

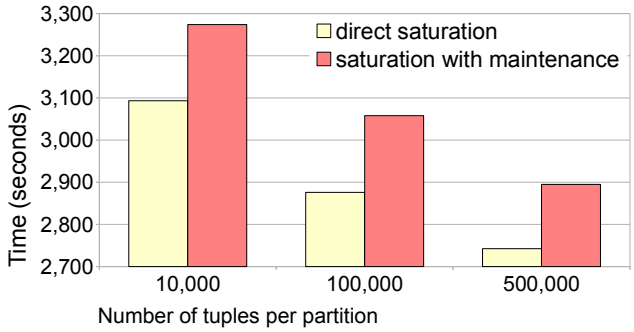
Figure 5.1 shows the time to saturate the DBLP, DBpedia and Barton graphs using different partition sizes. Since the difference between these times is less than 2% of the total time, in the following experiments we consider the saturation time obtained when reading the data in partitions of 500,000 triples (values already shown in Table 5.1).

Saturation process scalability. We now study the scalability of our saturation algorithms, with and without the provisions needed for incremental maintenance of the saturation. Figure 5.2 shows the running times of our `Saturate` and `Saturate`$_+$ algorithms for datasets of increasing sizes. The first graph shows results after saturating the $\frac{1}{4}$, $\frac{1}{2}$, and finally the whole DBLP instance-level data with the DBLP schema-level triples. The second graph reports results for the LUBM datasets, for which the data size increases by approximately a factor of 10. As can be seen in both graphs of Figure 5.2, the saturation time grows almost linearly as the data size increases, although its worst-case complexity is $O(\#\mathtt{db}^2)$ (Theorem 4.4).

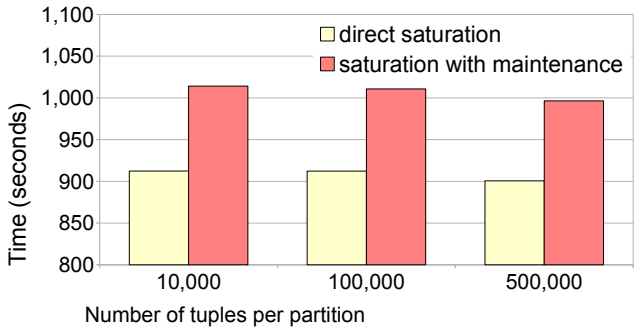
Comparison with other saturation methods. Our `Saturate` algorithm is quite straightforward and its performance is comparable to others from the literature (modulo



(a) Barton dataset



(b) DBpedia dataset

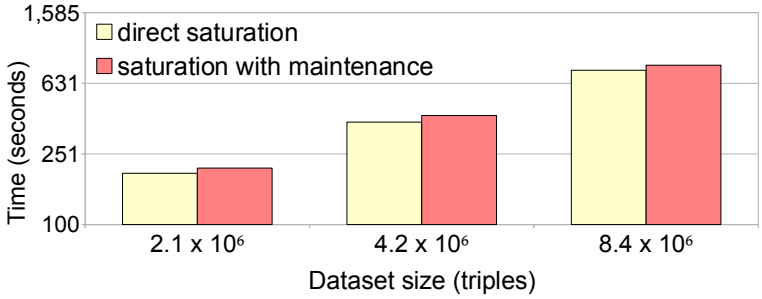


(c) DBLP dataset

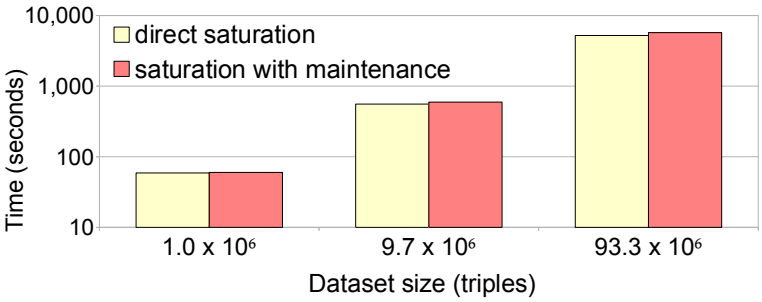
FIGURE 5.1: Saturation times using different data partitions.

the specific set of rules used). Our incremental $\texttt{Saturate}_+$ algorithm, on the other hand, is novel and outperforms existing saturation-based query answering techniques, relying on saturation maintenance. These either scale poorly [Broekstra03b] or perform more costly maintenance operations [Bishop11].

The maintenance method in [Broekstra03b], implemented in Sesame, uses a truth maintenance algorithm relying on managing the *justifications* (i.e., the proofs) for every entailed triple. Maintaining this set of justifications is problematic even for relatively small graphs (300.000 triples); maintenance after deletions is more costly than re-saturating the graph from scratch.



(a) DBLP datasets



(b) LUBM datasets

FIGURE 5.2: Saturation algorithms scalability.

The maintenance method in [Bishop11], implemented in the well-known BigOWLIM commercial system, improves over the maintenance-upon-delete method of [Broekstra03b] by computing justifications only at maintenance time, and only for the triples which may be impacted, instead of systematically computing and storing all entailed triples justifications. Still, this computes and stores much more data than our integer derivation counts, while achieving the same goal of correctly maintaining the saturation when the database changes. Furthermore, our algorithm is not tailored for a specific system and can be plugged on top of any RDBMS.

5.3 Query Answering Times

Table 5.2 shows the number of queries evaluated on each graph and information on the number of triples in each query. Most queries were hand-picked aiming at a variety of behavior when reformulated against each schema. In the case of DBpedia we considered queries of the forms typically asked in real life scenarios, as described in [Arias11]. While in the case of LUBM we also included the benchmark queries [Guo05] that were expressible as BGP queries. The queries are detailed in the Appendix B. The query answering times were similar on the two saturation tables, `Sat` and `SatM`, therefore from this point further only the `SatM` results will be discussed, since this table also allows incremental maintenance upon updates.

		Barton	*DBpedia*	*DBLP*	*LUBM*
a)	**number of queries**	17	21	26	36
b)	**minimum number of triples per query**	1	1	1	2
c)	**average number of triples per query**	2	2	6	4
d)	**maximum number of triples per query**	3	4	10	9

TABLE 5.2: Query characteristics.

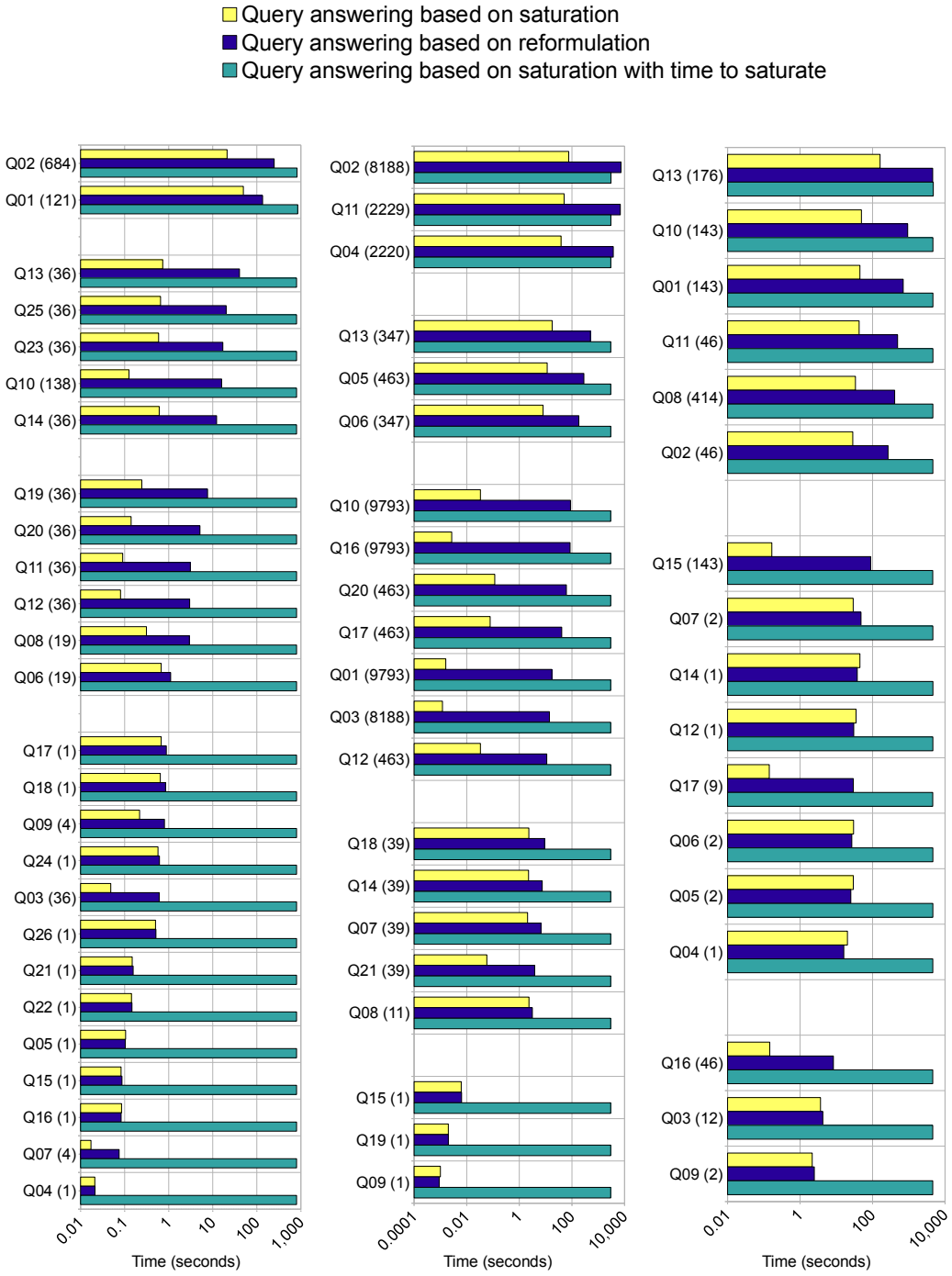
For a query q, we denote by $\mathbf{t^{sat}}(q)$ the time to answer q against the already saturated database `SatM`, and $\mathbf{t^{ref}}(q)$ the time to answer q by reformulating q and the (non-standard) evaluation of its reformulation against the `Triple` table.

The graphs in Figures 5.3 and 5.4 show, for each query:

- the number of union terms in the reformulated query (in parentheses after the query name);
- the time $\mathbf{t^{sat}}(q)$;
- the time $\mathbf{t^{ref}}(q)$;
- the sum $\mathbf{t^{sat}}(q) + \mathbf{t_{sat+}}$.

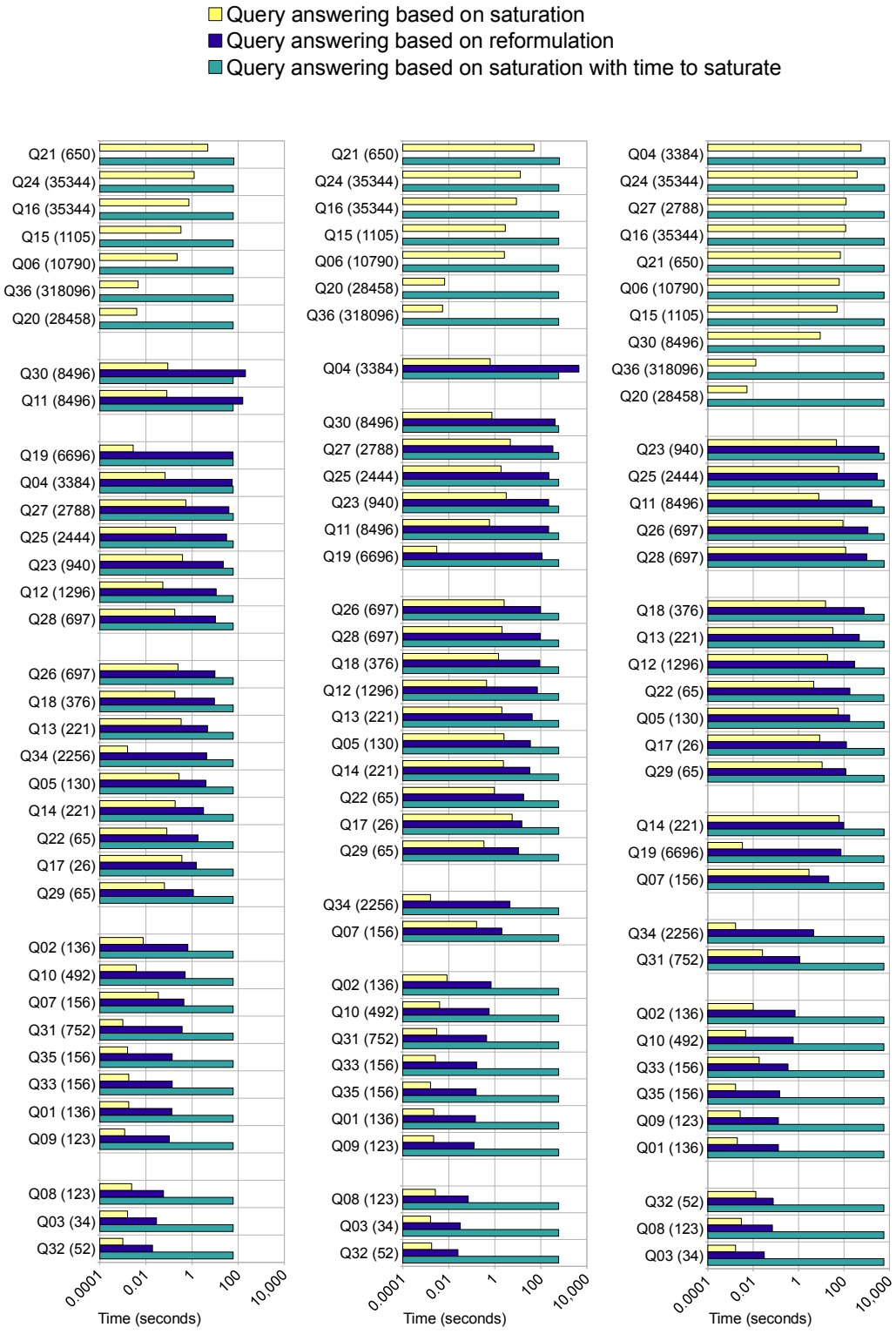
The query answering times are grouped in the decreasing order of $\mathbf{t^{ref}}$ and divided in groups by the value of $\mathbf{t^{ref}}$ compared with the thresholds $10^i, -4 \leq i \leq 4$ seconds. As expected, $\mathbf{t^{sat}}(q)$ is significantly smaller than $\mathbf{t^{ref}}(q)$ for queries with large reformulations. However, if one factors in the saturation time $\mathbf{t_{sat+}}$, the saturation-based approach becomes expensive. Obviously, saturation costs are paid only once, not for each query; we deepen this analysis below when discussing thresholds. Inspecting the results, we also found small-result queries have small $\mathbf{t^{sat}}$ and $\mathbf{t^{ref}}$, an encouraging sign that PostgreSQL's optimizer handled correctly both the original and the reformulated queries.

In the graph of Figure 5.3(a), the first group contains two extreme-case queries: Q_{01} returns the whole saturation of the database (11×10^6 tuples), while Q_{02} returns all (DBLP publication, attribute) pairs (5×10^6 tuples). The second group contains mostly general queries using upper-level classes or properties of the schema (entailed triples, thus RDF reasoning, strongly contribute to answering such queries). While such queries may be useful in certain contexts, they are not the usual queries asked by users knowing the schema. For example, if one is interested in all the articles that fit certain criteria, the query will be built restricting the answer to articles only (e.g., Q_{15}, Q_{16}), but if all publications (not just articles) are of interest, the user has no choice but to use the top concept of the ontology (e.g., Q_{11}, Q_{12}). Query selectivity also plays an important part, as can be seen by comparing the results for Q_{10} and Q_{13}. Finally, the third and fourth group contains mostly specific queries using lower-level classes or properties of the schema (thus, these queries' results are less impacted by reasoning). The results in Figures 5.3(b), 5.3(c), 5.4(a), 5.4(b) and 5.4(c) follow such patterns. Note that in the case of Figure 5.4 some of the queries could not be evaluated through reformulation by PostgreSQL due to the high number of reformulations and/or size of the intermediate



(a) DBLP dataset (b) DBpedia dataset (c) Barton dataset

FIGURE 5.3: Query answering times for the DBLP, DBpedia and Barton datasets.



(a) ≈ 1 mil. triples dataset (b) ≈ 10 mil. triples dataset (c) ≈ 100 mil. triples dataset

FIGURE 5.4: Query answering times for the LUBM datasets.

results[1]. While additional parameter tuning may enable the evaluation of such queries, the error was raised by many large-reformulation queries, signaling that their shape is problematic. We will reopen this subject while discussing ongoing works.

5.4 Comparison with Query Evaluation on Virtuoso

To emphasize the interest of query answering over standard RDBMSs we compare (in the same experimental setting) our PostgreSQL query evaluation over the saturated dataset with its Virtuoso counterpart (version 6.1.6 open source multi threaded edition). Figure 5.5 shows for each query evaluated over each dataset:

- the query evaluation time over the saturated dataset stored in PostgreSQL;
- the query evaluation time over the saturated dataset stored in Virtuoso.
- the number of query answers over the saturated dataset;

Figure 5.5 shows that for the majority of cases the PostgreSQL-based solution scales much better that the Virtuoso one. In some case, e.g., Q_{30} on the ≈ 100 million triples dataset, the query answering time is faster by up to three orders of magnitude. In the majority of remaining cases, where the Virtuoso query answering times are better, the evaluation times are comparable since they pass PostgreSQL-based evaluation by a relatively small margin only. With the exception of Q_4 on the ≈ 100 million triples dataset, we can see by analyzing each row of queries that as the data increases, the PostgreSQL solution scales better. This demonstrates the competitive performance of our implementation replying on SQL query evaluation and leveraging RDBMSs performance for handling large data.

5.5 Instance and Schema Updates

Updates have no impact on reformulation, but saturation needs to maintain the `SatM` table. To measure this overhead, we performed updates of one triple on the instance and on the schema.

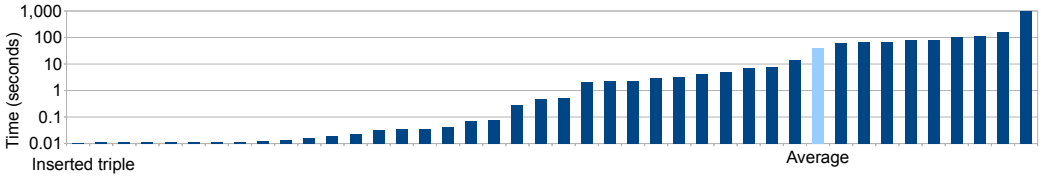
We made a random selection of instance triples from each dataset, considering all the distinct property values and the different classes. We obtained 42 triples for DBLP, 30 triples for LUBM, and hundreds of triples for Barton and DBpedia out of which we randomly selected 40 triples in each case. These triples were used in the instance deletion experiments. The same triples were modified by changing the subject or object or both with a new resource not present in the initial dataset, producing a new set of triples that was used for the instance insertions. The instance update experiments showed minor time variations between the different triples.

For the DBLP dataset, all the 41 triples in the schema were considered for schema deletions, while a set of the same triples with modified object values were used for

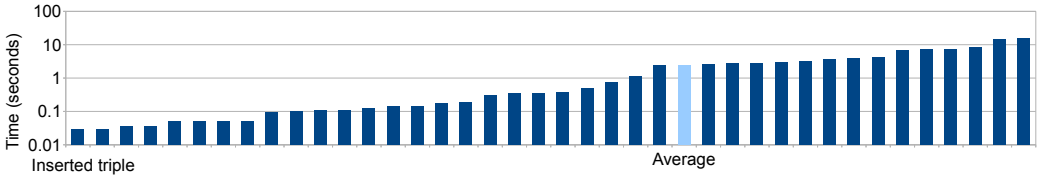
[1] Concretely, the system threw an I/O exception due to a failed attempt at materializing intermediary results.

(a) ≈ 1 mil. triples dataset (b) ≈ 10 mil. triples dataset (c) ≈ 100 mil. triples dataset

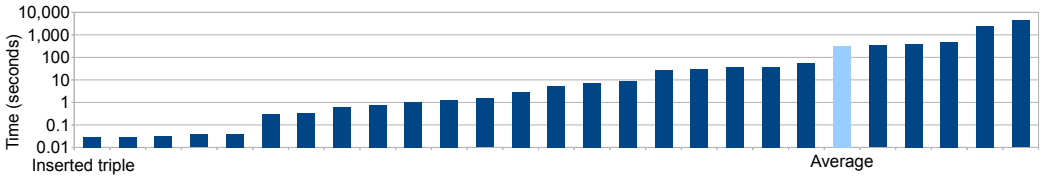
FIGURE 5.5: Saturation-based query answering through PostgreSQL and Virtuoso on the saturated LUBM datasets.



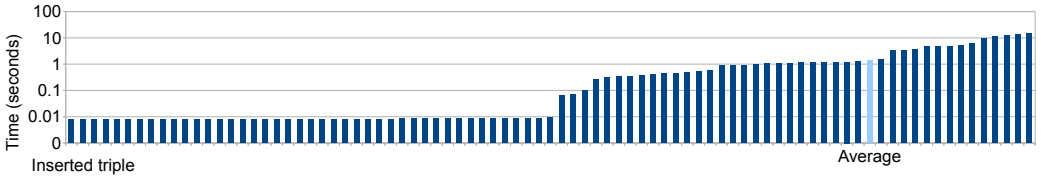
(a) DBLP dataset



(b) DBpedia dataset



(c) Barton dataset



(d) LUBM $\approx$ 1 million triples dataset

FIGURE 5.6: Schema triple insertion times.

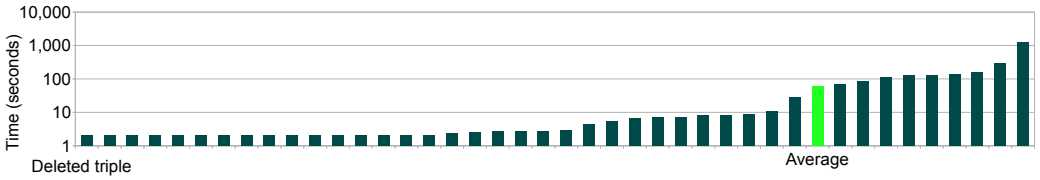
schema insertions. Similar experiments were run for all 84 schema triples from the LUBM dataset, a random selection of 26 triples from the Barton schema, and 39 triples from the DBpedia schema.

Schema updates present high variations among the different updated triples. This can be observed in Figures 5.6 and 5.7 (only one of the LUBM datasets is displayed since the others follow a similar pattern), which show:

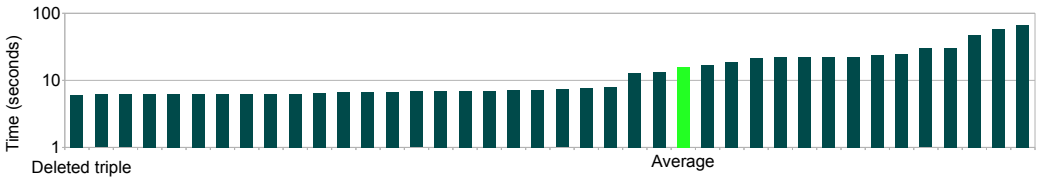
- for each of the schema triples considered for updates, the time to insert into, respectively delete from, and maintain the `SatM` table;
- the average update time for each dataset (the light colored column in each chart).

Finally, the graph in Figure 5.8 shows for each of the considered graphs:

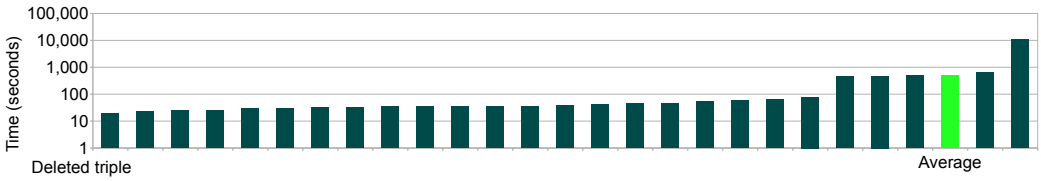
- the average time to insert into (respectively delete from) the `Triple` table;
- the average time to insert into (respectively delete from) *and maintain* the `SatM` table;



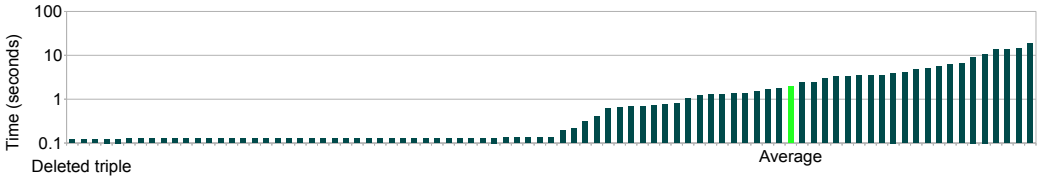
(a) DBLP dataset



(b) DBpedia dataset



(c) Barton dataset



(d) LUBM ≈ 1 million triples dataset

FIGURE 5.7: Schema triple deletion times.

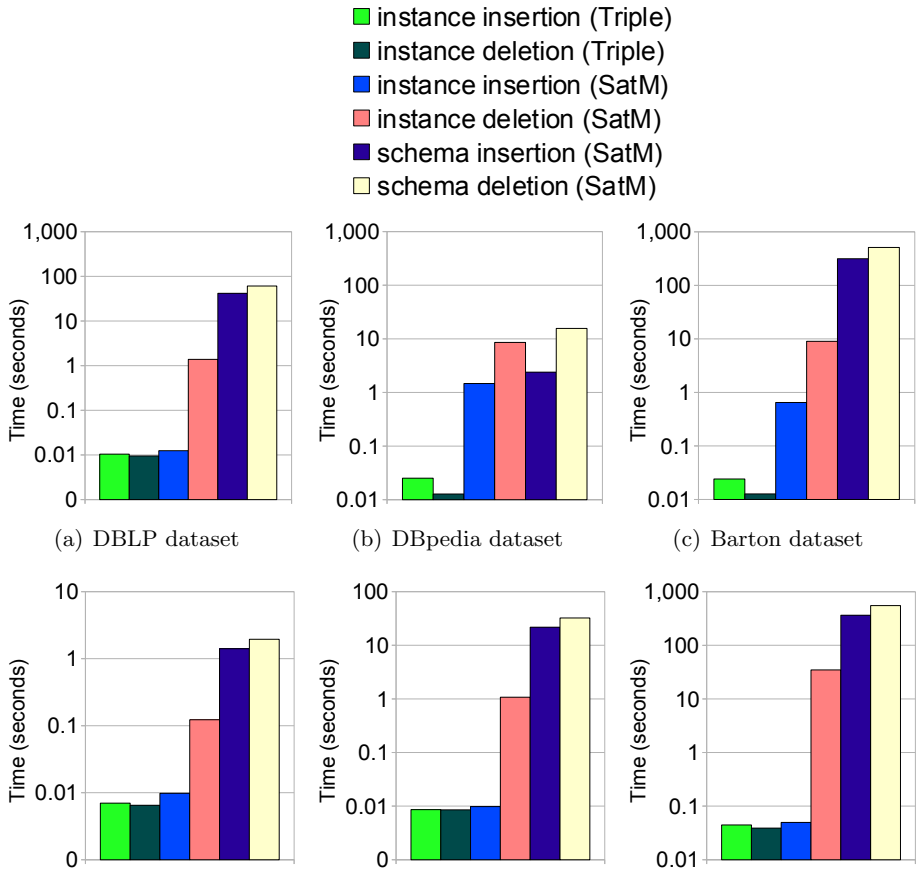
- the average time to *maintain* the `SatM` table after an insertion (respectively deletion) made on the schema.

We see that handling `SatM` is generally slower than updating `Triple`, by two orders of magnitude for instance deletions. This shows that while the algorithm $\texttt{Saturate}_+$ and the `SatM` table are required in order to avoid saturating from scratch, saturation maintenance may get costly due to the recursive nature of entailment. In particular, in the case of schema updates, maintaining the saturation may sometimes be more costly than re-saturating.

5.6 Saturation Thresholds

We now study *when saturation pays off* over multiple query runs. We call the *saturation threshold* of a query q, or $\mathbf{st}(q)$, the smallest integer n such that:

$$n \times \mathbf{t}^{\mathbf{ref}}(q) > n \times \mathbf{t}^{\mathbf{sat}}(q) + \mathbf{t}_{\mathbf{sat}_+}$$



(a) DBLP dataset (b) DBpedia dataset (c) Barton dataset

(d) LUBM ≈ 1 mil. dataset (e) LUBM ≈ 10 mil. dataset (f) LUBM ≈ 100 mil. dataset

FIGURE 5.8: Update times.

