Efficient ontology-based data management

Wafaa El Husseini

To cite this version:

Wafaa El Husseini. Efficient ontology-based data management. Databases [cs.DB]. Université de Rennes, 2023. English. NNT : 2023URENS102 . tel-04586804

HAL Id: tel-04586804
https://theses.hal.science/tel-04586804

Submitted on 24 May 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS, INFORMATIQUE, SIGNAL, SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE Nº 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *INFO : Informatique*

Par
Wafaa EL HUSSEINI

Efficient ontology-based data management

Gestion efficace de données à l'aide d'ontologies expressives

Thèse présentée et soutenue à Lannion, FRANCE, le 22 Décembre 2023
Unité de recherche : IRISA, UMR CNRS 6074 Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)

Rapporteurs avant soutenance :

Nicole BIDOIT-TOLLU	Professeure des Universités émérite, Université Paris-Saclay
Mohand-Saïd HACID	Professeur des Universités, Université Claude Bernard Lyon 1

Composition du Jury :

Président :	David GROSS-AMBLARD	Professeur des Universités, Université de Rennes
Examinateurs :	Nicole BIDOIT-TOLLU	Professeure des Universités émérite, Université Paris-Saclay
	David GROSS-AMBLARD	Professeur des Universités, Université de Rennes
	Mohand-Saïd HACID	Professeur des Universités, Université Claude Bernard Lyon 1
	Ioana MANOLESCU	Directrice de Recherche, INRIA Saclay
	Farouk TOUMANI	Professeur des Universités, Université Clermont-Auvergne
Dir. de thèse :	François GOASDOUÉ	Professeur des Universités, Université de Rennes
Co-dir. de thèse :	Hélène JAUDOIN	Maîtresse de Conférences, Université de Rennes

ACKNOWLEDGEMENT

"Data is a precious thing and will last longer than the systems themselves."

- Sir Tim Berners Lee

My journey of better understanding how to manipulate and reason on data in this PhD wasn't an easy one, and I would like to thank everyone who played a part of it.

Navigating the complexities of data manipulation and reasoning during my PhD was anything but smooth. I'd like to express my sincere thanks to everyone who played a role in this journey.

I would first like to thank my supervisors François Goasdoué and Hélène Jaudoin, whose support during the past years was more than I could have asked for. Thank you for being great role models.

I would like to thank the members of the jury for taking the time to read and evaluate the work done in this thesis.

Thank you to the SHAMAN family. A special thanks to Angélique who was always there to support us, even on short notice.

I would like to express my extreme gratitude towards my parents, Dunia and Adnan, my siblings Karim and Farah, as well as my soul-sister Sarah for believing in me and being my support system. I do not forget all my extended family members who were always there for me through thick and thin.

I would like to thank the friends I made in Lannion, who made this village feel like home. I especially like to mention: Véronne, Arthur, Ziad, Maram, Rahul, Cheikh Brahim, Thomas, Valentine, Hoan My & Adel.

I would like to thank my friends, scattered all around the world who were there for me during this journey. No matter the distance, we are never far apart: Malak, Rida, Zeina, Rasha, Christelle, Soukaynah, Rebbecca, Chadi, Nassim, Khaldun, Ximena...

Finally I would like to give a special thanks to Mrs. Leïla Shahid and M. Georges Morin. I do not forget Dr. Antoine Sabbagh and Dr. Fouad Zablith for their continuous mentorship throughout the years.

I would like to dedicate my work to the cherished memory of my beloved family members and friends who have left an indelible mark on my life and this journey.
My grandmother Jamila, my friend Grace, my uncles.

TABLE OF CONTENTS

List of Figures ix

List of Tables xi

Notations and Abbreviations xiii

Résumé en Français 1

Introduction 9

Context . . . 9

Ontology-mediated query answering . . . 10

Motivations . . . 12

Main Contributions . . . 13

Thesis Outline . . . 15

1 Preliminaries 17

1.1 Knowledge bases . . . 17

1.2 Data management tasks . . . 20

1.2.1 Consistency checking . . . 20

1.2.2 Query Answering . . . 21

1.3 Data management technique . . . 22

1.4 Conclusion . . . 27

2 Optimization framework for FO-Rewriting 29

2.1 Motivations . . . 29

2.2 Problem statement . . . 30

2.3 The Ω optimization function . . . 32

2.3.1 Rationale behind the Ω optimization function . . . 32

2.3.2 Identifying CQs with no answer on a database . . . 33

2.3.3 $(\wedge, \vee)$-CQ evaluation on databases . . . 35

2.4 Database summarization . . . 39

2.5 Conclusion 41

3 Experimental Results 43

3.1 Experimental setup 43
3.2 Database summarization 46
3.3 Query answering performance analysis 47
3.3.1 Queries 47
3.3.2 Performance analysis 49
3.4 Consistency checking performance analysis 60
3.4.1 Adding negative rules to $\mathcal{O}$ 60
3.4.2 Adding inconsistencies to databases 61
3.4.3 Performance analysis 62
3.5 Conclusion 69

4 OptiRef: An optimization Framework and Visualisation tool 71

4.1 Overview 71
4.1.1 Achitecture 71
4.1.2 Technologies 72
4.2 Implementation 73
4.2.1 Creating the datasets 73
4.2.2 The KB-Ref algorithm 74
4.3 OptiRef GUI 75
4.4 Conclusion 84

5 Related work 85

5.1 OMQA via FO-rewriting 85
5.2 Database Summaries 86
5.3 Summary Maintenance 87
5.4 Conclusion 87

Conclusion 89

Bibliography 91

A Appendix 97

A.1 Queries 97

A.1.1 Query Answering Queries 97
A.1.2 Consistency Checking Queries 100
A.2 Query Characteristics 101
A.2.1 UCQ Queries 101
A.2.2 USCQ Queries 102
A.2.3 JUCQ Queries 103
A.3 Execution Time 104
A.3.1 UCQ Queries 104
A.3.2 USCQ Queries 105
A.3.3 JUCQ Queries 106
A.4 Data 107

List of Figures

1 Standard OMQA via FO-rewriting 12
2 Standard and optimized OMQA via FO-rewriting 13

1.1 UCQ representation of the running example 24
1.2 USCQ representation of the running example 25
1.3 JUCQ representation of the running example 26

2.1 Optimizing the UCQ from our running example 38
2.2 Optimizing the USCQ from our running example 38
2.3 Optimizing the JUCQ from our running example 39

3.1 Query answering times (ms, log scale) - LUBM1M 50
3.2 Query answering times (ms, log scale) - LUBM10M 51
3.3 Query answering times (ms, log scale) - LUBM50M 52
3.4 Query answering times (ms, log scale) - LUBM100M 53
3.5 Query answering times (ms, log scale) - LUBM150M 54
3.6 Consistency checking times (ms, log scale) - LUBM1M 63
3.7 Consistency checking times (ms, log scale) - LUBM10M 64
3.8 Consistency checking times (ms, log scale) - LUBM50M 65
3.9 Consistency checking times (ms, log scale) - LUBM100M 66
3.10 Consistency checking times (ms, log scale) - LUBM150M 67

4.1 OptiRef Architecture 72
4.2 OptiRef GUI Settings 75
4.3 OptiRef GUI Bar Chart: QA8 asked on PostgreSQL using LUBM150M . . 76
4.4 UCQ/REF Reformulation Inspection of QA8 78
4.5 UCQ/DB Reformulation Inspection of QA8 79
4.6 UCQ/S Reformulation Inspection of QA8 80
4.7 USCQ/S Reformulation Inspection of QA7 82
4.8 JUCQ/S Reformulation Inspection of QA3 83

A.1 Raw Data in OWL format 107
A.2 Raw Data in DLP format 108

List of Tables

1 Main FO query languages used for FO-rewriting 11
2 Main related works on conjunctive query answering via FO-rewriting . . . 11

1.1 Positive and Negative DL-lite$_{\mathcal{R}}$ rules 19

3.1 Characteristics of our databases, their summaries, and summarization time per RDBMS (in seconds) 46
3.2 Characteristics of QA queries with $\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$ . . 48
3.3 Characteristics of CC queries with $\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$ . . 61

4.1 Size of LUBM databases 73

A.1 List of queries used to evaluate query answering and their description. . . . 97
A.2 List of queries used to evaluate consistency checking and their description. 100
A.3 QA queries characteristics - UCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) . . 101
A.4 CC queries characteristics - UCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) . . 101
A.5 QA queries characteristics - USCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) . 102
A.6 CC queries characteristics - USCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) . 102
A.7 QA queries characteristics - JUCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) . 103
A.8 QA queries performance per RDBMS - UCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) 104
A.9 QA queries performance per RDBMS - USCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) 105
A.10 QA queries performance per RDBMS - JUCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$) 106

Notations and Abbreviations

Notations

Notation	Meaning
$\mathcal{K}$	a knowledge base
$\mathcal{D}$	a database
$\mathcal{O}$	an ontology
$\mathcal{S}$	a summary of a database
q	a query
$q^{\mathcal{O}}$	a reformulation of a query q w.r.t. an ontology $\mathcal{O}$
$q^{\mathcal{K}}$	a query reformulation optimized for a knowledge base $\mathcal{K}$
$\bot$	the FO constant false

Abbreviations

Abbreviation	Meaning
CC	consistency checking
CQ	conjunctive queries
CWA	closed-world assumption
FO	first-order
JUCQ	join of unions of conjunctive queries
KB	knowledge base
OMQA	ontology-mediated query answering
OWA	open-world assumption
QA	query answering
UCQ	union of conjunctive queries
USCQ	union of semi-conjunctive queries

Résumé en Français

Cette thèse fait partie du projet ANR CQFD. Elle a été financée par *Lannion Trégor-Communauté* (LTC) et la *Région Bretagne* (ARED).

La gestion des données en présence d'ontologies, que ce soit pour interroger les données ou réaliser des tests de consistance des données, est un sujet très étudié dans les domaines des bases de données, de l'intelligence artificielle ainsi que celui du web sémantique. Ces tâches de gestion de données en présence d'ontologies consiste à exprimer des requêtes, à la manière de celles qu'on soumet usuellement à des bases de données relationnelles, sur des bases de connaissances, appelées Knowledge Base (KB) en anglais. Une base de connaissances est une théorie de la logique du premier ordre formée d'un ensemble de faits, appelé la base de données, qui modélise les données de l'application sous-jacente, et d'un ensemble d'axiomes appelé l'ontologie qui modélise les connaissances du domaine de l'application. La principale différence entre l'interrogation des bases de connaissances et celle des bases de données classiques réside dans le fait que pour répondre aux requêtes, il est nécessaire de tenir compte des données de la base de connaissances mais aussi de celles qu'on peut déduire à partir des données et de l'ontologie.

Deux techniques majeures pour répondre aux requêtes dans les bases de connaissances émergent de l'état de l'art. Toutes deux réduisent ce problème de calcul des réponses aux requêtes à un simple problème d'interrogation de bases de données. La première technique est celle de la réécriture de requête qui consiste à réécrire la requête exprimée sur la base de connaissances en une requête reformulée qui tient compte des contraintes de l'ontologie. Avec cette méthode, les réponses à une requête sont obtenues en exécutant la reformulation directement sur la base de données originale de la base de connaissances. La deuxième technique repose sur une procédure de matérialisation de la base de connaissances qui consiste à compléter la base de données avec tous les faits qui peuvent être déduits des données à l'aide de l'ontologie de la base de connaissances. Les réponses à une requête en suivant cette méthode sont obtenues en exécutant la requête originale sur la base de données augmentée des faits inférés. La combinaison de la technique de la réécriture avec celle de la matérialisation, appelée approche hybride ou combinée, a également été étudiée. Les deux techniques par réécriture et par matérialisation pour répondre aux requêtes sont

utiles et complémentaires. En effet, bien qu'il existe des cas simples d'interrogation de bases de connaissances dans lesquels elles peuvent être mises en concurrence, il existe des cadres formels plus expressifs de bases de connaissances dans lesquels une seule des techniques peut être utilisée.

Dans cette thèse, on focalise sur le problème de *répondre aux requêtes dans les bases de connaissances par la technique de réécriture*. Cette technique a été largement abordée dans l'état de l'art s'intéressant au problème de gestion de données dans les bases de connaissances, dans les contextes suivants :

— les requêtes sont des requêtes conjonctives (CQs) ;
— les bases de connaissances sont exprimées à l'aide de règles datalog±, de règles existentielles, en logique de description et OWL ou RDF/S.
— les réécritures de requêtes sont exprimées comme des unions de requêtes conjonctives (UCQ), ou comme des unions de requêtes semi-conjonctives (USCQ) ou encore comme des jointures d'unions de requêtes conjonctives (JUCQ).

Plus précisément, la réécriture d'une requête q adressée à une base de connaissances $\mathcal{K}$ consiste `i)` à produire une reformulation $q^{\mathcal{O}}$ de la requête q en utilisant l'ontologie $\mathcal{O}$ de $\mathcal{K}$, puis `ii)` à évaluer $q^{\mathcal{O}}$ sur la base de données $\mathcal{D}$ de $\mathcal{K}$ stockée dans un système de gestion de bases de données relationnelles (SGBDR). Les reformulations des requêtes $q^{\mathcal{O}}$ peuvent être grandes et assez complexes à évaluer en pratique. En effet, la réécriture d'une requête est dépendante de l'ontologie tout en étant indépendante des données. Cela implique que $q^{\mathcal{O}}$ doit s'adapter à toutes les bases de données possibles de $\mathcal{K}$ et ne peut être spécifique à une base de données particulière de $\mathcal{K}$.

Jusqu'à présent, de la même manière que l'optimisation sémantique des requêtes proposée dans les bases de données déductives, l'optimisation des reformulations de requêtes s'est concentrée sur l'étude de représentations des reformulations différentes (minimales ou plus compactes par exemple) mais équivalentes et pouvant être évaluées plus rapidement. Cependant, comme ces reformulations optimisées restent dépendantes de l'ontologie et indépendantes des données, celles-ci demeurent compliquées à évaluer. Étant des variantes syntaxiquement différentes mais sémantiquement équivalentes de la reformulation non optimisée d'une requête, elles sont génériques à toutes les bases de données possibles d'une base de connaissances, et pas uniquement adaptée à la base de données considérée pour l'interrogation.

Contributions

Cette thèse a pour but de contribuer à une gestion efficace de données dans les bases de connaissances. Elle définit un nouveau cadre pour l'optimisation de la gestion des données dans les bases de connaissances selon une approche par réécriture. Ce cadre repose sur les algorithmes de réécriture optimisée de l'état de l'art qui reste dépendants de l'ontologie et indépendants des données. Ces derniers produisent des reformulations génériques qui peuvent s'avérer très complexes à évaluer même dans les SGBDs les plus modernes. L'originalité des travaux de cette thèse est d'optimiser les reformulations, obtenues grâce aux outils de l'état de l'art, en les rendant dépendantes de la base de données de la base de connaissances, tout en s'assurant que les réécritures optimisées fournissent l'ensemble exact des réponses attendues, et ce, de manière efficace en termes de temps d'optimisation et d'évaluation des requêtes.

Préliminaires

Dans cette section, on effectue un rappel des notions de base permettant l'étude du problème de gestion de données décrites par les ontologies et notamment : les bases de connaissances, les tâches de gestion de données et la technique employée pour répondre à ces tâches par réécriture.

On considère des bases de connaissances de la logique du premier ordre, notées $\mathcal{K}$, exprimées à l'aide de règles datalog$\pm$ ou de règles existentielles. Une base de connaissances $\mathcal{K}$ est de la forme $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, où $\mathcal{O}$ est l'ontologie de $\mathcal{K}$, et $\mathcal{D}$ sa base de données. L'ontologie de $\mathcal{K}$ est un ensemble de règles de la forme $\forall\bar{x}(q_1(\bar{x}) \rightarrow q_2(\bar{x}))$, tel que q_1 et q_2 sont des CQs ayant le même ensemble $\bar{x}$ de variables de réponse. Ces règles sont soit positives de la forme $\forall\bar{x}(q_1(\bar{x}) \rightarrow q_2(\bar{x}))$, et elles permettent de dériver des faits impliqués dans une base de connaissance, soit négatives, de la forme $\forall\bar{x}(q_1(\bar{x}) \rightarrow \bot)$, et elles expriment des contraintes d'intégrité qui permettent de dériver des inconsistances dans les bases de connaissances. La base de données $\mathcal{D}$ de $\mathcal{K}$ est un ensemble de faits incomplets. En d'autres termes, ce sont des faits parmi lesquels des variables existentielles sont présentes et représentent des valeurs manquantes ou inconnues. On note qu'une base de connaissances peut être considérée comme une base de données relationnelle déductive et incomplète interprétée selon l'hypothèse du monde ouvert (OWA, pour open-world assumption en anglais).

On s'intéresse aux deux tâches majeures de gestion de données : répondre aux requêtes (QA pour query answering en anglais) et tester la consistance de la base de connaissances (CC pour consistency checking en anglais). Une base de connaissances $\mathcal{K}$ est consistante si et seulement si elle n'implique pas $\bot$ ($\mathcal{K} \not\models \bot$) où $\models$ désigne la relation d'implication de la logique du premier ordre. On remarque que, dans notre cadre théorique, l'inconsistance ne peut résulter que de l'interaction entre la base de données et l'ontologie : une ontologie seule est consistante car les règles négatives ont besoin de faits, éventuellement inférés, pour dériver $\bot$ dans une base de données, et une base de données seule est consistante car elle est constituée de fait positifs uniquement.

On considère des *requêtes de la logique du premier ordre* de la forme $q(\bar{x}) = \phi$, où ϕ est une formule logique dont l'ensemble des variables libres (non quantifiées) est exactement le tuple $\bar{x}$ de variables de réponses. L'arité d'une requête $q(\bar{x})$ est la cardinalité de $\bar{x}$; $q(\bar{x})$ est dite *Booléenne* si $\bar{x} = \emptyset$. Une réponse certaine à une requête $q(\bar{x})$ d'arité n sur une base de connaissances $\mathcal{K}$ est un tuple $\bar{\mathbf{t}}$ de n constantes de $\mathcal{K}$ tel que $\mathcal{K} \models q(\bar{\mathbf{t}})$, où $q(\bar{\mathbf{t}})$ est la requête booléenne obtenue en substituant $\bar{x}$ par $\bar{\mathbf{t}}$ dans q ; quand q est booléenne, $\bar{\mathbf{t}}$ est le tuple vide $\langle\rangle$.

En ce qui concerne la technique employée pour les tâches de gestion des données dans une base de connaissances $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, on considère, comme évoqué précédemment, celle basée sur la réécriture des requêtes en utilisant les règles de $\mathcal{O}$ et qui produit une requête reformulée directement exécutable sur la base $\mathcal{D}$ stockée dans un SGBD. On s'intéresse à l'optimisation des cadres d'interrogation des bases de connaissances $(\mathcal{L}_Q, \mathcal{L}_K)$, où $\mathcal{L}_Q$ définit un langage de requête et $\mathcal{L}_K$ le langage de la base de connaissances, qui ont la propriété d'être réécrivables. Un cadre d'interrogation $(\mathcal{L}_Q, \mathcal{L}_K)$ pour une base de connaissances $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ est dit réécrivable si pour toute requête q exprimée dans le langage $\mathcal{L}_Q$, et pour toute ontologie $\mathcal{O}$ exprimée dans $\mathcal{L}_K$, il existe une reformulation de q par rapport à $\mathcal{O}$, telle que pour toute $\mathcal{K}$ consistante : $ans(q, \mathcal{K}) = eval(q^{\mathcal{O}}, \mathcal{D})$, où $eval(q^{\mathcal{O}}, \mathcal{D})$ est l'évaluation relationnelle de $q^{\mathcal{O}}$ sur $\mathcal{D}$.

Les cadres de gestion de données dans les bases de connaissances bénéficiant de la propriété de réécrivabilité ont été très largement considérés dans l'état de l'art, notamment pour des langages de reformulations comme les UCQ, les USCQ et les JUCQ. Une propriété fondamentale des cadres de gestion de données dans les bases de connaissances présentés ci-dessus, et sur lesquels notre travail repose, est que la reformulation d'une requête $q^{\mathcal{O}}$ est sémantiquement équivalente à la requête q selon $\mathcal{O}$. En particulier, $q^{\mathcal{O}}$ est équivalente, quel que soit le langage utilisé pour l'exprimer, à l'union de toutes les

requêtes conjonctives (CQs) maximalement contenues dans q selon $\mathcal{O}$, soit à l'union de toutes les spécialisations (sous formes de CQs) les plus générales de q selon $\mathcal{O}$.

Un cadre d'optimisation pour la réécriture dans les bases de connaissances

Comme expliqué précédemment, la définition de la réécriture est indépendante des données : une seule reformulation $q^{\mathcal{O}}$ permet de répondre à une requête conjonctive q pour toutes les bases de données $\mathcal{D}$ d'une base de connaissances $\mathcal{K}$ ayant pour ontologie $\mathcal{O}$. La généralité de $q^{\mathcal{O}}$ a pour conséquence de produire des reformulations de requêtes qui sont complexes et difficiles à évaluer par les SGBDs en pratique. On s'intéresse alors au problème de calcul de reformulations de requêtes qui soient dépendantes des données d'une base de connaissances. Ces reformulations perdront de leur généricité au bénéfice d'une meilleure efficacité lors l'exécution des requêtes. Quand une requête q est adressée à une instance de KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ fixée, la base de données $\mathcal{D}$ représente une instance parmi toutes les bases de données possibles à laquelle $q^{\mathcal{O}}$ s'adapte. En particulier, dans les unions des requêtes conjonctives maximalement contenues (et équivalentes) à $q^{\mathcal{O}}$, de nombreuses requêtes conjonctives ne sont pas pertinentes par rapport à $\mathcal{D}$ car elles retournent des ensembles vides de réponse dans $\mathcal{D}$.

On propose un cadre d'optimisation pour la réécritures des requêtes dans les bases de connaissances respectant les propriétés suivantes : la *généralité* de l'approche afin qu'elle soit utilisable dans une majorité des cadres d'interrogation des bases de connaissance bénéficiant de la propriété de réécrivabilité, le maintien de l'*exactitude* de l'ensemble des réponses à une requête et *l'efficacité* de l'obtention des réécritures dépendantes des données et de leur exécution pour améliorer la performance des tâches de gestion de données dans les bases de connaissances considérées.

Notre cadre d'optimisation s'appuie sur une fonction Ω qui transforme une reformulation d'une requête donnée $q^{\mathcal{O}}$, exprimée comme une combinaison de conjonctions de disjonctions (ou inversement) de requêtes conjonctives, notée $(\wedge, \vee)$-CQ, en une reformulation optimisée par rapport à la base de données considérée $\mathcal{D}$. Cette reformulation optimisée est notée par la suite $q^{\mathcal{K}}$ puisqu'elle est spécifique à une base de connaissances $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ et donc à sa base $\mathcal{D}$. Ω va alors effectuer une réécriture dite "bottom-up" pour identifier et enlever les requêtes conjonctives imbriquées dans $q^{\mathcal{O}}$ et sans réponse sur $\mathcal{D}$ puis propager l'effet de leur suppression dans la requête reformulée. La reformulation

de requêtes libérées des sous-requêtes sans réponse sur $\mathcal{D}$ $q^{\mathcal{K}}$ ainsi obtenue, produit une requête contenue dans $q^{\mathcal{O}}$ mais dont l'évaluation fournira le même ensemble de réponses que $q^{\mathcal{O}}$.

L'identification des requêtes conjonctives vides peut s'effectuer facilement (avec l'opérateur EXISTS du langage SQL) en interrogeant directement la base $\mathcal{D}$ stockée dans un SGBD et de façon efficace. Ces systèmes sont en effet bien optimisés pour évaluer les requêtes conjonctives. En revanche, identifier un grand nombre de sous-requêtes conjonctives sans réponse dans les reformulations peut s'avérer coûteux en terme de temps, notamment quand on utilise de grandes bases de données. Pour cela, Ω va s'appuyer sur des résumés des bases de données définies comme des approximations homomorphiques des bases de données qu'ils résument et dont la taille est significativement plus petite que la base elle-même.