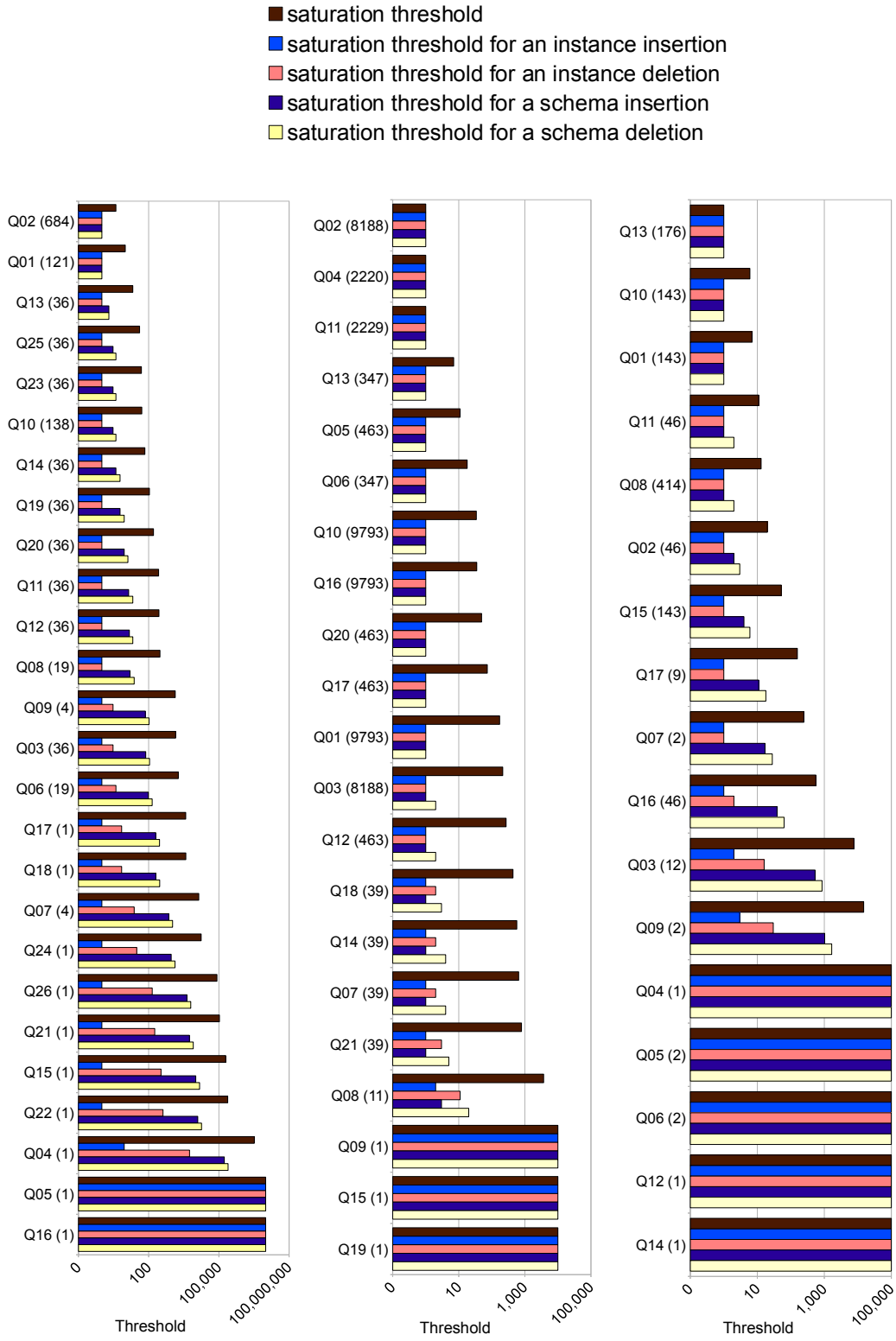
In other words, n is the minimum number of times one needs to run q in order for the whole saturation cost to amortize.

Similarly, we study how many times q should run in order for the *maintenance overhead due to one instance or schema update* to pay off. We formalize this as follows.

Concerning instance updates, let $\mathbf{t}^{+}_{\texttt{Triple}}$ be the time to insert one statement in `Triple`, and $\mathbf{t}^{+}_{\texttt{SatM}}$ be the time to propagate the insertion of one triple to the `SatM` relation. Then, the *saturation threshold for an instance insertion*, denoted $\mathbf{st}^{+}_{\mathbf{i}}(q)$, is the smallest n for which:

$$n \times \mathbf{t}^{\mathbf{ref}}(q) + \mathbf{t}^{+}_{\texttt{Triple}} > n \times \mathbf{t}^{\mathbf{sat}}(q) + \mathbf{t}^{+}_{\texttt{SatM}}$$

In other words, $\mathbf{st}^{+}_{\mathbf{i}}(q)$ is the minimum number of times one needs to run q in order for the maintenance overhead due to the insertion of one triple (recall Figure 5.8) to amortize. We similarly define the *saturation threshold for an instance deletion*, denoted $\mathbf{st}^{-}_{\mathbf{i}}(q)$.



(a) DBLP dataset (b) DBpedia dataset (c) Barton dataset

FIGURE 5.9: Saturation thresholds for the DBLP, DBpedia and Barton datasets.

Schema updates do not affect the `Triple` table, since the schema is kept in memory, but they can have a big impact on `SatM`. Similar to $\mathbf{st}(q)$, we define the *saturation threshold for a schema insertion* $\mathbf{st}_\mathbf{s}^+(q)$ and *deletion* $\mathbf{st}_\mathbf{s}^-(q)$, as the minimum number of times one needs to run q in order for the schema update cost to amortize.

Figure 5.9 shows the 5 saturation thresholds for our queries w.r.t. each considered graph. The vertical (log-scale) axis is limited to 10^7, for readability. The thresholds follow a similar trend, strongly determined by the size of the reformulated query (shown in parentheses on the x axis). The larger the reformulated query, the lower the threshold: saturation pays off faster when reformulation is expensive, and this tends to happen when the queries are syntactically complex.

Looking at $\mathbf{st}(q)$ in Figure 5.9(a), we see that it varies from 2 for Q_{02} to more than 10^5 for Q_{05} or Q_{22}; for queries such as Q_{04}, Q_{05} etc., which are left unchanged by reformulation, the saturation cost can only be compensated after $10^6 - 10^7$ runs. This shows that saturation is not always the most efficient way to go. While it is true saturation can be performed off-line, one needs to also keep in mind that saturation may require quite complex maintenance algorithms.

Comparing the thresholds among themselves, we notice that $\mathbf{st}$ is always higher than the update thresholds, which is expected since $\mathbf{st}$ runs need to offset *the complete saturation cost*, whereas $\mathbf{st}_\mathbf{i}^+$, $\mathbf{st}_\mathbf{i}^-$, $\mathbf{st}_\mathbf{s}^+$ and $\mathbf{st}_\mathbf{s}^-$ need to offset the cost of maintaining saturation for just one triple added or deleted. Finally, $\mathbf{st}_\mathbf{i}^+$ is lower than $\mathbf{st}_\mathbf{i}^-$, and $\mathbf{st}_\mathbf{s}^+$ is lower than $\mathbf{st}_\mathbf{s}^-$, meaning that saturation costs particularly penalize scenarios where deletions are frequent. Figures 5.9(b) and 5.9(c) show similar results for the Barton and DBpedia datasets, for the same reason the charts for the LUBM datasets were omitted.

5.7 Summary

Our experiments showed that `Saturate` and `Reformulate` can be used to *process BGP queries* efficiently by exploiting an off-the-shelf RDBMS. However, they perform very differently depending on the query selectivity and the impact of the schema through reasoning: saturation is best for large-reformulation queries, while reformulation is efficient for small-to-moderate reformulation.

With respect to *updates*, we showed that saturation can be maintained at a reasonable cost for instance-level updates, while schema-level updates are much more expensive. Updates, however, have a small impact on reformulation making it appropriate for high update rates. When considering also *repeated query runs*, we highlighted a number of thresholds determining when saturation pays off; these thresholds are strongly impacted by the query reformulation size and selectivity. While saturation is the default in many RDF platforms, our experiments demonstrate the practical interest of *reformulation-based BGP query answering*.

Concluding Remarks

In this first part of the thesis we presented a detailed analysis of the two main techniques for answering conjunctive queries against RDF databases, a significant fragment of RDF allowing both implicit and incomplete information. Figure 5.10 illustrates the positioning of our work w.r.t. to the related literature.

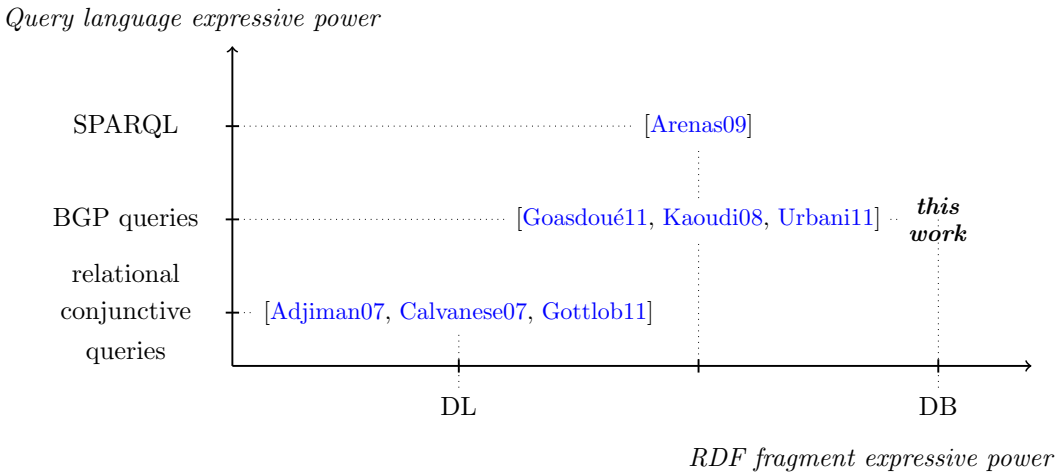


FIGURE 5.10: Outline of the positioning of our work.

We studied query answering over this DB fragment, when the data is saturated. Also, we proposed a data saturation algorithm robust to instance and schema updates, countering the main drawback of the technique. In contrast to the works presented in Section 3.1.2, the saturation maintenance technique presented in Section 4.2.2 is based on the *number of times* triples are entailed, facilitating data storage and manipulation. The subsequent work [Urbani13] also employes the use of derivation counts to facilitate saturation maintenance upon data deletions. In [Urbani13] the additional information stored for inferred triples counts the *distinct* direct entailments for that triple. While offering a higher degree of precision, it comes with the additional cost of keeping track of the entailment path. Our algorithms are tailored to work with the *total* number of different derivations, in order to avoid the computations necessary for distinguishing between different entailment paths. Our technique performs well for instance updates, and acceptably on schema updates. On average, it is worth maintaining the saturation. We note though that in some (rare) cases, when the updates affect upper-level classes or properties of the schema, saturation maintenance may be more costly than re-saturating.

We also devised a query reformulation approach for the identified fragment, and we described the requirements for obtaining correct answers when evaluating reformulated queries. The query reformulation algorithms of [Adjiman07, Goasdoué11] are restrictions of our `Reformulate`. The same holds for the algorithms in [Calvanese07, Gottlob11] when restricted to the DL fragment of RDF, whereas they are capable of handling complex DLs.

The algorithms in [Adjiman07, Calvanese07, Gottlob11] consider only our rules (4.14)–(4.17) to reformulate relational conjunctive queries, while the algorithm in [Goasdoué11] needs two additional rules for BGP queries. These two rules actually correspond to our rules (4.5)–(4.13), *under the simplifying assumption* that part of the information needed for reformulation have been pre-computed.

In [Kaoudi08, Urbani11], atomic BGP queries are reformulated using a standard backward-chaining algorithm [Russell10] on first order encodings of the entailment rules dedicated to RDFS statements.

In [Arenas09], SPARQL queries are reformulated into *nested* SPARQL, i.e., an extension of SPARQL in which properties in triples can be nested regular expressions. While such nested reformulated queries are more compact, the queries we produce are more practical, since their evaluation can be directly delegated to *any* off-the-shelf RDBMS, or to an RDF engine such as RDF-3X [Neumann08] even if it is unaware of reasoning.

Finally, we thoroughly compared the performance of the saturation and reformulation techniques and identified the factors impacting the comparison. Notably, our techniques can be directly deployed on top of any off-the-shelf RDBMS. We offered empirical proof that such approaches scale very well, even better than RDF dedicated stores.

Part II

Warehousing RDF Graphs

Chapter 6

RDF Data Warehousing Overview

In this chapter we present the main approaches proposed in the literature aiming at efficient analytics over multidimensional data, focusing on graph data and the semantic-rich RDF data model.

Data warehousing is a mature research field which has received significant attention. In Section 6.1 we discuss the main approaches to data warehousing, and then focus on the relational setting.

Works proposing warehousing techniques for RDF data have focused either on extracting tabular data from the RDF graphs or on proposing new RDF vocabularies. We describe these efforts in Section 6.2 and also examine the setting of graph data warehousing.

The sections above highlight the distinct need for data warehousing techniques taking into account the semantic rich web data. We detail this open issue in Section 6.3.

6.1 Multidimensional Relational Data Management

The multidimensional data model was first proposed in the 1990s with the aim of finding efficient techniques for analyzing large amounts of data. Databases of *facts*, each characterized by multiple *dimensions*, whose values are recorded in *measures*, are at the core of *multidimensional data warehouses* (DWs in short) [Jensen10]. The facts can then be analyzed by means of *aggregating* the measures, e.g., *"what is the average sale price of item A every month in every store?"*. One of the pioneer books on the topic is [Inmon92], which lists a set of data warehouse characteristics: the data is *integrated* (possibly through an *E*xtract-*T*ransform-*L*oad process that feeds the warehouse with well-structured data); data is *typically non volatile*, since a recorded fact or measure is unlikely to change in the future, data only gets added to the warehouse; finally, *time* is an important dimension in most DW applications.

Data warehouses are typically built to *analyze (some aspects of) an enterprise's business processes*. Thus, a first crucial task is *choosing* among the many data sources available

to the analyst, those that are interesting for a given class of business questions that the DW is designed for answering. The analysts then describe the facts, dimensions, and measures to be analyzed. Then, for each relevant business question, an *analytical query* is formulated, by (*i*) classifying facts along a set of dimensions and (*ii*) reporting the aggregated values of their measures. Such queries are commonly known as *cubes*.

The aim in data warehouses is to provide quick answers to complex multi-dimensional analytical queries. On-Line Analytical Processing (OLAP) [OLA] tools have been built for retrieving (and aggregating) large amounts of data, and also for viewing the data from multiple perspectives, e.g., using basic analytical operations to navigate through the data dimensions and hierarchies like slice, dice, roll-up, drill-down, etc.

Data warehousing and OLAP system implementations have been proposed from different perspectives. Multidimensional structures like arrays and matrixes have been considered for storing cube data. Such systems fit in the category of Multidimensional OLAP (MOLAP). The frequently adopted Relational OLAP (ROLAP) storage leverages relational database technologies (indexes, materialized views, etc.) for data analysis. Hybrid OLAP (HOLAP) approaches have also been considered, aiming at benefiting from both MOLAP and ROLAP techniques.

Relational data warehousing. For all its practical applications, data warehousing has attracted enormous interest, both from practitioners [Kimball02] and from the research community [Harinarayan96, Jarke99, Theodoratos97]; warehousing tools are now part of major relational database servers. The MD-join operator [Chatziantoniou01] was proposed for performing complex aggregations by separating the grouping computation from the aggregation. This operator's interaction with other relational operators union, selection, projections etc. allows for optimized computations. - can be used to express roll-ups. [Spyratos06] proposes a formal model for dimensional data analysis, using a functional algebra for data manipulation. Relational data warehousing is thus a pretty mature area.

Building data warehouses for unstructured data poses a new set of challenges [Inmon11]. Web data warehouses have been presented as interconnected corpora of XML documents and Web services [Abiteboul03], or as distributed knowledge bases [Abiteboul12]. In [Preda10], a large RDF knowledge base, Yago [Suchanek08], is enriched with information gathered from the Web, but do not consider RDF analytics.

6.2 RDF and Graph Data Analysis

The RDF language is increasingly being used in order to *export, share, and collaboratively author data* in many settings. For instance, it serves as a *metadata language* to describe cultural artifacts in large digital libraries, and to encode protein sequence data, as in the Uniprot dataset. RDF is a natural target for representing heterogeneous facts contributed by millions of Wikipedia users, gathered within the DBpedia data source, as well as for the *Linked (Open) Data* effort, aiming at connecting and sharing collectively produced data and knowledge.

While the current landscape of RDF data shows great potential for meaningful analysis, RDF-centric approaches have not yet been developed. In the following subsections we report on connected topics that have looked into the extraction of RDF data for the purpose of analysis (Section 6.2.1), proposing new vocabularies for publishing RDF analytical data (Section 6.2.2) and graph data warehousing (Section 6.2.3).

6.2.1 Extracting Multidimensional Data from RDF

The Journal on Data Semantics has shown its interest in Semantic Data Warehouses by proposing a special issue on this topic. The published works [Nebot09, Pinet09, Banerjee09, Niinimäki09, Skoutas09, Papastefanatos09] propose novel solutions to designing Semantic Data Warehouses using ontologies, but are mainly focused on the ETL process and how to create a relational schema for the data warehouse and to populate it. [Nebot12] also presents a semi-automated approach for deriving a RDW from an ontology.

The works mentioned above are oriented towards relational storage of RDF data, therefore preserving the data heterogeneity, and the ability to query the data semantics are not considered.

6.2.2 Vocabularies for RDF Data Analysis

[Etcheverry12, W3C14d] propose RDF(S) vocabularies (pre-defined classes and properties) for describing *relational* multidimensional data in RDF.

The vocabulary introduced by [Etcheverry12] is titled Open Cubes. It is used for the representation of the schemas and instances of OLAP cubes and allows applying operations to such representations of multidimensional RDF data. The work is motivated by decision making applications that require temporary Web data to complete the information already available in a given decision-support system. As such they allow (*i*) retrieving Web data in the Open Cubes vocabulary representation; (*ii*) applying analytical operations such as $roll-up$ and *slice* to align the data with the one already available in the decision support system; (*iii*) using the resulting data cubes for data analysis, and discarding this additional information when it is no longer needed/pertinent.

Notably, [Etcheverry12] provides an algorithm for mapping OLAP operations into SPARQL 1.1 queries. Preliminary experimental results are presented, mentioning that the Open Cubes vocabulary and proposed techniques are meant for handling specific information needs, therefore the data handled is assumed to be small. While retrieval of the web data is considered orthogonal to the work, they do sketch a procedure for exporting web cubes (a.k.a. the answer to the SPARQL queries) into a relational model-based OLAP server.

As of January 2014, the W3C proposes its own RDF Data Cube Vocabulary [W3C14d], recommended for publishing multi-dimensional data.

6.2.3 Graph Data Warehouses

Recent works [Zhao11, Bleco12] have focused on graph warehousing.

The model of [Zhao11] introduces the idea of defining independently the semantics of nodes and edges in an analytical schema ("graph cube" in their terminology). Analysis cubes and OLAP operations on cubes over graphs are also defined, considering a lattice structure for navigating between perspectives. However, their approach does not apply to *heterogeneous* graphs, and thus it cannot handle multi-valued attributes (e.g., a movie being both a comedy and a romance), nor data semantics, both central in RDF. Furthermore, the approach is focused on *counting edges*, not considering more complex aggregations.

In [Bleco12], graph data can be aggregated in a spatial fashion by grouping connected nodes into *regions* (think of a street map graph). This basic aggregation serves as a foundation for the proposed OLAP framework. The work makes a distinction between data and schema. While the schema is the graph built of all connections between nodes, data records are subgraphs of the schema with (cost) labeled edges (and occasionally nodes).

6.3 Summary

The current popularity of RDF raises interest in *models and tools for RDF data analytics*. For instance, consider applications seeking to *harvest, aggregate and analyze user data* from various sources (such as social networks, blog posts, comments on public Web sites etc.). The data is *heterogeneous*; it may include facts about the user such as age, gender or region, an endorsement of a restaurant the user liked etc. The data is *graph-structured*, since it describes relationships between users, places, companies etc. It comes from multiple sources and may have attached *semantics*, based on some ontologies for which RDF is an ideal format.

Despite the perceived need, there is currently *no satisfactory conceptual and practical solution for large-scale RDF analytics*. Relational DW tools are not easily adaptable, since loading RDF data in a relational analytical schema may lead to facts with unfilled or multiply-defined dimensions or measures; the latter does not comply with the relational multidimensional setting and DW tools. More important, to fully exploit RDF graphs, *the heterogeneity and rich semantics of RDF data should be preserved* through the warehouse processing chain and up to the analytical queries. In particular, RDF analytical queries should be allowed to *jointly query the schema and the data*, e.g., ask for most frequently specified properties of a CollegeStudent, or the three largest categories of Inhabitants. *Changes to the underlying database* (such as adding a new subclass of Inhabitant) *should not cause the warehouse schema to be re-designed; instead, the new resources (and their properties) should propagate smoothly to the analysis schema and cubes.*

In the next chapter we define a novel framework for RDF analytics, based on analytical schemas and queries that can be efficiently deployed on top of any RDF data management platform, to extend it with analytic capabilities.

Chapter 7

RDF Graph Analysis

The development of the Semantic Web (RDF) brings new requirements for data analytics tools and methods, going beyond querying to semantics-rich analytics through warehouse-style tools. The motivating scenario presented in Section 7.1 exemplifies the problems faced by developers using such data and highlights a set of emerging application needs.

In this chapter, we present a full (bottom-up) redesign of the core data analytics concepts and tools in the context of RDF data, leading to the first complete formal framework for warehouse-style RDF analytics.

Notably, we define analytical schemas tailored to heterogeneous, semantics-rich RDF graphs in Section 7.2. We introduce analytical queries which (beyond relational cubes) enables flexible querying of the data and the schema as well as powerful aggregation in Section 7.3. In Section 7.4 we discuss different query answering alternatives. Finally we introduce OLAP-style operations, allowing navigating through the data cubes in Section 7.5 and conclude.

This work has led to the publication of an article [Colazzo13b] and a demonstration [Colazzo13a] in the French database conference 29e journées Bases de Données Avancées *and a news article [Roatis14] in the magazine of the* European Community in Information Technology (ERCIM News). *The main results of this work have been published in the* Proceedings of the 23rd International World Wide Web Conference (WWW 2014) *[Colazzo14].*

7.1 Data Warehousing Scenario

In the following scenario, we identify by (i)-(v) a set of real-world application requirements, for further reference.

Alice is a software engineer working for an IT company responsible of developing **user applications based on open (RDF) data** from the region of Grenoble. From a dataset describing the region's restaurants, she must build a clickable map showing for each district of the region, "the number of restaurants and their average rating per type of cuisine".

The data is (*i*) heterogeneous, as information, such as the menu, opening hours or closing days, is available for some restaurants, but not for others. Fortunately, Alice studied data warehousing [Jarke99]. She thus designs a *relational data warehouse* (RDW, in short), writes some SPARQL queries to extract tabular data from the restaurant dataset (filled with nulls when data is missing), loads them in the RDW and builds the application using standard RDW tools.

The client is satisfied, and soon Alice is given two more datasets, on shops and museums; she is asked to (*ii*) merge them in the application already developed. Alice has a hard time: she had designed a classical *star schema* [Jensen10], centered on restaurants, which cannot accommodate shops. She builds a second RDW for shops and a third for museums.

The application goes online and soon bugs are noticed. When users search for landmarks in an area, they don't find anything, although there are multiple museums. Alice knows this happens because (*iii*) the RDW does not capture the fact that *a museum is a landmark*. With a small redesign of the RDW, Alice corrects this, but she is left with a nagging feeling that there may be many other relationships present in the RDF data which she missed in her RDW.

Further, the client wants the application to find (*iv*) the relationships between the region and famous people related to it, e.g., Stendhal was born in Grenoble. In Alice's RDWs, relationships between entities are part of the schema and statically fixed at RDW design time. In contrast, useful open datasets such as DBpedia [Lehmann14], which could be easily linked with the RDF restaurant dataset, may involve many relationships between two classes, e.g., *bornIn*, *gotMarriedIn*, *livedIn* etc.

Finally, Alice is required to support (*v*) a new type of aggregation: for each landmark, show how many restaurants are nearby. This is impossible in Alice's RDW designs of a separate star schema for each of restaurants, shops and landmarks/museums, as both restaurants and landmarks are central entities and Alice cannot use one as a measure for the other.

Alice's needs in setting up the application can be summarized as follows:

(*i*) support of *heterogeneous* data;

(*ii*) *multiple* central concepts, e.g., restaurants *and* landmarks above;

(*iii*) support for *RDF semantics* when querying the warehouse;

(*iv*) the possibility to *query the relationships between entities* (i.e., the schema);

(*v*) flexible choice of aggregation dimensions.

We address each of these requirements in the following sections.

7.2 Analytical Schemas and Instances

We model a schema for RDF graph analysis, called *analytical schema*, as a labeled directed graph.

From a classical data warehouse analytics perspective, *each node of our analytical schema represents a set of facts* that may be analyzed. Moreover, *the facts represented by an analytical schema node can be analyzed using (as either dimensions or measures) the schema nodes reachable from that node.* This makes our analytical schema model much more general than the traditional DW setting where facts (at the center of a star or snowflake schema) are analyzed according to a fixed set of dimensions and of measures.

From a Semantic Web perspective, an analytical schema node corresponds to an RDF class, while an analytical schema edge connecting two nodes corresponds to an RDF property. The instances of these classes and properties, modeling the DW contents to be further analyzed, are *intensionally* defined in the schema, following the well-know "Global As View" (GAV) approach for data integration [Halevy01].

Definition 7.1 (***Analytical Schema***)**.**
An *analytical schema* (*AnS*) is a labeled directed graph $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$ in which:

- $\mathcal{N}$ is the set of nodes;
- $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of directed edges;
- $\lambda : \mathcal{N} \cup \mathcal{E} \to U$ is an injective labeling function, mapping nodes and edges to URIs;
- $\delta : \mathcal{N} \cup \mathcal{E} \to \mathcal{Q}$ is a function assigning to each node $n \in \mathcal{N}$ a unary BGP query $\delta(n) = q(x)$, and to every edge $e \in \mathcal{E}$ a binary BGP query $\delta(e) = q(x, y)$.

Notation. We use n and e respectively (possibly with subscripts) to denote *AnS* nodes and edges. To emphasize that an edge connects two particular nodes we will place the nodes in subscript, e.g., $e_{n_1 \to n_2}$.

For simplicity, we assume that through λ, each node in the *AnS* defines a *new* class (not present in the original graph G), while each edge defines a new property[1]. Observe that using δ we define a GAV view for each node and edge in the analytical schema. Just as an analytical schema defines (and delimits) the data available to the analyst in a typical relational DW scenario, in our framework, *the classes and properties modeled by an AnS (defined using δ and labeled by λ) are the only ones visible to further RDF analytics*, that is: analytical queries will be formulated against the *AnS* and not against the base data (as Section 7.3 will show). Example 7.2 introduces the sample RDF graph used to illustrate new notions throughout this chapter, while in Example 7.3 we define an *AnS* for this graph.

Example 7.2 (***RDF graph – running example***)**.**
We consider the RDF graph G *depicted in Figure 7.1, comprising information about users and products. The graph features a resource* :user$_1$ *whose name is* :Bill *and whose age is* "28"*. Bill works with* :user$_2$ *and is a friend of* :user$_3$*. He is an active contributor to two blogs, one shared with his co-worker* :user$_2$*. Bill bought a* :SmartPhone *and rated it online etc. Moreover, the graph comes with a schema expressing semantic constraints like a* :Phone *is a* :Product*, a* :SmartPhone *is a* :Phone*, a* :Student *is a* :Person*, the domain and range of* :knows *is* :Person*, working with someone is one way of knowing her etc.*

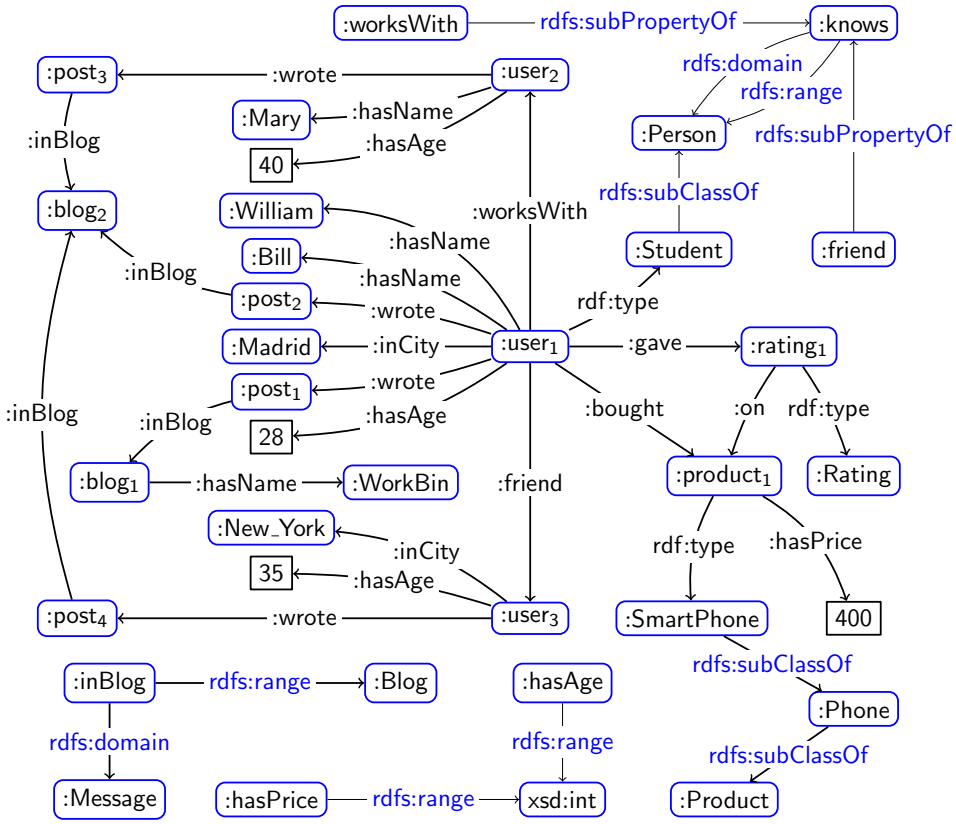


FIGURE 7.1: Running example: RDF graph.

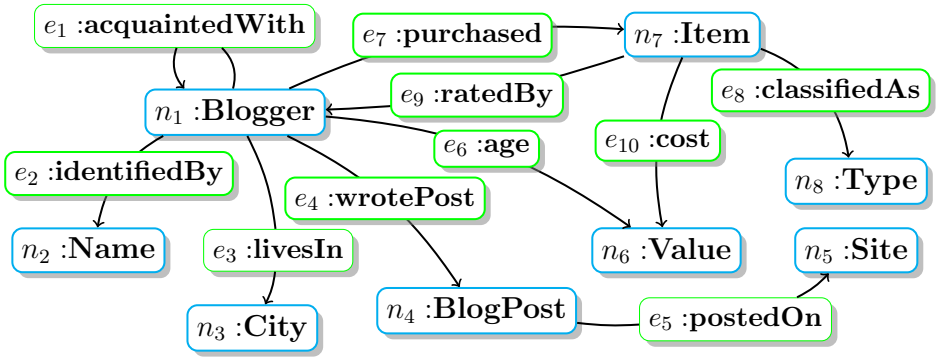


FIGURE 7.2: Sample analytical schema (*AnS*).

Example 7.3 (***Analytical Schema***)**.**
Figure 7.2 depicts an AnS for analyzing bloggers and items. The node and edge labels appear in the figure, while the BGP queries defining these nodes and edges are provided in Table 7.1. In Figure 7.2 a blogger (n_1) may have written posts (e_4) which appear on some site (e_5). A person may also have purchased items (e_7) which can be rated (e_9). The semantic of the remaining AnS nodes and edges can be easily inferred.

[1]In practice, nothing prevents λ from returning URIs of class/properties from G and/or the RDF model, e.g., rdf:type.

node	$\lambda(n)$	$\delta(n)$
n_1	**:Blogger**	$q(?x)$:- $?x$ rdf:type :Person, $?x$:wrote $?y$, $?y$:inBlog $?z$
n_2	**:Name**	$q(?x)$:- $?y$:hasName $?x$
n_3	**:City**	$q(?x)$:- $?y$:inCity $?x$
n_4	**:BlogPost**	$q(?x)$:- $?x$ rdf:type :Message, $?x$:inBlog $?z$, $?z$ rdf:type :Blog
n_5	**:Site**	$q(?x)$:- $?y$:inBlog $?x$, $?x$ rdf:type :Blog
n_6	**:Value**	$q(?x)$:- $?z$ rdfs:range xsd:int, $?y$ $?z$ $?x$
n_7	**:Item**	$q(?x)$:- $?x$ rdf:type $?y$, $?y$ rdfs:subClassOf :Product
n_8	**:Type**	$q(?x)$:- $?x$:rdfs:subClassOf :Product

edge	$\lambda(e)$	$\delta(e)$
e_1	**:acquaintedWith**	$q(?x, ?y)$:- $?z$ rdfs:subPropertyOf :knows, $?x$ $?z$ $?y$
e_2	**:identifiedBy**	$q(?x, ?y)$:- $?x$:hasName $?y$
e_3	**:livesIn**	$q(?x, ?y)$:- $?x$:inCity $?y$
e_4	**:wrotePost**	$q(?x, ?y)$:- $?x$:wrote $?y$, $?y$ rdf:type :Message
e_5	**:postedOn**	$q(?x, ?y)$:- $?x$ rdf:type :Message, $?x$:inBlog $?y$
e_6	**:age**	$q(?x, ?y)$:- $?x$ rdf:type :Person, $?x$:hasAge $?y$
e_7	**:purchased**	$q(?x, ?y)$:- $?x$:bought $?y$
e_8	**:classifiedAs**	$q(?x, ?y)$:- $?x$ rdf:type :Product, $?x$ rdf:type $?y$
e_9	**:ratedBy**	$q(?x, ?y)$:- $?y$:gave $?z$, $?z$ rdf:type :Rating, $?z$:on $?x$, $?x$ rdf:type :Product
e_{10}	**:cost**	$q(?x, ?y)$:- $?x$:hasPrice $?y$

TABLE 7.1: The labels λ and queries δ for the Figure 7.2 *AnS* nodes and edges.

The nodes and edges of an analytical schema define the perspective (or *lens*) through which to analyze an RDF dataset. This is formalized as follows:

Definition 7.4 (***Instance of an*** *AnS*)**.**
Let $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$ be an analytical schema and $\mathtt{G}$ an RDF graph. The *instance of* $\mathcal{S}$ *w.r.t.* $\mathtt{G}$ is the RDF graph $\mathcal{I}(\mathcal{S}, \mathtt{G})$ defined as:

$$\bigcup_{n \in \mathcal{N}} \{\mathtt{s}\ \text{rdf:type}\ \lambda(n) \mid \mathtt{s} \in q(\mathtt{G}^{\infty}) \wedge q = \delta(n)\}$$
$$\cup$$
$$\bigcup_{e \in \mathcal{E}} \{\mathtt{s}\ \lambda(e)\ \mathtt{o} \mid \mathtt{s}, \mathtt{o} \in q(\mathtt{G}^{\infty}) \wedge q = \delta(e)\}.$$

The above definition states that an instance of an *AnS* is a "new" RDF graph, consisting of the class and property assertions built through the BGP queries labeling the *AnS* nodes and edges. From now on, we denote the instance of an *AnS* either $\mathcal{I}(\mathcal{S}, \mathtt{G})$ or simply $\mathcal{I}$, when that does not lead to confusion.

Example 7.5 (***Analytical Schema Instance***)**.**
Below we show part of the instance of the analytical schema introduced in Example 7.3. We indicate at right of each triple the node (or edge) of the AnS which produced it.

$$\mathcal{I}(\mathcal{S}, \mathtt{G}') = \begin{array}{ll} \{\ :\mathsf{user}_1\ \mathsf{rdf{:}type}\ :\mathbf{Blogger}, & n_1 \\ \ :\mathsf{user}_1\ :\mathbf{acquaintedWith}\ :\mathsf{user}_2, & e_1 \\ \ :\mathsf{user}_1\ :\mathbf{identifiedBy}\ :\mathsf{Bill}, & e_2 \\ \ :\mathsf{post}_1\ :\mathbf{postedOn}\ :\mathsf{blog}_1, & e_5 \\ \ :\mathsf{user}_1\ :\mathbf{age}\ \text{“28”}, & e_6 \\ \ :\mathsf{product}_1\ \mathsf{rdf{:}type}\ :\mathbf{Item}, & n_7 \\ \ :\mathsf{SmartPhone}\ \mathsf{rdf{:}type}\ :\mathbf{Type}, & n_8 \\ \ :\mathsf{product}_1\ :\mathbf{cost}\ \text{“400”},\ \ldots\ \} & e_{10} \end{array}$$

Central to our notion of RDF warehouse is *the disjunctive semantics of an AnS*, materialized by the two levels of the union ($\cup$) in Definition 7.4. *Each node and each edge of an AnS populates $\mathcal{I}$ through an independent RDF query*, and the resulting triples are added to the union producing the *AnS* instance. Defining *AnS* nodes and edges independently of each other is crucial for allowing our warehouse to:

- be an actual *RDF graph* (in contrast to tabular data, possibly with many *null*s, which would result if we attempted to fit the RDF data in a relational warehouse). This addresses the requirement (*i*) from our motivating scenario (Section 7.1). It also guarantees that the *AnS* instance can be *shared, linked, and published* according to the best current Semantic Web practices;
- directly benefit from the *semantic-aware SPARQL query answering* provided by SPARQL engines. This answers our semantic-awareness requirement (*iii*), and also (*iv*) (ability to *query the schema*, notoriously absent from relational DWs);
- provide *as many entry points for analysis as there are AnS nodes*, in line with the flexible, decentralized nature of RDF graph themselves (requirement (*ii*)). As a consequence (see below), aggregation queries are very flexible, e.g., they can aggregate one entity in relation with another (count restaurants at proximity of landmarks, requirement (*v*) in Section 7.1);
- *support AnS changes easily* (requirement (*ii*)) since nodes and/or edge definitions can be freely added to (removed from) the *AnS*, with no impact on the other node/edge definitions, or their instances.

As an illustration of our point on heterogeneity ((*i*) above), consider the three users in the original graph G (Figure 7.1) and their properties: $:\mathsf{user}_1$, $:\mathsf{user}_2$ and $:\mathsf{user}_3$ are part of the :Blogger class in our *AnS* instance $\mathcal{I}$ (through n_1's query), although $:\mathsf{user}_2$ and $:\mathsf{user}_3$ lack a name. However, those user properties present in the original graph, are reflected by the *AnS* edges e_3, e_4 etc. Thus, RDF heterogeneity is accepted in the base data and present in the *AnS* instance.

Defining analytical schemas. As customary in data analysis/warehouse, analysts are in charge of defining the schema, with significant flexibility in our framework for doing so. Typically, schema definition starts with the choice of a few concepts of interest, to be turned into *AnS* nodes. These can come from the application, or be “suggested” based on the RDF data itself, e.g., *the most popular types in the dataset* (RDF classes together with the number of resources belonging to the class), which can be obtained

with a simple SPARQL query. Core concepts and edges may also be identified through RDF summarization as in e.g., [Campinas12]. Further, SPARQL queries can be asked to identify *the most frequent relationships to which the resources of an AnS node participate*, or *chains of relationships connecting instances of two AnS nodes* etc. In this incremental fashion, the *AnS* can be "grown" from a few nodes to a graph capturing all information of interest; throughout the process, SPARQL queries can be leveraged to assist and guide *AnS* design. Chapter 9 briefly describes this ongoing work.

Once the queries defining *AnS* nodes are known, the analyst may want to check that an edge is *actually connected to a node adjacent to the edge*, in the sense: some resources in the node extent also participate to the relationship defined by edge. Let $n_1, n_2 \in \mathcal{N}$ be *AnS* nodes and $e_{n_1 \to n_2} \in \mathcal{E}$ an edge between them. This condition can be easily checked through a SPARQL query ensuring that:

$$ans(\delta(n_1)) \cap \Pi_{\text{domain}}(ans(\delta(e_{n_1 \to n_2}))) \neq \emptyset$$

Such criteria are a useful means for testing the quality of any analytical schema, proposed by a data analyst or automatic schema creation methods.

Extensions. An *AnS* uses unary and binary BGP queries (introduced in Section 2.2.1) to define its instance, as the union of all *AnS* node/class and edge/property instances. This can be extended in a straightforward fashion to unary and binary (full) SPARQL queries (allowing disjunction, filter, regular expressions, etc.) in the setting of RDF analytics, and even to unary and binary queries from (a mix of) query languages (SQL, SPARQL, XQuery, etc.), in order to analyze data integrated from distributed heterogeneous sources.

7.3 Analytical Queries

Data warehouse analysis summarizes facts according to relevant criteria into so-called *cubes*. Formally, a cube (or analytical query) analyzes facts characterized by some *dimensions*, using a *measure*. We consider a set of dimensions $d_1, d_2, \ldots, d_n$, such that each dimension d_i may range over the value set $\{d_i^1, \ldots, d_i^{m_i}\}$; the Cartesian product of all dimensions $d_1 \times \cdots \times d_n$ defines a multidimensional space $\mathcal{M}$. To each tuple t in this multidimensional space $\mathcal{M}$ corresponds a *subset* $\mathcal{F}_t$ of the analyzed facts, having for each dimension $d_{i,1 \leq i \leq n}$, the value of t along d_i.

A *measure* is a set of values[2] characterizing each analyzed fact f. The facts in $\mathcal{F}_t$ are summarized by the *cube cell* $\mathcal{M}[t]$ by the result of an *aggregation* function $\oplus$ (e.g., count, sum, average, etc.) applied to the union of the measures of the $\mathcal{F}_t$ facts: $\mathcal{M}[t] = \oplus(\bigcup_{f \in \mathcal{F}_t} v_f)$, where v_f is a measure associated to each fact f.

An *analytical query* consists of two (rooted) queries and an aggregation function. The first query, known as a *classifier* in traditional data warehouse settings, defines the *dimensions* $d_1, d_2, \ldots, d_n$ according to which the facts matching the query root will be analyzed. The second query defines the *measure* according to which these facts will be

[2]It is a set rather than a single value, due to the structural heterogeneity of the *AnS* instance, which is an RDF graph itself: each fact may have zero, one, or more values for a given measure.

summarized. Finally, the aggregation function is used for *summarizing* the analyzed facts.

To formalize the connection between an analytical query and the *AnS* on which it is asked, we introduce a useful notion:

Definition 7.6 (***BGP Query to* AnS *Homomorphism***).
Let q be a BGP query whose labeled directed graph is $G_q = \langle \mathcal{N}, \mathcal{E}, \lambda \rangle$, and let $\mathcal{S} = \langle \mathcal{N}', \mathcal{E}', \lambda', \delta' \rangle$ be an *AnS*. An *homomorphism from q to $\mathcal{S}$* is a graph homomorphism $h : G_q \to \mathcal{S}$, such that:

- for every $n \in \mathcal{N}$, $\lambda(n) = \lambda'(h(n))$, or $\lambda(n)$ is a variable;
- for every $e_{n \to n'} \in \mathcal{E}$: (i) $e_{h(n) \to h(n')} \in \mathcal{E}'$ and (ii) $\lambda(e_{n \to n'}) = \lambda'(e_{h(n) \to h(n')})$, or $\lambda(e_{n \to n'})$ is a variable;
- for every $e_1, e_2 \in \mathcal{E}$, if $\lambda(e_1) = \lambda(e_2)$ is a variable, then $h(e_1) = h(e_2)$;
- for $n \in \mathcal{N}$ and $e \in \mathcal{E}$, $\lambda(n) \neq \lambda(e)$.

The above homomorphism is defined as a correspondence from the query to the *AnS* graph structure, which preserves labels when they are not variables (first two items), and maps all the occurrences of a given variable *labeling different query edges* to the same label value (third item). Observe that a similar condition referring to occurrences of a same variable *labeling different query nodes* is not needed, since by definition, all occurrences of a variable in a query are mapped to the same node in the query's graph representation. The last item (independent of h) follows from the fact that the labeling function of an *AnS* is injective. Thus, a query with a same label for a node and an edge cannot have an homomorphism with an *AnS*.

Next we introduce our analytical queries. In keeping with the spirit (but not the restrictions!) of classical RDWs [Jarke99, Jensen10], a *classifier* defines the level of data aggregation while a *measure* allows obtaining values to be aggregated using *aggregation functions*.

Definition 7.7 (***Analytical Query***).
Given an analytical schema $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$, an *analytical query* (*AnQ*) rooted in the node $r \in \mathcal{N}$ is a triple:

$$Q = \langle c(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus \rangle$$

where:

- $c(?x, ?d_1, \ldots, ?d_n)$ is a query rooted in the node r_c of its graph G_c, with $\lambda(r_c) = ?x$. This query is called the *classifier* of $?x$ w.r.t. the n dimensions $?d_1, \ldots, ?d_n$.
- $m(?x, ?v)$ is a query rooted in the node r_m of its graph G_m, with $\lambda(r_m) = ?x$. This query is called the *measure* of $?x$.
- $\oplus$ is a *function* computing a value (a literal) from an input set of values. This function is called the *aggregator* for the measure of $?x$ w.r.t. its classifier.

- For every homomorphism h_c from the classifier to $\mathcal{S}$ and every homomorphism h_m from the measure to $\mathcal{S}$, $h_c(r_c) = h_m(r_m) = r$ holds.

The last item above guarantees the "well-formedness" of the analytical query, that is: the facts for which we aggregate the measure are indeed those classified along the desired dimensions. From a practical viewpoint, this condition can be easily and naturally guaranteed by giving explicitly in the classifier and the measure either the type of the facts to analyze, using $?x$ rdf:type $\lambda(r)$, or a property describing those facts, using $?x$ $\lambda(e_{r\to n})$ o with $e_{r\to n} \in \mathcal{E}$. As a result, since the labels are unique in an AnS (its labeling function is injective), every homomorphism from the classifier (respectively the measure) to the AnS does map the query's root node labeled with $?x$ to the AnS's node r.

Notice that the above formalism allows the use of a classifier query $c(?x)$ with zero dimensions. Such a query can result from the application of typical cube operation discussed in Section 7.5. As a corner case, using a classifier without dimensions in an analytical query permits the analysis of unclassified sets of facts.

Example 7.8 (***Analytical Query***)**.**
The query below asks for the number of sites where each blogger posts, classified by the blogger's age and city:

$$\langle c(?x, ?y_1, ?y_2), m(?x, ?z), count\rangle$$

where the classifier and measure queries are defined by:

$$c(?x, ?y_1, ?y_2) \text{ :- } ?x \text{ :age } ?y_1, ?x \text{ :livesIn } ?y_2$$

$$m(?x, ?z) \text{ :- } ?x \text{ :wrotePost } ?y, ?y \text{ :postedOn } ?z$$

The semantics of an analytical query is:

Definition 7.9 (***Answer Set of an AnQ***)**.**
Let $\mathcal{S}$ be an analytical schema, whose instance $\mathcal{I}$ is defined w.r.t. an RDF graph G. And let $Q = \langle c(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus\rangle$ be an analytical query against $\mathcal{S}$. The *answer set* of Q against $\mathcal{I}$, denoted $ans(Q, \mathcal{I})$, is:

$$ans(Q, \mathcal{I}) = \{\langle d_1^j, \ldots, d_n^j, \oplus(q^j(\mathcal{I}))\rangle \mid \langle x^j, d_1^j, \ldots, d_n^j\rangle \in c(\mathcal{I}) \text{ and } q^j \text{ is defined as } q^j(?v) \text{ :- } m(x^j, ?v)\}$$

assuming that each value returned by $q^j(\mathcal{I})$ is of (or can be converted by the SPARQL rules [W3C13] to) the input type of the aggregator $\oplus$. Otherwise, the answer set is undefined.

In other words, the analytical query returns each tuple of dimension values found in the answer of the classifier query, together with the aggregated result of the measure query. The answer set of an AnQ can thus be represented as a cube of n dimensions, holding in each cube cell the corresponding aggregate measure. In the following, we focus on analytical queries whose answer sets are not undefined.

Example 7.10 (***Analytical Query Answer***).
Consider the query in Example 7.8, over the AnS in Figure 7.2. Some triples from the instance of this analytical schema were shown in Example 7.5. The classifier query' answer set is:

$$\{ \langle :\text{user}_1, \text{“28”}, :\text{Madrid} \rangle, \langle :\text{user}_3, \text{“35”}, :\text{New\_York} \rangle \}$$

while that of the measure query is:

$$\{\langle :\text{user}_1, :\text{blog}_1 \rangle, \langle :\text{user}_1, :\text{blog}_2 \rangle, \langle :\text{user}_2, :\text{blog}_2 \rangle, \langle :\text{user}_3, :\text{blog}_2 \rangle\}$$

Aggregating the blogs among the classification dimensions leads to the AnQ answer:

$$\{ \langle \text{“28”}, :\text{Madrid}, \mathbf{2} \rangle, \langle \text{“35”}, :\text{New\_York}, \mathbf{1} \rangle \}$$

For the sake of simplicity, we assume that an analytical query has only one measure. However, this can be easily relaxed, by introducing a set of measure queries with an associated set of aggregation functions.

7.4 Analytical Query Answering

The view-based definition of the warehouse *AnS* leaves open two concrete implementations of the warehouse. First, one can materialize (populate) the node and edge views, in the spirit of an ETL process; analytical queries are then evaluated over the materialized warehouse (as was done in Example 7.10). Alternatively, one could omit materialization and rewrite analytical queries using the views at runtime, following a mediator approach. We consider next such practical strategies for *AnQ* answering.

The *AnS* materialization approach. The simplest method consists of materializing the instance of the *AnS* (Definition 7.4) and storing it within an RDF data management system (or RDF-DMS, for short); recall that the *AnS* instance is an RDF graph itself defined using GAV views. Then, to answer an *AnQ*, one can use the RDF-DMS to process the classifier and measure queries, and the final aggregation. While effective, this solution has the drawback of storing the whole *AnS* instance; moreover, this instance may need maintenance when the analyzed RDF graph changes.

The *AnQ* reformulation approach. To avoid materializing and maintaining the *AnS* instance, we consider an alternative solution. The idea is to rewrite the *AnQ* using the GAV views of the *AnS* definition, so that evaluating the reformulated query returns exactly the same answer as if materialization was used. Using query rewriting, one can store the original RDF graph into an RDF-DMS, and use this RDF-DMS to answer the reformulated query.

Our reformulation technique below translates standard query rewriting using GAV views [Halevy01] to our RDF analytical setting.

Definition 7.11 (AnS *Reformulation of a Query*).
Given an analytical schema $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$, a BGP query $q(\bar{x})$:- $t_1, \ldots, t_m$ whose graph is $G_q = \langle \mathcal{N}', \mathcal{E}', \lambda' \rangle$, and the non-empty set $\mathcal{H}$ of all the homomorphisms from q to $\mathcal{S}$, the *reformulation of q w.r.t. $\mathcal{S}$* is the union of join queries:

$$q^{\bowtie}_{\mathcal{S}} = \bigcup_{h \in \mathcal{H}} q^{\bowtie}_h(\bar{x}) \ \text{:-} \ \bigwedge_{i=1}^{m} q_i(\bar{x}_i)$$

such that:

- for each triple $t_i \in q$ of the form s rdf:type $\lambda'(n_i)$, $q_i(\bar{x}_i)$ in $q^{\bowtie}_h$ is defined as $q_i = \delta(h(n_i))$ and $\bar{x}_i =$ s;
- for each triple $t_i \in q$ of the form s $\lambda'(e_i)$ o, $q_i(\bar{x}_i)$ in $q^{\bowtie}_h$ is defined as $q_i = \delta(h(e_i))$ and $\bar{x}_i =$ s, o.

This definition states that for a BGP query stated against an *AnS*, the reformulated query amounts to translating all its possible interpretations w.r.t. the *AnS* (modeled by all the homomorphisms from the query to the *AnS*) into a union of join queries modeling them. The important point is that *these join queries are defined onto the RDF graph* over which the *AnS* is wrapped. Also recall that we assume an implicit renaming of the non-distinguished variables was applied to the join query, prior to its creation (Section 2.2.2).

Example 7.12 (*AnS Reformulation of a Query*).
Let $q(?x, ?y_1)$ be a BGP query over the AnS in Figure 7.2.

$$q(?x, ?y_1) \text{ :- } ?x \text{ rdf:type :Blogger, } ?x \text{ :acquaintedWith } ?y_1$$

The first atom $?x$ rdf:type :Blogger in q is of the form s rdf:type $\lambda(n_1)$*, for the node n_1. Consequently, $q^{\bowtie}_{\mathcal{S}}$ contains as a conjunct the query:*

$$q(?x) \text{ :- } ?x \text{ rdf:type :Person, } ?x \text{ :wrote } ?y, \ ?y \text{ :inBlog } ?z$$

obtained from $\delta(n_1)$ in Table 7.1.

The second atom in q, $?x$:acquaintedWith $?y$ is of the form s $\lambda(e_1)$ o *for the edge e_1 in Figure 7.2, while the query defining e_1 is: $q(?x, ?y)$:- $?z$ rdfs:subPropertyOf :knows, $?x$ $?z$ $?y$. As a result, $q^{\bowtie}_{\mathcal{S}}$ contains the conjunct:*

$$q(?x, ?y_1) \text{ :- } ?z_1 \text{ rdfs:subPropertyOf :knows, } ?x \ ?z_1 \ ?y_1$$

Thus, the reformulated query amounts to:

$$q^{\bowtie}_{\mathcal{S}}(?x, ?y_1) \text{ :- } ?x \text{ rdf:type :Person, } ?x \text{ :wrote } ?y, \ ?y \text{ :inBlog } ?z,$$
$$?z_1 \text{ rdfs:subPropertyOf :knows, } ?x \ ?z_1 \ ?y_1$$

which can be evaluated directly on the graph G *in Figure 7.1.*

Theorem 7.13 states how BGP query reformulation w.r.t. an *AnS* can be used to answer analytical queries correctly.

Theorem 7.13 (***AnQ Reformulation-based Answering***)**.**
Let $\mathcal{S}$ be an analytical schema, whose instance $\mathcal{I}$ is defined w.r.t. an RDF graph G*. Let $Q = \langle c(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus\rangle$ be an analytical query against $\mathcal{S}$, $c^{\bowtie}_{\mathcal{S}}$ be the reformulation of Q's classifier query against $\mathcal{S}$, and $m^{\bowtie}_{\mathcal{S}}$ be the reformulation of Q's measure query against $\mathcal{S}$. We have:*

$$ans(Q, \mathcal{I}) = \{\ \langle d^j_1, \ldots, d^j_n, \oplus(q^j(\mathtt{G}^\infty))\rangle \mid \langle x^j, d^j_1, \ldots, d^j_n\rangle \in c^{\bowtie}_{\mathcal{S}}(\mathtt{G}^\infty) \text{ and } q^j \text{ is defined as } q^j(?v) \text{ :- } m^{\bowtie}_{\mathcal{S}}(x^j, ?v)\ \}$$

assuming that each value returned by $q^j(\mathtt{G}^\infty)$ is of (or can be converted by the SPARQL rules [W3C13] to) the input type of the aggregator $\oplus$.
Otherwise, the answer set is undefined.

The theorem states that in order to answer Q on $\mathcal{I}$, one first reformulates Q's classifier into $c^{\bowtie}_{\mathcal{S}}$ and answers it *directly against* G (not against $\mathcal{I}$ as in Definition 7.9): this is how reformulation avoids materializing $\mathcal{I}$. Then, for each tuple $\langle x^j, d^j_1, \ldots, d^j_n\rangle$ returned by the classifier, the following steps are applied: instantiate the reformulated measure query $m^{\bowtie}_{\mathcal{S}}$ with the fact x^j, leading to the query q^j; answer the latter against G; finally, aggregate its results through $\oplus$.

Appendix A.7 reports the proof for Theorem 7.13.

The trade-offs between materialization and reformulation have been thoroughly analyzed in the literature [Jarke99]; we leave the choice to the RDF warehouse administrator.

7.5 On-Line Analytical Processing on RDF Graphs

On-Line Analytical Processing (OLAP) [OLA] technologies enhance the abilities of data warehouses (so far, mostly relational) to answer multi-dimensional analytical queries.

The analytical model we introduced is specifically designed for graph-structured, heterogeneous RDF data. In this section, we demonstrate that our model is able to express RDF-specific counterparts of all the traditional OLAP concepts and tools known from the relational DW setting.

Typical OLAP operations allow transforming a cube into another. In our framework, a cube corresponds to an *AnQ*; for instance, the query in Example 7.8 models a bi-dimensional cube on the warehouse related to our sample *AnS* in Figure 7.2. Thus, we model traditional OLAP operations on cubes as *AnQ* rewritings, or more specifically, rewritings of *extended AnQ*s which we introduce below.

Definition 7.14 (***Extended AnQ***)**.**
As in Definition 7.7, let $\mathcal{S}$ be an *AnS*, and $?d_1, \ldots, ?d_n$ be a set of dimensions, each ranging over a non-empty finite set $V_{i,1\leq i\leq n}$. Let Σ be a total function over $\{?d_1, \ldots, ?d_n\}$

associating to each $?d_i$, either $\{?d_i\}$ or a non-empty subset of V_i. An *extended analytical query* Q is defined by a triple:

$$Q :\text{-} \langle c_\Sigma(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus \rangle$$

where (as in Definition 7.7) c is a classifier and m a measure query over $\mathcal{S}$, $\oplus$ is an aggregation operator, and moreover:

$$c_\Sigma(?x, ?d_1, \ldots, ?d_n) = \bigcup_{(\chi_1,\ldots,\chi_n)\in\Sigma(?d_1)\times\ldots\times\Sigma(?d_n)} c(?x, \chi_1, \ldots, \chi_n)$$

In the above, the extended classifier $c_\Sigma(?x, ?d_1, \ldots, ?d_n)$ is the set of all possible classifiers obtained by *substituting each dimension variable* $?d_i$ *with a value in* $\Sigma(?d_i)$. The function Σ is introduced to constrain some classifier dimensions, i.e., it plays the role of a filter-clause restricting the classifier result. The *semantics of an extended analytical query* is easily derived from the semantics of a standard *AnQ* (Definition 7.9) by replacing the tuples from $c(\mathcal{I})$ with tuples from $c_\Sigma(\mathcal{I})$. In other words, an extended analytical query can be seen as a union of a set of standard *AnQ*s, one for each combination of values in $\Sigma(?d_1), \ldots, \Sigma(?d_n)$. Conversely, an analytical query corresponds to an extended analytical query where Σ only contains pairs of the form $(?d_i, \{?d_i\})$.