Une base de donnée $\mathcal{S}$ est un résumé d'une base de données si et seulement s'il existe un homomorphisme σ de $\mathcal{D}$ vers $\mathcal{S}$ tel que σ associe les constantes de $\mathcal{D}$ avec des constantes de $\mathcal{S}$, et les variables de $\mathcal{D}$ avec des constantes ou des variables de $\mathcal{S}$. On note que $\mathcal{D}_\sigma = \mathcal{S}$ où $\mathcal{D}_\sigma$ est la base de données obtenue à partir de $\mathcal{D}$ en remplaçant les termes (variables ou constantes) dans $\mathcal{D}$ par leurs images dans $\mathcal{S}$ via σ. Concrètement, les résumés que nous utilisons avec notre fonction Ω sont une adaptation de l'opération de quotient de la théorie des graphes au cadre des bases de données relationnelles incomplètes considérée dans ce travail. De par sa définition, le résumé $\mathcal{S}$ garantit qu'une requête sans réponse sur $\mathcal{S}$ est également sans réponse sur $\mathcal{D}$. Cette propriété est fondamentale dans notre approche d'optimisation car elle assure de n'éliminer que les sous-requêtes inutiles d'une reformulation et donc de maintenir un ensemble de réponse égal à celui que fournit $q^{\mathcal{O}}$.

Expérimentations

Pour valider le travail effectué durant cette thèse, nous avons réalisé des expérimentations qui avaient pour but de mesurer le gain de performance apporté par notre cadre d'optimisation. Pour construire nos bases de connaissance, nous avons utilisé un benchmark reconnu de l'état de l'art : *extended LUBM benchmark*. Il s'agit d'une adaptation du benchmark LUBM à la logique de description DL-lite$_{\mathcal{R}}$. Notre intérêt s'est porté sur ce benchmark car il s'agit de celui qui le plus utilisé dans la littérature pour la réécriture de requêtes conjonctives dans les langages de reformulation UCQ, USCQ ou JUCQ. De plus, ceci nous a donné l'opportunité de réutiliser des requêtes disponibles et considérées dans

plusieurs articles de l'état de l'art. Nous avons généré plusieurs bases de données de taille croissante allant de 1 million de tuples jusqu'à 150 millions de tuples. Nous avons utilisé trois SGBDs : DB2 (v11.5.5), MySQL (v8.0.34) et PostgreSQL (v14.2), pour stocker les bases et leurs résumés, et réaliser les expérimentations. Nous avons utilisé et adapté des requêtes de l'état de l'art pour évaluer les performances de l'interrogation des données (*query answering*) et du test de consistance (*consistency checking*) des bases générées. Ces requêtes ont été reformulées dans les trois langages de reformulation UCQ, USCQ et JUCQ, et ont été traitées selon trois stratégies d'interrogation : application de la méthode de l'état de l'art par réécriture ($\mathcal{L}_{\mathbf{R}}$/REF), optimisation de l'état de l'art en utilisant une base de données ($\mathcal{L}_{\mathbf{R}}$/DB), optimisation de l'état de l'art en utilisant un résumé de la base de données ($\mathcal{L}_{\mathbf{R}}$/S).

Les résultats de nos expérimentations sont les suivants : notre cadre d'optimisation améliore la performance des reformulations dans le langage des UCQs jusqu'à trois ordres de grandeur pour les requêtes d'interrogation de la base (QA) et quatre ordres de grandeur pour les requêtes de test de consistance de la base (CC). Notre cadre d'optimisation n'a pas d'effet négatif pour le cas des reformulations dans le langage des USCQs, et il améliore généralement les reformulations dans le langage des JUCQs, et ce jusqu'à un ordre de grandeur.

Au cours de cette thèse, nous avons également développé un prototype, appelé OPTI-REF, qui utilise différents outils de réécriture de requête de l'état de l'art pour calculer des reformulations $q^{\mathcal{O}}$ dans les langages des UCQ, des USCQ et des JUCQ, puis optimise ces reformulations en des requêtes $q^{\mathcal{K}}$ à l'aide de résumés $\mathcal{S}$ ou des bases $\mathcal{D}$ qui sont stockés dans un SGBD et enfin évalue la requête optimisée sur la base $\mathcal{D}$. Cet outil bénéficie également d'une interface graphique pour exprimer des requêtes sur les bases générées pour nos expérimentations et d'un outil de visualisation des reformulations ainsi que des graphiques présentant et comparant les résultats des différentes approches de reformulation et des stratégies d'optimisation des réécritures.

État de l'art, conclusion et perspectives

Durant cette thèse nous avons conçu un nouveau cadre d'optimisation pour la réécriture de requête dans les bases de connaissances. Notre cadre repose sur les travaux d'optimisation de réécriture de requêtes proposés dans l'état de l'art jusqu'à présent. Ces optimisations sont dépendantes de l'ontologie mais indépendantes des données : ils

proposent des reformulations syntaxiquement différentes mais sémantiquement équivalentes. La nouveauté de notre cadre d'optimisation est d'ajouter une étape complémentaire d'optimisation dépendante des données aux outils de reformulations proposés dans l'état de l'art. Notre approche d'optimisation est générique dans le sens où elle s'applique à plusieurs cadres d'interrogation de bases de connaissance pour lesquels la propriété de réécrivabilité est vérifiée, et elle garantit la préservation de l'exactitude de l'ensemble des réponses aux requêtes exprimées dans les bases de connaissances. Pour les cadres de bases de connaissances que nous avons évalués, notre approche d'optimisation améliore considérablement les performances temporelles de l'interrogation des bases pour les reformulations de requêtes dans le langage des UCQs largement adoptées dans l'état de l'art, et pour les JUCQs. Par ailleurs, une originalité de notre approche est qu'elle s'appuie sur la fonction d'optimisation Ω qui réécrit une reformulation de requête en une reformulation plus simple, en supprimant les sous-requêtes inutiles à son évaluation pour une base de données à considérer. Les sous-requêtes inutiles sont rapidement identifiées à l'aide de résumés de bases de données que nous avons conçus en adaptant l'opération de quotient de graphe aux bases de données.

Plusieurs perspectives peuvent être considérées à l'issue de ce travail. Premièrement, en ce qui concerne les résumés, nous avons utilisé des résumés qui sont des approximations homomorphiques des bases de données originales, en adaptant l'opération de quotient aux bases de données relationnelles. Nous avons aussi défini une relation d'équivalence pour construire ces résumés. Une perspective intéressante serait d'étudier des résumés alternatifs à celui que nous avons proposé, et qui pourrait améliorer les performances obtenues dans le cadre la thèse. Une autre perspective serait d'étudier la maintenance de résumés lors de l'ajout ou de la suppression de données dans la base de connaissances. Finalement, nous pourrions également étudier la possibilité d'intégrer la phase d'optimisation des requêtes dès la phase de calcul des reformulations, et non pas après obtention de ces dernières.

INTRODUCTION

The work done in the context of this thesis is part of the ANR project CQFD and was financed by both *Lannion Trégor-Communauté* (LTC) and *Région Bretagne* (ARED).

Context

In an era defined by its abundance of information, the relentless growth of data presents both an opportunity and a challenge. As the volume and complexity of data continues to rapidly expand, the need for effectively and efficiently managing, retrieving and exploiting this asset grows with it. This challenge has lead to the convergence of two essential fields: databases and knowledge representation & reasoning; a pivotal area of study that finds its applications at the very heart of the Semantic Web.

Sir Tim Berners-Lee envisioned the Semantic Web to be an extension to the World Wide Web that aims to make the web content more meaningful, so that both humans and computers can understand it. The main idea is to add semantic meaning to web content. This is achieved through the use of knowledge representation languages, notably the W3C's RDF and OWL standards, which offer a formal and structured knowledge description of Web data that can be interpreted by machines. There is great potential when it comes to the Semantic Web for it to change the way we manage and reason on data which may lead to improving traditional web tasks such as search results, data exchange or decision making. This field is however a work in process and there's still great room for improvement.

An essential challenge in this field is the efficient data management, especially query answering. Many methods are being developed to provide accurate answers, efficiently, to queries over Semantic data. In this thesis we therefore tackle the problem of performing efficient management of data described by domain knowledge, as part of the work done in the ANR CQFD project.

Ontology-mediated query answering

Ontology-Mediated Query Answering (OMQA) [9] is a modern widely-investigated data management problem in Artificial Intelligence, Databases and Semantic Web. It involves asking database-style queries on a *Knowledge Base (KB)*.

Ontology-mediated Query Answering (OMQA)

OMQA consists in answering queries over a knowledge base, in order to improve the completeness of query answering with the use of background domain knowledge.

A KB is a First Order (FO) theory made of a *database*, which is a set of facts that models the application data, and an *ontology*, which is a set of axioms that holds on the database and that models the application's domain knowledge. OMQA has been mainly studied for KBs expressed using *datalog*± [15] and *existential rules* [8], description logics [6] thus OWL [1], or RDF [2]

Performing query answering tasks on a KB involves reasoning on the KB's database using the KB's ontology. The notable difference between this query answering setting and the traditional database one is that OMQA must take into account both the explicit facts stored in the KB's database and the implicit facts that can be deduced from the database using the ontology when computing answers to queries. To tackle this problem, several methods were devised by the literature, which involve embedding the ontological knowledge into either the data, the query or both.

There are currently two main OMQA techniques that are adopted by the literature. Both of them attempt to reduce the problem of query answering on a KB to the standard problem of query evaluation on relational databases.

The first technique is called *FO-rewriting* e.g., [19]. It involves rewriting an incoming query asked on a KB into a so-called query reformulation. Using this method, the answers to the initial query are obtained by evaluating the query reformulation on the KB's database.

1. `https://www.w3.org/OWL`
2. `https://www.w3.org/RDF`

Language	First-order logic syntax	Relational algebra syntax
CQ	$q(\bar{x}) = \exists \bar{y}\ \bigwedge_{i=1}^{n} atom_i$	$q(\bar{x}) = \Pi_{\bar{x}}(\bowtie_{i=1}^{n} atom_i)$
UCQ	$q(\bar{x}) = \bigvee_{i=1}^{n} \mathrm{CQ}_i$	$q(\bar{x}) = \bigcup_{i=1}^{n} \mathrm{CQ}_i$
JUCQ	$q(\bar{x}) = \bigwedge_{i=1}^{n} \mathrm{UCQ}_i$	$q(\bar{x}) = \Pi_{\bar{x}}(\bowtie_{i=1}^{n} \mathrm{UCQ}_i)$
SCQ	$q(\bar{x}) = \exists \bar{y} \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} atom_i^j$	$q(\bar{x}) = \Pi_{\bar{x}}(\bowtie_{i=1}^{n} \bigcup_{j=1}^{m_i} atom_i^j)$
USCQ	$q(\bar{x}) = \bigvee_{i=1}^{n} \mathrm{SCQ}_i$	$q(\bar{x}) = \bigcup_{i=1}^{n} \mathrm{SCQ}_i$

Table 1 – Main FO query languages used for FO-rewriting

	Query reformulation language			
KB language	UCQ	USCQ	JUCQ	datalog^{nr}
datalog±/existential rules	[38, 39, 46]	[60]		[53, 40]
description logics/OWL	[19, 54, 24, 61]		[14]	[55]
RDF/S	[37, 11]		[13]	

Table 2 – Main related works on conjunctive query answering via FO-rewriting

The second technique is called *materialization*, e.g., [1]. It involves completing the KB's database with all the facts that can be deduced from it using the KB's ontology. Using this method, the answers to the initial query are obtained by evaluating the (original) query on the augmented KB's database.

The combination of FO-rewriting and materialization, called the *combined* or *hybrid* approach, has also been investigated, e.g., [47].

Essentially, both FO-rewriting and materialization are useful because, although there exist some simple OMQA settings in which they compete, e.g., [3], there also exist many OMQA settings where only a single technique is applicable, e.g., [8].

In this thesis, we focus on FO-rewriting, which was introduced in [19]. This technique has been largely studied in OMQA settings consisting of:

- *Queries* expressed as conjunctive queries (CQs),
- *KBs* expressed using datalog± and existential rules, description logics and OWL, or RDF with RDFS schema (RDF/S),
- *Query reformulations* expressed as unions of conjunctive queries (UCQs) and non-recursive datalog programs (datalog^{nr}) that unfold to UCQs, unions of semi-conjunctive queries (USCQs), or joins of unions of conjunctive queries (JUCQs).

These languages and their respective syntaxes in both FO logic and relational algebra are recalled in Table 1 above. The main related works in these OMQA settings is given in Table 2 above. We consider all of these OMQA settings in this thesis.

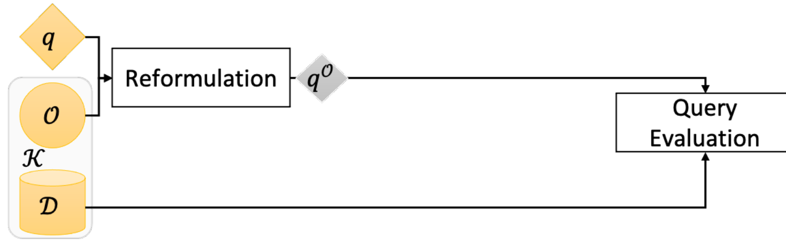


Figure 1 – Standard OMQA via FO-rewriting

Standard OMQA via FO-rewriting is illustrated in Figure 1 above. Given a query q asked on a KB $\mathcal{K}$ composed of an ontology $\mathcal{O}$ and a database $\mathcal{D}$, OMQA using FO-rewriting is performed over two steps:

1. **Reformulation step:** Rewrite q into a *query reformulation* $q^{\mathcal{O}}$ that compiles the axioms from $\mathcal{O}$ into q.
2. **Query evaluation step:** Compute the correct answers of q on $\mathcal{K}$ by evaluating $q^{\mathcal{O}}$ on an RDBMS that stores $\mathcal{D}$.

Motivations

The goal of this thesis is to optimize OMQA via FO-rewriting. The idea on which this thesis builds is that a query reformulation $q^{\mathcal{O}}$ may be large and complex to evaluate, e.g., [60, 14, 40]. We point out that FO-rewriting is indeed both ontology-dependent and data-independent (recall Figure 1), hence the query reformulation $q^{\mathcal{O}}$ must accommodate to all the possible databases, i.e., all the ways databases may store answers to q according to $\mathcal{O}$, and cannot be specific to the particular database $\mathcal{D}$ of $\mathcal{K}$.

So far, to the best of our knowledge, and similarly to semantic query optimization for deductive databases, e.g., [22], query optimization for FO-rewriting has focused on studying equivalent representations of query reformulations that can be evaluated faster: minimal reformulations (e.g., [24, 46]), compact reformulations (e.g., [40, 60]) or cost-based reformulations (e.g., [13, 14]). However, because these optimizations are ontology-dependent and data-independent, optimized query reformulations remain complex to evaluate. They correspond to syntactically different but semantically equivalent variants of non-optimized query reformulations. Consequently, these reformulations still need to accommodate to all the possible databases, and not to just the fixed database at hand.

The research question that this thesis addresses is therefore:

Research Question

Can we optimize FO-rewriting by making it data-dependent?

Main Contributions

The main contribution of this thesis is a novel optimization framework for OMQA via FO-rewriting. It is illustrated in Figure 2 below. The black edges represent the standard FO-rewriting used in the literature: solid edges are maintained within our framework, dashed edges are discarded. The blue edges represent our contribution via the proposed framework.

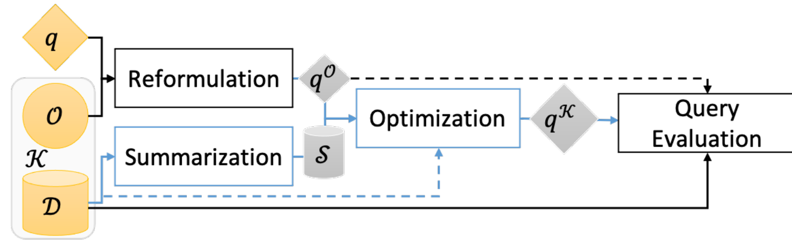


Figure 2 – Standard and optimized OMQA via FO-rewriting

This framework capitalizes on the ontology-dependent and data-independent query optimization for FO-rewriting that have been studied so far in the literature (Reformulation step in Figure 2). Its **originality** is to include complementary data-dependent query optimization for FO-rewriting (Summarization and Optimization steps in Figure 2). Its **purpose** is to optimize the query reformulation $q^{\mathcal{O}}$ produced by any off-the-shelf FO-rewriting algorithm into a query reformulation $q^{\mathcal{K}}$ that is optimized for the particular database $\mathcal{D}$ of $\mathcal{K}$: $q^{\mathcal{K}}$ is simpler than $q^{\mathcal{O}}$, as it just needs to accommodate to $\mathcal{D}$, so that it can be evaluated faster; at the same time it has the same answers as $q^{\mathcal{O}}$ on $\mathcal{D}$ in order to guarantee the correctness of query answering on $\mathcal{K}$. Crucially, $q^{\mathcal{O}}$ is optimized for $\mathcal{D}$ using a summary $\mathcal{S}$ of $\mathcal{D}$, which is a typically small approximation of $\mathcal{D}$. This allows a trade-off between optimization time and the extent to which $q^{\mathcal{K}}$ is optimized for $\mathcal{D}$.

More specifically, in this thesis, we provide the following contributions:

Formalizing the problem

We formalize the problem of data-dependent optimization of a query reformulation using the well-known notion of query containment [1]. To the best of our knowledge, data-dependent optimization of OMQA via FO-rewriting has never been considered in the literature.

The Ω optimization function

We devise an optimization function Ω that rewrites a query reformulation into a simpler contained one, i.e., a simpler more specific one, with the same answers on a fixed database. Containment and query answering correctness are ensured by appropriately removing useless sub-queries from the query reformulation, i.e., sub-queries that do not participate in producing answers on the given database while they may take time to be evaluated.

Database summaries

We define a summary of a database, which is a (typically much smaller) homomorphic database. A summary can be used by our Ω optimization function in place of the original database to perform faster a sound but incomplete identification and removal of useless sub-queries (i.e., some useless sub-queries may not be removed with a summary). Then, we also adapt the quotient operation from graph theory [41] in order to build concrete summaries of databases that are tailored to our needs: both small summaries for fast optimization time and precise summaries to limit the incompleteness of identifying useless sub-queries.

Experimental Evaluation

We experimentally evaluate our optimization framework on the well-established LUBM$^{\exists}$ benchmark for DL-lite$_{\mathcal{R}}$ KBs. DL-lite$_{\mathcal{R}}$ is the description logic that underpins the W3C's OWL2 QL profile for OMQA on large KBs [19]. We show that our optimization framework significantly improves query answering time performance in general (up to 3 orders

of magnitude), as well as consistency checking that turns out to be a simple case of query answering in DL-lite$_{\mathcal{R}}$ KBs (up to 4 orders of magnitude).

OptiRef tool

We discuss the implementation of our optimization framework, which implements the Ω function and relies on database summaries we define. We also propose a tool which allows users to visually examine query results from our experiments over a KB. We use intuitive visualizations to highlight the results of evaluating a query over several DBMS and several sizes as well as to show a breakdown of the query structure in a logical form. This tool allows users to better understand and to assess the benefits of our framework.

Thesis Outline

This thesis is organized as follows:

- Chapter 1 gives a background on OMQA. It introduces the notion of knowledge bases and data management tasks before going over the FO-rewriting technique on which we focus in this thesis.
- Chapter 2 presents the motivations behind our work, before formally defining the problem statement. This chapter then introduces the main contributions of our work: the Ω optimization function and the database summaries.
- Chapter 3 is an overview of our experiments and main results.
- Chapter 4 discusses the implementation of our framework as well as the GUI interface.
- Chapter 5 positions our work w.r.t. the state-of-the-art.

The work described in this thesis was published in the International Semantic Web Conference (ISWC) in 2021 [44], the French Conference on Advanced Databases (BDA) in 2022 [28, 29] and the ACM Web Conference (WWW) in 2023 and 2024 [30, 31].

Chapter 1

Preliminaries

Contents

1.1 Knowledge bases **17**
1.2 Data management tasks **20**
1.2.1 Consistency checking 20
1.2.2 Query Answering 21
1.3 Data management technique **22**
1.4 Conclusion **27**

In this chapter, we delve into the basics of ontology-mediated query answering. First, we define knowledge bases we consider (Section 1.1). Next, we discuss data management tasks (Section 1.2) which are namely consistency checking (Section 1.2.1) as well as query answering (Section 1.2.2). Finally, we focus on the FO-rewriting technique used for OMQA (Section 1.3).

1.1 Knowledge bases

A KB is a formal representation of knowledge in the form of an FO theory. It is composed of a *database*, which is a set of facts that models the application data, and an *ontology*, which is a set of deductive constraints that holds on the database and that models the application domain.

In this thesis, we consider FO KBs expressed using datalog± or existential rules [15, 17, 16, 18, 8], which we simply call rules hereafter. A KB $\mathcal{K}$ is of the form $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, with $\mathcal{O}$ the KB's *ontology* and $\mathcal{D}$ the KB's *database*.

Ontology: An ontology is a finite set of *rules* of the form:

$$\forall\bar{x}(q_1(\bar{x}) \rightarrow q_2(\bar{x})) \text{ or } \forall\bar{x}(q_1(\bar{x}) \rightarrow \bot)$$

In these rules, $\bot$ denotes the FO constant false, and q_1 and q_2 are CQs (recall Table 1, p. 11) with the same set $\bar{x}$ of answer variables. From a modeling point of view, rules of the form $\forall\bar{x}(q_1(\bar{x}) \rightarrow q_2(\bar{x}))$, which are said *positive*, are used to derive entailed facts in the KB, while rules of the form $\forall\bar{x}(q_1(\bar{x}) \rightarrow \bot)$, which are said *negative*, are integrity constraints used to derive inconsistencies in the KB.

Database: A database $\mathcal{D}$ is a finite set of *incomplete facts*, i.e., whose *terms* are constants and existential variables modeling missing or unknown values [45, 1], which we simply call *facts* from now.
The semantics of a KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ is that of the FO formula:

$$(\bigwedge_{\text{rule}\in\mathcal{O}} \text{rule}) \wedge \exists\bar{v}(\bigwedge_{\text{fact}\in\mathcal{D}} \text{fact})$$

where $\bar{v}$ is the set of variables that appear in $\mathcal{D}$.

We remark that a KB can be viewed as a deductive, incomplete relational database interpreted under the so-called open-world assumption (OWA) [1], i.e., under standard FO semantics. We recall that OWA departs from the closed-world assumption (CWA) typically used for relational databases. Under CWA a fact not stored in the database is considered to be false, while under OWA, a fact not stored in the database may be true if it can be deduced, a.k.a. *entailed*, from the stored facts and the constraints that hold on them.

Notation. We use small letters to denote constants, e.g., `f`, `h`, etc., and small italic letters to denote variables, e.g., x, y, etc. Also, we omit quantifiers in rules to simplify the notations since it follows from the FO semantics that existential variables solely appear on the right-hand side of $\rightarrow$ [1].

A note on the running example. While the work presented in this thesis can be applied to many KBs, to keep our running example simple, we choose to illustrate it using DL-lite$_\mathcal{R}$ KBs. DL-lite$_\mathcal{R}$ is the description logic that underpins the *W3C's OWL2 QL standard* for OMQA over large data volumes [63], due to a good balance between expressiveness and computational complexity [19]. In Table 1.1 (p. 19), we use rules to write DL-lite$_\mathcal{R}$ ontology constraints. It is worth noting that standard description logics [6], like DL-lite$_\mathcal{R}$, only use *unary* predicates called *concepts* and *binary* predicates called *roles*.

1. $\forall\bar{x}(\exists\bar{y} \wedge_{i=1}^{m} a_i \rightarrow \exists\bar{z} \wedge_{j=1}^{n} b_j) \Leftrightarrow \forall\bar{x}(\neg(\exists\bar{y} \wedge_{i=1}^{m} a_i) \vee \exists\bar{z} \wedge_{j=1}^{n} b_j) \Leftrightarrow \forall\bar{x}(\forall\bar{y}\neg(\wedge_{i=1}^{m} a_i) \vee \exists\bar{z} \wedge_{j=1}^{n} b_j) \Leftrightarrow \forall\bar{x}\forall\bar{y}(\wedge_{i=1}^{m} a_i \rightarrow \exists\bar{z} \wedge_{j=1}^{n} b_j)$

Positive Rules	**Negative Rules**
$\forall x(A_1(x) \rightarrow A_2(x))$	$\forall x(A_1(x) \wedge A_2(x) \rightarrow \bot)$
$\forall x(A_1(x) \rightarrow \exists y R_1(x,y))$	$\forall x \forall y(A_1(x) \wedge R_1(x,y) \rightarrow \bot)$
$\forall x(A_1(x) \rightarrow \exists y R_1(y,x))$	$\forall x \forall y(A_1(x) \wedge R_1(y,x) \rightarrow \bot)$
$\forall x \forall y(R_1(x,y) \rightarrow A_1(x))$	$\forall x \forall y(R_1(x,y) \wedge A_1(x) \rightarrow \bot)$
$\forall x \forall y(R_1(y,x) \rightarrow A_1(x))$	$\forall x \forall y(R_1(y,x) \wedge A_1(x) \rightarrow \bot)$
$\forall x \forall y(R_1(x,y) \rightarrow \exists z R_2(x,z))$	$\forall x \forall y \forall z(R_1(x,y) \wedge R_2(x,z) \rightarrow \bot)$
$\forall x \forall y(R_1(x,y) \rightarrow \exists z R_2(z,x))$	$\forall x \forall y \forall z(R_1(x,y) \wedge R_2(z,x) \rightarrow \bot)$
$\forall x \forall y(R_1(y,x) \rightarrow \exists z R_2(x,z))$	$\forall x \forall y \forall z(R_1(y,x) \wedge R_2(x,z) \rightarrow \bot)$
$\forall x \forall y(R_1(y,x) \rightarrow \exists z R_2(z,x))$	$\forall x \forall y \forall z(R_1(y,x) \wedge R_2(z,x) \rightarrow \bot)$
$\forall x \forall y(R_1(x,y) \rightarrow R_2(y,x))$	$\forall x \forall y(R_1(x,y) \wedge R_2(y,x) \rightarrow \bot)$
$\forall x \forall y(R_1(x,y) \rightarrow R_2(x,y))$	$\forall x \forall y(R_1(x,y) \wedge R_2(x,y) \rightarrow \bot)$

Table 1.1 – Positive and Negative DL-lite$_{\mathcal{R}}$ rules

Example 1 (Running example). *Let us consider the DL-lite$_{\mathcal{R}}$ KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, where:*

$$\begin{aligned} \mathcal{O} = \{ & r_1 = ww(x,y) \rightarrow ww(y,x), \\ & r_2 = sup(x,y) \rightarrow ww(x,y), \\ & r_3 = PhD(x) \rightarrow sup(y,x), \\ & r_4 = sup(x,y) \wedge sup(y,z) \rightarrow \bot \} \end{aligned}$$

and

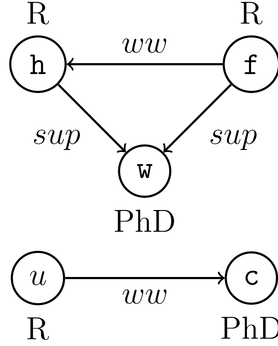
$$\mathcal{D} = \{R(\mathtt{f}), R(\mathtt{h}), sup(\mathtt{f},\mathtt{w}), sup(\mathtt{h},\mathtt{w}), PhD(\mathtt{w}), ww(\mathtt{f},\mathtt{h}), R(u), ww(u,\mathtt{c}), PhD(\mathtt{c})\}.$$

The ontology $\mathcal{O}$ is made of four rules which state the following:
- r_1 : working with (ww) someone is a symmetric relation,
- r_2 : supervising someone (sup) is a particular case of working with someone,
- r_3 : PhD students (PhD) are necessarily supervised by someone,
- r_4 : you cannot supervise and be supervised simultaneously.
Meanwhile, the database stores nine facts which state the following:

- *François* (`f`) *and Hélène* (`h`) *are researchers (R),*
- *u is an unknown researcher (R),*
- *Wafaa* (`w`) *and Cyrielle* (`c`) *are PhD students (PhD),*
- *François* (`f`) *and Hélène* (`h`) *supervise (sup) Wafaa* (`w`),
- *François* (`f`) *works with (ww) Hélène* (`h`),
- *u works with (ww) Cyrielle* (`c`).

For reading convenience, we also represent a database as a node-labeled and edge-labeled graph: nodes are the terms of the database, a node n has label L iff $L(n)$ is a fact in the database, there exists an edge with label L from n_1 to n_2 iff $L(n_1, n_2)$ is a fact in the database.

The graph representation of the above database $\mathcal{D}$ is:



◇

1.2 Data management tasks

We consider the two main data management tasks on KBs: *consistency checking* and *query answering*.

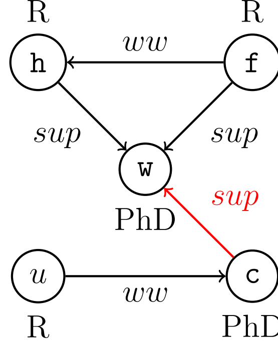
1.2.1 Consistency checking

A KB $\mathcal{K}$ is *consistent* if and only if it does not entail $\bot$, i.e., $\mathcal{K} \not\models \bot$ where $\models$ is the FO entailment relation.

We remark that, in our setting, inconsistency may only result from interactions between both the ontology and the database. Clearly, a database *alone* is consistent because it is a conjunction of *positive* facts, and an ontology *alone* is consistent because *negative* rules need (entailed) facts to derive $\bot$.

Example 2 (Cont.)**.** *$\mathcal{K}$ is consistent because $\bot$ is not derivable from the only negative rule r_4 and the (entailed) facts for the sup relation: $sup(\texttt{f}, \texttt{w})$ and $sup(\texttt{h}, \texttt{w})$ in $\mathcal{D}$ plus $\exists y_1\, sup(y_1, \texttt{w})$ and $\exists y_2\, sup(y_2, \texttt{c})$ that respectively follow from $PhD(\texttt{w})$ and r_3 and from $PhD(\texttt{c})$ and r_3.*

Let us consider the KB $\mathcal{K}_\bot = (\mathcal{O}, \mathcal{D}_\bot)$ where $\mathcal{D}_\bot$ is obtained by adding $sup(\texttt{c}, \texttt{w})$ to $\mathcal{D}$. Its graph representation is:



$\mathcal{K}_\perp$ is inconsistent *because r_4 derives $\perp$ from $sup(\mathtt{c}, \mathtt{w})$ in $\mathcal{D}_\perp$ together with the entailed fact $\exists y_2\, sup(y_2, \mathtt{c})$ that is derived from $PhD(\mathtt{c})$ and r_3.* $\diamond$

1.2.2 Query Answering

We consider *FO queries* of the form $q(\bar{x}) = \phi$, where q is the *query name*, $\bar{x}$ is the *tuple of answer variables* of q, and ϕ is the FO formula modeling what q is asking for. The set of the free (a.k.a. non-quantified or unbound) variables of ϕ exactly fits with the answer variables. The arity of a query $q(\bar{x})$ is the cardinality of $\bar{x}$. $q(\bar{x})$ is said *Boolean* if the set of answer variables is empty, and then $\bar{x}$ is the empty tuple $\langle\rangle$.

A *certain answer to a query* $q(\bar{x})$ of arity n on a KB $\mathcal{K}$ is a tuple $\bar{\mathbf{t}}$ of n constants from $\mathcal{K}$ such that $\mathcal{K} \models q(\bar{\mathbf{t}})$, where $q(\bar{\mathbf{t}})$ is the Boolean query obtained by instantiating $\bar{x}$ with $\bar{\mathbf{t}}$ in q; when q is Boolean, $\bar{\mathbf{t}} = \langle\rangle$. From now, we denote by $ans(q, \mathcal{K})$ the *answer set* of q on $\mathcal{K}$ and we remark that if q is Boolean then the answer is *true* when $ans(q, \mathcal{K}) = \{\langle\rangle\}$ and the answer is $false$ when $ans(q, \mathcal{K}) = \emptyset$.

Unless otherwise specified, we consider queries on KBs expressed in the FO query language of *conjunctive queries* (CQ for short), a.k.a. select-project-join queries. CQ is the most commonly-adopted language for querying KBs in the OMQA literature. A CQ is a conjunction of atoms and each variable in the CQ is either free or existentially quantified. The FO and relational algebra syntax of CQs is shown in Table 1 (p. 11).

Example 3 (Cont.)**.** *Let us consider the CQ asking for the supervisees who work with* $\mathtt{h}$ *that must be a researcher: $q(x) = \exists y\, R(\mathtt{h}) \wedge ww(\mathtt{h}, x) \wedge sup(y, x)$.*

Its answer set on $\mathcal{K}$ is $ans(q, \mathcal{K}) = \{\mathtt{w}\}$: $\mathtt{w}$ is obtained from $R(\mathtt{h}) \in \mathcal{D}$, $sup(\mathtt{f}, \mathtt{w}) \in \mathcal{D}$ or $sup(\mathtt{h}, \mathtt{w}) \in \mathcal{D}$, and the fact $ww(\mathtt{h}, \mathtt{w})$ entailed from $sup(\mathtt{h}, \mathtt{w}) \in \mathcal{D}$ and r_2.

Its answer set on $\mathcal{K}_\perp$ is $ans(q, \mathcal{K}) = \{\mathtt{f}, \mathtt{h}, \mathtt{w}, \mathtt{c}\}$, since $\mathcal{K}_\perp$ is inconsistent and therefore entails q for every tuple made of a single constant. We point out here that asking queries on an inconsistent KB is of little practical interest. ◇

1.3 Data management technique

We focus on optimizing OMQA via *FO-rewriting* [19]. As we shall see soon, this technique can also be used for consistency checking, which is a particular case of query answering in our KB setting.

FO-rewriting reduces query answering on KBs to query evaluation over relational database in *FO-rewritable OMQA settings.* An OMQA setting is a pair $(\mathcal{L}_Q, \mathcal{L}_K)$ of query and KB languages. Such a setting is FO-rewritable if for any $\mathcal{L}_Q$ query q and any $\mathcal{L}_K$ ontology $\mathcal{O}$, there exists an FO query $q^\mathcal{O}$, called a *reformulation of* q *w.r.t.* $\mathcal{O}$, such that for any consistent KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$: $ans(q, \mathcal{K}) = eval(q^\mathcal{O}, \mathcal{D})$, where $eval(q^\mathcal{O}, \mathcal{D})$ the relational evaluation of $q^\mathcal{O}$ on $\mathcal{D}$.

We remark that we only consider the practically relevant case of consistent KBs for query answering, as discussed in the preceding example. Furthermore, each FO-rewriting algorithm computes query reformulations in a fixed FO query dialect. Recall Table 2 (p. 11), for instance, where CQs are reformulated into UCQs, USCQs, JUCQs or datalog^{nr} programs. We therefore term *FO-rewriting setting* a triple of query language $\mathcal{L}_Q$, KB language $\mathcal{L}_K$ and query reformulation language $\mathcal{L}_R$, denoted by $(\mathcal{L}_Q, \mathcal{L}_K, \mathcal{L}_R)$, such that $(\mathcal{L}_Q, \mathcal{L}_K)$ is an FO-rewritable OMQA setting for which query reformulations are expressed in $\mathcal{L}_R$.

Moving forward, we focus on FO-rewriting settings with queries expressed in the language of CQs and query reformulations expressed in the languages of UCQs, USCQs and JUCQs. These setting are widely considered in the literature on FO-rewriting (e.g., Table 2, p. 11). We remark that datalog^{nr} reformulations must be unfolded into UCQs reformulations, which we consider, in order to be evaluated by RDBMSs.

A key property of the FO-rewriting settings that we consider, on which our work relies, is that a query reformulation $q^\mathcal{O}$ is equivalent to the CQ q w.r.t. the ontology $\mathcal{O}$. In particular, $q^\mathcal{O}$ is equivalent, regardless of the language used to express it, to the union

of all the *CQs that are maximally-contained in q w.r.t. $\mathcal{O}$*, i.e., the union of all the most general CQ specializations of q w.r.t. $\mathcal{O}$. In practice, these maximally-contained queries can also be seen as all the ways databases may store answers to q according to $\mathcal{O}$.

We recall that:

- A query q' is *contained in a query q*, noted $q' \subseteq q$, if and only if for each database $\mathcal{D}$, $eval(q', \mathcal{D}) \subseteq eval(q, \mathcal{D})$.
- A query q' is *contained in a query q w.r.t. an ontology $\mathcal{O}$*, noted $q' \subseteq_{\mathcal{O}} q$ if and only if for each KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, $ans(q', \mathcal{K}) \subseteq ans(q, \mathcal{K})$.
- A query q' is *maximally-contained in a query q w.r.t. an ontology $\mathcal{O}$* if and only if (i) $q' \subseteq_{\mathcal{O}} q$ and (ii) for any other query $q'' \subseteq_{\mathcal{O}} q$, if $q' \subseteq q''$ then $q'' \subseteq q'$ (i.e., q' and q'' are equivalent).

Notation. We omit the existential quantifications of variables to simplify the notation of queries: non-answer variables are existentially quantified in the query languages we consider (recall Table 1). For instance, the CQ of Example 3 is now written $q(x) = R(\mathtt{h}) \wedge ww(\mathtt{h}, x) \wedge sup(y, x)$.

Example 4 (Cont.). *The following query q^{UCQ} is the UCQ reformulation of q w.r.t. $\mathcal{O}$ computed by the Rapid tool [24]: it is the union of all the CQs that are maximally-contained in q w.r.t. $\mathcal{O}$.*

$$
\begin{aligned}
q^{\text{UCQ}}(x) = & (R(\mathtt{h}) \wedge ww(\mathtt{h}, x) \wedge sup(y, x)) && (1)\\
& \vee (R(\mathtt{h}) \wedge ww(\mathtt{h}, x) \wedge PhD(x)) && (2)\\
& \vee (R(\mathtt{h}) \wedge sup(\mathtt{h}, x)) && (3)\\
& \vee (R(\mathtt{h}) \wedge ww(x, \mathtt{h}) \wedge sup(y, x)) && (4)\\
& \vee (R(\mathtt{h}) \wedge ww(x, \mathtt{h}) \wedge PhD(x)) && (5)\\
& \vee (R(\mathtt{h}) \wedge sup(x, \mathtt{h}) \wedge sup(y, x)) && (6)\\
& \vee (R(\mathtt{h}) \wedge sup(x, \mathtt{h}) \wedge PhD(x)) && (7)
\end{aligned}
$$

We remark that, within q^{UCQ}, (1) *is q itself as q is trivially contained in itself w.r.t. $\mathcal{O}$ and* (2) *to* (7) *are CQs maximally-contained in q w.r.t. $\mathcal{O}$ that are obtained by specializing q through one or several specialization steps. Each step uses either a positive rule in $\mathcal{O}$ in a backward fashion or performs the unification of atoms. In our example:*

(2) is obtained from (1) *in one step by specializing $sup(y, x)$ into $PhD(x)$ using r_3;*

(3) is obtained from (1) *in two steps by first specializing $ww(\mathtt{h}, x)$ into $sup(\mathtt{h}, x)$ using r_2 and then by unifying $sup(\mathtt{h}, x)$ and $sup(y, x)$ into $sup(\mathtt{h}, x)$; the intermediate CQ*

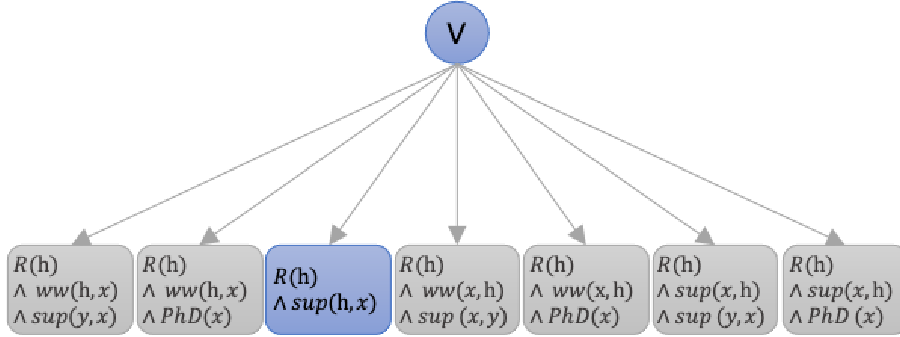


Figure 1.1 – UCQ representation of the running example

$R(\mathtt{h}) \wedge sup(\mathtt{h}, x) \wedge sup(y, x)$ *does not appear in* q^{UCQ} *because it is simply contained in* (3)*, thus redundant with it (it is actually equivalent to it);*

(4) is obtained from (1) *using* r_1*;*

(5) can be obtained either from (2) *using* r_1 *or from* (4) *using* r_3*;*

(6) is obtained from (4) *using* r_2*;*

(7) can be obtained either from (4) *using* r_2 *or from* (6) *using* r_3*.*

The answer to q *on* $\mathcal{K}$ *(i.e.,* `w`*) results only from* (3)*; this CQ is shown in blue. Figure 1.1 shows a graphical representation of this UCQ where nodes colored in gray represent the CQs with no answer on* $\mathcal{D}$ *and the blue node represents the CQ* (3) *having the answer* `w` *on* $\mathcal{D}$*.*

The following query q^{USCQ} *is the USCQ reformulation of* q *w.r.t.* $\mathcal{O}$ *computed by the Compact tool [60]; it is actually just an SCQ in this simple running example.*

$$\begin{aligned} q^{\mathrm{USCQ}}(x) = &(R(\mathtt{h})) \\ &\wedge (ww(\mathtt{h}, x) \vee sup(\mathtt{h}, x) \vee ww(x, \mathtt{h}) \vee sup(x, \mathtt{h})) \\ &\wedge (sup(y, x) \vee PhD(x)) \end{aligned}$$

USCQ query reformulations have been introduced to limit, to the extent possible, the repetition of the same atoms *across the CQs of UCQ reformulations. For instance, in* q^{UCQ}*,* $R(\mathtt{h})$ *occurs 7 times,* $sup(y, x)$ *and* $PhD(x)$ *occur 3 times each, etc. By contrast, in* q^{USCQ}*, each atom occurs only once. We note that the CQ* (3) *in* q^{UCQ} *is equivalently encoded here into the three pieces shown in blue in* q^{USCQ}*:* $R(\mathtt{h}) \wedge sup(\mathtt{h}, x)$ *in* q^{UCQ} *is equivalent to* $R(\mathtt{h}) \wedge sup(\mathtt{h}, x) \wedge sup(y, x)$ *where* $sup(y, x)$ *is superfluous, which is encoded in* q^{USCQ} *up to the distribution of the* $\wedge$*'s over the* $\vee$*'s.*

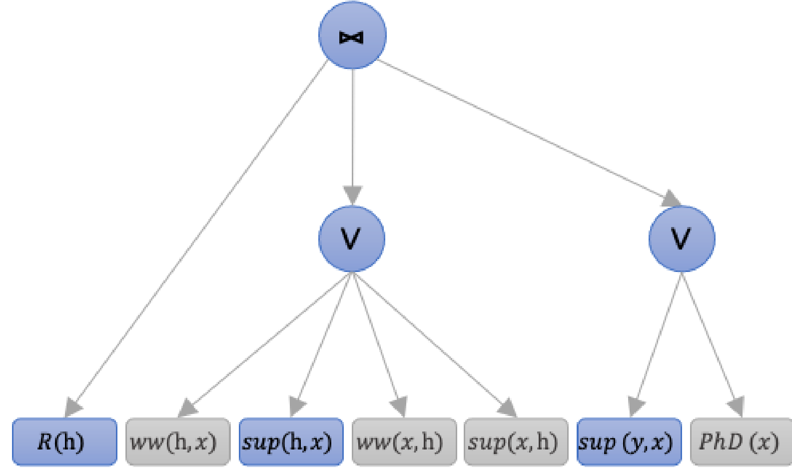


Figure 1.2 – USCQ representation of the running example

Figure 1.2 is a graphical representation of this USCQ where nodes colored in gray represent atoms with no answer on $\mathcal{D}$ and the blue nodes represent atoms (3) with an answer on $\mathcal{D}$.

The following query q^{JUCQ} is a JUCQ reformulation of q w.r.t. $\mathcal{O}$ computed by the GDL tool [14].

$$
\begin{aligned}
q^{\text{JUCQ}}(x) = (R(\mathtt{h})) \wedge (& (ww(\mathtt{h}, x) \wedge sup(y, x)) \\
& \vee (ww(\mathtt{h}, x) \wedge PhD(x)) \\
& \vee (sup(\mathtt{h}, x)) \\
& \vee (ww(x, \mathtt{h}) \wedge sup(y, x)) \\
& \vee (ww(x, \mathtt{h}) \wedge PhD(x)) \\
& \vee (sup(x, \mathtt{h}) \wedge sup(y, x)) \\
& \vee (sup(x, \mathtt{h}) \wedge PhD(x)))
\end{aligned}
$$

JUCQ reformulations have been introduced to limit the repetitions of sub-CQs *across the CQs of UCQ reformulations. A JUCQ reformulation of a query is obtained by:*

(i) Choosing a set of possibly overlapping sub-CQs that covers the whole query

(ii) Joining their UCQ reformulations

Note that not all query covers can be chosen, we refer to [14] for details.

In contrast to UCQ and USCQ reformulations, there exists more than just a single JUCQ reformulation for a given query (up to redundancy): the different sets of sub-CQs that cover the query lead to different JUCQ reformulations. This allows for exploring a space of equivalent JUCQ reformulations among which one with lowest estimated evalu-

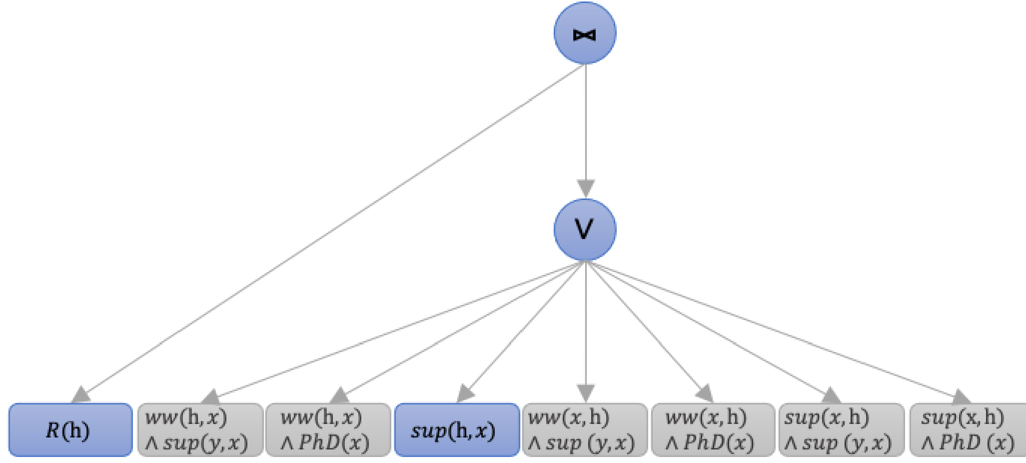


Figure 1.3 – JUCQ representation of the running example

ation cost, w.r.t. the RDBMS at hand, is chosen. q^{JUCQ} *results from joining the UCQ reformulations of* $R(\mathtt{h})$ *and of* $ww(\mathtt{h}, x) \wedge sup(y, x)$*. Here,* $R(\mathtt{h})$ *appears only once (like in* q^{USCQ}*) instead of 7 times (like in* q^{UCQ}*) if it were distributed on the reformulations of* $ww(\mathtt{h}, x) \wedge sup(y, x)$ *using* $\wedge$*. Also, the CQ (3) in* q^{UCQ} *is equivalently encoded into the two pieces shown in blue in* q^{JUCQ}*, up to the distribution of* $\wedge$ *over the* $\vee$*'s.*

Figure 1.3 is a graphical representation of this JUCQ where nodes colored in gray represent the CQs with no answer on $\mathcal{D}$ *and the blue nodes represent CQs (3) having an answer on* $\mathcal{D}$*.* ◇

Finally, consistency checking can be reduced to CQ answering in our KB setting because the negation of a negative rule $r = \forall \bar{x}(q_1(\bar{x}) \rightarrow \perp)$ is precisely the Boolean CQ denoted by $q_{\neg r}() = \exists \bar{x}\, q_1(\bar{x})$ [2] that checks if r derives $\perp$ [20]. In particular, a KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ is consistent if and only if none of the negative rules in $\mathcal{O}$ derives $\perp$ from the facts stored in $\mathcal{D}$ plus the facts entailed from $\mathcal{D}$ using the positive rules in $\mathcal{O}$, i.e., the negation of every negative rule in $\mathcal{O}$ has no answer on the consistent KB consisting of the positive rules in $\mathcal{O}$ and $\mathcal{D}$. Recall that, for a Boolean query, an empty answer set means false, while an answer set consisting of the empty tuple $\langle\rangle$ means true.