We can now define the classical *slice* and *dice* OLAP operations in our framework:

Slice. Given an extended query $Q = \langle c_\Sigma(?x, ?d_1, \ldots, ?d_n),\ m(?x, v),\ \oplus \rangle$, a slice operation over a dimension $?d_i$ with value :val returns the extended query $\langle c_{\Sigma'}(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus \rangle$, where $\Sigma' = (\Sigma \setminus \{\ (?d_i, \Sigma(?d_i))\ \}) \cup \{\ (?d_i, \{:\mathsf{val}\})\ \}$.

The intuition is that slicing restricts an aggregation dimension to one of its domain values.

Example 7.15 (***Slice***)**.**
Let Q be the extended query corresponding to the query-cube defined in Example 7.8, that is: $\langle c_\Sigma(?x, ?y_1, ?y_2), m(?x, ?z), count \rangle$, $\Sigma = \{\ (?y_1, \{?y_1\}),\ (?y_2, \{?y_2\})\ \}$ (the classifier and measure are as in Example 7.8). A slice operation on the age dimension $?y_1$ with value "35" results in replacing the extended classifier of Q with $c_{\Sigma'}(?x, ?y_1, ?y_2) = \{\ c(?x, \text{“35”}, ?y_2)\ \}$ where $\Sigma' = \Sigma \setminus \{\ (?y_1, \{?y_1\})\ \} \cup \{\ (?y_1, \{\text{“35”}\})\ \}$.

Dice. Similarly, a dice operation on Q over dimensions $\{?d_{i_1}, \ldots, ?d_{i_k}\}$ and corresponding sets of values $\{\ S_{i_1}, \ldots, S_{i_k}\ \}$, returns the query $\langle c_{\Sigma'}(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus \rangle$, where $\Sigma' = (\Sigma \setminus \bigcup_{j=i_1}^{i_k} \{\ (?d_j, \Sigma(?d_j))\ \}) \cup \bigcup_{j=i_1}^{i_k} \{\ (?d_j, S_j)\ \}$.

Intuitively, dicing restricts a set of aggregation dimensions to subsets of values of their respective domains.

Example 7.16 (***Dice***)**.**
Consider again the initial cube Q from Example 7.8 and a dice operation on both age and location dimensions with values { “28”} *for* $?y_1$ *and* { :Madrid, :Kyoto } *for* $?y_2$. *The dice operation replaces the extended classifier of Q with*

$$c_{\Sigma'}(?x, ?y_1, ?y_2) = \{\ c(?x, \text{“28”}, \text{:Madrid}),\ c(?x, \text{“28”}, \text{:Kyoto})\ \}$$

where $\Sigma' = \Sigma \backslash \{\ (?y_1, \{?y_1\}),\ (?y_2, \{?y_2\})\ \} \cup \{\ (?y_1, \{\text{“28”}\}),\ (?y_2, \{\text{:Madrid},\ \text{:Kyoto}\})\ \}$.

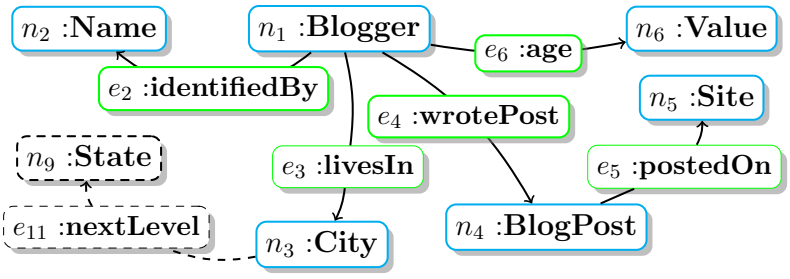
Drill-in and drill-out. These operations consist of adding to, respectively removing from, the classifier a dimension. Rewritings for drill operations can be easily formalized. We directly exemplify below a drill-in.

Example 7.17 (***Drill-in***)**.**
Consider the cube Q from Example 7.8, and a drill-in on the age dimension. The drill-in rewriting produces the query $Q = \langle c'_{\Sigma'}(?x, ?y_2), m(?x, ?z), count\rangle$ *with* $\Sigma' = \{\ (?y_2, \{?y_2\})\ \}$ *and* $c'(?x, ?y_2) = ?x$:livesIn $?y_2$.

Notice that a second drill-in applied to the obtained query in Example 7.17 leads to a classifier query $c(?x)$ with no dimensions. This is equivalent to the use of ALL in typical OLAP drill-in operations.

Dimension hierarchies. Typical relational warehousing scenarios feature hierarchical dimensions, e.g., a value of the *country* dimension corresponds to several *region*s, each of which contains many *cities* etc. Such hierarchies were not considered in our framework thus far[3].

To capture hierarchical dimensions, we introduce dedicated built-in *properties* to model the **nextLevel** relationship among parent-child dimensions in a hierarchy. For illustration, consider the addition of a new :State node and a new **nextLevel** edge to the *AnS* in Figure 7.2. Below, only part of that *AnS* is shown, highlighting the new nodes and edges with dashed lines:



In a similar fashion one could use the :nextLevel property to support ***hierarchies among edges***. For instance, relationships such as *isFriendsWith* and *isCoworkerOf* can be rolled up into a more general relationship *knows* etc.

Based on dimension hierarchies, *roll-up/drill-down* operations correspond to *adding to/removing from* the classifier, triple atoms navigating such :nextLevel edges.

[3]Dimension hierarchies should not be confused with the hierarchies built using the predefined RDF(S) properties, such as rdfs:subClassOf, e.g., in Figure 7.1.

Example 7.18 (*Roll-up*).
Recall the query in Example 7.8. A roll-up along the :City *dimension to the* :State *level yields* $\langle c'_{\Sigma'}(?x, ?y_1, ?y_3), m(?x, ?z), count\rangle$*, where:*

$$c'_{\Sigma'}(?x, ?y_1, ?y_3) \text{ :- } x \text{ :age } ?y_1, x \text{ :livesIn } ?y_2, ?y_2 \text{ :nextLevel } ?y_3.$$

The measure component remains the same, and Σ' *in the rolled-up query consists of the obvious pairs of the form* $(?d, \{?d\})$*. Note the change in both the head and body of the classifier, due to the roll-up.*

7.6 Summary

In this chapter we presented a *full redesign, from the bottom up, of the core data analytics concepts and tools*, leading to a *complete formal framework for warehouse-style analytics on RDF data*; in particular, this framework is especially suited to heterogeneous, semantic-rich corpora of Linked (Open) Data. We highlighted the requirements of modern-day applications using RDF data through a motivating scenario (Section 7.1). We addressed these requirements as follows:

- We devised a *full-RDF* warehousing approach, where the base data *and* the warehouse extent are RDF graphs. This answers to the needs (i), (iii) and (iv) stated in the motivating scenario (Section 7.1).
- We introduced *RDF Analytical Schemas (AnS)*, which are graphs of classes and properties themselves (Section 7.2), having nodes (classes) connected by edges (properties) with *no single central concept (node)*. This contrasts with the typical RDW star or snowflake schemas, and caters to requirement (ii) in the motivating scenario. The core idea behind many-node analytical schemas is to define *each node (respectively edge) by means of an independent query* over the base data.
- We define *Analytical Queries (AnQ)* over our decentralized analytical schemas (Section 7.3). Such queries are highly flexible in the choice of measures and classifiers (requirement (v)), while supporting all the classical analytical cubes and operations, i.e., slice, dice etc. (Section 7.3).

We fully implemented our approach in an operational prototype. We empirically demonstrate its interest and performance in the next chapter.

Chapter 8

The WaRG RDF Analytics Platform

We implemented the concepts and algorithms presented in the previous chapter within our WaRG tool [Colazzo13a]. In this chapter, we present experiments carried on this tool, demonstrating the practical interest and effectiveness of the RDF analytics framework proposed in Chapter 7.

Section 8.1 outlines our implementation and experimental settings.

We describe experiments on $\mathcal{I}$ materialization in Section 8.2, and on AnQ evaluation in Section 8.3. Next, we study the performance of query reformulation in Section 8.4 and OLAP operations in Section 8.5, then we conclude.

8.1 Implementation and Settings

We implemented our RDF analytics approach within the WaRG (for *Wa*rehousing *R*DF *G*raphs) tool, demonstrated in [Colazzo13a]. WaRG is built on top of kdb+ v3.0 (64 bits) [kdb], an in-memory column DBMS used in decision-support analytics. kdb+ provides arrays (tables), which can be manipulated through the q interpreted programming language. We store in kdb+ the RDF graph G, the *AnS* definitions, as well as the *AnS* instance, when we choose to materialize it. We translate BGP queries into q programs that kdb+ interprets; any engine capable of storing RDF and processing conjunctive RDF queries could be easily used instead.

Data organization. Figure 8.1 illustrates our data layout in kdb+. The URIs within the RDF dataset are encoded using integers; the mapping is preserved in a q dictionary data structure, named **dict**. The RDF graph saturation $\mathtt{G}^{\infty}$ (Section 2.2.1), is stored in the **db** table. Analytical schema *definitions* are stored as follows. The **asch** table stores the analytical schema triples: $\lambda(n)\ \lambda(e_{n\rightarrow n'})\ \lambda(n')$. The separate **query_dict** dictionary maps the labels λ for nodes and edges to their corresponding queries δ. Finally, we use the **dw** table to store the *AnS* instance $\mathcal{I}$, *or* several tables of the form **nX** and **eY** if a partitioned-table storage is used (see Section 8.2). While **query_dict** and **db** suffice to

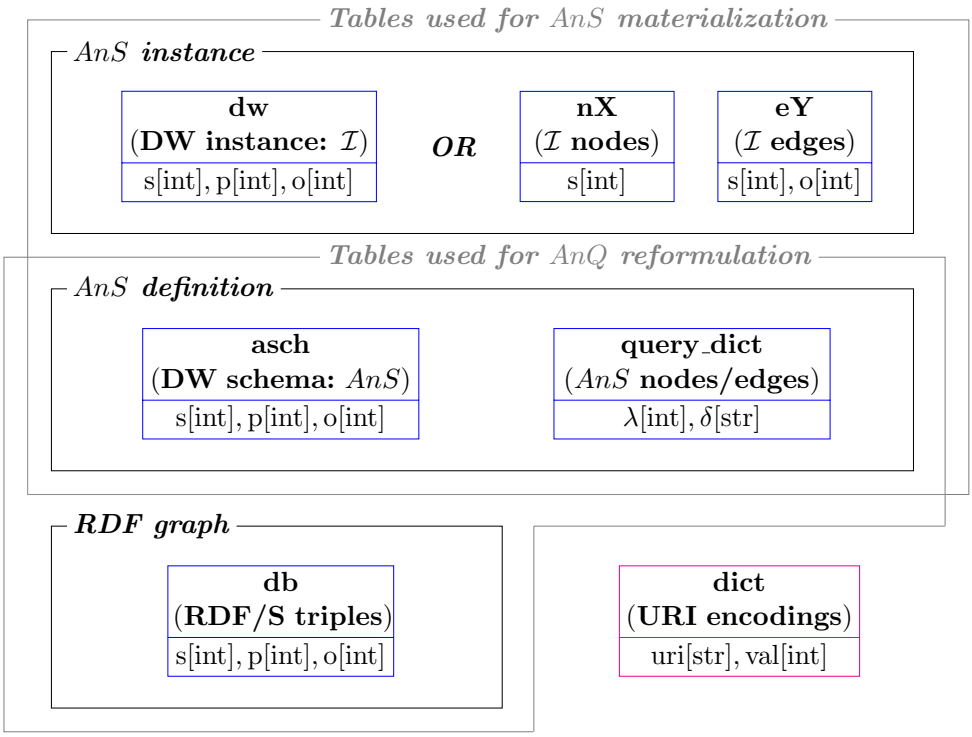


FIGURE 8.1: Data layout of the RDF warehouse.

G size	schema size	dictionary	G^∞ size
3.4×10^7 triples, 4.4 GB	5.5×10^3 triples, 746 KB	7×10^6 entries	3.8×10^7 triples

TABLE 8.1: Dataset characteristics.

create the instance, we store the analytical schema definition in **asch** to enable checking incoming analytical queries for correctness w.r.t. the *AnS*.

kdb+ stores each table column independently, and does not have a database-style query optimizer. It is quite fast since it is an in-memory system; at the same time, it relies on the q programmer's skills for obtaining an efficient execution. We try to avoid low-performance formulations of our queries in q, but further optimization is possible and more elaborate techniques (e.g., cost-based join reordering etc.) would further improve performance.

Dataset. Our experiments used the *Ontology* and *Ontology Infobox* datasets from the DBpedia Download 3.8; the data characteristics are summarized in Table 8.1. For our *scalability experiments* (Section 8.2), we replicated these datasets to study scalability in the database size.

Hardware. The experiments ran on an 8-core DELL server at 2.13 GHz with 16 GB of RAM, running Linux 2.6.31.14. All times we report are averaged over five executions.

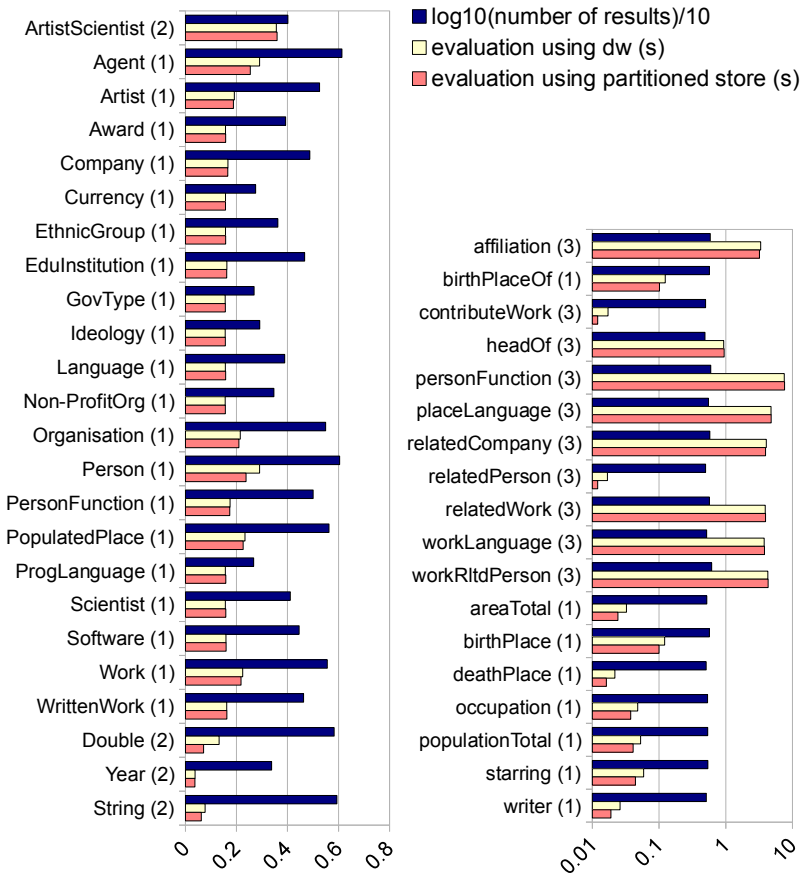


FIGURE 8.2: Evaluation time (s) and number of results for *AnS* node queries (left) and edge queries (right).

8.2 Analytical Schema Materialization

Loading the (unsaturated) $\mathtt{G}$ took about 3 minutes, and computing its full saturation $\mathtt{G}^{\infty}$ 22 minutes. We designed an *AnS* of 26 nodes and 75 edges, capturing a set of concepts and relationship of interest. *AnS* node queries have one or two atoms, while edge queries consist of one to three atoms.

We considered two ways of materializing the instance $\mathcal{I}$. First, we used a single table (**dw** in Figure 8.1). Second, inspired from RDF stores such as [Husain11], we tested a *partitioned data layout* for $\mathcal{I}$ as follows. For each distinct node (modeling triples of the form $\mathtt{s}$ $\mathtt{rdf{:}type}$ λ_X), we store a table with the subjects $\mathtt{s}$ declared of that type; this leads to a set of tables denoted **nX** (for **n**ode), with $X \in [1, 26]$. Similarly, for each distinct edge ($\mathtt{s}$ λ_Y $\mathtt{o}$) a separate table stores the corresponding triple subjects and objects, leading to the tables **eY** with $Y \in [1, 75]$.

Figure 8.2 shows for each node and edge query (labeled on the y axis by λ, chosen based on the name of a "central" class or property in the query):

(i) the number of query atoms (in parenthesis next to the label),

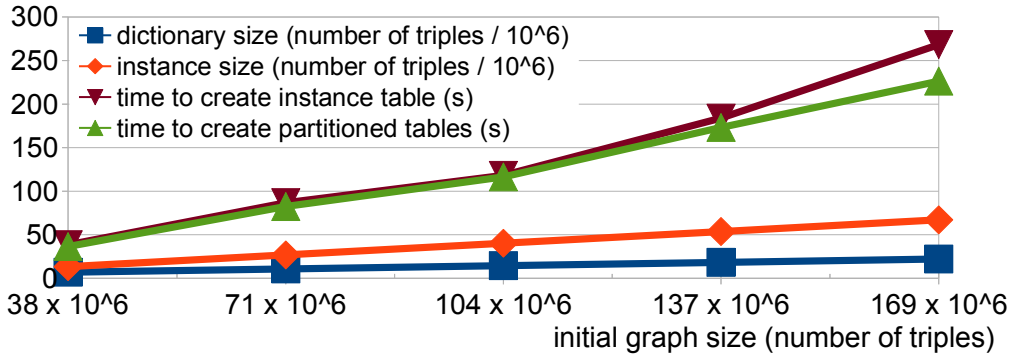


FIGURE 8.3: $\mathcal{I}$ materialization time vs. $\mathcal{I}$ size.

(*ii*) the number of query results (we show $log_{10}(\#res)/10$ to improve readability),

(*iii*) the evaluation time when inserting into a single **dw** table,

(*iv*) the time when inserting into the partitioned store.

For 2 node queries and 57 edge queries, the evaluation time is too small to be visible (below 0.01 s), and we omitted them from the plots. The *total time to materialize* the instance $\mathcal{I}$ (1.3×10^7 triples) was 38 seconds.

Scalability. We created larger RDF graphs such that the size of $\mathcal{I}$ would be multiplied by a factor of 2 to 5, with respect to the $\mathcal{I}$ obtained from the original graph `G`. The corresponding $\mathcal{I}$ materialization time are shown in Figure 8.3, demonstrating linear scale-up w.r.t. the data size.

8.3 Analytical Query Answering over $\mathcal{I}$

We consider a set of *AnQ*s, each adhering to a specific *query pattern*. A pattern is a combination of: (*i*) the number of atoms in the classifier query (denoted **c**), (*ii*) the number of dimension variables in the classifier query (denoted **v**), and (*iii*) the number of atoms in the measure query (denoted **m**). For instance, the pattern **c5v4m3** designates queries whose classifiers have 5 atoms, aggregate over 4 dimensions, and whose measure queries have 3 atoms. We used **12 distinct patterns** for a total of **1,097 queries**.

The graph at the top of Figure 8.4 shows for each query pattern, the number of queries in the set (in parenthesis after the pattern name), and the average, minimum and maximum number of query results. The largest result set (for c4v3m3) is 514, 240, while the second highest (for c1v1m3) is 160, 240. The graph at the bottom of Figure 8.4 presents the average, minimum and maximum query evaluation times among the queries of each pattern.

Figure 8.4 shows that query result size (up to hundreds of thousands) is the most strongly correlated with query evaluation time. Other parameters impacting the evaluation time are the number of atoms in the classifier and measure queries, and the number

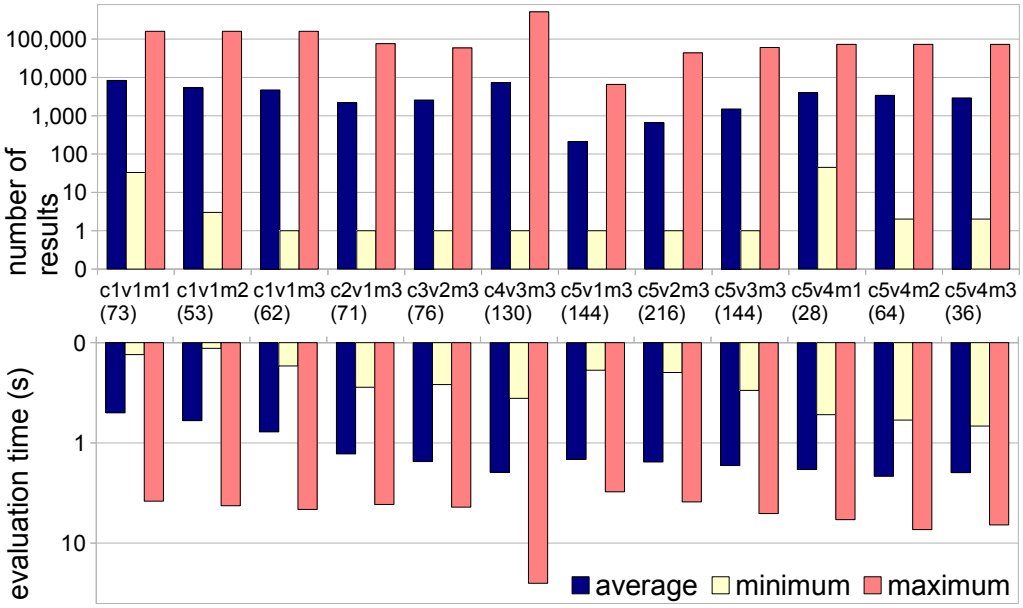


FIGURE 8.4: *AnQ* statistics for query patterns.

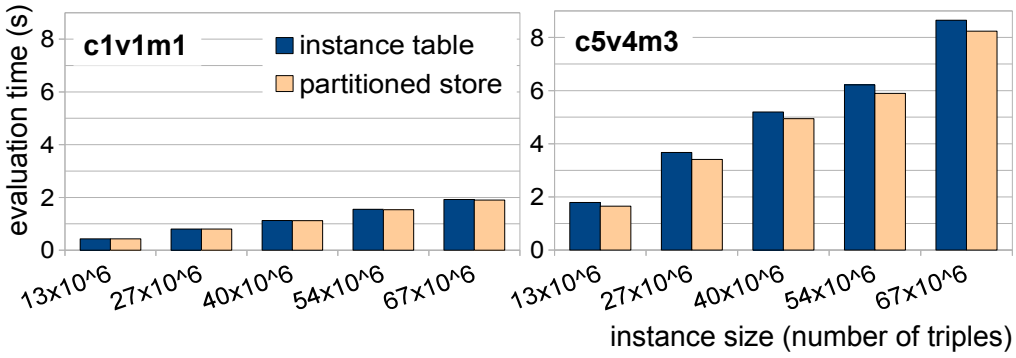


FIGURE 8.5: *AnQ* evaluation time over large datasets.

of aggregation variables. These parameters are to be expected in an in-memory execution engine such as kdb+. Observe the moderate time increase with the main query size metric (the number of atoms); this demonstrates robust performance even for complex *AnQ*s.

Figure 8.5 shows the average evaluation time for queries belonging to the sets c1v1m1 and c5v4m3 over increasing tables, using the instance triple table and the partitioned store implementations. In both cases the evaluation time increases linearly with the size of the dataset. The graph shows that the partitioned store brings a modest speed-up (about 10%); for small queries, the difference is unnoticeable. Thus, without loss of generality, in the sequel we consider only the single-table **dw** option.

Finally, we tested the impact of the $\mathcal{I}$ data layout on the query evaluation time. We compared the times obtained when $\mathcal{I}$ triples are stored in a single **dw** relation, with the times when a partitioned store is used. We found that the partitioned store brings

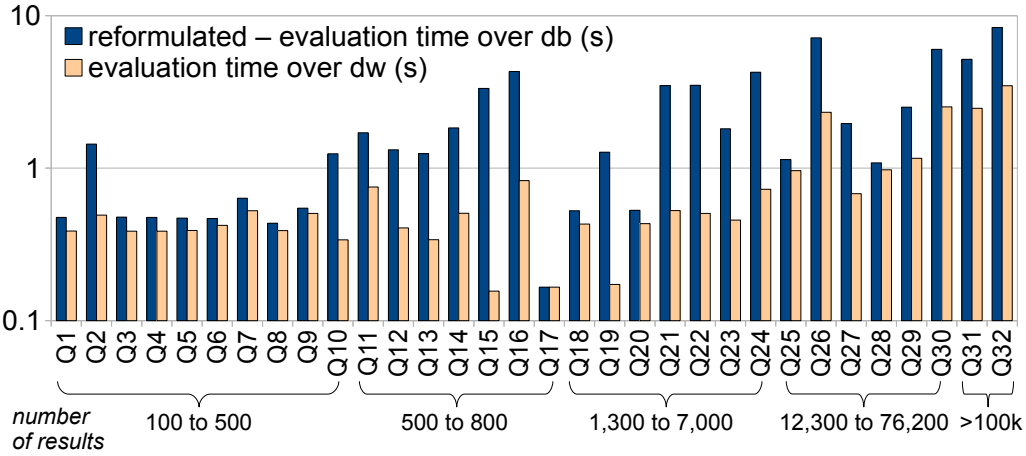


FIGURE 8.6: *AnQ* reformulation.

a modest speed-up (about 10%); for small queries, the difference is unnoticeable. This small difference is due to the efficient data access of kdb+ even when $\mathcal{I}$ is not partitioned and thus access is not selective. Thus, without loss of generality, in the sequel we consider only the single-table **dw** option.

8.4 Query Reformulation

We now study the performance of *AnQ* evaluation through reformulation (Section 7.4), through a set of 32 queries matching the pattern c1v1m1.

Figure 8.6 shows for each query, the number of answers (under the chart), the evaluation time over **db** when reformulated and the evaluation time over $\mathcal{I}$. As expected, reformulation-based evaluation is slower, because reformulated queries have to re-do some of the *AnS* materialization work. It turns out that the queries for which the difference is largest (such as Q_{15}, Q_{16} or Q_{19}) are those whose reformulation against the *AnS* definition have the largest numbers of atoms, one or more of which are of the form $x\ y\ z$. Evaluating complex joins including those of this form (matching all **dw**) is expensive, compared to evaluating them on the materialized $\mathcal{I}$. However, the extra-time incurred by query reformulation can be seen as the price to pay to avoid *AnS*'s instance maintenance time upon base data updates.

8.5 OLAP Operations

We now study the performance of OLAP operations on analytical queries (Section 7.5).

Slice and dice. In Figure 8.7, we consider three c5v4m3 queries: Q_1 having a small result size (455), Q_2 with a medium result size (1,251) and Q_3 with a large result size (73,242). For each query we perform a slice (dice) by restricting the number of answers for each of its 4 dimension variables, leading to the OLAP queries Q_{1s1} to Q_{1s4}, Q_{1d1} to

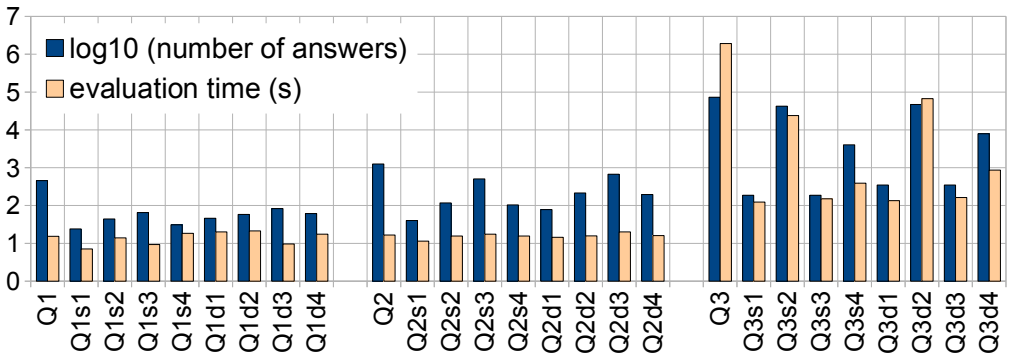


FIGURE 8.7: Slice and dice over *AnQ*s.

Q_{1d4} and similarly for Q_2 and Q_3. The figure shows that the slice/dice running time is strongly correlated with the result size, and is overall small (under 2 seconds in many cases, 4 seconds for Q_3 slice and dice queries having 10^4 results).