Formally, a KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ is consistent if and only if $\bigvee_{r \in \mathcal{O}_-} ans(q_{\neg r}, (\mathcal{O}_+, \mathcal{D})) = \emptyset$, where $\mathcal{O}_-$ and $\mathcal{O}_+$ denote the disjoint subsets of $\mathcal{O}$ consisting respectively of its negative and positive rules. Therefore, in FO-rewritable OMQA setting that allows CQ answering,

2. $\neg\forall\bar{x}(q_1(\bar{x}) \rightarrow \perp) \Leftrightarrow \neg\forall\bar{x}(\neg q_1(\bar{x}) \vee \perp) \Leftrightarrow \neg\forall\bar{x}\neg q_1(\bar{x}) \Leftrightarrow \exists\bar{x} q_1(\bar{x})$

$\mathcal{K} = (\mathcal{O}, \mathcal{D})$ is consistent if and only if $\bigvee_{r \in \mathcal{O}_-} eval(q_{\neg r}^{\mathcal{O}_+}, \mathcal{D}) = \emptyset$, with $q_{\neg r}^{\mathcal{O}_+}$ the reformulation of $q_{\neg r}$ w.r.t. $\mathcal{O}_+$ introduced above.

Example 5 (Cont.). *Consider now that we want to check the consistency of $\mathcal{K}$ and $\mathcal{K}_\perp$ via FO-rewriting. Since r_4 is the only negative rule in these KBs, the consistency of $\mathcal{K}$ and $\mathcal{K}_\perp$ is checked using a reformulation of the negation of r_4, i.e., a reformulation of the Boolean CQ $q_{\neg r_4}() = sup(x, y) \wedge sup(y, z)$.*

The UCQ, USCQ and JUCQ reformulations of $q_{\neg r_4}()$ w.r.t. $\mathcal{O}_+$, computed by the same tools as in the preceding example, are:

$$q_{\neg r_4}^{\text{UCQ}}() = q_{\neg r_4}^{\text{JUCQ}}() = (sup(x, y) \wedge sup(y, z)) \vee (PhD(y) \wedge sup(y, z))$$

$$q_{\neg r_4}^{\text{USCQ}}() = (sup(x, y) \vee PhD(y)) \wedge (sup(y, z))$$

We observe that due to the simplicity of $q_{\neg r_4}()$, a JUCQ reformulation must be either the UCQ one or the UCSQ one.

The KB $\mathcal{K}$ is checked consistent because:

$$eval(q_{\neg r_4}^{\text{UCQ}}, \mathcal{D}) = eval(q_{\neg r_4}^{\text{JUCQ}}, \mathcal{D}) = eval(q_{\neg r_4}^{\text{USCQ}}, \mathcal{D}) = \emptyset.$$

Meanwhile the KB $\mathcal{K}_\perp$ is checked inconsistent because:

$$eval(q_{\neg r_4}^{\text{UCQ}}, \mathcal{D}_\perp) = eval(q_{\neg r_4}^{\text{JUCQ}}, \mathcal{D}_\perp) = eval(q_{\neg r_4}^{\text{USCQ}}, \mathcal{D}_\perp) = \{\langle\rangle\}.$$

In the latter case, the empty tuple $\langle\rangle$ results from the sub-queries shown in blue. ◇

1.4 Conclusion

In this chapter, we went over the basics of OMQA, which constitute the building blocks of the work in this thesis. We also introduced OMQA via the FO-rewriting technique, on which we focus in this thesis.

In the following chapters, we investigate how OMQA performances can be significantly improved.

Chapter 2

Optimization framework for FO-Rewriting

Contents

2.1 Motivations **29**
2.2 Problem statement **30**
2.3 The Ω optimization function **32**
2.3.1 Rationale behind the Ω optimization function 32
2.3.2 Identifying CQs with no answer on a database 33
2.3.3 $(\wedge, \vee)$-CQ evaluation on databases 35
2.4 Database summarization **39**
2.5 Conclusion **41**

In this chapter, we formally define our optimization problem and motivations behind it, before moving on to propose an optimization framework for FO-rewriting, with the goal of achieving higher efficiency for OMQA. We first examine FO-rewriting more closely, in order to motivate the potential for optimization (Section 2.1), before formally stating the problem we study in this thesis (Section 2.2). Next, we focus on defining the contribution of this thesis: an optimization framework for FO-rewriting, which builds on a query optimization function Ω (Section 2.3) that relies on database summaries (Section 2.4).

2.1 Motivations

One of the main characteristics of FO-rewriting, as defined by the literature, is that query answering is performed *truly independently from* the queried data: *FO-rewriting is data-independent*. Under FO-rewriting, a query q is reformulated w.r.t. an ontology $\mathcal{O}$ into a query reformulation $q^{\mathcal{O}}$ such that ***for all*** databases $\mathcal{D}$ consistent with $\mathcal{O}$, $ans(q, (\mathcal{O}, \mathcal{D})) = eval(q^{\mathcal{O}}, \mathcal{D})$ holds. In other words, a single query reformulation $q^{\mathcal{O}}$ is

able to answer the query q on all the consistent KBs with ontology $\mathcal{O}$. However, we point out that when this query q is asked on a *particular KB* $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, the database $\mathcal{D}$ at hand is *just* ***one of all*** the databases the reformulation $q^{\mathcal{O}}$ accommodates to.

We argue that, although theoretically nice, the innate *universality* of $q^{\mathcal{O}}$ makes it *unnecessarily complex*, hence *needlessly difficult* to answer on a *particular* database $\mathcal{D}$. This generality of a query reformulation $q^{\mathcal{O}}$ follows from the fact that it is equivalent to the union of all the maximally-contained queries in the query q w.r.t. the ontology $\mathcal{O}$ (see Section 1, p. 17), which can essentially be seen as all the ways databases may store answers to q according to $\mathcal{O}$. As a consequence, a query reformulation may be large and complex to evaluate in practice [60, 13, 14]. For instance, the worst-case number of CQs that are maximally-contained in a CQ q w.r.t. a lightweight RDFS, DL-lite$_{\mathcal{R}}$ or datalog$\pm_0$ ontology, is exponential in the size of the CQ q (number of atoms) [37, 11, 19, 39].

We therefore study the data-dependent optimization of a query reformulation for a particular KB, in order to trade its universality for better OMQA performance. In particular, we stress that when the database $\mathcal{D}$ is fixed, within the union of all the maximally-contained queries a query reformulation is equivalent to, many may have no answer on $\mathcal{D}$ (i.e., $\mathcal{D}$ does not store answers to q w.r.t. $\mathcal{O}$ this way), in which case they are *irrelevant* to $\mathcal{D}$, and constitute a significant overhead to query evaluation. Consequently, the optimization we propose involves eliminating such irrelevant maximally-contained queries from query reformulations, to expedite their evaluation.

Example 6 (Cont.)**.** *Within q^{UCQ}, all the CQs contained in q w.r.t. $\mathcal{O}$ but the CQ (3) are irrelevant to $\mathcal{D}$, and similarly in q^{USCQ} and q^{JUCQ}, where these CQs are encoded up to the distribution of the $\wedge$'s over the $\vee$'s.* ◇

2.2 Problem statement

Following the previous discussion, we would like to devise an optimization framework for FO-rewriting that enjoys the following properties:

- **Generality:** to be used in as many FO-rewriting settings as possible,
- **Correctness:** to compute the exact answer set of a query,
- **Effectiveness:** to improve query answering time performance.

We propose a framework which relies on an optimization function Ω that turns a given query reformulation $q^{\mathcal{O}}$ into an *optimized query reformulation* for a given database $\mathcal{D}$.

Notation. This optimized query reformulation is hereafter noted $q^{\mathcal{K}}$, as it is specific to the KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$.

For the **generality** of our framework, the Ω function optimizes query reformulations from the language of $(\wedge, \vee)$-*combinations of CQs* (Definition 1 below). Our framework thus applies to FO-rewriting settings with reformulation languages included in $(\wedge, \vee)$-combinations of CQs, e.g., UCQ, USCQ and JUCQ.

Definition 1: $(\wedge, \vee)$-combination of CQs

A $(\wedge, \vee)$-*combination of CQs*, noted $(\wedge, \vee)$-CQ, is either a CQ or a conjunction or union of $(\wedge, \vee)$-CQs.

The Ω function computes an optimized query reformulation $q^{\mathcal{K}}$ contained in $q^{\mathcal{O}}$ (item (1) in Problem 1 below) since $q^{\mathcal{O}}$ is equivalent to a union of maximally-contained queries, in which we remove those irrelevant to a given database $\mathcal{D}$, and removing disjuncts from a union makes it more specific. However, this containment relationship only ensures that the answers to $q^{\mathcal{K}}$ form a subset of the answers to $q^{\mathcal{O}}$ on all the possible databases. For the **correctness** of our framework, Ω therefore computes an optimized query reformulation $q^{\mathcal{K}}$ with same answers as $q^{\mathcal{O}}$ on the particular database $\mathcal{D}$ (item (2) in Problem 1 below).

Finally, for the **effectiveness** of our framework, the Ω function optimizes $q^{\mathcal{O}}$ for $\mathcal{D}$ using a summary $\mathcal{S}$ of $\mathcal{D}$ (item (3) in Problem 1 below). This allows for a trade-off between the number of removed irrelevant maximally-contained queries and Ω's runtime, i.e., optimization time. As we shall see in our experiments, the optimization time may be too high to improve OMQA time performance when Ω identifies irrelevant maximally-contained queries in $q^{\mathcal{O}}$ with the database $\mathcal{D}$ instead of a typically much smaller summary $\mathcal{S}$ of it.

The above discussion can be summarized with the formal statement of the research problem we studied in this thesis.

Problem 1: Optimization for OMQA via FO-rewriting

Let $q^{\mathcal{O}}$ be a $(\wedge, \vee)$-CQ query reformulation and let $\mathcal{D}$ be a database. Define an optimization function Ω and a summary $\mathcal{S}$ of $\mathcal{D}$ so that the optimization of $q^{\mathcal{O}}$ for $\mathcal{D}$ using $\mathcal{S}$, denoted by $q^{\mathcal{K}}$ and computed by $\Omega(q^{\mathcal{O}}, \mathcal{S})$, satisfies:

1. $q^{\mathcal{K}} \subseteq q^{\mathcal{O}}$,
2. $eval(q^{\mathcal{K}}, \mathcal{D}) = eval(q^{\mathcal{O}}, \mathcal{D})$,
3. $\tau(\Omega(q^{\mathcal{O}}, \mathcal{S})) + \tau(eval(q^{\mathcal{K}}, \mathcal{D})) \leq \tau(eval(q^{\mathcal{O}}, \mathcal{D}))$, with $\tau(\cdot)$ the time to compute $\cdot$ in a fixed experimental setup.

We remark that, above, item 1 cannot be safely removed from Problem 1 since, with only items 2 and 3, $q^{\mathcal{K}}$ may be an arbitrary query with the same answer(s) as $q^{\mathcal{O}}$. E.g., "Where does The Web Conference 2024 take place?" may be optimized by "Where does Petra live?" just because they have the same answer, which is Singapore.

2.3 The Ω optimization function

The Ω function we define below, is a function that optimizes query reformulations from *all* the query reformulation languages considered so far for OMQA via FO-rewriting of CQs (i.e., UCQ, USCQ and JUCQ).

2.3.1 Rationale behind the Ω optimization function

When a query reformulation is seen as a $(\wedge, \vee)$-combination of CQs, these subCQs are parts of the maximally-contained CQs that the query reformulation models. Recall for instance Example 4 (p. 23) where the maximally-contained CQ (3) in the UCQ reformulation corresponds to the logical combinations of the subCQs shown in blue in the JUCQ and USCQ reformulations.

Removing subCQs from a query reformulation seen as $(\wedge, \vee)$-combinations of CQs obviously removes all the maximally-contained queries these subCQs are part of, and crucially for us, removing such subCQs with no answer on a particular database removes maximally-contained queries that are irrelevant to this database. E.g., in Example 4, removing from q^{USCQ} the subCQ $sup(x, \texttt{h})$ with no answer on $\mathcal{D}$ also removes from q^{USCQ}

the two irrelevant maximally-contained CQs $R(\mathtt{h}) \wedge sup(x, \mathtt{h}) \wedge sup(x, y)$ and $R(\mathtt{h}) \wedge sup(x, \mathtt{h}) \wedge PhD(x)$: without $sup(x, \mathtt{h})$, they cannot be recovered by distributing the $\wedge$'s over the $\vee$'s.

Rationale behind the Ω function

The Ω function optimizes a $(\wedge, \vee)$-CQ query reformulation for a given database by rewriting it from the bottom up to:

(i) Identify subCQs with no answer on this database.

(ii) Propagate the effect of removing such subCQs within the query reformulation.

2.3.2 Identifying CQs with no answer on a database

Checking whether a single CQ has no answer on a database can be done easily (e.g., using EXISTS in SQL) and efficiently in general since RDBMSs are highly-optimized for CQs, e.g., [56]. However, doing the same check for all the subCQs in a query reformulation may take significant time, especially when the database is large. To mitigate this issue, Ω uses database summaries that are (typically small) homomorphic approximations of the databases they summarize. Using such summaries instead of the databases allows for trading completeness of identifying CQs with no answer for efficiency, while retaining soundness.

Definition 2: Summary of a database

A database $\mathcal{S}$ is a *summary* of a database $\mathcal{D}$ iff (i) there exists a database-to-summary homomorphism σ from the terms [a] in $\mathcal{D}$ to the terms in $\mathcal{S}$, i.e., $\mathcal{D}_\sigma = \mathcal{S}$ where $\mathcal{D}_\sigma$ is the database obtained from $\mathcal{D}$ by replacing the terms in $\mathcal{D}$ by their images in $\mathcal{S}$ through σ, such that (ii) σ maps constants in $\mathcal{D}$ to constants in $\mathcal{S}$, while it maps variables in $\mathcal{D}$ to constants or variables in $\mathcal{S}$.

a. We recall that a term is either a variable or a constant in FO logic.

In the above definition, (i) ensures that $\mathcal{S}$ is a homomorphic approximation of $\mathcal{D}$, while (ii) ensures the soundness of identifying CQs with no answer on $\mathcal{D}$ using $\mathcal{S}$ established by Theorem 1 below. We also note that a database is a particular summary of itself: $\mathcal{D} = \mathcal{S}$ holds when the database-to-summary homomorphism σ maps each term to itself, i.e., when σ is the *identity* function.

Theorem 1

Let $\mathcal{D}$ be a database and $\mathcal{S}$ a summary of it with the homomorphism σ. Let q be a CQ asked on $\mathcal{D}$ and q_σ the CQ obtained from q by replacing its constants with their images through σ. If q_σ has no answer on $\mathcal{S}$, then q has no answer on $\mathcal{D}$.

Proof. We prove the theorem by showing that its contrapositive holds, i.e., *if q has some answer on $\mathcal{D}$, then q_σ has some answer on $\mathcal{S}$.*
If q has an answer on $\mathcal{D}$, then there exists a homomorphism h from q to $\mathcal{D}$ such that $h(q) \subseteq \mathcal{D}$ where every free variable is mapped to a constant, every existential variable is mapped to a constant or variable, and every constant is mapped to itself. Moreover, the composition $\sigma \circ h$ is a homomorphism from q to $\mathcal{S}$ such that $\sigma \circ h(q) \subseteq \mathcal{S}$ where, by definition of a database summary, every free variable is mapped to a constant, every existential variable is mapped to a constant or variable, and every constant is mapped to its image through σ.
Let us now build a homomorphism g from q_σ to $\mathcal{S}$ such that $g(q_\sigma) = \sigma \circ h(q) \subseteq \mathcal{S}$: it suffices that g maps every variable exactly as $\sigma \circ h$ does, while it maps every constant to itself (constants have already been replaced by their image through σ in q_σ). Since, defined this way, g maps free variables to constants, q_σ has an answer on $\mathcal{S}$. □

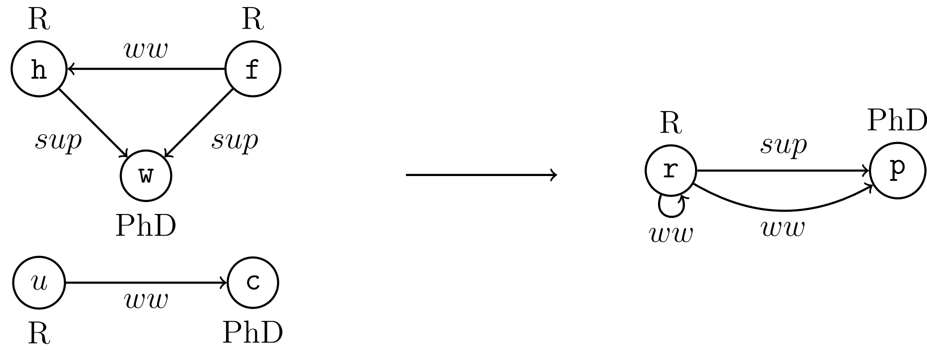
We emphasize that, as pointed out in the example below, if q_σ has no answer on $\mathcal{S}$, then q *certainly* has no answer on $\mathcal{D}$. On the other hand, if q_σ has an answer on $\mathcal{S}$, then q *may or may not* have an answer on $\mathcal{D}$.

Example 7 (Cont.). *Let us consider the summary $\mathcal{S}$ of $\mathcal{D}$ with $\mathcal{D}$-to-$\mathcal{S}$ homomorphism σ such that $\sigma(\mathtt{c}) = \sigma(\mathtt{w}) = \mathtt{p}$ and $\sigma(\mathtt{f}) = \sigma(\mathtt{h}) = \mathtt{r}$:*

$$\mathcal{S} = \{R(\mathtt{r}), sup(\mathtt{r}, \mathtt{p}), PhD(\mathtt{p}), ww(\mathtt{r}, \mathtt{r}), ww(\mathtt{r}, \mathtt{p})\}.$$

Let us consider now the CQs (1) *and* (5) *in* q^{UCQ}*, which we name* q^1 *and* q^5 *respectively. By Theorem 1, since* $q^1_\sigma(x) = R(\mathtt{r}) \wedge ww(\mathtt{r}, x) \wedge sup(y, x)$ *has an answer on the summary* $\mathcal{S}$ *(*$ans(q^1_\sigma, \mathcal{S}) = \{\mathtt{p}\}$*), then* q^1 may or may not *have an answer on* $\mathcal{D}$ *(here,* q^1 *has no answer on* $\mathcal{D}$*), while because* $q^5_\sigma(x) = R(\mathtt{r}) \wedge ww(x, \mathtt{r}) \wedge PhD(x)$ *has no answer on* $\mathcal{S}$*, then for sure* q^5 *has no answer on* $\mathcal{D}$*.*

The graph representation below clearly illustrates the compression obtained from summarizing $\mathcal{D}$ *into* $\mathcal{S}$ *in our example.*



◇

2.3.3 $(\wedge, \vee)$-CQ evaluation on databases

The Ω function builds on Theorem 1 to optimize a $(\wedge, \vee)$-CQ for a database $\mathcal{D}$. It rewrites a query while (*i*) identifying its subCQs with no answer on $\mathcal{D}$ using a summary $\mathcal{S}$ of it ((1) in Definition 3 below) and (*ii*) performing a bottom-up removal of the largest subqueries with no answer on $\mathcal{D}$ that these subCQs are the cause of ((2) and (3) in Definition 3 below).

Definition 3: Summary-based optimization of a $(\wedge, \vee)$-CQ

Let q be a $(\wedge, \vee)$-CQ asked on a database $\mathcal{D}$ and $\mathcal{S}$ be a summary of $\mathcal{D}$ with the database-to-summary homomorphism σ.
The *summary-based optimization of q for $\mathcal{D}$ using $\mathcal{S}$* is the $(\wedge, \vee)$-CQ obtained as the result of $\Omega(q, \mathcal{S})$, with Ω recursively defined as follows.
Below, $\varnothing$ denotes the empty relation with appropriate arity.
The optimization of a CQ q is:

$$\Omega(q, \mathcal{S}) = \begin{cases} \varnothing \text{ if } ans(q_\sigma, \mathcal{S}) = \emptyset \\ q \text{ otherwise} \end{cases} \quad (1)$$

where q_σ is the CQ obtained from q by replacing its constants by their images through σ.
The optimization of a conjunction of subqueries $\bigwedge_{i=1}^n q_i$ is:

$$\Omega(\bigwedge_{i=1}^{n} q_i, \mathcal{S}) = \begin{cases} \varnothing \text{ if } \exists i \in [1, n]\ \Omega(q_i, \mathcal{S}) = \varnothing \\ \bigwedge_{i=1}^n \Omega(q_i, \mathcal{S}) \text{ otherwise} \end{cases} \quad (2)$$

The optimization of a disjunction of subqueries $\bigvee_{i=1}^n q_i$ is:

$$\Omega(\bigvee_{i=1}^{n} q_i, \mathcal{S}) = \begin{cases} \varnothing \text{ if } \forall i \in [1, n]\ \Omega(q_i, \mathcal{S}) = \varnothing \\ \bigvee_{1 \leq i \leq n,\ \Omega(q_i, \mathcal{S}) \neq \varnothing} \Omega(q_i, \mathcal{S}) \text{ otherwise} \end{cases} \quad (3)$$

In the above definition, the rewriting rule (1) directly follows from the soundness of identifying CQs with no answer using a database summary (Theorem 1, p. 34), while the two other rewriting rules (2) and (3) follow from the semantics of the $\wedge$ and $\vee$ operators, respectively.

The next theorem establishes the two semantic relationships between a $(\wedge, \vee)$-CQ and its summary-based optimization, which correspond to items 1 and 2 in Problem 1 (p. 32). In particular, it states the *correctness* of summary-based optimization for $(\wedge, \vee)$-CQ evaluation on a database.

Theorem 2

Let $\mathcal{D}$ be a database, $\mathcal{S}$ a summary of $\mathcal{D}$, and q a $(\wedge, \vee)$-CQ asked on $\mathcal{D}$. Then, $\Omega(q, \mathcal{S}) \subseteq q$ and $eval(q, \mathcal{D}) = eval(\Omega(q, \mathcal{S}), \mathcal{D})$.

Proof. Let us first prove $\Omega(q, \mathcal{S}) \subseteq q$.