Drill-in and drill-out. The queries following the patterns c5v1m3, c5v2m3, c5v3m3 and c5v4m3 were chosen starting from the ones for c5v4m3 and eliminating one dimension variable from the classifier (without any other change) to obtain c5v3m3; removing one further dimension variable yielded the c5v2m3 queries etc. Recalling the definitions of drill-in and drill-out (Section 7.5), it follows that the queries in c5v**n**m3 are drill-ins of c5v**(n+1)**m3 for 1≤n≤3, and conversely, c5v**(n+1)**m3 result from drill-out on c5v**n**m3. Their evaluation times appear in Figure 8.4.

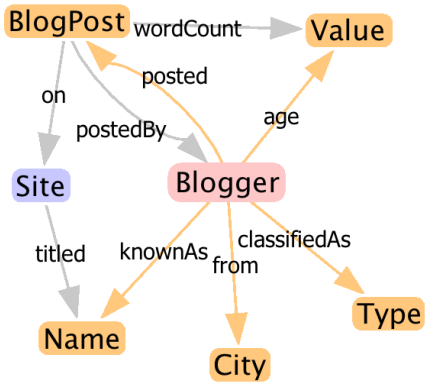
8.6 The WaRG Tool

As mentioned before, WaRG currently runs as an application on top of `kdb+` [kdb] v3.0. `kdb+` stores the RDF graph, *AnS* and $\mathcal{I}$, while BGP/*AnQ* queries are translated to the query language supported by `kdb+`, namely `q`. The WaRG system is provided with a graphic user interface (GUI), implemented in Java 1.6 and based on the Prefuse [Heer05] toolkit for visualizing and interacting with data. A video teaser and screenshots can be found at `https://team.inria.fr/oak/warg`. The WaRG tool allows doing the following:

(1) The user can select an RDF graph to analyze. The graph triples (or subsets, for large graphs) are displayed by the GUI giving a first glimpse at the data.

(2) The user may choose an *AnS* from a set of such schemas, created beforehand for the respective graph, or design one from scratch with the help of our *AnS* editor GUI. Once the *AnS* is selected, WaRG materializes its instance $\mathcal{I}$ over which the user can pose analytical queries. Figure 8.8(a) shows a sample analytical schema viewed within our GUI.

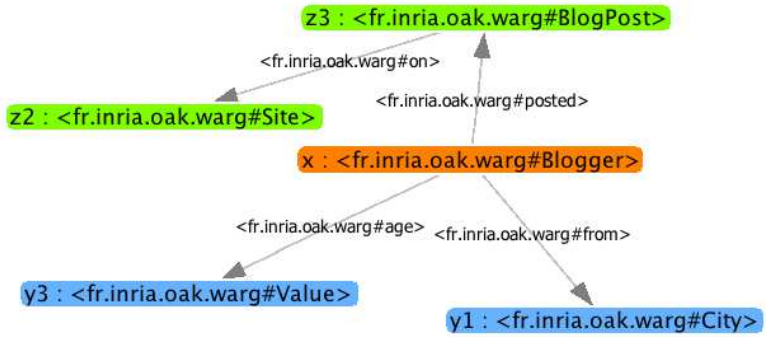
(2') We are currently adding to WaRG a schema recommendation feature (part of our ongoing work), e.g., proposing as *AnS* nodes the classes most frequent in the input graph etc.This extension can be seen as an alternative to step **(2)**.



(a) *AnS* view.

(b) Node information.

(c) Edge information.



(d) *AnQ* view.

FIGURE 8.8: WaRG visualization of a sample *AnS* and *AnQ*.

(3) On the chosen *AnS*, the user may select through the GUI an *AnS* node (class) or *AnS* edge (property), see the query defining it, and visualize its extent, i.e., the RDF resources whose type is the selected *AnS* class, respectively the resource pairs connected by the *AnS* property. This retrieves from kdb+ the corresponding query answer; a bounded subset of this answer is displayed by the GUI (Figures 8.8(b) and 8.8(c)).

(4) The user may choose a previously defined *AnQ* over the current *AnS*, or edit a new query. The GUI's point-and-click interface allows incrementally designing the classifier and measure BGP queries in an *AnQ*. The user must first select an *AnS* node, designating the *set of facts* to be analyzed. This leads to all *eligible edges* (outgoing from the selected node and having non-empty extents) to become highlighted. The user can continue forming the query by selecting one such edge, which leads to highlighting its target node; and then selecting from amongst the highlighted nodes and edges etc. The query is complete after its output variables are also chosen. This process ensures that an *AnQ* is over connected nodes from the *AnS*. An *AnQ* is specified as one BGP classifier and one BGP measure (starting from the same initial node – set of facts), while the aggregation operation (sum, max etc.) is selected from a drop-down list. Figure 8.8(d) shows the GUI's visualization for an analytical query (similar to the one in Figure 7.8), where the classifier is depicted in blue, the measure in green, while the common set of facts is the node x (orange). Each node in this query graph represents a variable, whose type appears after the colon.

(5) The user can trigger the evaluation of the chosen AnQ, again delegated to kdb+, and inspect the results. We are currently working on implementing several visualization options for exploring these results.

8.7 Summary

Our experiments demonstrate the feasibility of our full RDF warehousing approach, which exploits standard RDF functionalities such as triple storage, conjunctive query evaluation, and reasoning. We showed robust scalable performance when loading and saturating G, and building $\mathcal{I}$ in time linear in the input size (even for complex, many-joins node and edge queries). Finally, we proved that OLAP operations can be evaluated quite efficiently in our RDF cube (AnQ) context. While further optimizations are possible, our experiments confirmed the interest and good performance of our proposed all-RDF Semantic Web warehousing approach.

Concluding Remarks

Data warehouse models and techniques have had a strong impact on the usages and usability of data. In this work, we proposed the first approach for specifying and exploiting an *RDF data warehouse*, notably by

(*i*) defining an analytical schema that captures the information of interest, and

(*ii*) formalizing analytical queries (or cubes) over the analytical schema.

Importantly, instances of analytical schemas are *RDF graphs themselves*, which permits exploiting the semantics and heterogeneous structure (e.g., jointly query the schema and the data) that make RDF data interesting.

This novel framework for RDF analytics, can be efficiently deployed on top of any RDF data management platform, to extend it with analytic capabilities. We fully implemented our approach in an operational prototype and empirically demonstrated its interest and performance.

Compared to other works in the literature, our approach is not focused on a specific vocabulary like [Etcheverry12, W3C14d]. Also, in contrast to [Nebot12], in our work, the analytical schema instance is an RDF graph itself thus seamlessly preserves the heterogeneity, semantics, and ability to query the schema together with the data, present in RDF.

Unlike the recent graph warehousing approaches [Zhao11, Bleco12], our warehouse is built to handle *heterogeneous* graphs, and keep the data semantics, both central in RDF. Moreover, our analytical queries provide more flexibility, by allowing diverse aggregation functions.

Multidimensional modeling based on an object-oriented paradigm [Boukraâ13] bears some similarities with our work, but it is not dedicated to Semantic Web data, and more importantly, it does not allow defining analytical schema edges independently from nodes. As we explained in Section 7.2, this independent definition is crucial in meeting the RDF analytics requirement we identified in Section 7.1.

Our approach of separating grouping and aggregation in the analytical queries (by the use of a classifier and a measure query) is in line with the MD-join operator [Chatziantoniou01] for relational data warehouses.

The techniques for transforming OLAP queries into SPARQL, proposed by [Kämpgen13] can be added to our framework in order to further optimize analytical query answering. Moreover, deploying our framework on an efficient SPARQL 1.1 [W3C13] platform (featuring SQL-style grouping and aggregation) enables taking advantage both of its efficiency and of the high-level, expressive, flexible RDF graph analysis concepts introduced in this work.

Chapter 9

Conclusion

The expanding world of Semantic Web data is made truly valuable by the tools developed for exploring and processing it. Handling the unstructured and semi-structured modern day data, together with its potential for inferring new information is a challenge faced by data management systems. In a context where data publication far exceeds the current tools' capacities for analyzing it, it is essential to find ways to take advantage of the available technologies.

In this thesis we propose efficient algorithms and pertinent formalizations for handling RDF data complexity, while still allowing easy portability to existing relational database management systems. We analyze two critical problems, query answering over data subject to semantic constraints and complex analytics on heterogeneous, semantic-rich data.

9.1 Saturation vs. Reformulation

Ontology languages are used to add semantic constraints to RDF data. Such constraints model implicit information that is expected to be included in the answers to queries. The literature proposes two main approaches for querying data in the presence of semantic constraints. The first and simplest approach is to alter the data by making explicit, all its inferable information. The alternative is to build a new query that uses the constraints to reshape the question in such a way as to obtain the correct set of answers.

Previous works have mostly viewed the two techniques as orthogonal problems and have focused on improving either one or the other. In contrast we formalize a common setting for comparing the two approaches, while also improving the state of the art for each. Notably we propose a new data saturation algorithm that is robust to changes brought to the data instance and schema. Also, we describe query reformulation for the considered RDF fragment, which extends those in the literature by the inclusion of blank nodes.

The choice of RDF data fragment was made in an informed fashion, taking into account the portability to relational database management systems. We implemented our algorithms on top of such a systems and presented a thorough experimental comparison of the two approaches. In particular we showed that the implementation on top of an

RDBMS rivals the use of dedicated systems and is oftentimes more efficient. Moreover our experimental comparison of the two approached quantifies their strengths and weaknesses, allowing a database administrator to make an informed choice between them.

The work in this thesis was focused on individually extending the saturation and reformulation approaches and comparing them. However, mixed approaches have also been proposed in the literature. In [Urbani11] the authors leverage the benefits of both forward and backward chaining, proposing a mixed approach that makes use of precomputed reasoning steps to reduce the number of query reformulations and consequently its run time. The results of Section 5.6 could be extended to also compare saturation or reformulation with such mixed approaches.

Improving query reformulation. Recall that some of our large-reformulation queries could not be evaluated by the RDBMS (Figure 5.4). These reformulated queries present a great number of common subexpressions that are evaluated multiple times in the current implementation. Optimizing the current state-of-the-art query reformulation language for the DB fragment is ongoing work as the topic of a separate PhD thesis.

9.2 RDF Analytics

Heterogeneity significantly complicates RDF data analytics. Existing works tackle the problem by normalizing the data in the Extract Transform Load process, occasionally also allowing null values and nesting. In contrast, we view heterogeneity as an essential desired feature of RDF data, that should be propagated to the data warehouse storing it.

Moreover, we go beyond the classical star (or snowflake) data warehouse schemas, where facts of a single kind can be analyzed based on a specified set of dimensions and measures. In our analytical setting the facts, dimensions and measures are chosen at query run time. This allows a great flexibility in the choice of analyses, in particular even enabling the analysis of core concepts by means of other core concepts.

Notably, in our framework, the warehousing process does not change the structure of the data. Hence, the RDF semantics are maintained and can even be used inside analyses. In particular our setting facilitates querying the schema to find and analyze the relationships between entities.

To the best of our knowledge, our framework is the first one to keep the data entirely in RDF while still providing meaningful data analysis. In this thesis we demonstrated both the theoretical benefits of such an approach, and its practicability.

Automatic analytical schema design. In our proposed framework, analytical schemas are designed by the data analyst. This task may have daunting complexity, given the very numerous alternative ways of choosing analytical schema nodes and edges. At the same time, the choice of an analytical schema conditions (determines) any analysis that may be subsequently performed, thus it is very important that the schema be carefully chosen. In this context, one can devise a set of metrics characterizing the interest and relevance of an analytical schema with respect to a given RDF dataset. Based on these

metrics, a search algorithm can then be devised to identify schemas having good properties. This work has started in the group as part of a Master thesis and is ongoing at the time of the writing.

Analytical queries as views. The OLAP operations described thus far were applied on analytical queries and evaluated against the data warehouse instance. We are interested in improving performance of such operations by evaluating them directly on the materialized results of previous analytical queries (significantly reducing the input data and benefiting from the regular-structure analytical query results). We started analyzing the situations where such "shortcuts" are applicable and devising concrete algorithms for computing the results of an analytical query based on the previously materialized results of another analytical query. This work has started and is ongoing, as part of a distinct Master thesis.

9.3 Perspectives

In addition to the ongoing work mentioned above, we identify several avenues for potential follow-up works. These mostly focus on: the optimization of the described techniques; and the automation of data analysis.

Extending the RDF fragment and query language. The main contributions in Part I relate to answering instance-level queries over the introduced DB fragment of RDF. We see several paths for extending our algorithms, such as evaluating both instance and schema-level queries, considering a larger set of entailment rules and expanding the query language to also allow disjunctions and negations.

Benchmarking RDF Schema updates. Maintaining the data saturation after a schema update may lead to diverse outcomes, namely altering from a few triples to significant portions of the database. To our knowledge at this date no works have proposed a benchmark for RDF Schema updates. Most works, as in our case, make a best effort at illustrating the variety of outcomes that can occur. I am interested in exploring a standardized approach to evaluate schema updates over RDF data.

Dynamic choice of inference technique. The saturation thresholds introduced in Section 5.6, can be extended to compare saturation-based query answering with the optimized reformulation-based technique described above. These thresholds allow the database administrator to make an informed choice regarding which inference technique to adopt. However, a workload for computing such thresholds is not always available. In particular, in the case of the Semantic Web, the interest is to continuously add new data, infer new semantics and explore the available data through queries. We are interested in leveraging the information gained through the computation of thresholds to make the choice of inference technique dynamically, at run-time, and benefit from both query answering approaches. For this we envision a global system threshold that is adjusted as queries and updates arrive. It may lead the system to saturate the data or to reformulate the queries. By keeping track of the schema triples for which saturation was already applied,

we can further optimize query reformulation to use only subparts of the schema, while still returning the correct and complete query answers.

Parallel analytics. The broader area of *data analytics*, related to data warehousing, albeit with a significantly extended set of goals and methods, is the target of very active research now, especially in the context of massively parallel Map-Reduce processing. We are interested in extending such techniques to our analytical schemas and queries to further improve the deployment of the data warehouse and analytical query evaluation. Efficient methods for deploying our analytical schemas and evaluating analytical queries in such a parallel context are part of our future work.

Integrating known vocabularies. The W3C has proposed recommendations for publishing cube data on the Web [W3C14d]. Though orthogonal to our work, this topic has a great potential for integration with our analytics framework. Therefore, manipulating such data within our framework is a venue worth exploring.

Code release. Finally, to increase the visibility of our analytics framework, we plan to make a public version available to users that want to profit from the interactive interface and the high potential for data analysis. Also, we envision expanding the WaRG tool to be pluggable on top of multiple database backends.

Appendix A

Theorem Proofs

A.1 Proof of Theorem 4.2

For one direction ($\Leftarrow$), the proof is trivial as the rules of Table 2.3(d) are among those defining $\mathtt{db}^{\propto}$.

For the converse direction ($\Rightarrow$), let us call a *derivation* of t any sequence of immediate entailment rules that produces the entailed triple t, starting from db. Let us consider, without loss of generality, a *minimal* derivation (i.e., in which removing a step of rule application does not allow deriving t anymore). A derivation can be minimized by gradually removing steps producing entailed triples that are not further reused in the entailment sequence of t. We show for such a *minimal* derivation of an entailed triple t that any step using a rule that is not in Table 2.3(d) can be replaced by a sequence of steps using only rules from Table 2.3(d), leading to another derivation of t. Applying exhaustively the above replacement on the minimization of obtained derivations obviously leads to a derivation of t using the rules in Table 2.3(d) only.

Consider a minimal derivation of t using the immediate entailment rule from Table 2.3(d):

$$\mathtt{s}\ \text{rdfs:subClassOf}\ \mathtt{o},\ \mathtt{s}_1\ \text{rdf:type}\ \mathtt{s}\ \vdash_{\text{RDF}}\ \mathtt{s}_1\ \text{rdf:type}\ \mathtt{o}$$

While the triple $\mathtt{s}_1$ rdf:type s is either in db or produced by a rule from Table 2.3(d) (only the rules in Table 2.3(d) produce such a triple), the triple s rdfs:subClassOf o may result from the triples:

$$\{\mathtt{s}\ \text{rdfs:subClassOf}\ \mathtt{o}_n,\ \mathtt{o}_n\ \text{rdfs:subClassOf}\ \mathtt{o}_{n-1},\ \ldots,\ \mathtt{o}_1\ \text{rdfs:subClassOf}\ \mathtt{o}\} \subseteq \mathtt{db}$$

and n applications of the rule:

$$\mathtt{s}\ \text{rdfs:subClassOf}\ \mathtt{o},\ \mathtt{o}\ \text{rdfs:subClassOf}\ \mathtt{o}_1\ \vdash_{\text{RDF}}\ \mathtt{s}\ \text{rdfs:subClassOf}\ \mathtt{o}_1$$

from Table 2.3(c) (only that rule produces triples of the form s rdfs:subClassOf o).

Observe that we do not have to consider the rules from Table 2.3(b) in a *minimal* derivation. It is therefore easy to see that the application of:

$$\mathsf{s}\ \text{rdfs:subClassOf}\ \mathsf{o}, \mathsf{s}_1\ \text{rdf:type}\ \mathsf{s} \vdash_{\text{RDF}} \mathsf{s}_1\ \text{rdf:type}\ \mathsf{o}$$

in the derivation of t can be replaced by the following sequence:

s rdfs:subClassOf o_n, s_1 rdf:type s $\vdash_{\text{RDF}}$ s_1 rdf:type o_n ,
o_n rdfs:subClassOf o_{n-1}, s_1 rdf:type o_n $\vdash_{\text{RDF}}$ s_1 rdf:type o_{n-1} ,
. . . ,
o_1 rdfs:subClassOf o, s_1 rdf:type o_1 $\vdash_{\text{RDF}}$ s_1 rdf:type o .

The rest of the proof is omitted as it amounts to showing, similarly as above, that the claim also holds for the three other immediate entailment rules of Table 2.3(d).

A.2 Proof of Theorem 4.4

A close examination of the saturation rules exhibits producer-consumer dependencies among rules. For instance, triples produced by the rule (4.1) can only be used to further apply the same rule. Hence, rule (4.1) can only *feed* itself. One can similarly see that rules (4.2) and (4.3) can only feed rule (4.1), and rule (4.4) can only feed itself plus rules (4.2) and (4.3).

Given the two possible forms of instance-level triples, the saturation schemes based on the above dependencies can be written, using regular expressions, as follows.

Instance-level triple	Saturation scheme
`s` rdf:type `o`	$(4.1)^*$
`s p o`	$(4.4)^*.((4.2) + (4.3)).(4.1)^*$

This said, we provide now an upper bound for the size of `Saturate(db)`, when `db` contains the single instance-level triple `s` rdf:type `o` or `s p o`. Assume `db` $= \langle$`S`, `D`$\rangle$ and let #`S` and #`D` be the sizes (number of triples) of `S` and `D` respectively.

- The triple `s` rdf:type `o` can be transformed at most #`S` times by the sequence of rules $(4.1)^*$ (there are at most #`S` schema-level triples in `db`).
- The triple `s p o` can be transformed at most #`S` times by the sequence of rules $(4.4)^*.(4.2).(4.1)^*$ and also at most #`S` times by the sequence of rules $(4.4)^*.(4.3).(4.1)^*$ (there is at most #`S` schema-level triples in `db`).

As a result, the worst-case is for a triple of the form `s p o`. Therefore, the overall upper bound for the size of the output of `Saturate(db)` is $2 * \#\texttt{S}$.

The above result easily generalizes to a database whose instance-level is of size #`D`, namely $2 * \#\texttt{S} * \#\texttt{D}$. Given that $\#\texttt{db} = \#\texttt{S} + \#\texttt{D}$, the size of the output of `Saturate(db)` is in $O(\#\texttt{db}^2)$.

A.3 Proof of Proposition 4.5

$\texttt{Saturate}(\texttt{db}) = \texttt{set}(\texttt{Saturate}_{+}(\texttt{db}))$ is shown by induction on the number k of saturation steps by proving: $\texttt{Saturate}^{k}(\texttt{db}) = \texttt{set}(\texttt{Saturate}^{k}_{+}(\texttt{db}))$.

Base step:
By definition, $\texttt{Saturate}^{0}(\texttt{db}) = \texttt{db}$ and $\texttt{Saturate}^{0}_{+}(\texttt{db}) = \texttt{db}$.
Thus $\texttt{Saturate}^{0}(\texttt{db}) = \texttt{set}(\texttt{Saturate}^{0}_{+}(\texttt{db}))$ holds.

Inductive step:
Suppose that $\texttt{Saturate}^{k}(\texttt{db}) = \texttt{set}(\texttt{Saturate}^{k}_{+}(\texttt{db}))$ for $k < \alpha$ and let us show that it still holds for $k = \alpha$.

By definition,
$\texttt{Saturate}^{\alpha}(\texttt{db}) = \texttt{Saturate}^{\alpha-1}(\texttt{db}) \cup$
$\{t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i)
on some $\{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}(\texttt{db})$
yields t_3 with $t_2 \notin \texttt{Saturate}^{j<\alpha-1}(\texttt{db})\}$.

By the induction hypothesis,
$\texttt{Saturate}^{\alpha}(\texttt{db}) = \texttt{set}(\texttt{Saturate}^{\alpha-1}_{+}(\texttt{db})) \cup$
$\{t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i)
on some $\{t_1, t_2\} \subseteq \texttt{set}(\texttt{Saturate}^{\alpha-1}_{+}(\texttt{db}))$
yields t_3 with $t_2 \notin \texttt{set}(\texttt{Saturate}^{j<\alpha-1}_{+}(\texttt{db}))\}$ holds.

That is, given the semantics of the set and $\uplus$ operators,
$\texttt{Saturate}^{\alpha}(\texttt{db}) = \texttt{set}(\texttt{Saturate}^{\alpha-1}_{+}(\texttt{db}) \uplus$
$\{t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i)
on some $\{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_{+}(\texttt{db})$
yields t_3 with $t_2 \notin \texttt{Saturate}^{j<\alpha-1}_{+}(\texttt{db})\}) = \texttt{set}(\texttt{Saturate}^{\alpha}_{+}(\texttt{db}))$ holds.

A.4 Proof of Theorem 4.7

Let us consider the three cases for insertions.

- **Insertion Case 1:** $t \in \texttt{db}$.
  Because db is a set of triples and $t \in \texttt{db}$, we have $\texttt{db} \cup \{t\} = \texttt{db}$.
  Thus $\texttt{Saturate}_+(\texttt{db} \cup \{t\}) = \texttt{Saturate}_+(\texttt{db})$.

- **Insertion Case 2:** $t \notin \texttt{db}$ **is an instance-level triple.**
  Given that $t \notin \texttt{db}$ and t is an instance-level triple,

  $$\texttt{Saturate}_+(\texttt{db} \cup \{t\}) = \texttt{Saturate}_+(\texttt{db}) \uplus [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]$$

  is proved by showing the more general result:

  $$\texttt{Saturate}_+(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle) = \texttt{Saturate}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ].$$

  Indeed, observe that $\texttt{Saturate}_+(\texttt{db} \cup \{t\}) = \texttt{Saturate}_+(\langle \texttt{S}, \texttt{D} \uplus \{t\}\rangle)$, since $t \notin \texttt{db}$.
  The proof is by induction on the number k of saturation steps:

  $$\texttt{Saturate}^k_+(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle) = \texttt{Saturate}^k_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^k_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ].$$

  Base step:
  By definition $\texttt{Saturate}^0_+(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle) = \langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle$ and

  $$\begin{aligned} \texttt{Saturate}^0_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^0_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ] &= \langle \texttt{S}, \texttt{D}_1\rangle \uplus [\ \langle \texttt{S}, \texttt{D}_2\rangle \setminus_+ \texttt{S}\ ] \\ &= \langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle. \end{aligned}$$

  Thus, the following equation holds:

  $$\texttt{Saturate}^0_+(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle) = \texttt{Saturate}^0_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^0_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ].$$

  Inductive step:
  Suppose that the following equation holds for $k < \alpha$

  $$\texttt{Saturate}^k_+(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2\rangle) = \texttt{Saturate}^k_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^k_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ]$$

  and let us show that it still holds for $k = \alpha$.

  By definition, we have:

  $$\begin{aligned} &\texttt{Saturate}^\alpha_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ] \\ &= [\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \\ &\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \texttt{D}_1\rangle)\ \}\ ] \\ &\uplus [\ (\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \\ &\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \texttt{D}_2\rangle)\ \}\ ) \setminus_+ \texttt{S}\ ]. \end{aligned}$$

  By the semantics of the $\uplus$ and $\setminus_+$ operators, it holds that:

  $$\begin{aligned} &\texttt{Saturate}^\alpha_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ] \\ &= [\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus (\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S})\ ] \\ &\uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \text{ on some} \\ &\quad \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ] \\ &\quad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \texttt{D}_1\rangle) \uplus [\ \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \texttt{D}_2\rangle) \setminus_+ \texttt{S}\ ]\ \}. \end{aligned}$$

  By the induction hypothesis, we get:

$\texttt{Saturate}^{\alpha}_{+}(\langle \texttt{S}, \texttt{D}_1 \rangle) \uplus [\ \texttt{Saturate}^{\alpha}_{+}(\langle \texttt{S}, \texttt{D}_2 \rangle) \setminus_{+} \texttt{S}\ ]$
$= \texttt{Saturate}^{\alpha-1}_{+}(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2 \rangle)$
$\uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i) on some
$\{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_{+}(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2 \rangle)$
yields t_3 with $t_2 \notin \texttt{Saturate}^{j<\alpha-1}_{+}(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2 \rangle)\ \}$.

Therefore, it holds that:
$\texttt{Saturate}^{\alpha}_{+}(\langle \texttt{S}, \texttt{D}_1 \rangle) \uplus [\ \texttt{Saturate}^{\alpha}_{+}(\langle \texttt{S}, \texttt{D}_2 \rangle) \setminus_{+} \texttt{S}\ ] = \texttt{Saturate}^{\alpha}_{+}(\langle \texttt{S}, \texttt{D}_1 \uplus \texttt{D}_2 \rangle)$.

- **Insertion Case 3:** $t \notin \texttt{db}$ **is a schema-level triple.**
Given that $t \notin \texttt{db}$ and t is a schema-level triple, we prove that

$\texttt{Saturate}_{+}(\texttt{db} \cup \{t\}) = \texttt{Saturate}_{+}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{t' \in \texttt{D}'} [\ \texttt{Saturate}_{+}(\langle \texttt{S}, \{t'\} \rangle) \setminus_{+} \texttt{S}\ ]$, where the multiset $\texttt{D}'$ is $\{\ t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i) on $\{t, t_2\}$ with $t_2 \in \texttt{Saturate}_{+}(\texttt{db})$ yields $t_3\}$.

We show this by induction on the number k of saturation steps:

$$\texttt{Saturate}^{k}_{+}(\texttt{db} \cup \{t\}) \quad = \quad \texttt{Saturate}^{k}_{+}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{k} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}^{k-l}_{+}(\langle \texttt{S}, \{t'_l\} \rangle) \setminus_{+} \texttt{S}\ ],$$

where the multiset $\texttt{D}'_l$ is $\emptyset$ for $l = 0$ and $\{\ t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i) on $\{t, t_2\}$ yields t_3 with $t_2 \in \texttt{Saturate}^{l-1}_{+}(\texttt{db})$ and $t_2 \notin \texttt{Saturate}^{j<l-1}_{+}(\texttt{db})\}$ for $l > 0$. Observe that, by definition, $\texttt{D}' = \biguplus_{l=0}^{\infty} \texttt{D}'_l$, i.e., whenever the saturation fixed-point is reached.

Base step:
By definition, $\texttt{Saturate}^{0}_{+}(\texttt{db} \cup \{t\}) = \texttt{db} \cup \{t\}$ holds. In turn, by definition,

$\texttt{Saturate}^{0}_{+}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{0} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}^{0-l}_{+}(\langle \texttt{S}, \texttt{D}'_l \rangle) \setminus_{+} \texttt{S}\ ]$
$= \texttt{db} \uplus \{t\}$
$= \texttt{db} \cup \{t\}$ since $t \notin \texttt{db}$.

Therefore, the following equation holds:

$$\texttt{Saturate}^{0}_{+}(\texttt{db} \cup \{t\}) \quad = \quad \texttt{Saturate}^{0}_{+}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{0} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}^{0-l}_{+}(\langle \texttt{S}, \texttt{D}'_l \rangle) \setminus_{+} \texttt{S}\ ].$$

Inductive step:
Suppose that the following equation holds for $k < \alpha$

$$\texttt{Saturate}^{k}_{+}(\texttt{db} \cup \{t\}) \quad = \quad \texttt{Saturate}^{k}_{+}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{k} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}^{k-l}_{+}(\langle \texttt{S}, \{t'_l\} \rangle) \setminus_{+} \texttt{S}\ ]$$

and let us show that it still holds for $k = \alpha$.