We prove this by induction on the depth d of q defined as the maximal nesting of the $\wedge$ and $\vee$ operators on top of CQs, with the *induction hypothesis* that Ω performs rewritings (rules (1), (2) and (3) in Definition 3) that are contained in the rewritten query.

Base case, $d = 0$: rule (1) rewrites q either by (second case) itself or by (first case) $\varnothing$, and clearly, q is contained in itself and $\varnothing$ is contained in q.

Induction step, $d > 0$: rule (2) rewrites a conjunction either by (second case) a contained one (induction) or by (first case) $\varnothing$ that is by definition contained in the rewritten conjunction; rule (3) rewrites a disjunction either by (second case) a contained one (induction), or by $\varnothing$ (first case) that is by definition contained in the rewritten disjunction. Let us now prove that $eval(q, \mathcal{D}) = eval(\Omega(q, \mathcal{S}), \mathcal{D})$.

Again, we prove this by induction on the depth d of q defined as the maximal nesting of $\wedge$ and $\vee$ operators on top of CQs, with the *induction hypothesis* that Ω performs rewritings (rules (1), (2) and (3) in Definition 3) that are equivalent w.r.t. the database $\mathcal{D}$.

Base case, $d = 0$: rule (1) rewrites q either by (second case) itself or by (first case) $\varnothing$ if q has no answer on $\mathcal{S}$, hence on $\mathcal{D}$ according to Theorem 1, i.e., q is equivalent to $\varnothing$ on $\mathcal{D}$.

Induction step, $d > 0$: rule (2) rewrites a conjunction either by (second case) an equivalent one (induction) or by (first case) $\varnothing$ if a q_i subquery has no answer on $\mathcal{D}$ (induction), hence the conjunction is equivalent to $\varnothing$ on $\mathcal{D}$; rule (3) rewrites a disjunction either by (second case) an equivalent one (induction), or by $\varnothing$ (first case) if all its subqueries have no answer on $\mathcal{D}$, hence the disjunction is equivalent to $\varnothing$ on $\mathcal{D}$. $\square$

Example 8 (Cont.). *It can be easily checked that the summary-based optimization of* q^{UCQ}, q^{USCQ} *and* q^{JUCQ} *for* $\mathcal{D}$ *using* $\mathcal{S}$ *corresponds to the following UCQ, USCQ and JUCQ, respectively. We also show in* purple *the disjuncts that would have been additionally removed if* Ω *had used* $\mathcal{D}$ *instead of* $\mathcal{S}$*. However, as we shall see in our experiments, the optimization time may not amortize when* Ω *uses a database instead of a summary of it.*

$$q_{\mathcal{S}}^{\mathrm{UCQ}}(x) = \Omega(q^{\mathrm{UCQ}}, \mathcal{S}) = (R(\mathtt{h}) \wedge ww(\mathtt{h}, x) \wedge sup(y, x)) \\ \vee (R(\mathtt{h}) \wedge ww(\mathtt{h}, x) \wedge PhD(x)) \\ \vee (R(\mathtt{h}) \wedge sup(\mathtt{h}, x))$$

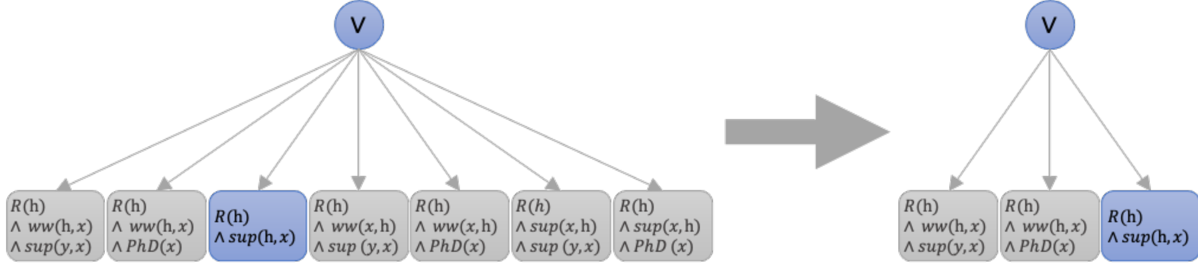


Figure 2.1 – Optimizing the UCQ from our running example

Figure 2.1 shows the transformation of $q^{\mathrm{UCQ}}(x)$ *into* $q_{\mathcal{S}}^{\mathrm{UCQ}}(x)$*; 4 empty CQs (in gray) were discarded, leaving only 2 empty CQs in* $q_{\mathcal{S}}^{\mathrm{UCQ}}(x)$*.*

$$q_{\mathcal{S}}^{\mathrm{USCQ}}(x) = \Omega(q^{\mathrm{USCQ}}, \mathcal{S}) = (R(\mathtt{h})) \\ \wedge (ww(\mathtt{h}, x) \vee sup(\mathtt{h}, x) \vee ww(x, \mathtt{h})) \\ \wedge (sup(y, x) \vee PhD(x))$$

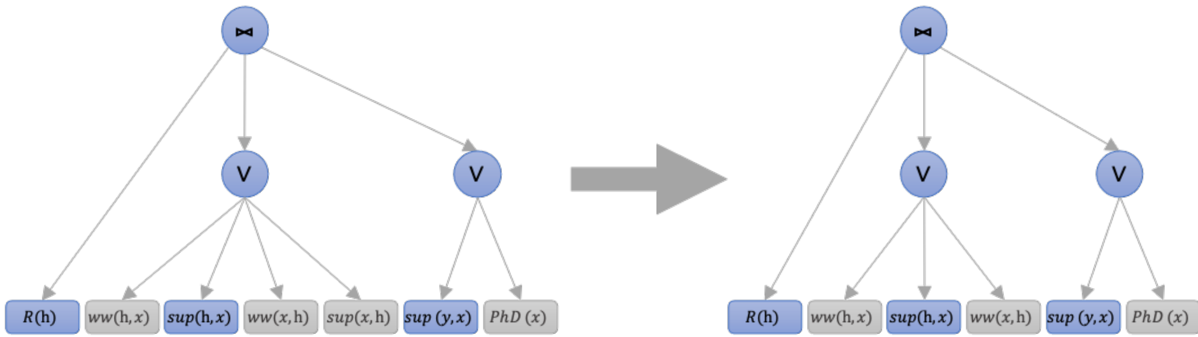


Figure 2.2 – Optimizing the USCQ from our running example

Figure 2.2 shows the transformation of $q^{\mathrm{USCQ}}(x)$ *into* $q_{\mathcal{S}}^{\mathrm{USCQ}}(x)$*; one empty CQ (in gray) was discarded, leaving only 3 empty CQs in* $q_{\mathcal{S}}^{\mathrm{USCQ}}(x)$*.*

$$q_{\mathcal{S}}^{\mathrm{JUCQ}}(x) = \Omega(q^{\mathrm{JUCQ}}, \mathcal{S}) = (R(\mathtt{h})) \wedge \Big((ww(\mathtt{h}, x) \wedge sup(y, x)) \\ \vee (ww(\mathtt{h}, x) \wedge PhD(x)) \\ \vee (sup(\mathtt{h}, x))\Big)$$

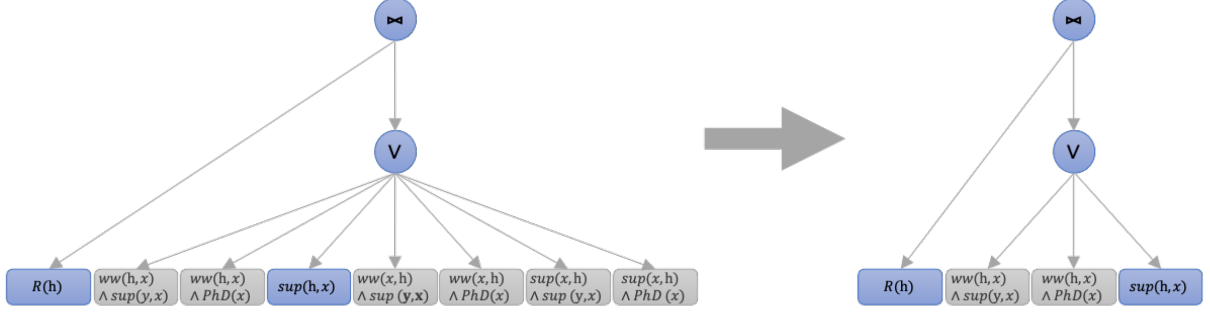


Figure 2.3 – Optimizing the JUCQ from our running example

Figure 2.3 shows the transformation of $q^{\mathrm{JUCQ}}(x)$ *into* $q_{\mathcal{S}}^{\mathrm{JUCQ}}(x)$*; 4 empty CQs (in gray) was discarded, leaving only 2 empty CQs in* $q_{\mathcal{S}}^{\mathrm{JUCQ}}(x)$*.*

Above, the optimized UCQ and JUCQ reformulations encode 3 contained CQs w.r.t. $\mathcal{O}$ *instead of 7, while the USCQ reformulation encodes 6 contained CQs w.r.t.* $\mathcal{O}$ *instead of 7.*

For $\mathcal{L}_R \in \{\mathrm{UCQ}, \mathrm{USCQ}, \mathrm{JUCQ}\}$*,* $\Omega(q^{\mathcal{L}_R}, \mathcal{S}) \subseteq q^{\mathcal{L}_R}$ *holds since* Ω *makes unions more specific by removing disjuncts, and* $eval(\Omega(q^{\mathcal{L}_R}, \mathcal{S}), \mathcal{D}) = eval(q^{\mathcal{L}_R}, \mathcal{D})$ *holds since both* $q^{\mathcal{L}_R}$ *and* $\Omega(q^{\mathcal{L}_R}, \mathcal{S})$ *model the CQ (3) that produces the sole answer* `w`*.*

Let us now consider consistency checking for the consistent KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$*, which is checked using the Boolean CQ* $q_{\neg r_4}$ *(recall Example 5).*

The summary-based optimization of $q_{\neg r_4}^{\mathrm{UCQ}}$*,* $q_{\neg r_4}^{\mathrm{USCQ}}$ *and* $q_{\neg r_4}^{\mathrm{JUCQ}}$ *for* $\mathcal{D}$ *using* $\mathcal{S}$ *are:*

$$\Omega(q_{\neg r_4}^{\mathrm{UCQ}}, \mathcal{S}) = \Omega(q_{\neg r_4}^{\mathrm{JUCQ}}, \mathcal{S}) = \varnothing \textit{ and } \Omega(q_{\neg r_4}^{\mathrm{USCQ}}, \mathcal{S}) = q_{\neg r C_4}^{\mathrm{USCQ}}.$$

We remark that the optimized UCQ and JUCQ reformulations indicate that $q_{\neg r_4}$ *has no answer on* $\mathcal{D}$ *without requiring any further access to* $\mathcal{D}$*, while the USCQ reformulation is not optimized by* Ω *and requires to be (fully) evaluated on* $\mathcal{D}$ *to find out that it has no answer on* $\mathcal{D}$*.* ◇

2.4 Database summarization

We now introduce the *concrete* database summaries that we use with the Ω optimization function from the preceding section. They are defined by adapting the *quotient operation* from graph theory [41] to the incomplete relational databases we consider

in this work. The quotient operation has been widely investigated in the literature for graph database summarization [50, 21]. Besides, it offers an elegant summarization technique by decoupling the summarization method, which basically fuses equivalent nodes, from the high-level specification of equivalent nodes, defined by an equivalence relation [1], e.g., bisimilarity [2].

Assuming we have an equivalence relation between database terms (the one we use will be discussed shortly), we define a *quotient database*, i.e., a *quotient-based summary*, as follows.

Definition 4: Quotient database

Let $\mathcal{D}$ be a database, $\equiv$ be some equivalence relation between terms, and let $c^1_\equiv, \ldots, c^k_\equiv$ denote, by a slight abuse of notation, both the equivalence classes of the terms in $\mathcal{D}$ w.r.t. $\equiv$ and the terms used to represent these equivalence classes.
The *quotient database of* $\mathcal{D}$ *w.r.t.* $\equiv$ is the database $\mathcal{D}_\equiv$ such that:

- $R(c^{\alpha_1}_\equiv, \cdots, c^{\alpha_n}_\equiv) \in \mathcal{D}_\equiv$ iff there exists $R(term_1, \cdots, term_n) \in \mathcal{D}$ with $term_i \in c^{\alpha_i}_\equiv$ and $1 \leq \alpha_i \leq k$, for $1 \leq i \leq n$,
- the term $c^j_\equiv$ in $\mathcal{D}_\equiv$, for $1 \leq j \leq k$, is a variable if all the equivalent terms in $\mathcal{D}$ it represents according to $\equiv$ are variables, otherwise it is a constant.

The next proposition establishes that quotient databases can be used by the optimization function Ω to identify CQs with no answer on databases. This proposition follows from the fact that in the above definition, $\equiv$ defines an *implicit function* that maps the terms in $\mathcal{D}$ to the terms in $\mathcal{D}_\equiv$, which turns out to be the database-to-summary homomorphism σ in Definition 2 (p. 33) such that $\mathcal{D}_\equiv = \mathcal{D}_\sigma$, because the first and second items in the above definition enforce respectively the conditions (i) and (ii) in Definition 2.

Proposition 1

Quotient databases are database summaries.

1. An equivalence relation is a binary relation that is reflexive, symmetric, and transitive.

We now introduce the equivalence relation $\equiv_\Omega$ used to build our summaries, i.e., how database terms are fused into summary terms. The rationale behind its definition is that, similarly to ER or UML languages, ontology languages [6, 23, 17, 8] are centered on *concepts* modeled by unary predicates, which are then interrelated using *relationships* modeled by n-ary predicates (with $n \geq 2$). We therefore choose to adopt a summarization *centered on the instances of concepts* stored in a database: all the terms that are instances of a same concept in a database are represented by a single term in the database summary ((i) in Definition 5 below), and all the concepts with common instances in the database (i.e., that join) have the same single term that represents all their instances in the database summary ((ii) in Definition 5 below).

As we shall see in our experiments, $\equiv_\Omega$ achieves a good tradeoff between size reduction (≥90%) and completeness of identifying CQs with no answer on the summarized database (≥92% on average).

Definition 5: $\equiv_\Omega$ equivalence relation

$\equiv_\Omega$ is the equivalence relation such that two terms $term_1$ and $term_2$ are equivalent within a database $\mathcal{D}$, denoted by $term_1 \equiv_\Omega term_2$, iff ($i$) both $term_1$ and $term_2$ are instances of the same *unary predicate*, i.e., concept, or (ii) there exists a term $term_3$ in $\mathcal{D}$ such that $term_1 \equiv_\Omega term_3$ and $term_2 \equiv_\Omega term_3$.

Example 9 (Cont.). *The summary $\mathcal{S}$ in Example 7 is actually the quotient database of $\mathcal{D}$ w.r.t. $\equiv_\Omega$: it defines two equivalence classes, one for the researchers in $\mathcal{D}$, i.e., $\{\mathtt{f}, \mathtt{h}, u\}$, and one for the PhD students in $\mathcal{D}$, i.e., $\{\mathtt{w}, \mathtt{c}\}$; these two classes are represented in $\mathcal{S}$ by the constants $\mathtt{r}$ and $\mathtt{p}$, respectively.* ◇

We discuss the need of summary maintenance in case of database updates in Chapter 5, as well as how our particular summaries (quotient databases w.r.t. $\equiv_\Omega$) can be efficiently updated.

2.5 Conclusion

In this chapter, we discussed the limitations of FO-rewriting for OMQA: the universality of query reformulations creates an overhead which affects OMQA performance. This

discussion lead us to define the problem statement: propose a framework that optimizes query reformulations w.r.t. to the database the KB accommodates to, all while retaining correctness, effectiveness as well as generality.

Next, we discussed the main contributions of our work. We defined the Ω optimization function, which optimizes query reformulations by discarding from them irrelevant subqueries w.r.t. to the database we are interested in. We also defined the database summaries we use with our Ω function in place of the databases, to allow a tradeoff between optimization time and the completeness of discarding irrelevant subqueries from query reformulations.

The next chapters will focus on the experiments we conducted and to evaluate the benefits of our optimization framework.

Chapter 3

Experimental Results

Contents

3.1 Experimental setup **43**
3.2 Database summarization **46**
3.3 Query answering performance analysis **47**
3.3.1 Queries 47
3.3.2 Performance analysis 49
3.4 Consistency checking performance analysis **60**
3.4.1 Adding negative rules to $\mathcal{O}$ 60
3.4.2 Adding inconsistencies to databases 61
3.4.3 Performance analysis 62
3.5 Conclusion **69**

We carried out a set of experiments to evaluate the performance improvements that our work on optimizing FO-rewriting for query answering may bring in practice. We comment on the results below.

First, we briefly describe our experimental setup (Section 3.1). Next, we comment on our summarization technique (Section 3.2). Then, we discuss the results we obtained for for query answering (Section 3.3) and consistency checking (Section 3.4).

Finally, we conclude on our findings (Section 3.5).

3.1 Experimental setup

We implemented our optimization framework in a prototype called OptiRef, using Java 14. OptiRef generates optimized reformulations over two steps, according to Figure 2:

(i) **Summarization:** OptiRef computes quotient databases w.r.t. the $\equiv_\Omega$ equivalence relation (recall Section 2.4, p. 39).

(ii) **Optimization:** OptiRef optimizes the query reformulations using the Ω function (recall Section 2.3, p. 32). The optimization is made either with respect to the original database or with respect to the summarized database (a.k.a. the quotient database).

We used the Ω function for summary-based optimization of query reformulations in the following FO-rewritable OMQA settings: $(\text{CQ}, \text{DL-lite}_{\mathcal{R}}, \text{UCQ})$, $(\text{CQ}, \text{DL-lite}_{\mathcal{R}}, \text{USCQ})$, $(\text{CQ}, \text{DL-lite}_{\mathcal{R}}, \text{JUCQ})$.

In order to conduct our experiments, we had to rely on several tools. These tools allowed us to build our KBs and store them and then perform query reformulations according to the methods provided in the literature. These tools are detailed below.

Knowledge Bases (KB) For our KBs, we relied on the well-established *extended LUBM benchmark* [51] a.k.a. LUBM$^{\exists}$. It is an adaptation of the Leight University benchmark [42], a.k.a. LUBM, to the DL-lite$_{\mathcal{R}}$ description logic [19].

We chose this benchmark for two reasons. First, DL-lite$_{\mathcal{R}}$ is the most expressive KB language for which the reformulation of CQs into UCQ, USCQ and JUCQ reformulations has been studied. Second, this benchmark is widely-considered in the OMQA literature and provides opportunities to reuse and adapt many available queries to our needs.

The ontology We used the LUBM$^{\exists}_{n}$ ontology with default value $n = 20$. In the ontology name LUBM$^{\exists}_{n}$, $\exists$ indicates that *incomplete* facts can be derived using the positive rules and n indicates that every unary relation a.k.a. concept in the original LUBM ontology is *specialized into* n subconcepts in LUBM$^{\exists}_{n}$, e.g., Course, Professor, Student, etc. The default value $n = 20$ is considered reasonably challenging and is widely adopted in the literature (the higher the n value, the higher the number of contained queries encoded in query reformulations).

The LUBM$^{\exists}_{20}$ ontology is made of 449 positive rules over 163 relations: 128 unary relations or concepts, and 35 binary relations or relationships. We note that LUBM$^{\exists}_{n}$ ontologies do not contain negative rules, i.e., integrity constraints.

The data We used the EUGen (v0.1b) data generator provided with the Extended LUBM benchmark to generate databases of increasing sizes. The tool generates synthetic, repeatable and customizable data over the LUBM ontology. We are able to select the seed for random number generation and the number of universities to consider, allowing us to

generate random, potentially very large datasets.

RDBMSs We used the popular, commercial or open-source RDBMSs to store our databases and their summaries, notably: DB2 (v11.5.5), MySQL Community (v8.0.34) and PostgreSQL (v14.2). Our choice was based on the RDBMSs' popularity as robust systems and because they are commonly used in the OMQA literature.

Data Layout We adopted the data layout of [14], for both the databases and the summaries, which was found to be the most efficient for evaluating query reformulations on DL-lite$_{\mathcal{R}}$ KB's database. DL-lite$_{\mathcal{R}}$ uses unary relations for concepts and binary relations for relationships. As such, unary relations are stored as *unary tables* and binary relations are stored as *binary tables*. Moreover, all the values stored in these tables are *dictionary-encoded* into integers; the dictionary is stored as a *binary table*. Finally, for a database summary, the database-to-summary homomorphism σ, which maps the database terms to the summary terms, is stored as a *binary table*. For the above tables, each unary table has *an index on its unique attribute* and each binary table has *the two two-attributes indexes*.

Query Reformulation Tools To compute query reformulations, we used the Rapid (v0.93) [24], Compact (v1.0b6) [60] and GDL (v1.0) [14] FO-rewriting tools that respectively compute UCQ, USCQ and JUCQ reformulations of CQs w.r.t. DL-lite$_{\mathcal{R}}$ ontologies. They load and keep in memory the ontology w.r.t. which CQs are reformulated.

While Compact and GDL are the only options to respectively compute USCQ and JUCQ query reformulations, there are other tools besides Rapid that can be used to compute UCQ query reformulations, e.g., Graal [7], Iqaros [61], Nyaya [62], Presto [55], Requiem [54], etc. Choosing Rapid instead of another tool does not affect our conclusions, as reformulation time is negligible w.r.t. both optimization and evaluation times: reformulation is performed w.r.t. the in-memory ontology, while optimization and evaluation is performed w.r.t. the on-disk data (summary and database).

Hardware We used Ubuntu 20.04.2 Linux server with Intel Xeon 4215R 3.20GHz CPU, 128GB of RAM, and 7TB of fast HDD.

Further implementation details will be provided in the next chapter.

3.2 Database summarization

Database $\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{S}\rvert$	size reduction (%)	DB2	MySQL	PostgreSQL
LUBM1M	1,186,832	93,440	92.127%	60	58	15
LUBM10M	10,793,976	843,526	92.185%	273	609	86
LUBM50M	53,328,357	4,159,768	92.2%	3,121	2,899	308
LUBM100M	106,596,211	8,315,828	92.199%	3,523	5,925	699
LUBM150M	159,899,201	12,474,239	92.199%	8,693	8,661	1,100

Table 3.1 – Characteristics of our databases, their summaries, and summarization time per RDBMS (in seconds)

We generated five LUBM databases of increasing size: LUBM1M, LUBM10M, LUBM50M, LUBM100M and LUBM150M. The name of a database indicates the number of stored facts in millions. Databases are created such that LUBM1M $\subseteq$ LUBM10M $\subseteq$ LUBM50M $\subseteq$ LUBM100M $\subseteq$ LUBM150M, where $\subseteq$ means set inclusion, so that query answering becomes harder as data grows.

For database summarization, we relied on the *union-find* data structure for disjoint sets [25], since equivalence classes of database terms w.r.t. $\equiv_\Omega$ are disjoint sets of equivalent terms w.r.t. $\equiv_\Omega$. This data structure supports two main operations, *union* and *find*, in optimal constant amortized time complexity [57, 58], i.e., time complexity is almost constant over a sequence of union or find operations. *Union* is used to state which values must be in a same set, and results in merging the sets these values belong to. *Find* returns the representative value of the set a given value belongs to.