By definition,

$\texttt{Saturate}^{\alpha}_{+}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}^{\alpha-l}_{+}(\langle \texttt{S}, \{t'_l\} \rangle) \setminus_{+} \texttt{S}\ ]$
$= (\ \texttt{Saturate}^{\alpha-1}_{+}(\texttt{db}) \uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i)
on some $\{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_{+}(\texttt{db})$
yields t_3 with $t_2 \notin \texttt{Saturate}^{j<\alpha-1}_{+}(\texttt{db})\ \}\ )$
$\uplus \{t\} \uplus \biguplus_{t'_\alpha \in \texttt{D}'_\alpha} [\ \texttt{Saturate}^{0}_{+}(\langle \texttt{S}, \{t'_\alpha\} \rangle) \setminus_{+} \texttt{S}\ ]$
$\uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l} [\ (\ \texttt{Saturate}^{\alpha-1-l}_{+}(\langle \texttt{S}, \{t'_l\} \rangle) \uplus$
$\{\ t_3 \mid \exists\, i \in [4.1, 4.4]$ such that applying rule (i)
on some $\{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1-l}_{+}(\langle \texttt{S}, \{t'_l\} \rangle)$
yields t_3 with $t_2 \notin \texttt{Saturate}^{j<\alpha-1-l}_{+}(\langle \texttt{S}, \{t'_l\} \rangle)\ \}\ ) \setminus_{+} \texttt{S}\ ]$ holds.

That is, by the semantics of the $\uplus$ and $\setminus_+$ operators,

$$\begin{aligned}
&\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ] \\
&= \texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ] \\
&\uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\quad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \\
&\quad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\ \} \\
&\uplus \biguplus_{t'_\alpha \in \texttt{D}'_\alpha} [\ \langle \texttt{S}, \{t'_\alpha\}\rangle \setminus_+ \texttt{S}\ ] \\
&\uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l} \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\ \} \text{ holds.}
\end{aligned}$$

By the induction hypothesis,

$$\begin{aligned}
&\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ] \\
&= \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \cup \{t\}) \uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\ \} \\
&\uplus \texttt{D}'_\alpha \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l} \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\ \} \text{ holds.}
\end{aligned}$$

By definition of $\texttt{D}'_{0 \le i \le \alpha}$,

$$\begin{aligned}
&\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ] \\
&= \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \cup \{t\}) \uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db} \cup \{t\}) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db} \cup \{t\})\ \} \text{ holds.}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\texttt{Saturate}_+^{\alpha}(\texttt{db}) \uplus \{t\} \uplus \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ] \\
&= \texttt{Saturate}_+^{\alpha}(\texttt{db} \cup \{t\}) \text{ holds.}
\end{aligned}$$

Let us now consider the three cases for deletions.

- **Deletion Case 1:** $t \in \texttt{db}$**.**
  Because db is a set of triples and $t \notin \texttt{db}$, we have $\texttt{db} \setminus \{t\} = \texttt{db}$.
  Thus $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+(\texttt{db})$.

- **Deletion Case 2:** $t \notin \texttt{db}$ **is an instance-level triple.**
  Given that $t \in \texttt{db}$ and t is an instance-level triple,

  $$\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]$$

  is shown on the number k of saturation steps:

  $$\texttt{Saturate}_+^{k}(\texttt{db} \setminus \{t\}) = \texttt{Saturate}_+^{k}(\texttt{db}) \setminus_+ [\ \texttt{Saturate}_+^{k}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ].$$

  Base step:
  By definition $\texttt{Saturate}_+^{0}(\texttt{db} \setminus \{t\}) = \texttt{db} \setminus \{t\}$ and

$$\begin{aligned}\texttt{Saturate}^0_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^0_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] &= \texttt{db} \setminus_+ [\ \langle \texttt{S}, \{t\}\rangle \setminus_+ \texttt{S}\ ] \\ &= \texttt{db} \setminus_+ \{t\} \\ &= \texttt{db} \setminus \{t\} \text{ since } t \in \texttt{db}.\end{aligned}$$

Thus, $\texttt{Saturate}^0_+(\texttt{db} \setminus \{t\}) = \texttt{Saturate}^0_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^0_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]$ holds.

Inductive step:

Suppose that the following equation holds for $k < \alpha$

$$\texttt{Saturate}^k_+(\texttt{db} \setminus \{t\}) = \texttt{Saturate}^k_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^k_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]$$

and let us show that it still holds for $k = \alpha$.

By definition the following holds:

$$\begin{aligned}&\texttt{Saturate}^\alpha_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] \\ &= (\ \texttt{Saturate}^{\alpha-1}_+(\texttt{db}) \ \uplus\ \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\texttt{db}) \\ &\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\texttt{db})\ \}\ ) \\ &\setminus_+ [\ (\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \ \uplus\ \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \\ &\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle)\ \}\ ) \setminus_+ \texttt{S}\ ]\end{aligned}$$

Since $\texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \subseteq \texttt{Saturate}^\alpha_+(\texttt{db})$, by the semantics of the $\uplus$ and $\setminus_+$ operators,

$$\begin{aligned}&\texttt{Saturate}^\alpha_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] \\ &= (\ \texttt{Saturate}^{\alpha-1}_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]) \\ &\uplus (\ \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\texttt{db}) \\ &\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\texttt{db})\ \} \\ &\quad \setminus_+ \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \\ &\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle)\ \}\ ) \text{ holds.}\end{aligned}$$

By the induction hypothesis,

$$\begin{aligned}&\texttt{Saturate}^\alpha_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] = \texttt{Saturate}^{\alpha-1}_+(\texttt{db} \setminus \{t\}) \\ &\uplus (\ \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\texttt{db}) \\ &\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\texttt{db})\ \} \\ &\quad \setminus_+ \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \\ &\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle)\ \}\ ) \text{ holds.}\end{aligned}$$

That is,

$$\begin{aligned}&\texttt{Saturate}^\alpha_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] = \texttt{Saturate}^{\alpha-1}_+(\texttt{db} \setminus \{t\}) \\ &\uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\quad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^{\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] \\ &\quad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^{j<\alpha-1}_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ]\ \} \text{ holds.}\end{aligned}$$

By the induction hypothesis,

$$\begin{aligned}&\texttt{Saturate}^\alpha_+(\texttt{db}) \setminus_+ [\ \texttt{Saturate}^\alpha_+(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] = \texttt{Saturate}^{\alpha-1}_+(\texttt{db} \setminus \{t\}) \\ &\uplus \{\ t_3 \mid \exists\, i \in [4.1, 4.4] \text{ such that applying rule } (i) \\ &\quad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}^{\alpha-1}_+(\texttt{db} \setminus \{t\}) \\ &\quad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}^{j<\alpha-1}_+(\texttt{db} \setminus \{t\})\ \} \text{ holds.}\end{aligned}$$

Therefore, $\texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ [\ \texttt{Saturate}_+^{\alpha}(\langle \texttt{S}, \{t\}\rangle) \setminus_+ \texttt{S}\ ] = \texttt{Saturate}_+^{\alpha}(\texttt{db} \cup \{t\})$ holds.

- **Deletion Case 3:** $t \notin \texttt{db}$ **is a schema-level triple.**
  Given that $t \in \texttt{db}$ and t is a schema-level triple, $\texttt{Saturate}_+(\texttt{db} \setminus \{t\}) = (\ \texttt{Saturate}_+(\texttt{db}) \setminus_+ \{t\}\ ) \setminus_+ (\ \biguplus_{t' \in \texttt{D}'} [\ \texttt{Saturate}_+(\langle \texttt{S}, \{t'\}\rangle) \setminus_+ \texttt{S}\ ]\ )$, where the multiset $\texttt{D}'$ is $\{\ t_3 \mid \exists\ i \in [4.1, 4.4]$ such that applying rule (i) on $\{t, t_2\}$ with $t_2 \in \texttt{Saturate}_+(\texttt{db})$ yields $t_3\}$. We actually show this by induction on the number k of saturation steps: $\texttt{Saturate}_+^k(\texttt{db} \setminus \{t\}) = (\ \texttt{Saturate}_+^k(\texttt{db}) \setminus_+ \{t\}\ ) \setminus_+ (\ \biguplus_{l=0}^{k} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{k-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ]\ )$, where the multiset $\texttt{D}'_l$ is $\emptyset$ for $l = 0$ and $\{\ t_3 \mid \exists\ i \in [4.1, 4.4]$ such that applying rule (i) on $\{t, t_2\}$ yields t_3 with $t_2 \in \texttt{Saturate}_+^{l-1}(\texttt{db})$ and $t_2 \notin \texttt{Saturate}_+^{j<l-1}(\texttt{db})\ \}$ for $l > 0$. Indeed, observe that, by definition, $\texttt{D}' = \biguplus_{l=0}^{\infty} \texttt{D}'_l$, i.e., whenever the saturation fixed-point is reached.

  Base step:
  By definition, $\texttt{Saturate}_+^0(\texttt{db} \setminus \{t\}) = \texttt{db} \cup \{t\}$ holds. In turn, by definition,

  $$\begin{aligned}&(\ \texttt{Saturate}_+^0(\texttt{db}) \setminus_+ \{t\}\ ) \setminus_+ (\ \biguplus_{l=0}^{0} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{0-l}(\langle \texttt{S}, \texttt{D}'_l\rangle) \setminus_+ \texttt{S}\ ]\ )\\ &= \texttt{db} \setminus_+ \{t\}\\ &= \texttt{db} \setminus \{t\} \text{ since } t \notin \texttt{db}.\end{aligned}$$

  Therefore,

  $$\begin{aligned}\texttt{Saturate}_+^0(\texttt{db} \setminus \{t\}) = {}&(\ \texttt{Saturate}_+^0(\texttt{db}) \setminus_+ \{t\})\\ &\setminus_+ (\ \biguplus_{l=0}^{0} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{0-l}(\langle \texttt{S}, \texttt{D}'_l\rangle) \setminus_+ \texttt{S}\ ]\ ) \text{ holds.}\end{aligned}$$

  Inductive step:
  Suppose that the following equation holds for $k < \alpha$

  $$\begin{aligned}\texttt{Saturate}_+^k(\texttt{db} \setminus \{t\}) = {}&(\ \texttt{Saturate}_+^k(\texttt{db}) \setminus_+ \{t\}\ )\\ &\setminus_+ (\ \biguplus_{l=0}^{k} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{k-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ]\ )\end{aligned}$$

  and let us show that it still holds for $k = \alpha$.

  By definition, the following equation holds

  $$\begin{aligned}&(\ \texttt{Saturate}_+^{\alpha}(\texttt{db}) \setminus_+ \{t\}\ ) \setminus_+ (\ \biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ]\ )\\ &= (\ [\ \texttt{Saturate}_+^{\alpha-1}(\texttt{db}) \uplus \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i)\\ &\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})\\ &\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1}(\texttt{db})\ \}\ ] \setminus_+ \{t\}\ )\\ &\setminus_+ (\ \biguplus_{t'_\alpha \in \texttt{D}'_\alpha} [\ \texttt{Saturate}_+^0(\langle \texttt{S}, \{t'_\alpha\}\rangle) \setminus_+ \texttt{S}\ ]\\ &\quad \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l} [\ (\ \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\\ &\qquad \uplus \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i)\\ &\qquad\quad \text{on some } \{t_1, t_2\} \subseteq \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\\ &\qquad\quad \text{yields } t_3 \text{ with } t_2 \notin \texttt{Saturate}_+^{j<\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle)\ \}\ ) \setminus_+ \texttt{S}\ ]\ ).\end{aligned}$$

  That is, by the semantics of the $\uplus$ and $\setminus_+$ operators and since $\biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \texttt{D}'_l} [\ \texttt{Saturate}_+^{\alpha-1-l}(\langle \texttt{S}, \{t'_l\}\rangle) \setminus_+ \texttt{S}\ ] \subseteq \texttt{Saturate}_+^{\alpha-1}(\texttt{db})$ holds due to $t \in \texttt{db}$,

$$
\begin{aligned}
&(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) \\
&= [(\mathtt{Saturate}_+^{\alpha-1}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} (\mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S})] \\
&\uplus \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\quad \text{on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db}) \\
&\quad \text{yields } t_3 \text{ with } t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\mathtt{db}) \ \} \\
&\setminus_+ (\biguplus_{t'_\alpha \in \mathtt{D}'_\alpha} [\mathtt{Saturate}_+^{0}(\langle \mathtt{S}, \{t'_\alpha\}\rangle) \setminus_+ \mathtt{S}] \\
&\quad \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \ \} \ ) \text{ holds.}
\end{aligned}
$$

By the induction hypothesis,

$$
\begin{aligned}
&(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) \\
&= \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db} \setminus \{t\}) \uplus \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db}) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\mathtt{db}) \ \} \\
&\setminus_+ (\mathtt{D}'_\alpha \uplus \biguplus_{l=0}^{\alpha-1} \biguplus_{t'_l \in \mathtt{D}'_l} \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \ \} \ ) \text{ holds.}
\end{aligned}
$$

By definition of $\mathtt{D}'_{0 \le i \le \alpha}$,

$$
\begin{aligned}
&(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}]) \\
&= \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db} \setminus \{t\}) \uplus \{\ t_3 \mid \exists\ i \in [4.1, 4.4] \text{ such that applying rule } (i) \\
&\qquad\qquad \text{on some } \{t_1, t_2\} \subseteq \mathtt{Saturate}_+^{\alpha-1}(\mathtt{db} \setminus \{t\}) \\
&\qquad\qquad \text{yields } t_3 \text{ with } t_2 \notin \mathtt{Saturate}_+^{j<\alpha-1}(\mathtt{db} \setminus \{t\}) \ \} \text{ holds.}
\end{aligned}
$$

Therefore, $(\mathtt{Saturate}_+^{\alpha}(\mathtt{db}) \setminus_+ \{t\}) \setminus_+ (\biguplus_{l=0}^{\alpha} \biguplus_{t'_l \in \mathtt{D}'_l} [\mathtt{Saturate}_+^{\alpha-l}(\langle \mathtt{S}, \{t'_l\}\rangle) \setminus_+ \mathtt{S}])$ $= \mathtt{Saturate}_+^{\alpha}(\mathtt{db} \setminus \{t\})$ holds.

A.5 Proof of Theorem 4.14

As in the proof for Theorem 4.4, a close examination of the reformulation rules also exhibits producer-consumer dependencies among rules.

Given the possible forms of query triples that trigger reformulation rules, we write the reformulation schemes based on these dependencies using regular expressions as follows.

Query triple	Reformulation scheme
s ?y o	[(4.5).((4.9) + (4.10) + (4.11) + (4.12) + (4.13)).(4.14)*.((4.15) +(4.16)).(4.17)*] + [((4.6) + (4.7) + (4.8)).(4.17)*]
s rdf:type ?z	((4.9) + (4.10) + (4.11) + (4.12) + (4.13)).(4.14)*.((4.15)+ 4.16)).(4.17)*
s rdf:type **c**	(4.14)*.((4.15) + (4.16)).(4.17)*
s **p** o	(4.17)*

This said, we provide now an upper bound for the size of $\texttt{Reformulate}(q, \texttt{db})$, when q contains a single triple s ?y o, s rdf:type ?z, s rdf:type **c**, or s **p** o Assume $\texttt{db} = \langle \texttt{S}, \texttt{D} \rangle$ and let #S and #D the sizes (number of triples) of S and D respectively.

The triple s ?y o can be either s ?y *val* or s ?y ?z, depending whether o is a value or a variable. In the former case, its reformulation scheme is reduced to [(4.5).(4.14)*.((4.15) + (4.16)).(4.17)*] + [((4.6) + (4.7) + (4.8)).(4.17)*], while in the latter case its reformulation scheme is that shown in the above table.

As for s ?y *val*,

- reformulating s ?y *val* with (4.5) leads to 1 triple of the form s rdf:type *val*, which can be reformulated at most $2 * \#\texttt{S}$ times by the sequence (4.14)*.((4.15) + (4.16)).(4.17)*, i.e., at most #S times for (4.14)*.(4.15).(4.17)* and for (4.14)*.(4.16).(4.17)*, as rules (4.14)–(4.17) are based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (4.5) is at most: $1 + 2 * \#\texttt{S}$.
- reformulating s ?y *val* with rule (4.6) leads to at most #D triples of the form s **p** *val* (there is at most #D instance-level triples in db). In turn, those triples can be reformulated at most #S times by the sequence (4.17)*, as rule (4.17) is based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (4.6) is at most: $\#\texttt{D} * (1 + \#\texttt{S})$.
- reformulating s ?y *val* with rules (4.7) and (4.8) leads to at most $2 * \#\texttt{S}$ triples of the form s **p** *val* (there is at most #S schema-level triples in db, with at two properties per triple). In turn, those triples can be reformulated at most #S times by the sequence (4.17)*, as rule (4.17) is based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (4.7) or (4.8) is at most: $2 * \#\texttt{S} * (1 + \#\texttt{S})$.

Summing up, the overall number of reformulations of s ?y *val* is at most: $2 * \#\texttt{S}^2 + 5 * \#\texttt{S} + \#\texttt{D} + 1$.

As for s ?y ?z,

- reformulating s ?y ?z with (4.5) leads to 1 triple of the form s rdf:type ?z In turn, that triple can be reformulated using rule (4.9) in at most #D triples of the form s rdf:type **c** (there is at most #D instance-level triples in db). The triple s rdf:type ?z can also be reformulated using the rules (4.10)–(4.13) in at most $2 * \#\mathtt{S}$ triples of the form s rdf:type **c** (there is at most #S schema-level triples in db, with at most two classes per triple). Finally, the triples resulting from rules (4.9)–(4.13) can be reformulated at most $2 * \#\mathtt{S}$ times by the sequence (4.14)*.((4.15)+(4.16)).(4.17)*, i.e., at most #S times for (4.14)*.(4.15).(4.17)* and for (4.14)*.(4.16).(4.17)*, as rules (4.14)–(4.17) are based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (4.5) is at most: $1 + (\#\mathtt{D} + 2 * \#\mathtt{S}) * (1 + 2 * \#\mathtt{S})$.
- reformulating s ?y ?z with rule (4.6) leads to at most #D triples of the form s **p** ?z (there is at most #D instance-level triples in db). In turn, those triples can be reformulated at most #S times by the sequence (4.17)*, as rule (4.17) is based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (4.6) is at most: $\#\mathtt{D} * (1 + \#\mathtt{S})$.
- reformulating s ?y ?z with rule (4.7) or (4.8) leads to at most $2 * \#\mathtt{S}$ triples of the form s **p** ?z (there is at most #S schema-level triples in db, with at two properties per triples). In turn, those triples can be reformulated at most #S times by the sequence (4.17)*, as rule (4.17) is based on schema-level triples (there is at most #S schema-level triples in db). Summing up, the number of reformulations obtained starting from rule (4.7) or (4.8) is at most: $2 * \#\mathtt{S} * (1 + \#\mathtt{S})$.

Summing up, the overall number of reformulations of s ?y ?z is at most: $6 * \#\mathtt{S}^2 + 3 * \#\mathtt{D} * \#\mathtt{S} + 2 * \#\mathtt{D} + 4 * \#\mathtt{S} + 1$.

We therefore get that the worst-case number of reformulations for s ?y o is actually that of s ?y ?z.

By proceeding analogously with the other query triples, we show that the worst-case number of reformulations for a query triple is precisely that of s ?y ?z.

That is, for a query q made of a single triple, the overall upper bound for the size of the output of $\mathtt{Reformulate}(q, \mathtt{db})$ is: $1 + (6 * \#\mathtt{S}^2 + 3 * \#\mathtt{D} * \#\mathtt{S} + 2 * \#\mathtt{D} + 4 * \#\mathtt{S} + 1)$, where the leading 1 accounts for q itself.

The above result easily generalizes to a query of any size $\#q$. Given that $\#\mathtt{db} = \#\mathtt{S} + \#\mathtt{D}$, the size of the output of $\mathtt{Reformulate}(q, \mathtt{db})$ is in $O((6 * \#\mathtt{db}^2)^{\#q})$.

A.6 Proof of Theorem 4.18

Let us first show that $q(\mathtt{db}^\infty) \supseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ holds. We actually show that $\tilde{q}'_{\sigma'}(\mathtt{db}^\infty) \subseteq q(\mathtt{db}^\infty)$ for any $q'_{\sigma'} \in \mathtt{Reformulate}(q, \mathtt{db})$, since $\tilde{q}'_{\sigma'}(\mathtt{db}) \subseteq \tilde{q}'_{\sigma'}(\mathtt{db}^\infty)$ (1. in Property 4.17). The proof is by induction on the length l of a sequence of reformulation rules leading to $q'_{\sigma'}$, starting from $\langle \mathtt{db}, q \rangle$.

Base step:
For $l = 0$, we have $q'_{\sigma'} = q$ and $\tilde{q}'_{\sigma'}(\mathtt{db}^\infty) = q(\mathtt{db}^\infty)$, since q is blank node free (2. in Property 4.17).

Inductive step:
For $l < \alpha$, suppose that $\tilde{q}'_{\sigma'}(\mathtt{db}^\infty) \subseteq q(\mathtt{db}^\infty)$ holds. Now at $l = \alpha$, $q'_{\sigma'}$ has been produced from $q''_{\sigma''}$ by the application a given rule. In turn, $q''_{\sigma''}$ has been produced from q by a sequence of rules starting from $\langle \mathtt{db}, q \rangle$. That sequence being of length $< \alpha$, we get $\tilde{q}''_{\sigma''}(\mathtt{db}^\infty) \subseteq q(\mathtt{db}^\infty)$ by the induction hypothesis. We show that $\tilde{q}'_{\sigma'}(\mathtt{db}^\infty) \subseteq \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$ to prove our claim.
Let $\bar{x}_{\sigma'}$ be the (partially instantiated) output of $q'_{\sigma'}$ and $\bar{x}_{\sigma''}$ be that of $q''_{\sigma''}$.

- Consider the case where $q'_{\sigma'}$ is obtained from $q''_{\sigma''}$ by a rule of the form $\frac{\langle t_1 \in \mathtt{db}, t_2 \in q_\sigma \rangle}{q_{\sigma \cup \nu}}$ that binds a variable of $q''_{\sigma''}$ using an assignment ν. Observe here that $\sigma' = \sigma'' \cup \{\nu\}$ holds. If $(\bar{x}_{\sigma'})_\mu \in \tilde{q}'(\mathtt{db}^\infty)$, then $\mu \cup \{\nu\}$ is a total assignment of the variables of $q''_{\sigma''}$ such that $(\bar{x}_{\sigma'})_\mu = (\bar{x}_{\sigma''})_{\mu \cup \{\nu\}} \in \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$.
- Consider the case where $q'_{\sigma'}$ is obtained from $q''_{\sigma''}$ by a rule of the form $\frac{\langle t_1 \in \mathtt{db}, t_2 \in q_\sigma \rangle}{q_{\sigma[t_2/t_3]}}$ that replaces a triple in $q''_{\sigma''}$ by another one. Observe here that $\sigma' = \sigma''$ holds. If $(\bar{x}_{\sigma'})_\mu \in \tilde{q}'_{\sigma'}(\mathtt{db}^\infty)$, then $(t_{3\sigma'})_\mu \in \mathtt{db}^\infty$ and the immediate entailment rule $t_1, (t_{3\sigma'})_\mu \vdash^i_{\mathrm{RDF}} (t_{2\sigma''})_\mu$ applies. As a result, $(t_{2\sigma''})_\mu \in \mathtt{db}^\infty$ and μ is a total assignment of the variables of $q''_{\sigma''}$ (that may also assign an extra variable generated by the rule leading from $q''_{\sigma''}$ to $q'_{\sigma'}$) such that $(\bar{x}_{\sigma'})_\mu = (\bar{x}_{\sigma''})_\mu \in \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$.

As there is no other form of rule that leads from $q''_{\sigma''}$ to $q'_{\sigma'}$, we get $\tilde{q}'_{\sigma'}(\mathtt{db}^\infty) \subseteq \tilde{q}''_{\sigma''}(\mathtt{db}^\infty)$ which concludes the proof of $q(\mathtt{db}^\infty) \supseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$.

Let us show now that $q(\mathtt{db}^\infty) \subseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ holds.
We actually show that $\tilde{q}_\sigma(\mathtt{db}^\infty) \subseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ holds with q_σ a possibly partially instantiated query, for which $q(\mathtt{db}^\infty) \subseteq \bigcup_{q'_{\sigma'} \in \mathtt{Reformulate}(q,\mathtt{db})} \tilde{q}'_{\sigma'}(\mathtt{db})$ is a special case when q_σ is not partially instantiated ($\sigma = \emptyset$) and does no contain blank nodes ($\tilde{q}(\mathtt{db}^\infty) = q(\mathtt{db}^\infty)$, 2. in Property 4.17).

Provided that q_σ is of the form $q(\bar{x}_\sigma) \coloneq (t_1, \ldots, t_n)_\sigma$, suppose that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}_\sigma(\mathtt{db}^\infty)$ and let us show that there exists $q'_{\sigma'} \in \mathtt{Reformulate}(q, \mathtt{db})$ such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$. We show this by induction on the length l of a (minimal) sequence of immediate entailment rules such that $((t_1, \ldots, t_n)_\sigma)_\mu \subseteq \mathtt{db}^l$: for any l there exists a (possibly empty) sequence of reformulation rules leading to $q'_{\sigma'}$, starting from $\langle \mathtt{db}, q_\sigma \rangle$.

Base step:
For $l = 0$, we have $((t_1, \ldots, t_n)_\sigma)_\mu \subseteq \mathtt{db}^0$, thus $((t_1, \ldots, t_n)_\sigma)_\mu \subseteq \mathtt{db}$, so for the empty sequence of reformulation rules we have $q'_{\sigma'} = q_\sigma$, and $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

Inductive step:
For $l < \alpha$, suppose that the above claim holds. Now at $l = \alpha$, for $1 \leq i \leq n$, $t_{i\sigma}$ matches $(t_{i\sigma})_\mu$: it is either $(t_{i\sigma})_\mu$ or a generalization of $(t_{i\sigma})_\mu$ using one, two, or three (distinct) variables. Moreover, $(t_{i\sigma})_\mu$ has been added to the saturation at $l \leq \alpha$ by an entailment rule of Table 2.3(d). Indeed, the entailment rules of the other Figures do not need to be considered due to Theorem 4.2.

Consider the case of the rule: s rdfs:subClassOf o, s_1 rdf:type $\mathsf{s} \vdash^i_{\text{RDF}} \mathsf{s}_1$ rdf:type o. Assume that $(t_{i\sigma})_\mu = v$ rdf:type $\mathbf{c}_1$, i.e., has been produced from:

$$\{\ \mathbf{c}_2 \text{ rdfs:subClassOf } \mathbf{c}_1,\ v \text{ rdf:type } \mathbf{c}_2\ \} \subseteq \mathsf{db}^{\alpha-1}.$$

Observe that v, c_1, and c_2 can be blank nodes.

- If $t_{i\sigma} = v$ rdf:type $\mathbf{c}_1$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rule (4.14), in which v rdf:type $\mathbf{c}_1$ is replaced by v rdf:type $\mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathsf{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathsf{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathsf{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathsf{db})$.
- If $t_{i\sigma} = ?x$ rdf:type $\mathbf{c}_1$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rule (4.14), in which $?x$ rdf:type $\mathbf{c}_1$ is replaced by $?x$ rdf:type $\mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathsf{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathsf{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathsf{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathsf{db})$.
- If $t_{i\sigma} = v$ rdf:type $?x$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (4.11) then (4.14), in which v rdf:type $?x$ is replaced by $?x$ rdf:type $\mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathsf{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathsf{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathsf{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathsf{db})$.
- If $t_{i\sigma} = v\ ?x\ \mathbf{c}_1$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (4.5) then (4.14), in which $v\ ?x\ \mathbf{c}_1$ is replaced by v rdf:type $\mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathsf{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathsf{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathsf{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathsf{db})$.
- If $t_{i\sigma} = ?x\ ?y\ \mathbf{c}_1$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (4.5) then (4.14), in which $?x\ ?y\ \mathbf{c}_1$ is replaced by $?x$ rdf:type $\mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathsf{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathsf{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathsf{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathsf{db})$.
- If $t_{i\sigma} = ?x$ rdf:type $?y$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (4.11) then (4.14), in which $?x$ rdf:type $?y$ is replaced by $?x$ rdf:type $\mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathsf{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathsf{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathsf{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathsf{db})$.

- If $t_{i\sigma} = v\ ?x\ ?y$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (4.5), then (4.11) then (4.14), in which $v\ ?x\ ?y$ is replaced by v $\mathsf{rdf{:}type}\ \mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.
- If $t_{i\sigma} = ?x\ ?y\ ?z$ then consider the query $q''_{\sigma''}$ obtained from q using the reformulation rules (4.5), then (4.11) then (4.14), in which $?x\ ?y\ ?z$ is replaced by $?x\ \mathsf{rdf{:}type}\ \mathbf{c}_2$. As a result, $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}''(\mathtt{db}^{\alpha-1})$ and, by the induction hypothesis, there exists a sequence of reformulation rules leading to $q'_{\sigma'}$ starting from $\langle \mathtt{db}, q''_{\sigma''} \rangle$, thus from $\langle \mathtt{db}, q_\sigma \rangle$, such that $(\bar{x}_\sigma)_\mu$ is in $\tilde{q}'_{\sigma'}(\mathtt{db})$.

The rest of the proof is omitted here as it amounts to show, similarly as above, that the claim also holds at $l = \alpha$ for the three other entailment rules of Table 2.3(d).

A.7 Proof of Theorem 7.13

Given an analytical query $Q = \langle c(?x, ?d_1, \ldots, ?d_n), m(?x, ?v), \oplus \rangle$ against an analytical schema $\mathcal{S}$, for any analytical schema instance $\mathcal{I}$ (built from $\mathcal{S}$ and an RDF graph $\mathtt{G}$), by Definition 7.9, the answer of Q is

$$ans(Q, \mathcal{I}) = \{\langle d_1^j, \ldots, d_n^j, \oplus(q^j(\mathcal{I}))\rangle \mid \langle x^j, d_1^j, \ldots, d_n^j\rangle \in c(\mathcal{I}) \\ \text{and } q^j \text{ is defined as } q^j(?v) \text{ :- } m(x^j, ?v)\}$$

Let us show that the same answer is obtained through the reformulation of Q against $\mathcal{S}$ (Definition 7.11), i.e., let us show:

$$ans(Q, \mathcal{I}) = \{\ \langle d_1^j, \ldots, d_n^j, \oplus(q^j(\mathtt{G}^\infty))\rangle \mid \langle x^j, d_1^j, \ldots, d_n^j\rangle \in c_{\mathcal{S}}^{\bowtie}(\mathtt{G}^\infty) \\ \text{and } q^j \text{ is defined as } q^j(?v) \text{ :- } m_{\mathcal{S}}^{\bowtie}(x^j, ?v)\ \}$$

This amounts showing that, above, $c(\mathcal{I}) = c_{\mathcal{S}}^{\bowtie}(\mathtt{G}^\infty)$ and $m(\mathcal{I}) = m_{\mathcal{S}}^{\bowtie}(\mathtt{G}^\infty)$. This follows from the Proposition A.1 below (proved in Appendix A.8).

Proposition A.1. *Given an analytical schema $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta\rangle$ and an RDF graph $\mathtt{G}$, for any BGP query q and its corresponding reformulation $q_{\mathcal{S}}^{\bowtie}$ w.r.t. $\mathcal{S}$:*

$$q_{\mathcal{S}}^{\bowtie}(\mathtt{G}^\infty) = q(\mathcal{I}(\mathcal{S}, \mathtt{G})) \text{ holds.}$$

A.8 Proof of Proposition A.1

Proposition A.1 follows by two-way inclusion between $q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty})$ and $q(\mathcal{I}(\mathcal{S},\mathtt{G}))$.

1. We start by proving that $q(\mathcal{I}(\mathcal{S},\mathtt{G})) \subseteq q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty})$.

   Let q be the BGP query $q(\bar{x}) \coloneq t_1, \ldots, t_m$, $m \geq 1$.

   The answer set of q over $\mathcal{I}(\mathcal{S},\mathtt{G})$ is the union of the answer sets of all its different interpretations q_h based on the homomorphisms in $h \in \mathcal{H}$ from q to $\mathcal{S}$ (Definition 7.6). A query q_h is built using the graph homomorphism h by replacing the variables in the query q which correspond to nodes/edges in $\mathcal{S}$ with their corresponding labels. Hence $q_h(\bar{x}) \coloneq t^h_1, \ldots, t^h_m$ has triples of the form $t^h_{i,1\leq i\leq m} \in \{\ \mathtt{s}\ \mathsf{rdf{:}type}\ \lambda(n)\ ,\ \mathtt{s}\ \lambda(e)\ \mathtt{o}\ \}$ where $\mathtt{s},\mathtt{o}$ may be variables.

   $$q(\mathcal{I}(\mathcal{S},\mathtt{G})) = \bigcup_{h\in\mathcal{H}} q_h(\mathcal{I}(\mathcal{S},\mathtt{G})),\ q_h(\bar{x}) \ \coloneq \bigwedge_{i=1}^{m} t^h_i \tag{A.1}$$

   By Definition 7.4, $\mathcal{I}(\mathcal{S},\mathtt{G}) = \bigcup_{n\in\mathcal{N}}\{\mathtt{s}\ \mathsf{rdf{:}type}\ \lambda(n) \mid \mathtt{s} \in q(\mathtt{G}^{\infty}) \wedge q = \delta(n)\} \cup \bigcup_{e\in\mathcal{E}}\{\mathtt{s}\ \lambda(e)\ \mathtt{o} \mid \mathtt{s},\mathtt{o} \in q(\mathtt{G}^{\infty}) \wedge q = \delta(e)\}$.

   It follows that

   - for each triple $t^h_i \in q_{h,\,h\in\mathcal{H}}$ of the form $\mathtt{s}\ \mathsf{rdf{:}type}\ \lambda(n)$, the set of values matching its variables $\mathtt{s}$, denoted $\mathtt{Val}_{t^h_i}(\mathcal{I})$, is included in the set of values matching the node n, denoted $\mathtt{Val}_{q_i}(\mathtt{G}^{\infty}) = \{(\mathtt{s}) \in q_i(\mathtt{G}^{\infty}) \wedge q_i = \delta(n)\}$.
   - for each triple $t^h_i \in q_{h,\,h\in\mathcal{H}}$ of the form $\mathtt{s}\ \lambda(e)\ \mathtt{o}$, the set of values matching its variables $\mathtt{s},\mathtt{o}$, denoted $\mathtt{Val}_{t^h_i}(\mathcal{I})$, is included in the set of values matching the edge e, denoted $\mathtt{Val}_{q_i}(\mathtt{G}^{\infty}) = \{(\mathtt{s},\mathtt{o}) \in q_i(\mathtt{G}^{\infty}) \wedge q_i = \delta(e)\}$.

   We conclude that $\mathtt{Val}_{t^h_i}(\mathcal{I}) \subseteq \mathtt{Val}_{q_i}(\mathtt{G}^{\infty})$ for any $t^h_{i,\,1\leq i\leq m}$ and its corresponding q_i described above. It follows that

   $$q_h(\mathcal{I}) \subseteq q^{\star}_h(\mathtt{G}^{\infty}),\ q^{\star}_h(\bar{x}) \ \coloneq \bigwedge_{i=1}^{m} q_i \tag{A.2}$$

   Examining the definitions of q_i above and Definition 7.11, we conclude that

   $$\bigcup_{h\in\mathcal{H}} q^{\star}_h(\bar{x}) \equiv \bigcup_{h\in\mathcal{H}} q^{\bowtie}_h(\bar{x}) = q^{\bowtie}_{\mathcal{S}}(\bar{x}) \tag{A.3}$$

   Finally, from equations (A.1), (A.2) and (A.3) we have

   $$q(\mathcal{I}(\mathcal{S},\mathtt{G})) = \bigcup_{h\in\mathcal{H}} q_h(\mathcal{I}(\mathcal{S},\mathtt{G})) \subseteq \bigcup_{h\in\mathcal{H}} q^{\bowtie}_h(\mathtt{G}^{\infty}) = q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty})$$

   therefore proving that $q(\mathcal{I}(\mathcal{S},\mathtt{G})) \subseteq q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty})$.

2. We now proceed to showing that $q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty}) \subseteq q(\mathcal{I}(\mathcal{S},\mathtt{G}))$.

From *AnS* reformulation of a query (Definition 7.11) it follows that to answer $q^{\bowtie}_{\mathcal{S}}$ we must answer all $q^{\bowtie}_h, h \in \mathcal{H}$:

$$q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty}) = \bigcup_{h\in\mathcal{H}} q^{\bowtie}_h(\mathtt{G}^{\infty}),\ q^{\bowtie}_h(\bar{x}) \ \coloneq \bigwedge_{i=1}^{m} q_i(\bar{x}_i). \tag{A.4}$$

Considering BGP query answering (Section 2.2.1) in the case of join queries (Definition 2.7) we see that to answer each query $q^{\bowtie}_h, h \in \mathcal{H}$ over the graph $\mathtt{G}^{\infty}$ we must evaluate its component queries over the graph, $q_i(\mathtt{G}^{\infty}), 1 \leq i \leq m$, join their results on the common variables and project out the answer $\bar{x}$. (A.5)

From Definition 7.11 we can also deduce that:

- for $\bar{x}_i = \mathtt{s}$ we have $q_i(\mathtt{G}^{\infty}) = \{\mathtt{s} \in q(\mathtt{G}^{\infty}) \wedge q = \delta(n), h(n_i) = n \in \mathcal{S}\}$; and
- for $\bar{x}_i = \mathtt{s},\mathtt{o}$ we have $q_i(\mathtt{G}^{\infty}) = \{\mathtt{s},\mathtt{o} \in q(\mathtt{G}^{\infty}) \wedge q = \delta(e), h(e_i) = e \in \mathcal{S}\}$;

Considering the *AnS* instance (Definition 7.4), it follows that

- for $\bar{x}_i = \mathtt{s}$ we have $q_i(\mathtt{G}^{\infty}) \subseteq \{\mathtt{s} \mid \mathtt{s}\ \mathsf{rdf{:}type}\ \lambda(n) \in \mathcal{I}(\mathcal{S},\mathtt{G})\}$; and
- for $\bar{x}_i = \mathtt{s},\mathtt{o}$ we have $q_i(\mathtt{G}^{\infty}) \subseteq \{\mathtt{s},\mathtt{o} \mid \mathtt{s}\ \lambda(e)\ \mathtt{o} \in \mathcal{I}(\mathcal{S},\mathtt{G})\}$;

Naming q^t_i the query having as head all the variables in the triple t_i and as body the triple t_i itself (where the triple t_i corresponds to q_i in Definition 7.11), we have that

$$q_i(\mathtt{G}^{\infty}) \subseteq q^t_i(\mathcal{I}(\mathcal{S},\mathtt{G})) \tag{A.6}$$

From (A.5) and (A.6) it follows that

$$q^{\bowtie}_h(\mathtt{G}^{\infty}) \subseteq q^{\star}_h(\mathcal{I}(\mathcal{S},\mathtt{G})),\ q^{\star}_h(\bar{x}) \ \coloneq \bigwedge_{i=1}^{m} q^t_i(\bar{x}_i) \tag{A.7}$$

Since the body of each q^t_i is made of a single triple belonging to the initial query q, we have:

$$q^{\star}_h(\bar{x}) \equiv q_h(\bar{x}),\ q_h(\bar{x}) \ \coloneq \bigwedge_{i=1}^{m} t^h_i \tag{A.8}$$

Therefore we conclude that $q^{\bowtie}_h(\mathtt{G}^{\infty}) \subseteq q_h(\mathcal{I}(\mathcal{S},\mathtt{G}))$ for any $h \in \mathcal{H}$.

It follows that

$$\bigcup_{h\in\mathcal{H}} q^{\bowtie}_h(\mathtt{G}^{\infty}) \subseteq \bigcup_{h\in\mathcal{H}} q_h(\mathcal{I}(\mathcal{S},\mathtt{G}))$$

which together with A.1 and A.4 finally prove that $q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty}) \subseteq q(\mathcal{I}(\mathcal{S},\mathtt{G}))$.

Since $q(\mathcal{I}(\mathcal{S},\mathtt{G})) \subseteq q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty})$ and $q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty}) \subseteq q(\mathcal{I}(\mathcal{S},\mathtt{G}))$, it follows that $q(\mathcal{I}(\mathcal{S},\mathtt{G})) = q^{\bowtie}_{\mathcal{S}}(\mathtt{G}^{\infty})$ proving Proposition A.1.

Appendix B

Queries used in the Experiments of Chapter 5

The queries evaluated over the DBLP [DBL], DBpedia [Lehmann14], Barton [wwwb] and LUBM [Guo05] datasets are respectively shown in Appendix B.1, Appendix B.2, Appendix B.3 and Appendix B.4. The number of queries in the union forming the reformulated query computed by Algorithm `Reformulate` *is shown in parenthesis next to each query.*

The resource namespaces are abbreviate as follows:

elements	http://purl.org/dc/elements/1.1/
foaf	http://xmlns.com/foaf/0.1/
dblp	http://sw.deri.org/ aharth/2004/07/dblp/dblp.owl#
mods	http://simile.mit.edu/2006/01/ontologies/mods3#
language	http://simile.mit.edu/2006/01/language/iso639 − 2b/
role	http://simile.mit.edu/2006/01/role/
info	info:marcorg/
resource	http://dbpedia.org/resource/
ontology	http://dbpedia.org/ontology/
owl	http://www.w3.org/2002/07/owl#
lubm	http://www.lehigh.edu/ zhp2/2004/0401/univ − bench.owl#
:Univ**X**	http://www.University**X**.edu
:Dep**X**.Univ**Y**	http://www.Department**X**.University**Y**.edu

B.1 BGP Queries over the DBLP Dataset

(121) $Q_{01}(?x, ?y, ?z)$:- $?x$ $?y$ $?z$

(684) $Q_{02}(?x, ?y)$:- $?x$ rdf:type dblp:Document ,
$?x$ dblp:datatypeField $?y$

(36) $Q_{03}(?x)$:- $?x$ rdf:type dblp:Document ,
$?x$ elements:publisher “Springer”

(1) $Q_{04}(?x)$:- $?x$ rdf:type dblp:Book ,
$?x$ elements:publisher "Springer"

(1) $Q_{05}(?y, ?z)$:- $?x$ rdf:type foaf:Person ,
$?x$ foaf:name $?y$,
$?x$ foaf:homepage $?z$

(19) $Q_{06}(?y, ?u, ?t)$:- $?x$ dblp:datatypeField "Algorithmica" ,
$?x$ elements:title $?y$,
$?x$ elements:creator $?u$,
$?x$ elements:date $?t$

(4) $Q_{07}(?y, ?u, ?t)$:- $?x$ dblp:objectField "Algorithmica" ,
$?x$ elements:title $?y$,
$?x$ elements:creator $?u$,
$?x$ elements:date $?t$

(19) $Q_{08}(?u, ?z)$:- $?x$ dblp:editor $?y$,
$?y$ foaf:name $?z$,
$?x$ elements:title $?u$,
$?x$ dblp:datatypeField $?v$,
$?x$ elements:publisher "Springer"

(4) $Q_{09}(?u, ?z)$:- $?x$ dblp:editor $?y$,
$?y$ foaf:name $?z$,
$?x$ elements:title $?u$,
$?x$ dblp:objectField $?v$,
$?x$ elements:publisher "Springer"

(138) $Q_{10}(?t, ?n, ?m)$:- $?x$ rdf:type dblp:Document ,
$?x$ dblp:editor $?y$,
$?x$ elements:creator $?z$,
$?y$ foaf:name $?n$,
$?z$ foaf:name $?m$,
$?x$ elements:title $?t$,
$?x$ dblp:objectField $?u$

(36) $Q_{11}(?t, ?y, ?j, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name "Alison Cawsey" ,
$?p$ elements:title $?t$,
$?p$ dblp:year $?y$,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal $?j$,
$?p$ rdf:type dblp:Document

(36) $Q_{12}(?t, ?y, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name "Hugh Darwen" ,
$?p$ elements:title $?t$,
$?p$ dblp:year $?y$,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal "SIGMOD Record" ,
$?p$ rdf:type dblp:Document

(36) $Q_{13}(?t, ?j, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name "Dana Randall" ,
$?p$ elements:title $?t$,
$?p$ dblp:year "2006" ,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal $?j$,
$?p$ rdf:type dblp:Document

(36) $Q_{14}(?n, ?t, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name $?n$,
$?p$ elements:title $?t$,
$?p$ dblp:year "1966" ,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal "Information and Control" ,
$?p$ rdf:type dblp:Document

(1) $Q_{15}(?t, ?y, ?j, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name "Alison Cawsey" ,
$?p$ elements:title $?t$,
$?p$ dblp:year $?y$,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal $?j$,
$?p$ rdf:type dblp:Article

(1) $Q_{16}(?t, ?y, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name "Hugh Darwen" ,
$?p$ elements:title $?t$,
$?p$ dblp:year $?y$,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal "SIGMOD Record" ,
$?p$ rdf:type dblp:Article

(1) $Q_{17}(?t, ?j, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name "Dana Randall" ,
$?p$ elements:title $?t$,
$?p$ dblp:year "2006" ,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal $?j$,
$?p$ rdf:type dblp:Article

(1) $Q_{18}(?n, ?t, ?x)$:- $?p$ elements:creator $?a$,
$?a$ foaf:name $?n$,
$?p$ elements:title $?t$,
$?p$ dblp:year "1966" ,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal "Information and Control" ,
$?p$ rdf:type dblp:Article

(36) $Q_{19}(?n, ?t, ?k, ?p, ?x)$:- $?b$ elements:creator $?a$,
$?a$ foaf:name $?n$,
$?b$ elements:title $?t$,
$?b$ dblp:year "1991" ,
$?b$ dblp:pages $?x$,
$?b$ dblp:Booktitle $?k$,
$?b$ elements:publisher $?p$,
$?b$ rdf:type dblp:Document

(36) $Q_{20}(?t, ?y, ?p, ?x)$:- $?b$ elements:creator $?a$,
$?a$ foaf:name "William J. Frawley" ,
$?b$ elements:title $?t$,
$?b$ dblp:year $?y$,
$?b$ dblp:pages $?x$,
$?b$ dblp:Booktitle "Knowledge Discovery in Databases" ,
$?b$ elements:publisher $?p$,
$?b$ rdf:type dblp:Document

(1) $Q_{21}(?n, ?t, ?k, ?p, ?x)$:- $?b$ elements:creator $?a$,
$?a$ foaf:name $?n$,
$?b$ elements:title $?t$,
$?b$ dblp:year "1991" ,
$?b$ dblp:pages $?x$,
$?b$ dblp:Booktitle $?k$,
$?b$ elements:publisher $?p$,
$?b$ rdf:type dblp:Book

(1) $Q_{22}(?t, ?y, ?p, ?x)$:- $?b$ elements:creator $?a$,
$?a$ foaf:name "William J. Frawley" ,
$?b$ elements:title $?t$,
$?b$ dblp:year $?y$,
$?b$ dblp:pages $?x$,
$?b$ dblp:Booktitle "Knowledge Discovery in Databases" ,
$?b$ elements:publisher $?p$,
$?b$ rdf:type dblp:Book

(36) $Q_{23}(?t, ?y, ?j, ?x)$:- $?p$ elements:creator $?a_1$,
$?a_1$ foaf:name "Grzegorz Rozenberg" ,
$?p$ elements:creator $?a_2$,
$?a_2$ foaf:name "Azriel Rosenfeld" ,
$?p$ elements:title $?t$,
$?p$ dblp:year $?y$,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal $?j$,
$?p$ rdf:type dblp:Document

(1) $Q_{24}(?t, ?y, ?j, ?x)$:- $?p$ elements:creator $?a_1$,
$?a_1$ foaf:name "Grzegorz Rozenberg" ,
$?p$ elements:creator $?a_2$,
$?a_2$ foaf:name "Azriel Rosenfeld" ,
$?p$ elements:title $?t$,
$?p$ dblp:year $?y$,
$?p$ dblp:pages $?x$,
$?p$ dblp:journal $?j$,
$?p$ rdf:type dblp:Article

(36) $Q_{25}(?t, ?y, ?k, ?p, ?x)$:- $?b$ elements:creator $?a_1$,
$?a_1$ foaf:name "Christopher J. Matheus" ,
$?b$ elements:creator $?a_2$,
$?a_2$ foaf:name "William J. Frawley" ,
$?b$ elements:title $?t$,
$?b$ dblp:year $?y$,
$?b$ dblp:pages $?x$,
$?b$ dblp:Booktitle $?k$,
$?b$ elements:publisher $?p$,
$?b$ rdf:type dblp:Document

(1) $Q_{26}(?t, ?y, ?k, ?p, ?x)$:- $?b$ elements:creator $?a_1$,
$?a_1$ foaf:name "Christopher J. Matheus" ,
$?b$ elements:creator $?a_2$,
$?a_2$ foaf:name "William J. Frawley" ,
$?b$ elements:title $?t$,
$?b$ dblp:year $?y$,
$?b$ dblp:pages $?x$,
$?b$ dblp:Booktitle $?k$,
$?b$ elements:publisher $?p$,
$?b$ rdf:type dblp:Book

B.2 BGP Queries over the DBpedia Dataset

(9793) $Q_{01}(?x, ?y)$:- resource:France $?x$ $?y$

(8188) $Q_{02}(?x, ?y)$:- $?x$ rdf:type $?y$

(8188) $Q_{03}(?x)$:- resource:France rdf:type $?x$

(2220) $Q_{04}(?x)$:- $?x$ rdf:type owl:Thing

(463) $Q_{05}(?x)$:- $?x$ rdf:type ontology:Person

(347) $Q_{06}(?x)$:- $?x$ rdf:type ontology:Organisation

(39) $Q_{07}(?x)$:- $?x$ rdf:type ontology:Company

(11) $Q_{08}(?x)$:- $?x$ rdf:type ontology:Animal

(1) $Q_{09}(?x)$:- resource:France ontology:currency $?x$

(9793) $Q_{10}(?x, ?y, ?z)$:- resource:France $?x$ $?y$,
$?y$ foaf:name $?z$

(2229) $Q_{11}(?x, ?y, ?z)$:- $?y$ $?x$ ontology:Place ,
$?y$ foaf:name $?z$

(463) $Q_{12}(?x)$:- $?x$ rdf:type ontology:Person ,
resource:Cubix ontology:starring $?x$

(347) $Q_{13}(?x, ?y)$:- $?x$ foaf:homepage $?y$,
$?x$ rdf:type ontology:Organisation

(39) $Q_{14}(?y, ?x)$:- $?y$ rdf:type ontology:Company ,
$?y$ ontology:headquarter $?x$

(1) $Q_{15}(?y, ?x)$:- $?y$ foaf:name $?x$,
$?y$ foaf:page resource:Trinity

(9793) $Q_{16}(?x, ?y, ?z, ?t)$:- resource:Eurosport $?x$ $?y$,
resource:Eurosport ontology:country $?z$,
resource:Eurosport foaf:homepage $?t$

(463) $Q_{17}(?y, ?z)$:- $?y$ rdf:type ontology:Person ,
resource:Cubix ontology:starring $?y$,
$?y$ ontology:occupation $?z$

(39) $Q_{18}(?x, ?y, ?z)$:- $?x$ rdf:type ontology:Company ,
$?x$ ontology:headquarter $?y$,
$?x$ foaf:homepage $?z$

(1) $Q_{19}(?x, ?y, ?z)$:- resource:Eurosport foaf:name $?x$,
resource:Eurosport ontology:country $?y$,
resource:Eurosport foaf:homepage $?z$

(463) $Q_{20}(?x, ?y, ?z)$:- $?y$ rdf:type ontology:Person ,
$?y$ ontology:nationality $?z$,
$?y$ ontology:occupation resource:Author ,
$?y$ ontology:hometown $?x$

(39) $Q_{21}(?x, ?y, ?z)$:- $?x$ rdf:type ontology:Company ,
$?x$ ontology:headquarter $?y$,
$?x$ ontology:alliance resource:Star_Alliance ,
$?x$ foaf:homepage $?z$

B.3 BGP Queries over the Barton Dataset

(143) $Q_{01}(?x, ?y)$:- $?x$ rdf:type $?y$

(46) $Q_{02}(?x)$:- $?x$ rdf:type mods:Item

(12) $Q_{03}(?x)$:- $?x$ rdf:type mods:Name

(1) $Q_{04}(?x)$:- $?x$ rdf:type mods:Text

(2) $Q_{05}(?x, ?y)$:- $?x$ mods:language $?y$

(2) $Q_{06}(?x, ?y)$:- $?x$ mods:description $?y$

(2) $Q_{07}(?x, ?z)$:- $?x$ rdf:type mods:Text ,
$?x$ mods:language $?z$

(414) $Q_{08}(?y, ?z)$:- $?x$ rdf:type mods:Text ,
$?x$ $?y$ $?z$,
$?x$ mods:language language:fre

(2) $Q_{09}(?x, ?z)$:- $?x$ rdf:type mods:Text ,
$?x$ role:creator $?z$,
$?x$ mods:language language:fre

(143) $Q_{10}(?x, ?y)$:- $?x$ mods:origin info:DLC ,
$?x$ mods:records $?z$,
$?z$ rdf:type $?y$

(46) $Q_{11}(?x, ?z)$:- $?x$ mods:origin info:DLC ,
$?x$ mods:records $?z$,
$?z$ rdf:type mods:Item

(1) $Q_{12}(?x, ?z)$:- $?x$ mods:origin info:DLC ,
$?x$ mods:records $?z$,
$?z$ rdf:type mods:Text

(176) $Q_{13}(?y)$:- $?x$ $?y$ $?z$,
$?x$ mods:records $?t$,
$?t$ rdf:type mods:Text

(1) $Q_{14}(?z)$:- $?x$ mods:created $?z$,
$?x$ mods:records $?t$,
$?t$ rdf:type mods:Text

(143) $Q_{15}(?x, ?y, ?z)$:- $?x$ mods:point "end" ,
$?x$ mods:encoding $?y$,
$?x$ rdf:type $?z$

(46) $Q_{16}(?x, ?y)$:- $?x$ mods:point "end" ,
$?x$ mods:encoding $?y$,
$?x$ rdf:type mods:Item

(9) $Q_{17}(?x, ?y)$:- $?x$ mods:point "end" ,
$?x$ mods:encoding $?y$,
$?x$ rdf:type $mods{:}Date$

B.4 BGP Queries over the LUBM Datasets

(136) $Q_{01}(?x, ?y)$:- $?x$ rdf:type lubm:Employee ,
$?x$ lubm:worksFor :Dep0.Univ0 ,
$?x$ lubm:degreeFrom $?y$

(136) $Q_{02}(?x, ?y, ?u, ?v, ?w)$:- $?x$ rdf:type lubm:Employee ,
$?x$ lubm:worksFor :Dep0.Univ0 ,
$?x$ lubm:degreeFrom $?y$,
$?x$ lubm:name $?u$,
$?x$ lubm:emailAddress $?v$,
$?x$ lubm:telephone $?w$

(34) $Q_{03}(?x, ?y)$:- $?x$ rdf:type lubm:Employee ,
$?x$ lubm:worksFor :Dep0.Univ0 ,
$?x$ lubm:doctoralDegreeFrom $?y$

(3,384) $Q_{04}(?x, ?y, ?z)$:- $?x$ rdf:type $?y$,
$?u$ rdf:type lubm:University ,
$?x$ lubm:doctoralDegreeFrom $?u$,
$?x$ lubm:memberOf $?z$

(130) $Q_{05}(?x, ?y, ?z)$:- $?x$ rdf:type lubm:Student ,
$?y$ rdf:type lubm:Faculty ,
$?z$ rdf:type lubm:Course ,
$?x$ lubm:advisor $?y$,
$?y$ lubm:teacherOf $?z$,
$?x$ lubm:takesCourse $?z$

(10,790) $Q_{06}(?x, ?w, ?y, ?z)$:- $?x$ rdf:type $?w$,
$?y$ rdf:type lubm:Faculty ,
$?z$ rdf:type lubm:Course ,
$?x$ lubm:advisor $?y$,
$?y$ lubm:teacherOf $?z$,
$?x$ lubm:takesCourse $?z$

(156) $Q_{07}(?x, ?y)$:- $?x$ rdf:type lubm:Faculty ,
$?x$ lubm:degreeFrom $?y$,
$?x$ lubm:memberOf $?y$

(123) $Q_{08}(?x)$:- $?x$ rdf:type lubm:Person ,
$?x$ lubm:memberOf :Dep0.Univ0

(123) $Q_{09}(?x)$:- $?x$ rdf:type lubm:Person ,
$?x$ lubm:doctoralDegreeFrom $?y$,
$?x$ lubm:memberOf :Dep0.Univ0

(492) $Q_{10}(?x, ?y)$:- $?x$ rdf:type lubm:Person ,
$?x$ lubm:degreeFrom $?y$,
$?x$ lubm:memberOf :Dep0.Univ0

$(8, 496)$ $Q_{11}(?x, ?y)$:- $?x$ rdf:type lubm:Professor ,
$?y$ rdf:type lubm:Professor ,
$?x$ lubm:degreeFrom $?u$,
$?y$ lubm:degreeFrom $?v$,
$?x$ lubm:memberOf $?v$,
$?y$ lubm:memberOf $?u$

$(1, 296)$ $Q_{12}(?x, ?y)$:- $?x$ rdf:type lubm:Professor ,
$?y$ rdf:type lubm:Lecturer ,
$?x$ lubm:degreeFrom $?u$,
$?y$ lubm:degreeFrom $?u$,
$?x$ lubm:memberOf $?v$,
$?y$ lubm:memberOf $?v$

(221) $Q_{13}(?w, ?x, ?y)$:- $?w$ rdf:type lubm:Lecturer ,
$?x$ rdf:type lubm:GraduateStudent ,
$?y$ rdf:type lubm:Faculty ,
$?w$ lubm:Lecturer $?x$,
$?w$ lubm:Lecturer $?y$