We first compute the homomorphism σ from the database $\mathcal{D}$ to the summary $\mathcal{S}$ (Definition 2) w.r.t. the $\equiv_\Omega$ equivalence relation (Definition 5, p. 41). Given a union-find data structure for disjoint sets of integers, we use *union* to state that the (integer-encoded) terms stored in each unary relation in $\mathcal{D}$ must be in a same set, as these terms are equivalent w.r.t. $\equiv_\Omega$ (condition (*i*) in Definition 5). By definition of *union*, this ensures that if unary relations share some terms, in which case all the terms of these relations are equivalent w.r.t. $\equiv_\Omega$ (condition (*ii*) in Definition 5), then these terms end up in the same set. Finally, since *find* returns a representative term for the set of equivalent terms a given term belongs to, it models the homomorphism σ from the database $\mathcal{D}$ to its summary $\mathcal{S}$ w.r.t. $\equiv_\Omega$. The computation of σ is therefore linear in the size of the data: it needs a worst-case number of calls to *union* in the size of $\mathcal{D}$, each of which is performed in

constant amortized time. Then, the summary $\mathcal{S}$ of the database $\mathcal{D}$ w.r.t. $\equiv_\Omega$ is computed by rewriting $\mathcal{D}$ into $\mathcal{S}$ as per Definition 4 (p. 40): every fact in $\mathcal{D}$ is rewritten by replacing each term by its image through σ, i.e., through *find*. The computation of $\mathcal{S}$ is therefore linear in the size of the data: it needs a worst-case number of calls to *find* in the size of $\mathcal{D}$ (one or two calls per fact), each of which is performed in constant amortized time.
Table 3.1 shows for each database $\mathcal{D}$ we generated:

- $|\mathcal{D}|$: the size of $\mathcal{D}$, or the number of facts contained in $\mathcal{D}$
- $|\mathcal{S}|$: the size of $\mathcal{S}$, or the number of facts contained in $\mathcal{S}$.
- The $\mathcal{D}$-to-$\mathcal{S}$ size reduction given by the formula: $1 - |\mathcal{S}|/|\mathcal{D}|$
- The summarization time per RDBMS which includes the computation and storage time of σ as well as the time needed to generate and store $\mathcal{S}$.

We observe that $\equiv_\Omega$ achieves significant size reduction ($\geq 90\%$) and that summarization time scales linearly in the size of the data.

3.3 Query answering performance analysis

We now comment on the performance of queries of type **query answering** we used throughout our experiments.

3.3.1 Queries

We used ten queries (CQs) adapted from [51, 14] to conduct our experiments on query answering. This allowed us to obtain CQs having a a variety of numbers of maximally-contained CQs w.r.t. $\mathcal{O}$ that query reformulations model (recall Section 1) and a variety of numbers of answers. We also consider queries that contain both constants and variables.

The full list of queries we used in our experiments to evaluate query answering, in dlp format, and their descriptions, is available in Table A.1 (p. 97).

For each database (LUBM1M, LUBM10M, LUBM50M, LUBM100M, LUBM150M), and using each of the RDBMSs (DB2, MySQL, PostgreSQL), we processed every query using three query answering strategies, per $\mathcal{L}_R$ query reformulation languages, used by FO-rewriting tools we considered:

- $\mathcal{L}_R$ = UCQ for Rapid.
- $\mathcal{L}_R$ = USCQ for Compact.
- $\mathcal{L}_R$ = JUCQ for GDL.

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,759	1,949	1,701	1,151	719	495	299	183	143	31
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
optim. ratio for UCQ/S	81.92	100	99.88	83.8	82.05	79.84	52.53	78.02	69.92	75
optim. ratio for USCQ/S	100	100	100	100	100	100	100	100	100	100
optim. ratio for JUCQ/S	100	100	100	94.12	100	97.33	88.89	78.89	100	100

Table 3.2 – Characteristics of QA queries with $\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$

$\mathcal{L}_\mathbf{R}$/REF. The first strategy, denoted by $\mathcal{L}_R$/REF, simply consists in computing the $\mathcal{L}_R$ query reformulation with the FO-rewriting tool and then evaluating it using the RDBMS; this is how OMQA is performed via FO-rewriting, hence the state-of-the-art baseline.

$\mathcal{L}_\mathbf{R}$/DB. The second strategy, denoted by $\mathcal{L}_R$/DB, departs from $\mathcal{L}_R$/REF by optimizing the query reformulation for the database $\mathcal{D}$ before evaluating it. For this strategy, our Ω function optimizes the query reformulation using the database $\mathcal{D}$.

$\mathcal{L}_\mathbf{R}$/S. The last strategy, denoted by $\mathcal{L}_R$/S, is similar to $\mathcal{L}_R$/DB except that our Ω function optimizes the query reformulation for $\mathcal{D}$ using the summary $\mathcal{S}$ of $\mathcal{D}$.

The main characteristics of our Query Answering (QA) queries are shown in Table 3.2 above. This table describes, for each query:

- The number of atoms making up the query.
- The number of CQs contained in the query w.r.t. $\mathcal{O}$, regardless of the adopted query reformulation language.
- The number of answers to the query on the KB, i.e., the size of $ans(q, \mathcal{K})$.
- The optimization ratio for UCQ, USCQ and JUCQ obtained with $\mathcal{L}_R$/S, for $\mathcal{D} =$ LUBM150M. We define it as the percentage of CQs with no answer on $\mathcal{D}$ that are identified and removed by Ω using $\mathcal{S}$. It is given by the following formula:

(number of CQs with NO answer on $\mathcal{S}$ / number of CQs with NO answer on $\mathcal{D}$)

We note that the optimization ratio is 0% with $\mathcal{L}_R$/REF and 100% with $\mathcal{L}_R$/DB. This is because, the baseline method is not optimized and using Ω over the database (as opposed to its summary) allows us to identify all empty CQs.

We observe that optimization ratios are high in general, 92% on average with $QA6$ having the lowest value of 52.53% using UCQ, thus our summaries are effective in identifying

CQs with no answers. Similar results are obtained on LUBM1M, LUBM10M, LUBM50M, and LUBM100M.

Further details on QA queries characteristics, can be found in Appendix A.2 (p. 101).

3.3.2 Performance analysis

The query answering times we measured when processing the above-mentioned strategies are reported for all the databases we considered: LUBM1M (Figure 3.1, p. 50), LUBM10M (Figure 3.2, p. 51), LUBM50M (Figure 3.3, p. 52), LUBM100M (Figure 3.4, p. 53) and finally LUBM150M (Figure 3.5, p. 54).

For a given strategy, the measured time is defined as optimization time (for $\mathcal{L}_R$/DB and $\mathcal{L}_R$/S only) + evaluation time (recall item 3 in Problem 1, p. 32). Every reported time is an average over 5 "hot" query runs, s.t. the first "cold" query run is discarded. Times are reported in a logarithmic scale as follows:

- Each RDBMS data is reported independently s.t. the figures for DB2 are on the top, in the middle for MySQL and on the bottom for PostgreSQL.
- Each query QAk (with $0 \leq k \leq 9$) is represented by a set of 9 bars s.t. : For each reformulation language (UCQ in green, USCQ in blue and JUCQ in red), the first bar represents the reformulated query without optimization ($\mathcal{L}_R$/REF), the next one represents the reformulated query optimized using the database $\mathcal{D}$ ($\mathcal{L}_R$/DB) and the third bar represents the reformulated query optimized using $\mathcal{S}$ ($\mathcal{L}_R$/S).

We also note that missing bars indicate system errors, where the queries could not be executed. For example, for DB2, regardless of the database size, the UCQ/REF light green bars for $QA0, QA1, QA2$ are missing because the system throws the error: "The [SQL] statement is too long or too complex."

Additionally Appendix A.3 (p. 104) is a detailed breakdown of all performance results we obtained when asking the queries on the different RDBMSs.

$\mathcal{L}_R$/S vs. the state-of-the-art baseline $\mathcal{L}_R$/REF

We now comment on the results we obtained when query reformulations are optimized with Ω using the summary $\mathcal{S}$ of $\mathcal{D}$.

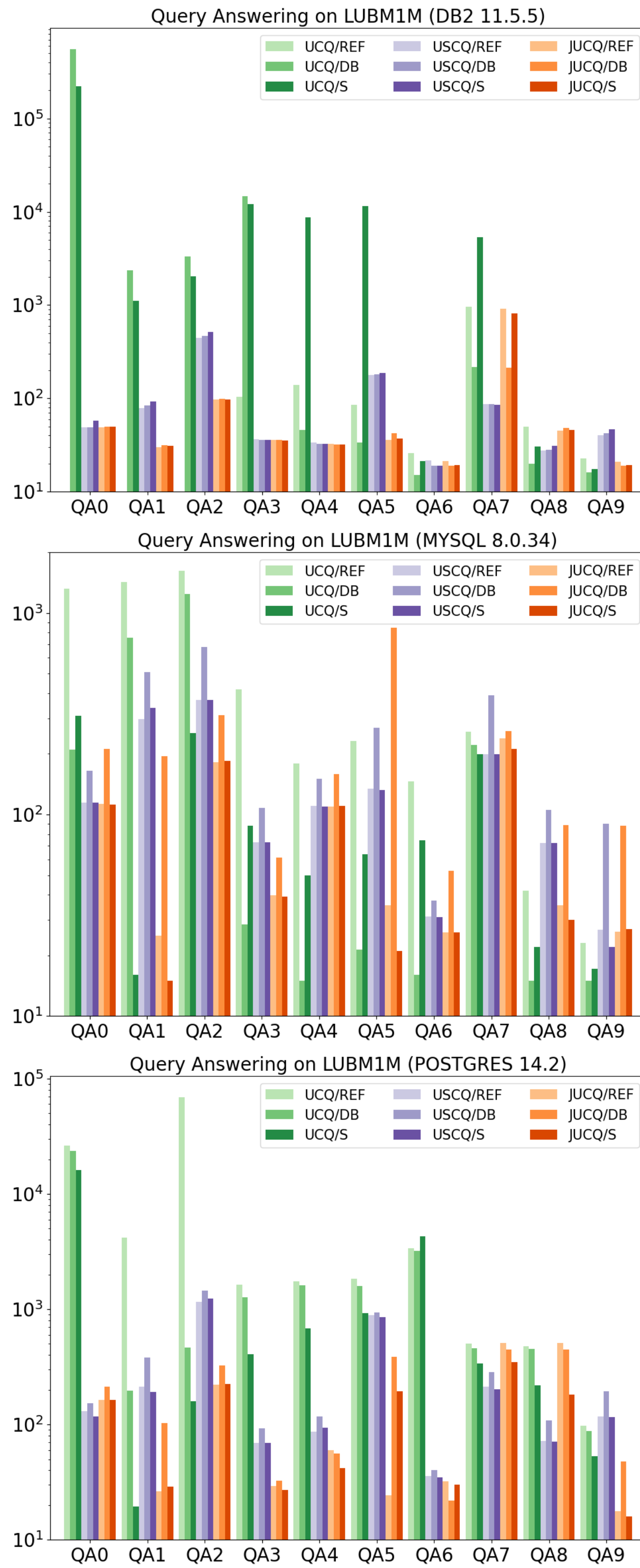


Figure 3.1 – Query answering times (ms, log scale) - LUBM1M

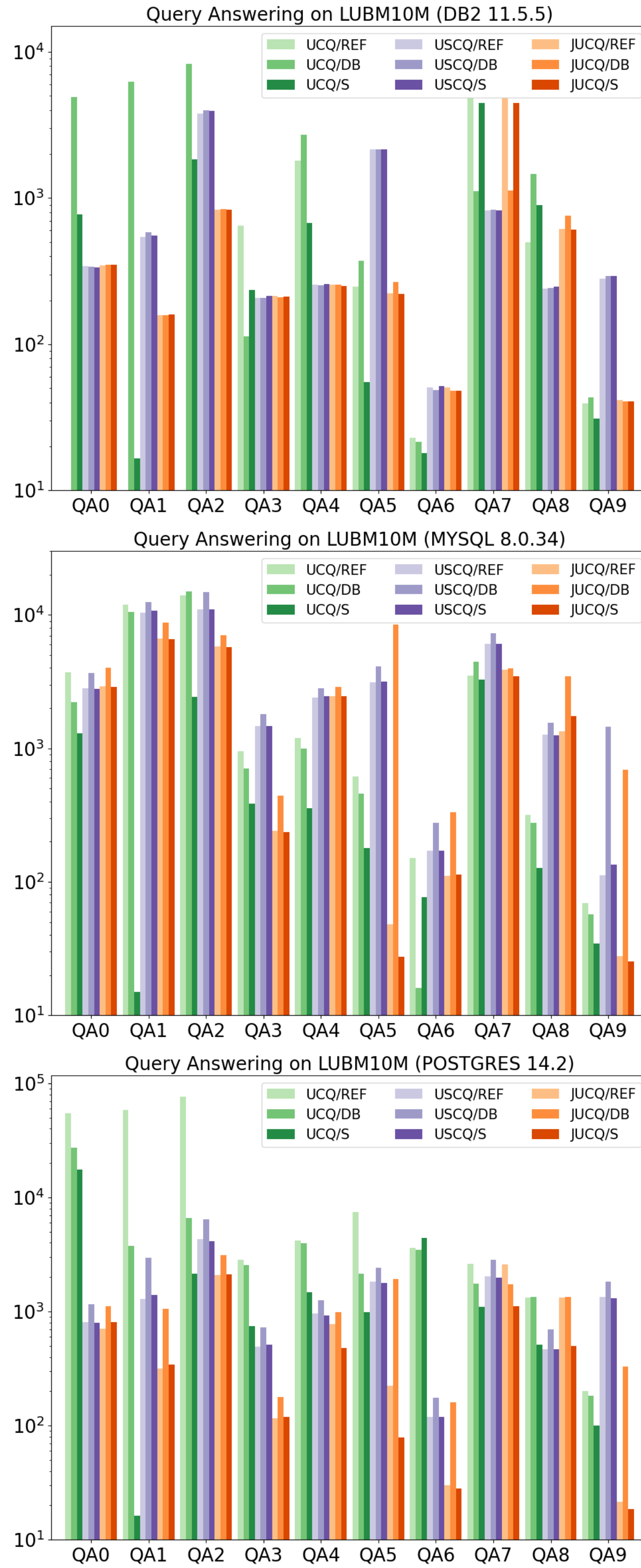


Figure 3.2 – Query answering times (ms, log scale) - LUBM10M

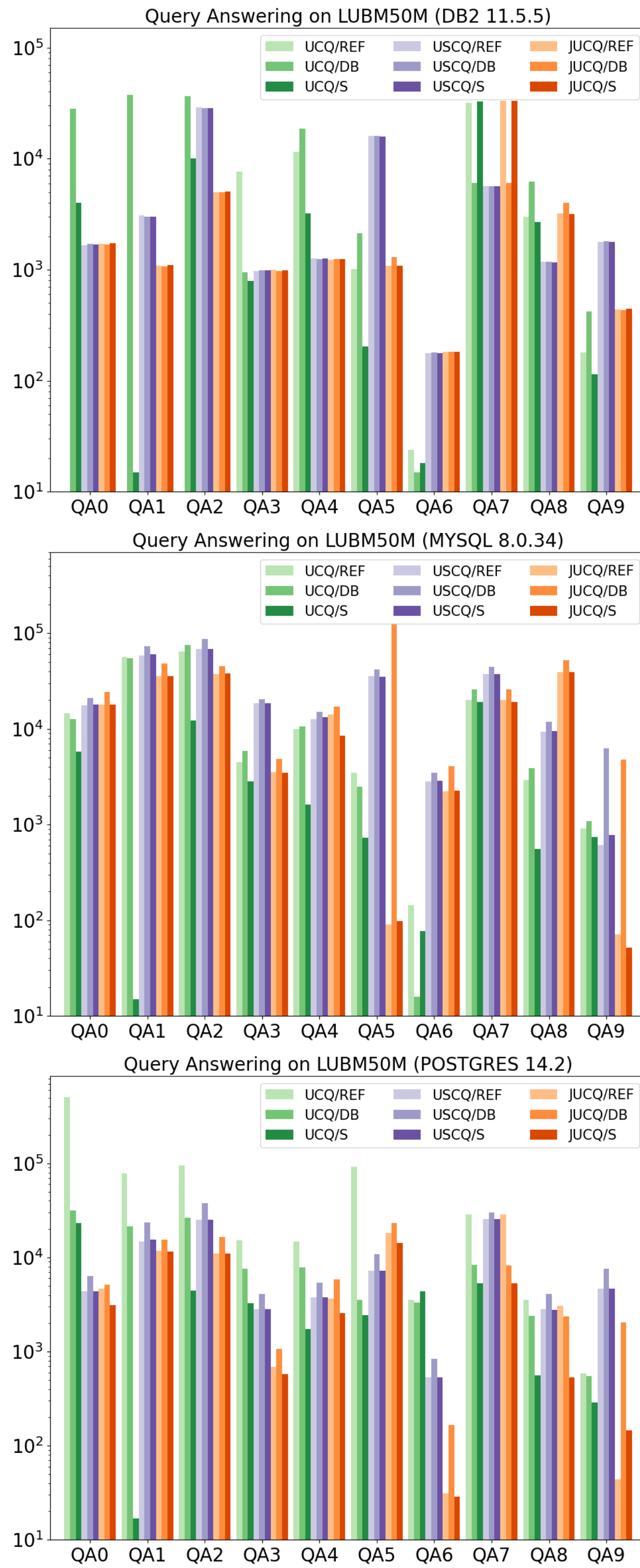


Figure 3.3 – Query answering times (ms, log scale) - LUBM50M

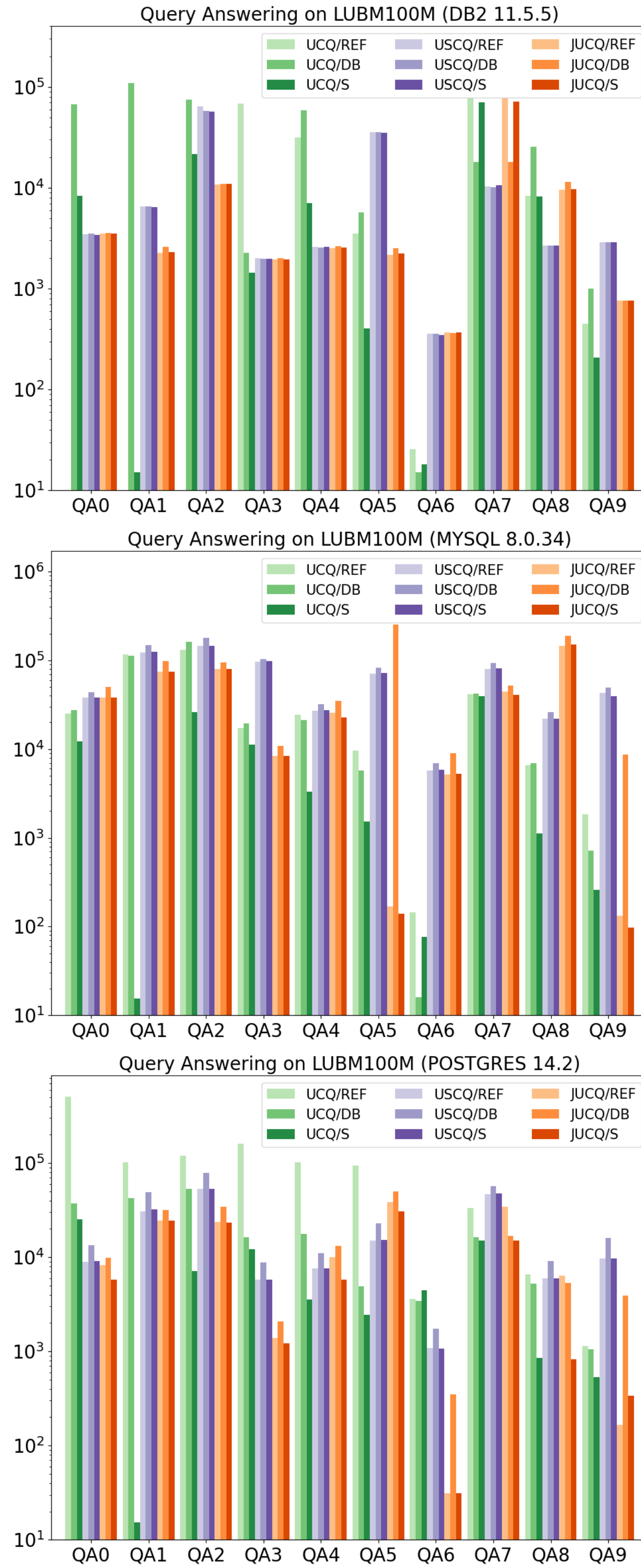


Figure 3.4 – Query answering times (ms, log scale) - LUBM100M

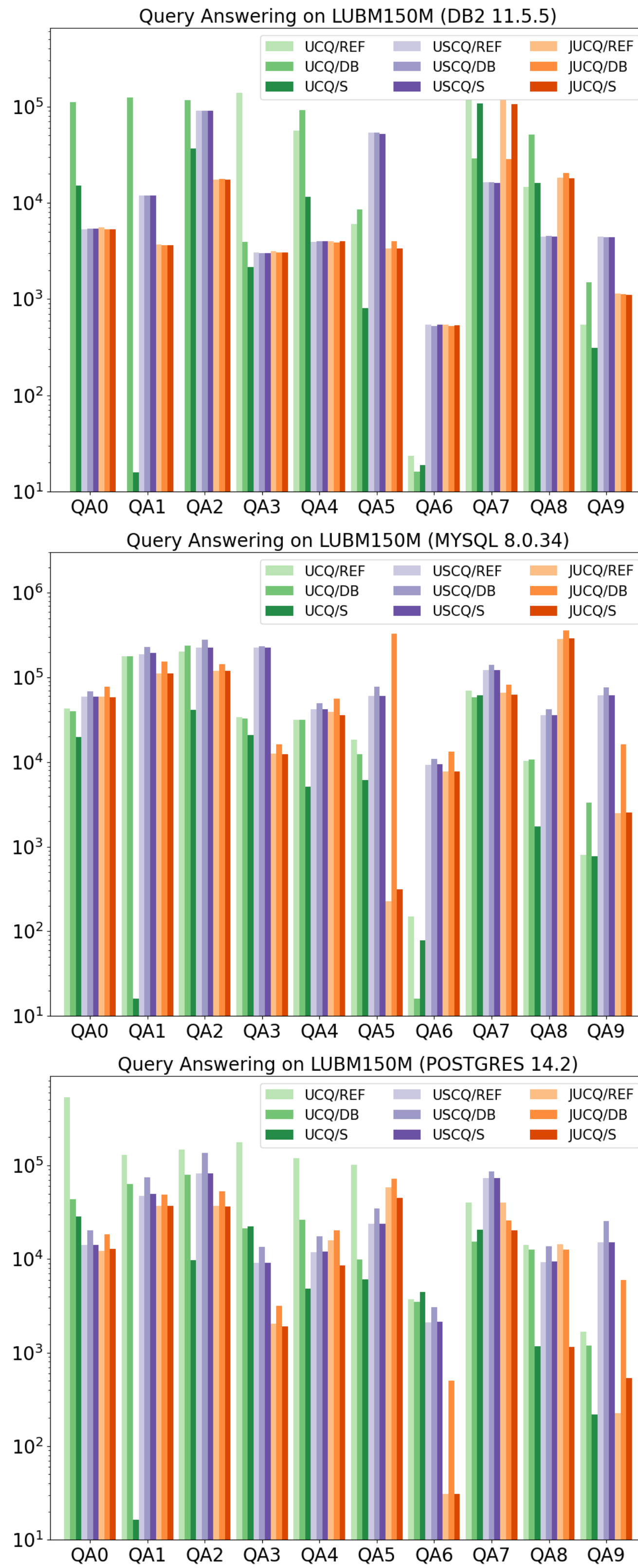


Figure 3.5 – Query answering times (ms, log scale) - LUBM150M

Observations:

Observation 1: UCQ/S vs UCQ/REF

UCQ/S consistently (almost always) and significantly improves the performance of UCQ/REF by **up to 3 orders of magnitude**.

With the exception of $QA6$ on PostgreSQL, we observe that the optimization of UCQs using Ω w.r.t. $\mathcal{S}$ improves the performance of **UCQ/REF** by **up to 3 orders of magnitude**. This is particularly true for queries with a large number of contained CQs. $QA1$, for example, is a large query with 1949 contained CQs and zero answers on all instances of $\mathcal{D}$; it is very costly when asked on an RDBMS without optimization, however after optimizing it with $\mathcal{S}$ its performance is always improved by at least one order of magnitude and in some instances by more than three orders of magnitude (e.g., on LUBM150M using MySQL and PostgreSQL). We highlight that for this particular query the optimization ratio is at 100%. Similarly, $QA0$ and $QA2$ to $QA5$, which have an optimization ratio higher than 79%, perform significantly better after optimization w.r.t. to $\mathcal{S}$.