(221) $Q_{14}(?w, ?x, ?y)$:- $?w$ rdf:type lubm:Lecturer ,
$?x$ rdf:type lubm:GraduateStudent ,
$?y$ rdf:type lubm:Faculty ,
$?x$ lubm:advisor $?y$,
$?w$ lubm:Lecturer $?x$,
$?w$ lubm:Lecturer $?y$

$(1, 105)$ $Q_{15}(?w, ?x, ?y)$:- $?w$ rdf:type lubm:Lecturer ,
$?x$ rdf:type lubm:GraduateStudent ,
$?y$ rdf:type lubm:Faculty ,
$?z$ rdf:type lubm:Course ,
$?x$ lubm:advisor $?y$,
$?y$ lubm:teacherOf $?z$,
$?x$ lubm:takesCourse $?z$,
$?w$ lubm:Lecturer $?x$,
$?w$ lubm:Lecturer $?y$

$(35,344)$ $Q_{16}(?u,?v)$:- $?x$ rdf:type $?u$,
$?y$ rdf:type $?v$,
$?x$ lubm:advisor $?y$

(26) $Q_{17}(?z)$:- $?x$ rdf:type lubm:Student ,
$?y$ rdf:type lubm:GraduateStudent ,
$?z$ rdf:type lubm:Faculty ,
$?x$ lubm:advisor $?z$,
$?y$ lubm:advisor $?z$

(376) $Q_{18}(?z,?w)$:- $?x$ rdf:type lubm:Student ,
$?y$ rdf:type lubm:GraduateStudent ,
$?z$ rdf:type $?w$,
$?x$ lubm:advisor $?z$,
$?y$ lubm:advisor $?z$

$(6,696)$ $Q_{19}(?x)$:- $?x$ rdf:type lubm:University ,
$?y$ rdf:type lubm:Article ,
$?u$ rdf:type $?z$,
$?y$ lubm:Lecturer $?u$,
$?u$ lubm:memberOf $?x$

$(28,458)$ $Q_{20}(?x)$:- $?x$ rdf:type lubm:University ,
$?y$ rdf:type lubm:Lecturer ,
$?u$ rdf:type $?z$,
$?y$ lubm:Lecturer $?u$,
$?u$ lubm:memberOf $?x$

(650) $Q_{21}(?z)$:- $?z$ rdf:type lubm:Faculty ,
$?x$ rdf:type lubm:Student ,
$?y$ rdf:type lubm:GraduateStudent ,
$?u$ rdf:type lubm:Course ,
$?v$ rdf:type lubm:Course ,
$?z$ lubm:teacherOf $?u$,
$?z$ lubm:teacherOf $?v$,
$?x$ lubm:takesCourse $?u$,
$?y$ lubm:takesCourse $?v$

(65) $Q_{22}(?x)$:- $?x$ rdf:type lubm:Faculty ,
$?y$ rdf:type lubm:GraduateCourse ,
$?z$ rdf:type lubm:Course ,
$?x$ lubm:teacherOf $?y$,
$?x$ lubm:teacherOf $?z$

(940) $Q_{23}(?x,?w)$:- $?x$ rdf:type $?w$,
$?y$ rdf:type lubm:GraduateCourse ,
$?z$ rdf:type lubm:Course ,
$?x$ lubm:takesCourse $?y$,
$?x$ lubm:takesCourse $?z$

$(35,344)$ $Q_{24}(?x,?y,?z,?w)$:- $?x$ rdf:type $?y$,
$?z$ rdf:type $?w$,
$?z$ lubm:Lecturer $?x$

$(2,444)$ $Q_{25}(?x,?z,?w)$:- $?x$ rdf:type lubm:Faculty ,
$?z$ rdf:type $?w$,
$?z$ lubm:Lecturer $?x$

(697) $Q_{26}(?x,?y)$:- $?x$ rdf:type lubm:Person ,
$?y$ rdf:type lubm:Lecturer ,
$?y$ lubm:Lecturer $?x$

$(2,788)$ $Q_{27}(?x,?y,?z)$:- $?x$ rdf:type lubm:Person ,
$?y$ rdf:type lubm:Lecturer ,
$?x$ lubm:degreeFrom $?z$,
$?y$ lubm:Lecturer $?x$

(697) $Q_{28}(?x,?y,?z)$:- $?x$ rdf:type lubm:Person ,
$?y$ rdf:type lubm:Lecturer ,
$?x$ lubm:doctoralDegreeFrom $?z$,
$?y$ lubm:Lecturer $?x$

(65) $Q_{29}(?x,?y)$:- $?x$ rdf:type lubm:Faculty ,
$?y$ rdf:type lubm:Course ,
$?x$ lubm:doctoralDegreeFrom $?z$,
$?x$ lubm:teacherOf $?y$

$(8,496)$ $Q_{30}(?x,?y)$:- $?x$ rdf:type lubm:Professor ,
$?y$ rdf:type lubm:Professor ,
$?x$ lubm:degreeFrom $?u$,
$?y$ lubm:degreeFrom $?u$,
$?x$ lubm:memberOf $?v$,
$?y$ lubm:memberOf $?v$

(752) $Q_{31}(?x,?y)$:- $?x$ rdf:type $?y$,
$?x$ lubm:degreeFrom :Univ0

(52) $Q_{32}(?x)$:- $?x$ rdf:type lubm:Faculty ,
$?x$ lubm:degreeFrom :Univ0

(156) $Q_{33}(?x,?y)$:- $?x$ rdf:type lubm:Faculty ,
$?x$ lubm:degreeFrom :Univ532 ,
$?x$ lubm:memberOf $?y$

$(2,256)$ $Q_{34}(?x,?y)$:- $?x$ rdf:type $?y$,
$?x$ lubm:degreeFrom :Univ532 ,
$?x$ lubm:memberOf :Dep1.Univ7

(156) $Q_{35}(?x)$:- $?x$ rdf:type lubm:Faculty ,
$?x$ lubm:degreeFrom :Univ532 ,
$?x$ lubm:memberOf :Dep1.Univ7

$(318,096)$ $Q_{36}(?x, ?u, ?y, ?v, ?z)$:- $?x$ rdf:type $?u$,
$?y$ rdf:type $?v$,
$?x$ lubm:mastersDegreeFrom :Univ532 ,
$?y$ lubm:doctoralDegreeFrom :Univ532 ,
$?x$ lubm:memberOf $?z$,
$?y$ lubm:memberOf $?z$

Bibliography

[Abadi07] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, Kate Hollenbach. "Scalable Semantic Web Data Management Using Vertical Partitioning". In *PVLDB*. 2007.

[Abiteboul95] Serge Abiteboul, Richard Hull, Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Abiteboul03] Serge Abiteboul. "Managing an XML Warehouse in a P2P Context". In *CAiSE*, pages 4–13. 2003.

[Abiteboul11] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart. *Web Data Management*. Cambridge University Press, New York, NY, USA, 2011.

[Abiteboul12] Serge Abiteboul, Emilien Antoine, Julia Stoyanovich. "Viewing the Web as a Distributed Knowledge Base". In *ICDE*, pages 1–4. 2012.

[Adjiman07] Philippe Adjiman, François Goasdoué, Marie-Christine Rousset. "SomeRDFS in the Semantic Web". *JODS*, 2007.

[Ailamaki10] Anastasia Ailamaki, Verena Kantere, Debabrata Dash. "Managing Scientific Data". *Commun. ACM*, 53(6):pages 68–78, 2010.

[Arenas09] Marcelo Arenas, Claudio Gutierrez, Jorge Pérez. "Foundations of RDF Databases". In *Reasoning Web*. 2009.

[Arias11] Mario Arias, Javier D. Fernández, Miguel A. Martínez-Prieto, Pablo de la Fuente. "An Empirical Study of Real-World SPARQL Queries". *CoRR*, abs/1103.5043, 2011.

[Atre10] Medha Atre, Vineet Chaoji, Mohammed J. Zaki, James A. Hendler. "Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data". In *WWW*, pages 41–50. 2010.

[Baader03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[Banerjee09] Sandipto Banerjee, Karen C. Davis. "Modeling Data Warehouse Schema Evolution over Extended Hierarchy Semantics". In *J. Data Semantics* [Spaccapietra09], pages 72–96.

[Berners-Lee01] Tim Berners-Lee, James Hendler, Ora Lassila. "The Semantic Web". *Scientific American Magazine*, 2001.

[Berners-Lee08] Tim Berners-Lee. "Linked Open Data". http://www.w3.org/2008/Talks/0617-lod-tbl/, 2008. Talk.

[Bishop11] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, Ruslan Velkov. "OWLIM: A family of scalable semantic repositories". *Semantic Web*, 2(1), 2011.

[Bleco12] Dritan Bleco, Yannis Kotidis. "Business intelligence on complex graph data". In *EDBT/ICDT Workshops*, pages 13–20. 2012.

[Boukraâ13] Doulkifli Boukraâ, Omar Boussaïd, Fadila Bentayeb, Djamel Eddine Zegour. "A Layered Multidimensional Model of Complex Objects". In *CAiSE*, pages 498–513. 2013.

[Broekstra02] Jeen Broekstra, Arjohn Kampman, Frank van Harmelen. "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema". In *International Semantic Web Conference*, pages 54–68. 2002.

[Broekstra03a] Jeen Broekstra, Arjohn Kampman. "Inferencing and Truth Maintenance in RDF Schema: Exploring a naive practical approach". In *PSSS*. 2003.

[Broekstra03b] Jeen Broekstra, Arjohn Kampman. "Inferencing and Truth Maintenance in RDF Schema: Exploring a naive practical approach". In *PSSS Workshop*. 2003.

[Calvanese07] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati. "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family". *Journal of Automated Reasoning (JAR)*, 2007.

[Campinas12] Stephane Campinas, Thomas E. Perry, Diego Ceccarelli, Renaud Delbru, Giovanni Tummarello. "Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation". In *Proceedings of the 2012 23rd International Workshop on Database and Expert Systems Applications*, DEXA '12, pages 261–266. IEEE Computer Society, Washington, DC, USA, 2012.

[Chatziantoniou01] Damianos Chatziantoniou, Michael O. Akinde, Theodore Johnson, Samuel Kim. "The MD-join: An Operator for Complex OLAP". In *ICDE*, pages 524–533. 2001.

[Chebotko09] Artem Chebotko, Shiyong Lu, Farshad Fotouhi. "Semantics Preserving SPARQL-to-SQL Translation". *Data Knowl. Eng.*, 68(10):pages 973–1000, 2009.

[Chong05] Eugene Inseok Chong, Souripriya Das, George Eadon, Jagannathan Srinivasan. "An Efficient SQL-based RDF Querying Scheme". In *VLDB*, pages 1216–1227. 2005.

[Christophides03] Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, Sotirios Tourtounis. "On labeling schemes for the semantic web". In *WWW*, pages 544–555. 2003.

[Codd70] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". *Commun. ACM*, 13(6):pages 377–387, 1970.

[Colazzo13a] Dario Colazzo, I. Ghosh, Tushar, François Goasdoué, Ioana Manolescu, Alexandra Roatis. "WaRG: Warehousing RDF Graphs". In *Bases de Données Avancées*. Nantes, France, 2013. Demonstration.

[Colazzo13b] Dario Colazzo, François Goasdoué, Ioana Manolescu, Alexandra Roatis. "Warehousing RDF Graphs". In *Bases de Données Avancées*. Nantes, France, 2013.

[Colazzo14] Dario Colazzo, François Goasdoué, Ioana Manolescu, Alexandra Roatis. "RDF analytics: lenses over semantic graphs". In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 467–478. 2014.

[DBL] "DBLP RDF dataset". http://kdl.cs.umass.edu/data/dblp/dblp-info.html.

[Dritsou11] Vicky Dritsou, Panos Constantopoulos, Antonios Deligiannakis, Yannis Kotidis. "Optimizing Query Shortcuts in RDF Databases". In *ESWC*, pages 77–92. 2011.

[Duan11a] Songyun Duan, Anastasios Kementsietsidis, Kavitha Srinivas, Octavian Udrea. "Apples and oranges: a comparison of RDF benchmarks and real RDF datasets". In *SIGMOD Conference*, pages 145–156. 2011.

[Duan11b] Songyun Duan, Anastasios Kementsietsidis, Kavitha Srinivas, Octavian Udrea. "Apples and oranges: A comparison of rdf benchmarks and real rdf datasets". In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 145–156. ACM, New York, NY, USA, 2011.

[Erling07] Orri Erling, Ivan Mikhailov. "RDF Support in the Virtuoso DBMS". In *CSSW*, pages 59–68. 2007.

[Erling09] Orri Erling, Ivan Mikhailov. "Virtuoso: RDF Support in a Native RDBMS". In *Semantic Web Information Management*. 2009.

[Erling12] Orri Erling. "Virtuoso, a Hybrid RDBMS/Graph Column Store". *IEEE Data Eng. Bull.*, 2012.

[Etcheverry12] Lorena Etcheverry, Alejandro A. Vaisman. "Enhancing OLAP Analysis with Web Cubes". In *ESWC*, pages 469–483. 2012.

[Fernández13] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, Mario Arias. "Binary RDF representation for publication and exchange (HDT)". *J. Web Sem.*, 19:pages 22–41, 2013.

[Gantz12] John Gantz, David Reinsel. "THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East". http://idcdocserv.com/1414, 2012.

[Giacomo12] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, Domenico Fabio Savo. "MASTRO: A Reasoner for Effective Ontology-Based Data Access". In *ORE*. 2012.

[Goasdoué10] François Goasdoué, Konstantinos Karanasos, Julien Leblay, Ioana Manolescu. "RDFViewS: A Storage Tuning Wizard for RDF Applications". In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1947–1948. ACM, New York, NY, USA, 2010.

[Goasdoué11] François Goasdoué, Konstantinos Karanasos, Julien Leblay, Ioana Manolescu. "View Selection in Semantic Web Databases". *PVLDB*, 2011.

[Goasdoué12a] François Goasdoué, Ioana Manolescu, Alexandra Roatis. "BGP Query Answering against Dynamic RDF Databases". Rapport de recherche RR-8018, INRIA, 2012. URL http://hal.inria.fr/hal-00719641.

[Goasdoué12b] François Goasdoué, Ioana Manolescu, Alexandra Roatis. "Getting more RDF support from relational databases". In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 515–516. 2012.

[Goasdoué12c] François Goasdoué, Ioana Manolescu, Alexandra Roatis. "Répondre aux requêtes par reformulation dans les bases de données RDF". In *Actes de la conférence RFIA 2012*, pages 978–2–9539515–2–3. Lyon, France, 2012. Session "Posters".

[Goasdoué13] François Goasdoué, Ioana Manolescu, Alexandra Roatis. "Efficient query answering against dynamic RDF databases". In *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 299–310. 2013.

[Gottlob11] Georg Gottlob, Giorgio Orsi, Andreas Pieris. "Ontological Queries: Rewriting and Optimization". In *ICDE*. 2011. Keynote.

[Guo05] Yuanbo Guo, Zhengxiang Pan, Jeff Heflin. "LUBM: A benchmark for OWL knowledge base systems". *J. Web Sem.*, 3(2-3):pages 158–182, 2005.

[Gupta99] Ashish Gupta, Iderpal Singh Mumick, editors. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, Cambridge, MA, USA, 1999.

[Gutierrez06] Claudio Gutierrez, Carlos Hurtado, Alejandro Vaisman. "The Meaning of Erasing in RDF under the Katsuno-Mendelzon Approach". In *In Proceedings WebDB-2006*. 2006.

[Gutierrez11] Claudio Gutierrez, Carlos A. Hurtado, Alejandro A. Vaisman. "RDFS Update: From Theory to Practice". In *ESWC*, pages 93–107. 2011.

[Haase04] Peter Haase, Jeen Broekstra, Andreas Eberhart, Raphael Volz. "A Comparison of RDF Query Languages". In *International Semantic Web Conference*, pages 502–517. 2004.

[Halevy01] Alon Y. Halevy. "Answering Queries Using Views: A Survey". *The VLDB Journal*, 10(4):pages 270–294, 2001.

[Harinarayan96] Venky Harinarayan, Anand Rajaraman, Jeffrey D. Ullman. "Implementing Data Cubes Efficiently". In *SIGMOD Conference*, pages 205–216. 1996.

[Heer05] Jeffrey Heer, Stuart K. Card, James A. Landay. "Prefuse: a toolkit for interactive information visualization". In *CHI*, pages 421–430. 2005.

[Huang11] Jiewen Huang, Daniel J. Abadi, Kun Ren. "Scalable SPARQL Querying of Large RDF Graphs". *PVLDB*, 4(11):pages 1123–1134, 2011.

[Husain11] Mohammad Farhan Husain, James P. McGlothlin, Mohammad M. Masud, Latifur R. Khan, Bhavani M. Thuraisingham. "Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing". *IEEE Trans. Knowl. Data Eng.*, 23(9):pages 1312–1327, 2011.

[Imielinski84] Tomasz Imielinski, Witold Lipski Jr. "Incomplete Information in Relational Databases". *JACM*, 1984.

[Inmon92] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[Inmon11] W.H. Inmon, K. Krishnan. *Building the Unstructured Data Warehouse: Architecture, Analysis, and Design*. Technics Publications Llc, 2011.

[Jarke99] Matthias Jarke, Y. Vassiliou, P. Vassiliadis, M. Lenzerini. *Fundamentals of Data Warehouses*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[Jensen10] Christian S. Jensen, Torben Bach Pedersen, Christian Thomsen. *Multidimensional Databases and Data Warehousing*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[Kämpgen13] Benedikt Kämpgen, Andreas Harth. "No Size Fits All - Running the Star Schema Benchmark with SPARQL and RDF Aggregate Views". In *ESWC*, pages 290–304. 2013.

[Kaoudi08] Zoi Kaoudi, Iris Miliaraki, Manolis Koubarakis. "RDFS Reasoning and Query Answering on Top of DHTs". In *International Semantic Web Conference*, pages 499–516. 2008.

[Kaoudi14] Zoi Kaoudi, Ioana Manolescu. "RDF in the Clouds: A Survey". *VLDB journal*, 2014.

[kdb] "[kx] white paper". kx.com/papers/KdbPLUS_Whitepaper-2012-1205.pdf.

[Kimball02] Ralph Kimball, Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.

[Lehmann14] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, Christian Bizer. "DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia". *Semantic Web Journal*, 2014.

[Levandoski09] Justin J. Levandoski, Mohamed F. Mokbel. "RDF Data-Centric Storage". In *ICWS*, pages 911–918. 2009.

[Maduko07] Angela Maduko, Kemafor Anyanwu, Amit P. Sheth, Paul Schliekelman. "Estimating the cardinality of RDF graph patterns". In *WWW*, pages 1233–1234. 2007.

[Maduko08] Angela Maduko, Kemafor Anyanwu, Amit P. Sheth, Paul Schliekelman. "Graph Summaries for Subgraph Frequency Estimation". In *ESWC*, pages 508–523. 2008.

[Matono05] Akiyoshi Matono, Toshiyuki Amagasa, Masatoshi Yoshikawa, Shunsuke Uemura. "A Path-based Relational RDF Database". In *ADC*, pages 95–103. 2005.

[Matono06] Akiyoshi Matono, Said Mirza Pahlevi, Isao Kojima. "RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores". In *DBISP2P*, pages 323–330. 2006.

[Mohan13] C. Mohan. "History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla". In *EDBT*, pages 11–16. 2013.

[Nebot09] Victoria Nebot, Rafael Berlanga Llavori, Juan Manuel Pérez-Martínez, María José Aramburu, Torben Bach Pedersen. "Multidimensional Integrated Ontologies: A Framework for Designing Semantic Data Warehouses". In *J. Data Semantics* [Spaccapietra09], pages 1–36.

[Nebot12] Victoria Nebot, Rafael Berlanga Llavori. "Building data warehouses with semantic web data". *Decision Support Systems*, 52(4):pages 853–868, 2012.

[Neumann08] Thomas Neumann, Gerhard Weikum. "RDF-3X: A RISC-style Engine for RDF". *Proc. VLDB Endow.*, 1(1):pages 647–659, 2008.

[Neumann09] Thomas Neumann, Gerhard Weikum. "Scalable Join Processing on Very Large RDF Graphs". In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 627–640. ACM, New York, NY, USA, 2009.

[Neumann10a] Thomas Neumann, Gerhard Weikum. "The RDF-3X Engine for Scalable Management of RDF Data". *The VLDB Journal*, 19(1):pages 91–113, 2010.

[Neumann10b] Thomas Neumann, Gerhard Weikum. "x-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases". *PVLDB*, 2010.

[Neumann11] Thomas Neumann, Guido Moerkotte. "Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins". In *ICDE*, pages 984–994. 2011.

[Niinimäki09] Marko Niinimäki, Tapio Niemi. "An ETL Process for OLAP Using RDF/OWL Ontologies". In *J. Data Semantics* [Spaccapietra09], pages 97–119.

[OLA] "OLAP Council White Paper". http://www.olapcouncil.org/research/resrchly.htm.

[ora] "Oracle Spatial and Graph white papers". http://www.oracle.com/technetwork/database/options/spatialandgraph/.

[Papastefanatos09] George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, Yannis Vassiliou. "Policy-Regulated Management of ETL Evolution". In *J. Data Semantics* [Spaccapietra09], pages 147–177.

[Pettey11] Christy Pettey. "Gartner Symposium Press Release". http://www.gartner.com/newsroom/id/1824919, 2011.

[Pichler08] Reinhard Pichler, Axel Polleres, Fang Wei, Stefan Woltran. "dRDF: Entailment for Domain-Restricted RDF". In *ESWC*, pages 200–214. 2008.

[Pinet09] François Pinet, Michel Schneider. "A Unified Object Constraint Model for Designing and Implementing Multidimensional Systems". In *J. Data Semantics* [Spaccapietra09], pages 37–71.

[Polleres07] Axel Polleres. "From SPARQL to Rules (and Back)". In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 787–796. ACM, New York, NY, USA, 2007.

[Preda10] Nicoleta Preda, Gjergji Kasneci, Fabian M. Suchanek, Thomas Neumann, Wenjun Yuan, Gerhard Weikum. "Active knowledge: dynamically enriching RDF knowledge bases by web services". In *SIGMOD Conference*, pages 399–410. 2010.

[Ramakrishnan95] Raghu Ramakrishnan, Jeffrey D. Ullman. "A Survey of Deductive Database Systems". *Journal of Logic Programming*, 23(2):pages 125–149, 1995.

[Recommendation04] W3C Recommendation. "RDF Semantics". http://www.w3.org/TR/rdf-mt/, 2004.

[Roatis14] Alexandra Roatis. "Analysing RDF data: A realm of new possibilities". *ERCIM News*, 2014(96), 2014.

[Rosati10] Riccardo Rosati, Alessandro Almatelli. "Improving Query Answering over DL-Lite Ontologies". In *KR*. 2010.

[Russell10] Stuart J. Russell, Peter Norvig. *Artificial Intelligence - A Modern Approach*. Pearson Education, 2010.

[Sidirourgos08] Lefteris Sidirourgos, Romulo Goncalves, Martin Kersten, Niels Nes, Stefan Manegold. "Column-store Support for RDF Data Management: Not All Swans Are White". *Proc. VLDB Endow.*, 1(2):pages 1553–1563, 2008.

[Skoutas09] Dimitrios Skoutas, Alkis Simitsis, Timos K. Sellis. "Ontology-Driven Conceptual Design of ETL Processes Using Graph Transformations". In *J. Data Semantics* [Spaccapietra09], pages 120–146.

[Spaccapietra09] Stefano Spaccapietra, Esteban Zimányi, Il-Yeol Song, editors. *Journal on Data Semantics XIII*, volume 5530 of *Lecture Notes in Computer Science*. Springer, 2009.

[Spyratos06] Nicolas Spyratos. "A Functional Model for Data Analysis". In *FQAS*, pages 51–64. 2006.

[Stuckenschmidt05] Heiner Stuckenschmidt, Jeen Broekstra. "Time - Space Trade-Offs in Scaling up RDF Schema Reasoning". In *WISE Workshops*, pages 172–181. 2005.

[Suchanek08] Fabian M. Suchanek, Gjergji Kasneci, Gerhard Weikum. "YAGO: A Large Ontology from Wikipedia and WordNet". *J. Web Sem.*, 6(3):pages 203–217, 2008.

[terHorst05] Herman J. ter Horst. "Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary". *J. Web Sem.*, 3(2-3):pages 79–115, 2005.

[Theodoratos97] Dimitri Theodoratos, Timos K. Sellis. "Data Warehouse Configuration". In *VLDB*, pages 126–135. 1997.

[Tsialiamanis12] Petros Tsialiamanis, Lefteris Sidirourgos, Irini Fundulaki, Vassilis Christophides, Peter A. Boncz. "Heuristics-based query optimisation for SPARQL". In *EDBT*, pages 324–335. 2012.

[Udrea07] Octavian Udrea, Andrea Pugliese, V. S. Subrahmanian. "GRIN: A Graph Based RDF Index". In *AAAI*, pages 1465–1470. 2007.

[Urbani09] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, Frank van Harmelen. "Scalable Distributed Reasoning Using MapReduce". In *International Semantic Web Conference*, pages 634–649. 2009.

[Urbani10] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, Henri E. Bal. "OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples". In *ESWC*, pages 213–227. 2010.

[Urbani11] Jacopo Urbani, Frank van Harmelen, Stefan Schlobach, Henri E. Bal. "QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases". In *International Semantic Web Conference (1)*, pages 730–745. 2011.

[Urbani12] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, Henri E. Bal. "WebPIE: A Web-scale Parallel Inference Engine using MapReduce". *J. Web Sem.*, 10:pages 59–75, 2012.

[Urbani13] Jacopo Urbani, Alessandro Margara, Ceriel J. H. Jacobs, Frank van Harmelen, Henri E. Bal. "DynamiTE: Parallel Materialization of Dynamic RDF Data". In *ISWC*, pages 657–672. 2013.

[Virgilio12] Roberto De Virgilio. "A Linear Algebra Technique for (de)Centralized Processing of SPARQL Queries". In *ER*, pages 463–476. 2012.

[W3Ca] W3C. "Entailment Regimes". http://www.w3.org/TR/sparql11-entailment/.

[W3Cb] W3C. "Resource Description Framework". http://www.w3.org/RDF.

[W3Cc] W3C. "Simple Knowledge Organization System". http://www.w3.org/TR/skos-primer/.

[W3Cd] W3C. "SPARQL Protocol and RDF Query Language". http://www.w3.org/TR/rdf-sparql-query.

[W3Ce] W3C. "Web Ontology Language". http://www.w3.org/TR/owl2-overview.

[w3cf] "RDF Schema". http://www.w3.org/TR/rdf-schema/.

[W3C13] W3C. "SPARQL 1.1 Query Language". http://www.w3.org/TR/sparql11-query/, 2013.

[W3C14a] W3C. "N-Triples [A line-based syntax for an RDF graph]". http://www.w3.org/TR/n-triples/, 2014.

[W3C14b] W3C. "RDFa [Rich Structured Data Markup for Web Documents]". http://www.w3.org/TR/rdfa-primer/, 2014.

[W3C14c] W3C. "RDF/XML [XML syntax for RDF]". http://www.w3.org/TR/rdf-syntax-grammar/, 2014.

[W3C14d] W3C. "The RDF Data Cube Vocabulary". http://www.w3.org/TR/vocab-data-cube/, 2014.

[W3C14e] W3C. "Turtle [Terse RDF Triple Language]". http://www.w3.org/TR/turtle/, 2014.

[Weaver09] Jesse Weaver, James A. Hendler. "Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples". In *International Semantic Web Conference*, pages 682–697. 2009.

[Weiss08] Cathrin Weiss, Panagiotis Karras, Abraham Bernstein. "Hexastore: Sextuple Indexing for Semantic Web Data Management". *PVLDB*, 1(1), 2008.

[Wilkinson03] Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, Dave Reynolds. "Efficient RDF Storage and Retrieval in Jena2". In *SWDB*, pages 131–150. 2003.

[Win] "WinterCorp White Papers". http://www.wintercorp.com/download-any-wp.

[Wu12] Zhe Wu, Karl Rieb, George Eadon, Ankesh Khandelwal, Vladimir Kolovski. "Advancing the Enterprise-class OWL Inference Engine in Oracle Database". In *ORE*. 2012.

[wwwa] "AllegroGraph". http://franz.com/agraph/allegrograph/.

[wwwb] "Barton". http://simile.mit.edu/rdf-test-data/barton.

[wwwc] "Jena". http://jena.sourceforge.net.

[wwwd] "Neo4j". http://www.neo4j.org/.

[wwwe] "PostgreSQL". http://www.postgresql.org/.

[wwwf] "OWLIM". http://owlim.ontotext.com.

[wwwg] "Sesame". http://www.openrdf.org.

[wwwh] "Virtuoso". http://virtuoso.openlinksw.com.

[Zhao11] Peixiang Zhao, Xiaolei Li, Dong Xin, Jiawei Han. "Graph cube: on warehousing and OLAP multidimensional networks". In *SIGMOD Conference*, pages 853–864. 2011.

[Zou11] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Özsu, Dongyan Zhao. "gStore: Answering SPARQL Queries via Subgraph Matching". *PVLDB*, 4(8):pages 482–493, 2011.