It is also worth noting that, for some queries, the performance improvement of UCQ/S is more significant when the database grows larger. We particularly observe that when comparing the performances for $QA8$ and $QA9$ over LUBM1M and LUBM150M. These queries are also the ones with the lowest number of contained CQs in our experiments and an optimization ratio lower than 79 %.

In contrast, if we take a closer look at query $QA6$, we can note that it is a query which contains 8 atoms and has 300 total contained CQs, only two of which produce the final answer on $\mathcal{D} = LUBM150M$. Also, as can be seen in Table A.1 (p. 97), it is the only query that contains several constants which specify the name, phone number, email address, etc. For these reasons, $QA6$ has a low optimization ratio, as we are only able to dismiss around 52% of its contained CQs using $\mathcal{S}$ for LUBM150M. On PostgreSQL, this low optimization ratio translates to a high optimization time which does not amortize and therefore leads to a slightly worse performance for **UCQ/S** compared to **UCQ/REF**. This is not the case for DB2 and MySQL.

Observation 2: JUCQ/S vs JUCQ/REF

JUCQ/S *frequently* (around half the time) improves the performance of JUCQ/REF by **up to 1 order of magnitude**.

Query answering performance frequently improves for JUCQs (in half of the cases overall), up to one order of magnitude (e.g., JUCQ/S for $QA8$ using PostgreSQL), otherwise performance is marginally affected.

We remark that when the performance visibly degrades (e.g., $QA9$ on PostgreSQL) it is just in the order of a few tens of ms.

Observation 3: USCQ/S vs USCQ/REF

Query answering performance is marginally affected for USCQs.

In the case of USCQs, we observe that query performance is marginally affected. Although, Ω always completely discards empty queries from USCQs (optimization ratio of 100%, see Table 3.2, p. 48), there is no significant difference between the bars of USCQ/S and USCQ/REF as they exhibit similar performances.

Analysis: The previous observations can be explained with the two following facts, and the optimization ratios obtained with our summaries (Table 3.2, p. 48).

1. Optimizing reformulations with Ω removes CQs with no answer from the top union in UCQs and from the unions on which the top join is performed in JUCQs. In USCQs, single-atom CQs are removed from unions on top of which joins are performed, on top of which the top union is performed.
2. Removing CQs with no answer from a union improves its evaluation time (as it may take time for an RDBMS to find out that a CQ has no answer), while it does not change the size of its output hence the number of tuples to process after this union.

Therefore:

- Optimizing a UCQ reformulation with Ω speeds up its entire evaluation since Ω optimizes its top union. Also, because our summaries allow high optimization ratios for UCQ/S, query answering performance is significantly improved in general. We remark that performance degrades for $QA6$ because the optimization time does not amortize with a low optimization ratio (52.53% using LUBM150M).

- Optimizing a JUCQ reformulation with Ω speeds up the evaluation of its sub-UCQs but does not affect the evaluation time of the top join (as the same tuples must be joined). JUCQ reformulations are thus more difficult to optimize than UCQ ones. This is why query answering performance is "only" frequently improved (in half of the cases) and marginally affected otherwise, even with high optimization ratios for JUCQ/S (84.34% on average using LUBM150M).

- Optimizing a USCQ reformulation with Ω only removes atomic CQs from its inner unions while it does not take time for an RDBMS to figure out that these atomic CQs are empty. The optimization thus marginally affects the evaluation time of these inner unions, and the evaluation time of the subsequent joins and top union is not affected. USCQ reformulations are thus more difficult to optimize than UCQ and JUCQ ones. This is why query answering performance is marginally affected in general, even with maximal optimization ratios for USCQ/S (100% using LUBM150M).

$\mathcal{L}_R$/DB **versus** $\mathcal{L}_R$/REF **and** $\mathcal{L}_R$/S

We now comment on the results we obtained when query reformulations are optimized with Ω using the database $\mathcal{D}$ ($\mathcal{L}_R$/DB), as opposed to using the summary $\mathcal{S}$ ($\mathcal{L}_R$/S), compared with the baseline ($\mathcal{L}_R$/REF).

Observations:

Observation 4: UCQ/DB vs UCQ/REF

Query answering performance is marginally to significantly better with UCQ/DB than with the baseline UCQ/REF, on small databases. However, UCQ/DB *may perform worse than* UCQ/REF when the database size grows.

Depending on the RDBMS considered, UCQ/DB may perform better than UCQ/REF up to 1 order of magnitude on DB2 (e.g., QA3) and two orders of magnitude on PostreSQL (e.g., QA2).

When the database size grows, *UCQ/DB may perform worse than UCQ/REF* because performing query optimization on $\mathcal{D}$ requires significant time which may not amortize. We observe this on MySQL for queries $QA2$ and $QA7$, and on DB2 for QA4, QA5, QA8, QA9. $QA2$ in particular has a significant number of contained CQs (1702), and without optimization, its execution time on MySQL for LUBM150M is around 203 seconds. Meanwhile, optimizing $QA2$ w.r.t. $\mathcal{D} = LUBM150M$ on MySQL requires around 200 seconds and then an additional 39 seconds to execute it making it less efficient than UCQ/REF.

Observation 5: UCQ/DB vs. UCQ/S

UCQ/DB *may perform better than* UCQ/S, **frequently** on small databases and **rarely** on on larger databases.

When comparing the performance of UCQ/DB to UCQ/S, we observe the following: UCQ/DB may perform better on smaller databases while UCQ/S performs better on larger databases. For example, UCQ/DB performs better than UCQ/S on $\mathcal{D} = LUBM1M$ using MySQL for queries: $QA0, QA3, QA4, QA5, QA6, QA8, QA9$. Meanwhile, on $\mathcal{D} = LUBM150M$ using MySQL, UCQ/S performs better than UCQ/DB for all queries except for $QA6$ (due to the low optimization ratio of 52.53%) and $QA7$ (UCQ/S and UCQ/DB have a similar performance).

The reason behind this behavior is that, as the database size increases, UCQ/DB dedicates increasingly more time to optimizing reformulations compared to UCQ/S. However,

this additional time may not pay off at evaluation time, especially when optimization ratios are not low. In other words, when the reformulations of UCQ/S are only moderately more complex than those of UCQ/DB.

Observation 6: JUCQ/REF vs. JUCQ/DB vs. JUCQ/S

JUCQ/DB performance is always worse than the baseline JUCQ/REF and almost always worse than with JUCQ/S

Our experiments show that JUCQ/DB almost always worsens the query performance time as compared to JUCQ/REF, sometimes up to several orders of magnitude (e.g., QA5 on MySQL). The only exceptions to this are QA7 and QA8 on PostgreSQL and DB2 as JUCQ/DB performs similarly or better than JUCQ/REF. Also, in the case of QA7 on DB2, JUCQ/DB performs better than both JUCQ/REF and JUCQ/S.

Observation 7: USCQ/REF vs. USCQ/DB vs. USCQ/S

USCQ/DB performance is always worse than with the baseline USCQ/REF and with USCQ/S.

USCQ/DB always worsens the performance of USCQ/S, hence of USCQ/REF, because with optimization ratios of 100% for USCQ/S for all the queries and all the databases, USCQ/DB and USCQ/S produce the *very same* optimized reformulations, but in general with higher optimization time if Ω uses $\mathcal{D}$ instead of $\mathcal{S}$.

Analysis: These observations are explained by the following:

The extra-time spent by $\mathcal{L}_R$/DB w.r.t. $\mathcal{L}_R$/S in completely optimizing a query reformulation using the database (recall that optimization ratios are of 100% for $\mathcal{L}_R$/DB) does not amortize: optimization time with $\mathcal{L}_R$/DB is in general significantly higher than with $\mathcal{L}_R$/S, because a database is much larger than its summary, while at the same time $\mathcal{L}_R$/DB provides a moderate gain in optimization ratios because they are already very high with $\mathcal{L}_R$/S in general. This is why $\mathcal{L}_R$/DB performs worse than $\mathcal{L}_R$/S overall, and worse than $\mathcal{L}_R$/REF when optimization time is higher than the time saved when the optimized reformulation is evaluated.

QA Performance Conclusion:

Our experiments show that our summaries can be computed fast (linear in data size), small (<10% of data size) and effective in identifying CQs with no answer on a database (92% on average). They also show that, when our Ω optimization function uses summaries, OMQA time performance can be significantly improved (item 3 in Problem 1) for UCQ reformulations in general and frequently for JUCQ reformulations, while performance is marginally affected for USCQ ones.

3.4 Consistency checking performance analysis

We also experimentally studied the performance of our summary-based optimization approach on consistency checking as it turns to be a particular case of query answering in our KB setting (see Chapter 1). Consistency checking queries in our experimental framework are reduced in size since DL-lite$_{\mathcal{R}}$ description logic used to express negative rules with 2 atoms only. Therefore, consistency checking is performed via FO-rewriting of CQs made of two atoms (recall Section 1.3).The aim of this experimental phase is to check whether our summary-based optimization approach brings further improvements to the consistency checking time compared to those of the query answering.

3.4.1 Adding negative rules to $\mathcal{O}$

We handcrafted 5 negative rules which we added to $\mathcal{O}$, as LUBM$_n^{\exists}$ ontologies do not contain negative rules. They express inherent integrity constraints, such as *persons and organizations are disjoint* or *persons and publications are disjoint*, all of which are satisfied by the EUGen database generator:

- $\forall x(Person(x) \wedge Organization(x) \rightarrow \bot)$
- $\forall x(Organization(x) \wedge Student(x) \rightarrow \bot)$
- $\forall x(Organization(x) \wedge Publication(x) \rightarrow \bot)$
- $\forall x(Professor(x) \wedge Department(x) \rightarrow \bot)$
- $\forall x(Professor(x) \wedge Publication(x) \rightarrow \bot)$

Each rule leads to two *equivalent* Ckc and Cki Boolean CQs, for $0 \leq k \leq 4$, that are used to check the performance of consistency checking when the negative rule k is

Query	C0c|C0i	C1c|C1i	C2c|C2i	C3c|C3i	C4c|C4i
#atoms	2	2	2	2	2
#contained CQs w.r.t. $\mathcal{O}$	3,645	1,035	765	630	510
# answers in $ans(q, \mathcal{K})$	0|1	0|1	0|1	0|1	0|1
compl. ratio for UCQ/S	100|100	100|100	100|99.21	100|96.97	100|100
compl. ratio for USCQ/S	100|100	100|100	100|100	100|100	100|100
compl. ratio for JUCQ/S	100|100	100|100	100|99.21	100|96.97	100|100

Table 3.3 – Characteristics of CC queries with $\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$

satisfied or not (recall Section 2): Ckc is asked on every consistent database $\mathcal{D}$ shown in Table 3.1, while Cki is asked on every inconsistent version of these databases (discussed shortly).

The characteristics of these CQs are shown in Table 3.3:

- The *number of atoms*: they correspond to 2 in all cases because of the allowed rules in DL-lite$_{\mathcal{R}}$ (recall Table 1.1, p. 19).
- The *number of contained CQs w.r.t.* $\mathcal{O}$ that a reformulation must encode regardless of the adopted query reformulation language.
- The *number of answers on* $\mathcal{K} = (\mathcal{O}, \mathcal{D})$, which in this case is either 0 or 1 as they are *Boolean* CQs.

Further details on CC queries characteristics, can be found in Appendix A.2 (p. 101).

3.4.2 Adding inconsistencies to databases

We built inconsistent versions of the databases in Table 3.1 as follows. For each Cki, we randomly picked a CQ in its UCQ reformulation that models all the CQs contained in Cki w.r.t. $\mathcal{O}$, i.e., *all* the possible ways of looking for violations of the corresponding negative rule k, which we use to insert an inconsistent fact. Additionally, we created five sets of databases (with sizes ranging from 1M to 150M), each of which is inconsistent w.r.t. just a given Cki as to avoid any biases. For instance, for the negative rule 0 stating that *persons and organizations are disjoint*, we randomly picked within the UCQ reformulation of $C0i$ the CQ $q() = \text{Faculty}(x) \wedge \text{University}(x)$, based on which we randomly selected a faculty f in $\mathcal{D}$ in order to add it as a university in $\mathcal{D}$. We do that for all database sizes.

3.4.3 Performance analysis

The consistency checking times we measured are reported for all the databases we considered: LUBM1M (Figure 3.6, p. 63), LUBM10M (Figure 3.7, p. 64), LUBM50M (Figure 3.8, p. 65), LUBM100M (Figure 3.9, p. 66) and finally LUBM150M (Figure 3.10, p. 67).

We followed the same strategy as the one used to perform query answering experiments (Recall Section 3.3, p. 47). Because in DL-lite$_{\mathcal{R}}$ the left-hand side of a negative rule consists of a conjunction of two atoms only, for the purpose of consistency checking the JUCQ and UCQ reformulations of a query are equivalent. We therefore only report the results of UCQ and USCQ using the 3 strategies: $\mathcal{L}_R$/REF, $\mathcal{L}_R$/DB and $\mathcal{L}_R$/S.

Similarly to query answering, every reported time is an average over 5 "hot" query runs, i.e., the first "cold" query run is discarded. Times are reported per RDBMS, Ckc and Cki for $0 \leq k \leq 4$, query reformulation language, and according to the reformulation strategy.

Finally, we note that UCQ/REF light green bars for $C0c/C0i$ are missing for DB2 because the system throws the error "The [SQL] statement is too long or too complex."

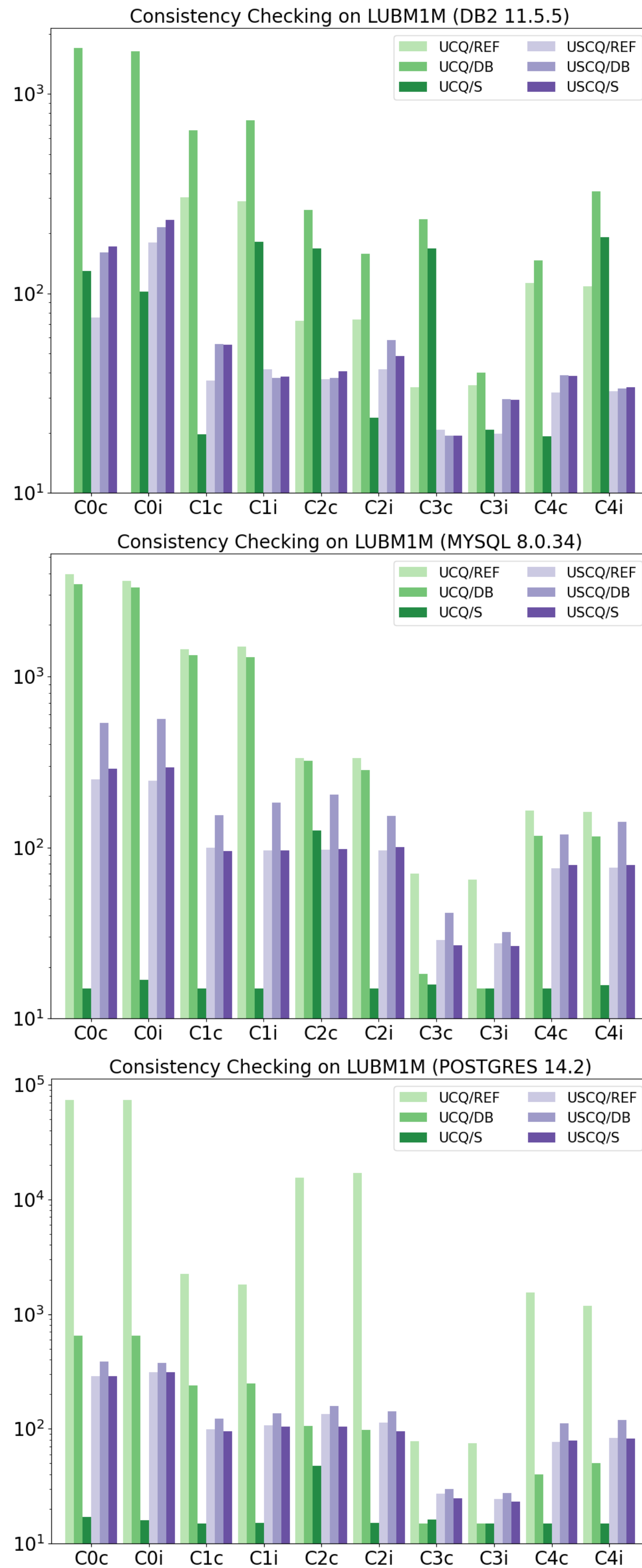


Figure 3.6 – Consistency checking times (ms, log scale) - LUBM1M

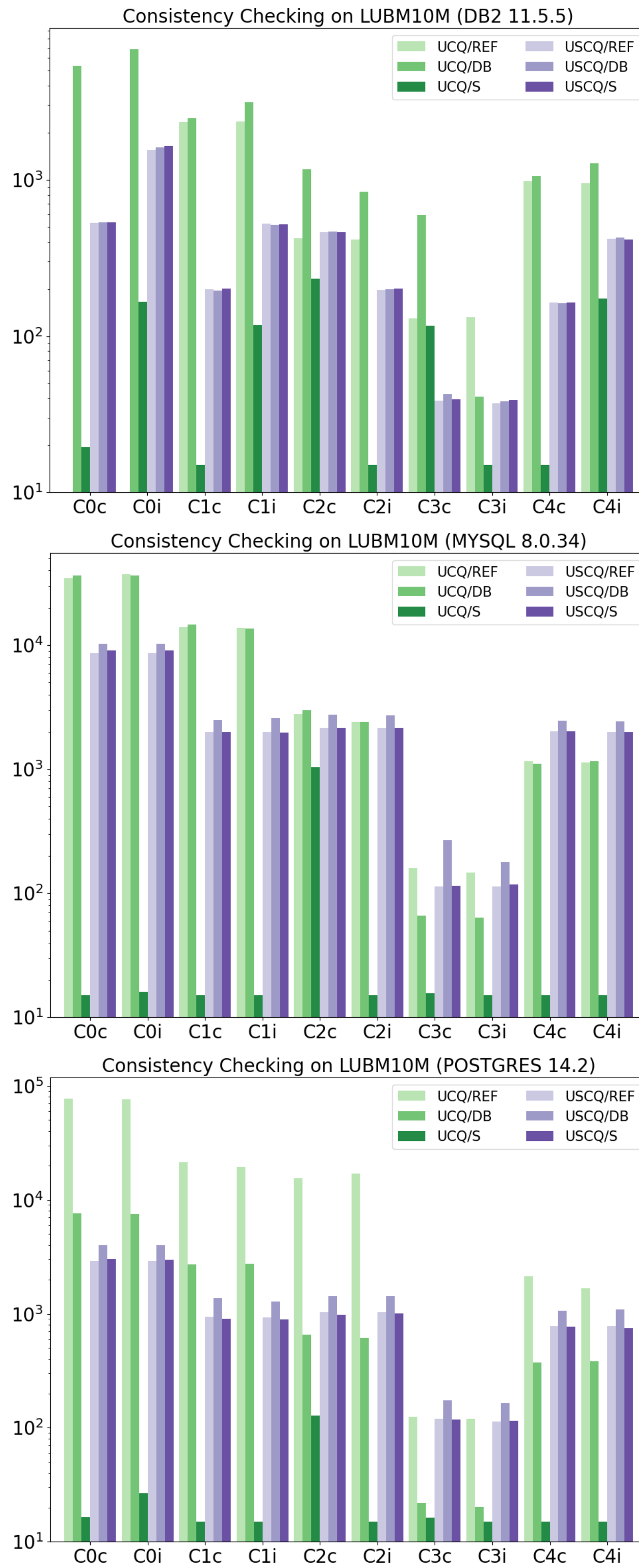


Figure 3.7 – Consistency checking times (ms, log scale) - LUBM10M

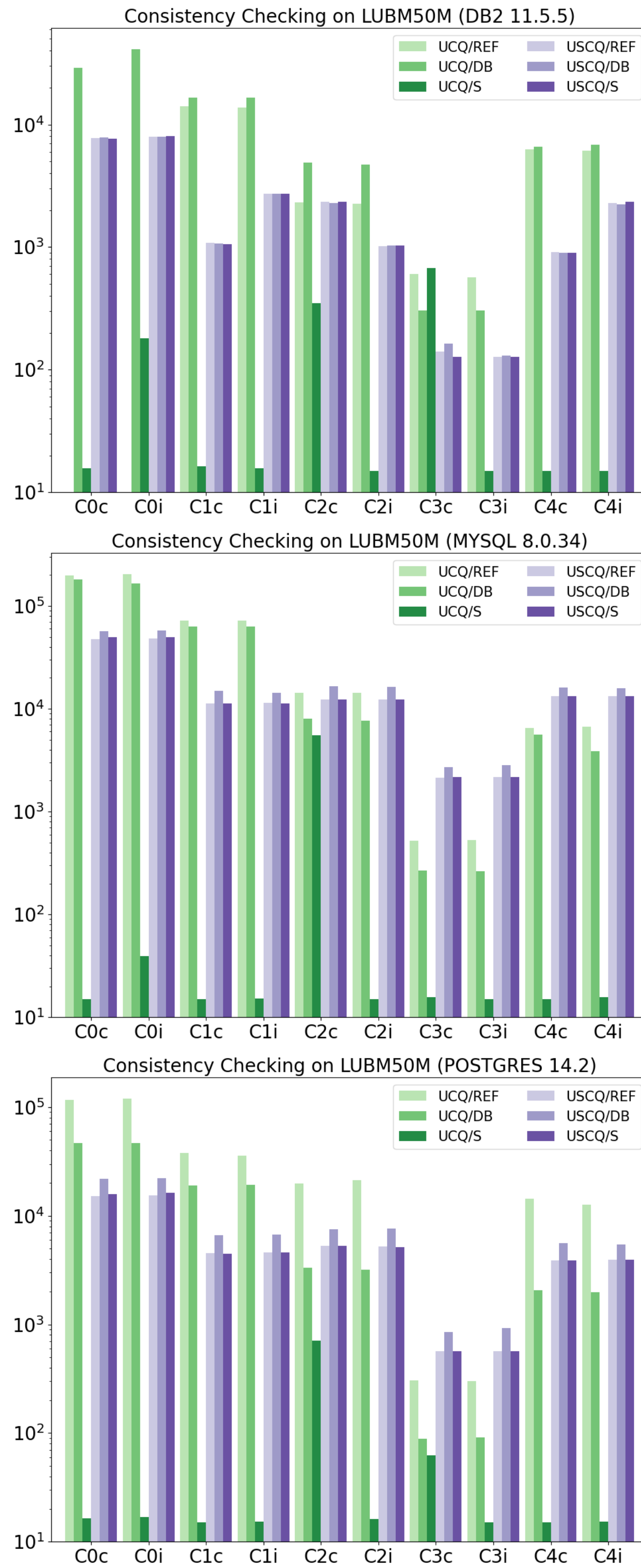


Figure 3.8 – Consistency checking times (ms, log scale) - LUBM50M

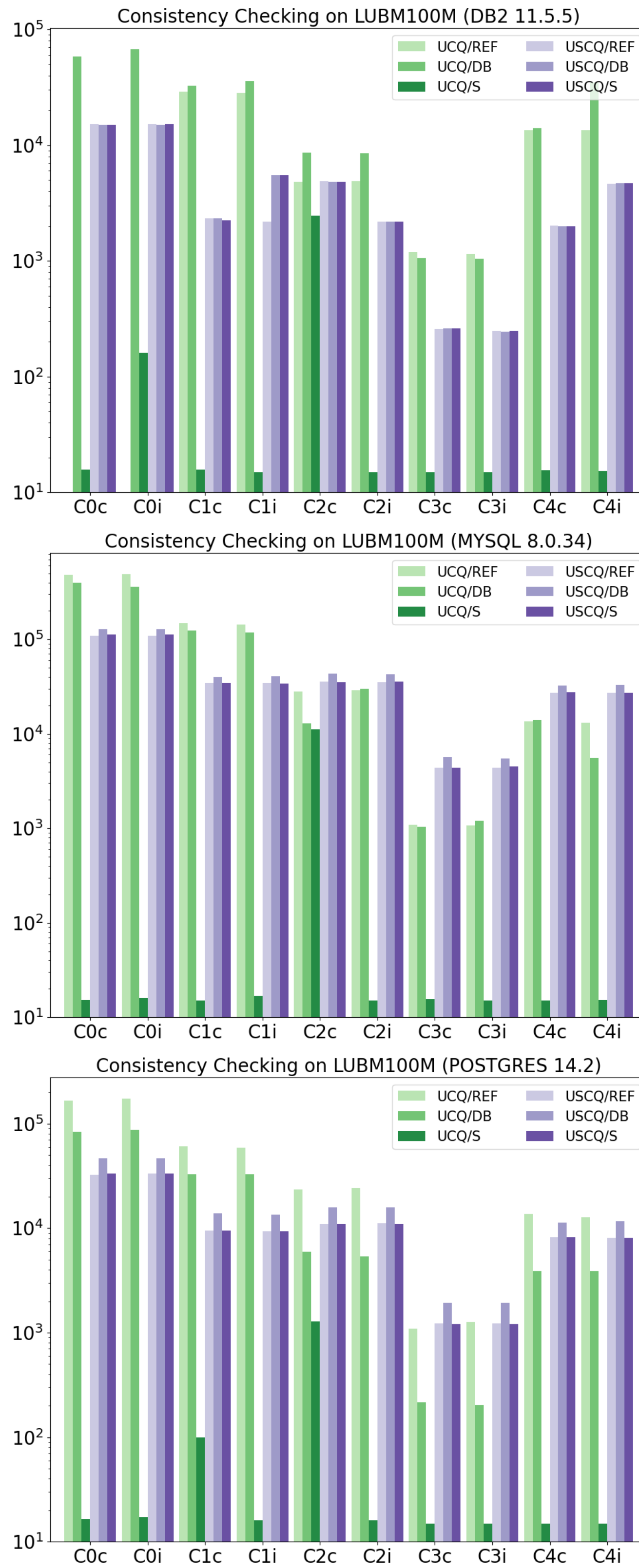


Figure 3.9 – Consistency checking times (ms, log scale) - LUBM100M

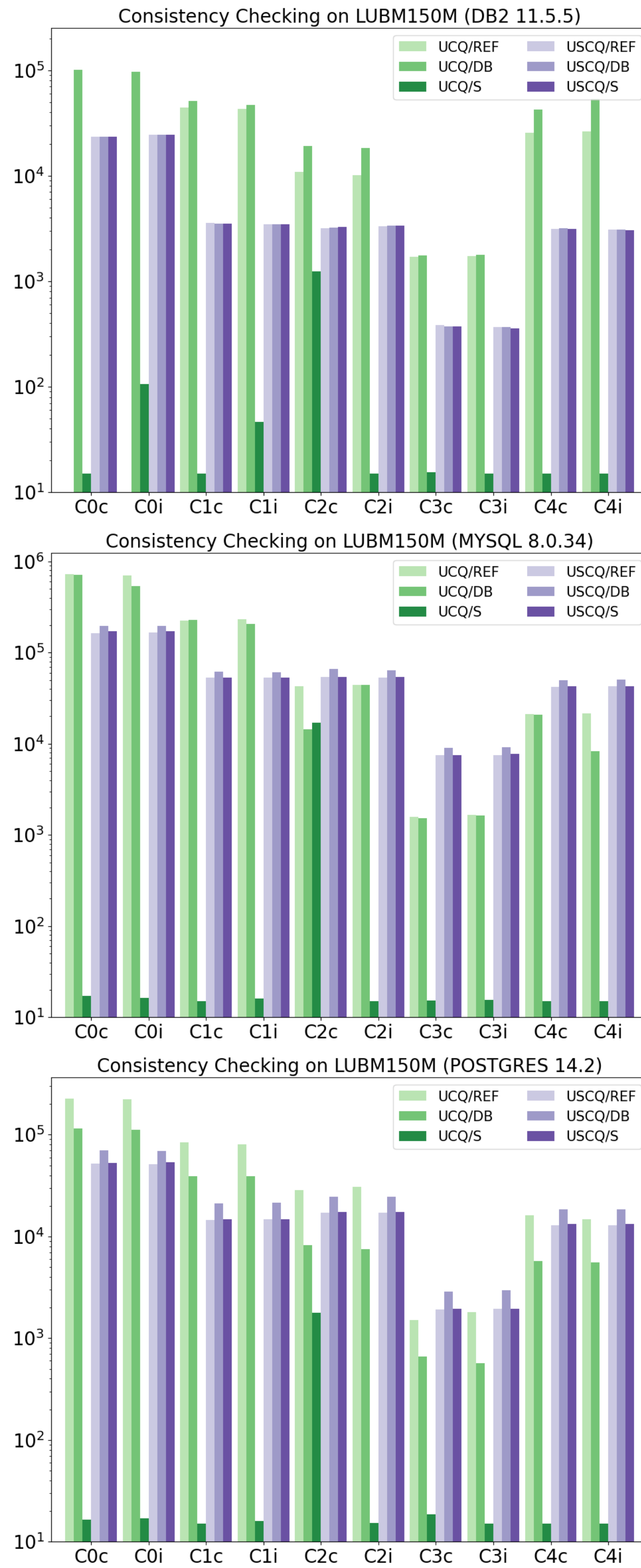


Figure 3.10 – Consistency checking times (ms, log scale) - LUBM150M

$\mathcal{L}_R$/S vs. the state-of-the-art baseline $\mathcal{L}_R$/REF

Observations:

Observation 8: UCQ/S vs UCQ/REF

UCQ/S consistently (almost always) and significantly improves the performance of UCQ/REF by **up to 4 orders of magnitude**.

Similarly to our observations on query answering, we observe that UCQ queries optimized with Ω using $\mathcal{S}$ are significantly improved, up to four orders of magnitude (e.g., C0c and C0i on PostgreSQL and MySQL). The only exceptions being C3c on DB2 for LUBM1M and LUBM50M.

Observation 9: USCQ/S vs USCQ/REF

Consistency checking query performance is marginally affected for USCQs, regardless of the consistency of the database.

Analysis: We observe similar behaviors when it comes to query evaluation of consistency checking queries, as compared to query answering queries. These observations can be explained by the following:

Consistency checking queries are a simpler (particular case) of query answering queries. But the size of their query reformulations are relatively high. In the case of UCQs, Ω is able to completely and rapidly discard all irrelevant contained queries using the summary $\mathcal{S}$ on consistent databases and most of them on inconsistent databases (>96.97% on LUBM150M), which translates into a better performance.

$\mathcal{L}_R$/DB versus $\mathcal{L}_R$/REF and $\mathcal{L}_R$/S

Observations:

Observation 10: UCQ/DB vs UCQ/REF and UCQ/S

UCQ/DB may occasionally perform better than UCQ/REF but always performs worse than UCQ/S.

When using MySQL and PostgreSQL, UCQ/DB generally slightly improves the performance of UCQ/REF. In contrast, when using DB2, UCQ/DB always performs worse than UCQ/S; the only exception being C3c on DB2 for LUBM50M.

Observation 11: USCQ/DB vs USCQ/S vs USCQ/REF

USCQ/DB may significantly worsen the performance of USCQ/REF.

On both consistent and inconsistent databases, we observe that USCQ/DB generally has a worse performance than both USCQ/REF and USCQ/S, especially when using MySQL and PostgreSQL.

Analysis: Similarly to query answering, these observations can be explained because of the extra-time spent by $\mathcal{L}_R$/DB w.r.t. $\mathcal{L}_R$/S in completely optimizing a query reformulation using the database (recall that optimization ratios are of 100% for $\mathcal{L}_R$/DB) which does not amortize.

CC Performance Conclusion:

Similarly to query answering, the UCQ and JUCQ reformulations of consistency checking queries are significantly improved when using our Ω optimization function.

3.5 Conclusion

In this chapter, we discussed the results of our experiments. We showed that the time performance of query answering and consistency checking via FO-rewriting, can be significantly improved in general when UCQ and JUCQ reformulations are used.

In the next chapter, we discuss the implementation of our optimization framework in our OptiRef prototype.

Chapter 4

OptiRef: An optimization Framework and Visualisation tool

Contents

4.1 Overview **71**
4.1.1 Achitecture 71
4.1.2 Technologies 72
4.2 Implementation **73**
4.2.1 Creating the datasets 73
4.2.2 The KB-Ref algorithm 74
4.3 OptiRef GUI **75**
4.4 Conclusion **84**

In this chapter, we discuss the technical implementation of our work in the OptiRef tool. First, we give an overview of the whole project (Section 4.1), then we discuss the implementation (Section 4.2), before finally showcasing our OptiRef GUI (Section 4.3).

4.1 Overview

4.1.1 Achitecture

The technical work done in the context of this thesis was implemented according to Figure 4.1 and can be divided into two main parts:

Front-end We designed the OptiRef GUI, which is a user-friendly visual interface that allows users to visualize the outcome of our conducted experiments. It also provides users with the capability of interacting with the KBs directly by asking custom queries.

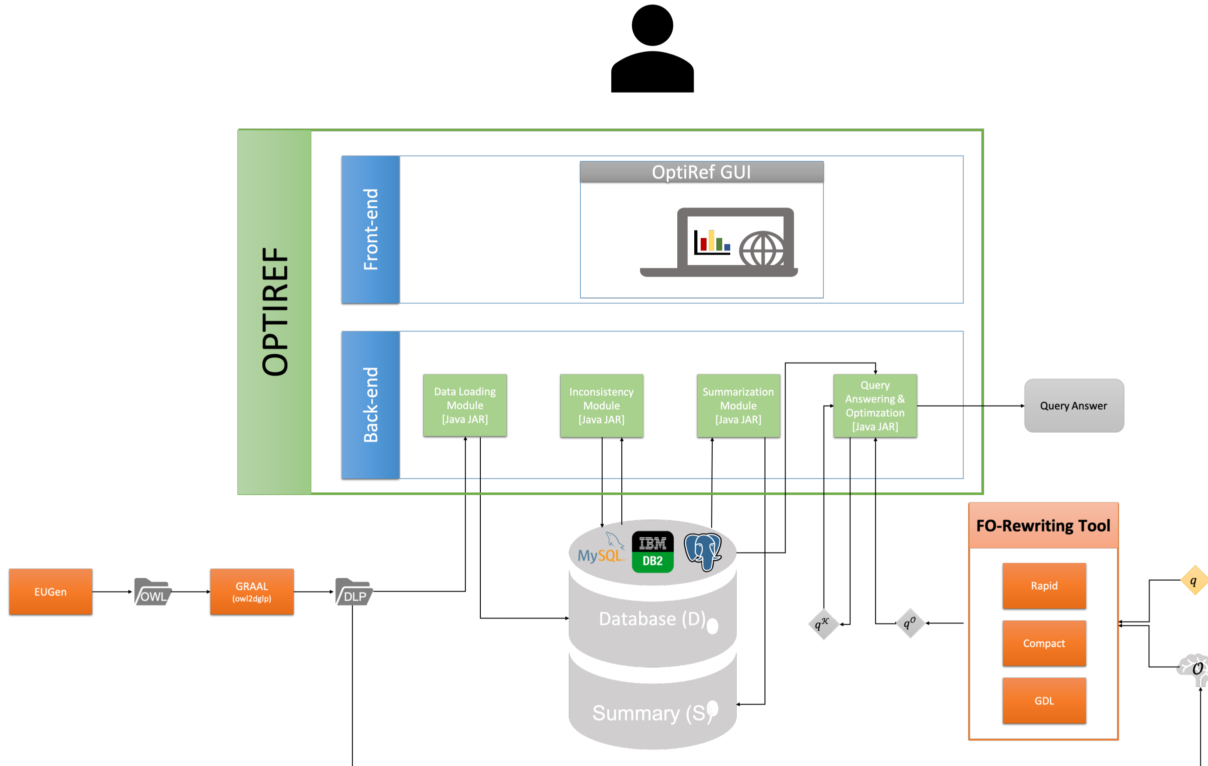


Figure 4.1 – OptiRef Architecture

Back-end We developed several modules that allow us to set up and handle KB databases on our server:

- A module that generates databases from the raw data files and stores them in an RDBMS.
- A module that introduces inconsistencies to databases.
- A module that creates a summary of a given database according to our equivalence relation $\equiv_\Omega$ (Recall Section 2.4, p. 39).
- A module that handles query answering on our KBs according to the different reformulation languages and to the optimization method considered.

4.1.2 Technologies

Back-end technologies Since most of the tools we rely on in our experiments are available as Java libraries (e.g., Rapid, Compact,GDL), we used Java 14 to deploy these tools and to develop our modules.

Front-end technologies The OptiRef GUI interface is developed independently from the back-end and uses the following technologies:
- PHP version 7.4.3 which allows the interface to communicate with the back-end and fetch the data needed for the visualization.
- jQuery, Chartjs and D3.js which are the necessary Javascript tools needed to generate both simple and complex charts.

4.2 Implementation

4.2.1 Creating the datasets

From owl to integer-encoded databases To create the databases we used for our experiments, we proceeded according to Figure 4.1 (p. 72).

We used the EUGen generator which allows us to create synthetic data in *owl* format using the Univ-Bench ontology, and to generate datasets that can grow as large as needed, depending on the number of universities we consider. We generated data for a total of 1200 universities, which turns out to be the equivalent of a little over 150 million database tuples. Each university is represented by a set of around 20± *owl* files containing a varying number of instances. Figure A.1 (p. 107) is an extract from the *owl* files we generated.

We consider a database of 150 million tuples to be reasonably large enough as to allow us to fairly evaluate the different approaches we consider. It also allows us to better observe the performance benefits brought forth by our proposed method.

After generating the complete set of .owl files, we selected subsets of them in order to generate databases of increasing sizes. Five databases were generated s.t. : LUBM1M $\subseteq$ LUBM10M $\subseteq$ LUBM50M $\subseteq$ LUBM100M $\subseteq$ LUBM150M.

Database Name	Database Size	# Universities	# Tuples in $\mathcal{D}$
LUBM1M	1 million triples	9	1,186,832
LUBM10M	10 million triples	75	10,793,976
LUBM50M	50 million triples	373	53,328,357
LUBM100M	100 million triples	745	106,596,211
LUBM150M	150 million triples	1118	159,899,201

Table 4.1 – Size of LUBM databases

Table 4.1 shows the name of the database, its approximate size (or the order of mag-

nitude), the number of universities it corresponds to and the number of tuples that are actually in it.

To obtain the data layout from [14], we proceeded over several steps:

1. We convert our .owl files (e.g., Figure A.1, p. 107) into .dlp files (e.g., Figure A.2, p. 108) using the owl2dlgp program, which is an easier format to handle data in [59].
2. Using the .dlp files, we generate relational databases s.t. unary relations are modeled using unary tables and binary relations are modeled using binary tables. All values are encoded using a dictionary *binary* table.
3. We store the resulting databases in PostgreSQL, MySQL and DB2.

This method was used to generate all the databases for the purpose of our experiments.

4.2.2 The KB-Ref algorithm

Algorithm 1: optimization algorithm for OMQA via FO-rewriting

Input : a CQ q,
a $\mathcal{L}_K$ KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$,
a summary $\mathcal{S}$ of $\mathcal{D}$, and
a query reformulation algorithm O−ref
for the FO-rewriting setting $(CQ, \mathcal{L}_K, \mathcal{L}_R)$ with $\mathcal{L}_R$ in $(\wedge, \vee)$-CQs

Output: a reformulation $q^{\mathcal{K}}$ of q w.r.t. $\mathcal{K}$

1 $q^{\mathcal{O}} \leftarrow \text{O}{-}\text{ref}(q, \mathcal{O})$; *// reformulation of q w.r.t. $\mathcal{O}$*
2 $q^{\mathcal{K}} \leftarrow \Omega(q^{\mathcal{O}}, \mathcal{S})$; *// reformulation of q w.r.t. $\mathcal{O}$ that is optimized for $\mathcal{D}$*
3 **return** $q^{\mathcal{K}}$

We propose the KB-Ref algorithm which optimizes $(\wedge, \vee)$-CQ reformulations of CQs asked on KBs. This algorithm uses the Ω function we defined in Section 2.3 and the summaries from Section 2.4 to compute an optimized version of an incoming query.

Given *any* off-the-shelf query reformulation algorithm O−ref for FO-rewriting, KB−ref starts by using O−ref to compute a reformulation $q^{\mathcal{O}}$ of a CQ q w.r.t. the ontology $\mathcal{O}$ of a $\mathcal{L}_K$ KB $\mathcal{K} = (\mathcal{O}, \mathcal{D})$ (line 1). Then, it uses our Ω function to optimize the reformulation $q^{\mathcal{O}}$ for the database $\mathcal{D}$ of the KB $\mathcal{K}$ (line 2).

4.3 OptiRef GUI

We developed the OptiRef GUI, which is a web-based interface designed for experimenting with FO-rewriting and its optimization. This tool provides an intuitive platform for users to explore the intricacies of FO-rewriting, making it accessible to both novices and experts. In this section, we present the tool in details in order to highlight its benefits.

A live demo of the GUI is available at: https://shaman.enssat.fr/optiref/.

Figure 4.2 – OptiRef GUI Settings

The landing page of our interface (Figure 4.2) is a settings page which allows users to select the data they would like to visualize. Users can select the *benchmark* as well as the *database* size from the set of databases we generated, and the *RDBMS(s)* over which they were generated. They can also choose to either select a predefined and pre-computed *query* or to write their own query for it to be processed directly on the database. Finally, users can choose the *reformulation language* they are interested in visualizing and whether they want to see the the results of optimized FO-rewriting w.r.t. to $\mathcal{S}$ *(Summary-based optimization)* and/or w.r.t. to $\mathcal{D}$ *(Database-based optimization)*.

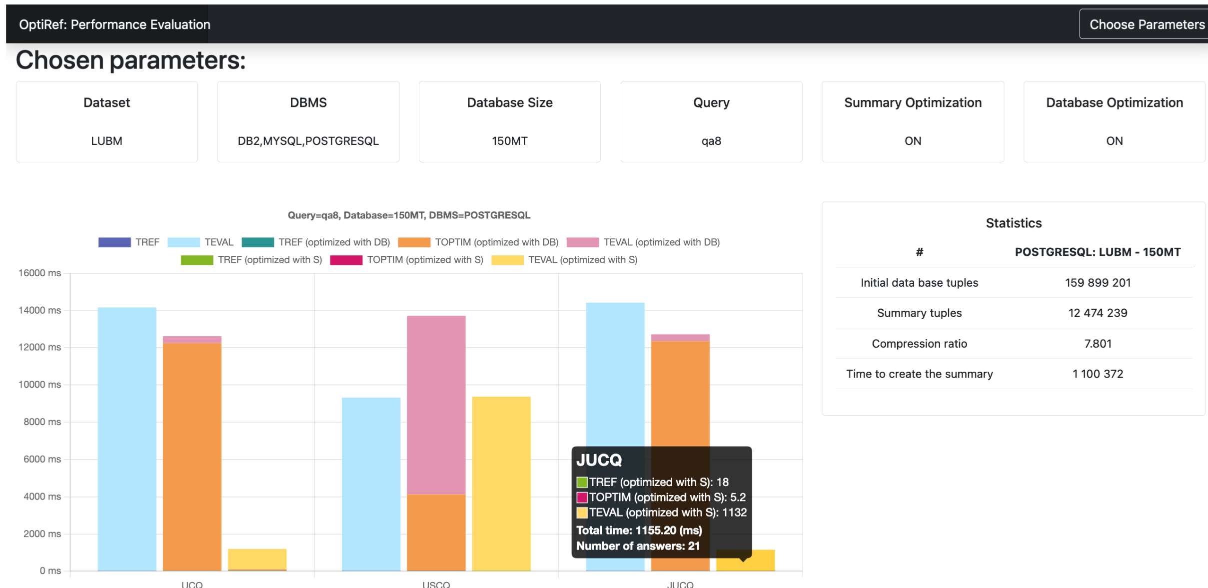


Figure 4.3 – OptiRef GUI Bar Chart: QA8 asked on PostgreSQL using LUBM150M

Figure 4.3 is a screen capture of what the user can visualize according to a chosen set of parameters. In this case, we show the results of query QA8 asked on the *LUBM 150M* database using PostgreSQL. For each of the reformulation languages (UCQ, USCQ and JUCQ) we considered, the OptiRef tool reports the performance time in form of a bar chart s.t. :

- The first bar represents the performance of the reformulated query before optimization.
- The second bar represents the performance of the reformulated query after optimization w.r.t. the database.
- The third bar represents the performance of the reformulated query after optimization w.r.t. the summary.

Each bar is a sum of: reformulation time (TREF), optimization time (TOPTIM) and evaluation time (TEVAL). Time is reported in a linear scale.

To further understand the benefits of our optimization framework, it is recommended to discard the reformulation time (TREF), focusing only on the optimization time (TOPTIM) and the evaluation time (TEVAL), because in our framework, query reformulations are considered as an input. This corresponds to the reported results in Chapter 3. The interface allows us to therefore filter the parts of the bars we may consider as irrelevant to

our analysis by clicking on the elements we would like to discard in the legend. Focusing only on evaluation time (TEVAL) on the other hand may bias the analysis of the user: $\mathcal{L}_{\mathbf{R}}$/DB is evaluated faster than $\mathcal{L}_{\mathbf{R}}$/S because it is the optimal version of an incoming query, however the time to compute it is generally quite costly.

The bar chart on the left hand side, allows us to conduct a performance analysis over the considered queries. We are able to compare the different reformulation languages from the literature to each other: we conclude that UCQs generally have the worst performance. We can also compare reformulation languages to their optimized versions: we observe that optimizing the queries generally improves the performance of the query.

Furthermore, the statistics on the right hand side offer additional information about the database we consider: the number of tuples it contains, the number of tuples in its summary, the compression ratio and the time necessary to compute our summary.

A good use-case of OptiRef, is to examine the bars over three steps:

1. A first examination of the state-of-the art baseline (i.e., by leaving *optimizations* unchecked) which corresponds to $\mathcal{L}_{\mathbf{R}}$/REF in our experiments.
2. A next examination of the comparison between $\mathcal{L}_{\mathbf{R}}$/REF and the optimization w.r.t. the database, which corresponds to $\mathcal{L}_{\mathbf{R}}$/DB in our experiments.
3. A last examination being a comparison between $\mathcal{L}_{\mathbf{R}}$/REF, $\mathcal{L}_{\mathbf{R}}$/DB and the optimization w.r.t. the summary which which corresponds to $\mathcal{L}_{\mathbf{R}}$/S in our experiments.

By doing so, we are able to first, visually compare the different FO-rewriting from the literature to each other. Next we can see the performance benefits or drawbacks of using the database when optimizing a query with the Ω function. Finally, we can examine the benefits of using the summary when optimizing a query with the Ω function.

OptiRef also allows us to inspect queries in details when clicking on any of the bars, e.g., clicking on the first UCQ bar for QA8 in Figure 4.3, p. 76 above shows a breakdown of this UCQ according to the $\mathcal{L}_{\mathbf{R}}$/REF strategy.

Figure 4.4 is an inspection of the UCQ reformulation of QA8, generated using Rapid, that is obtained after clicking on the first UCQ bar from figure 4.3. We chose to represent a UCQ in a tree format with a "Union" as the central root and a the CQs as the leaves.

In this example, the UCQ is a union of 144 CQs (leaf nodes) most of which are empty and are color coded in *red*, and only 11 of which contribute to the final answer and are

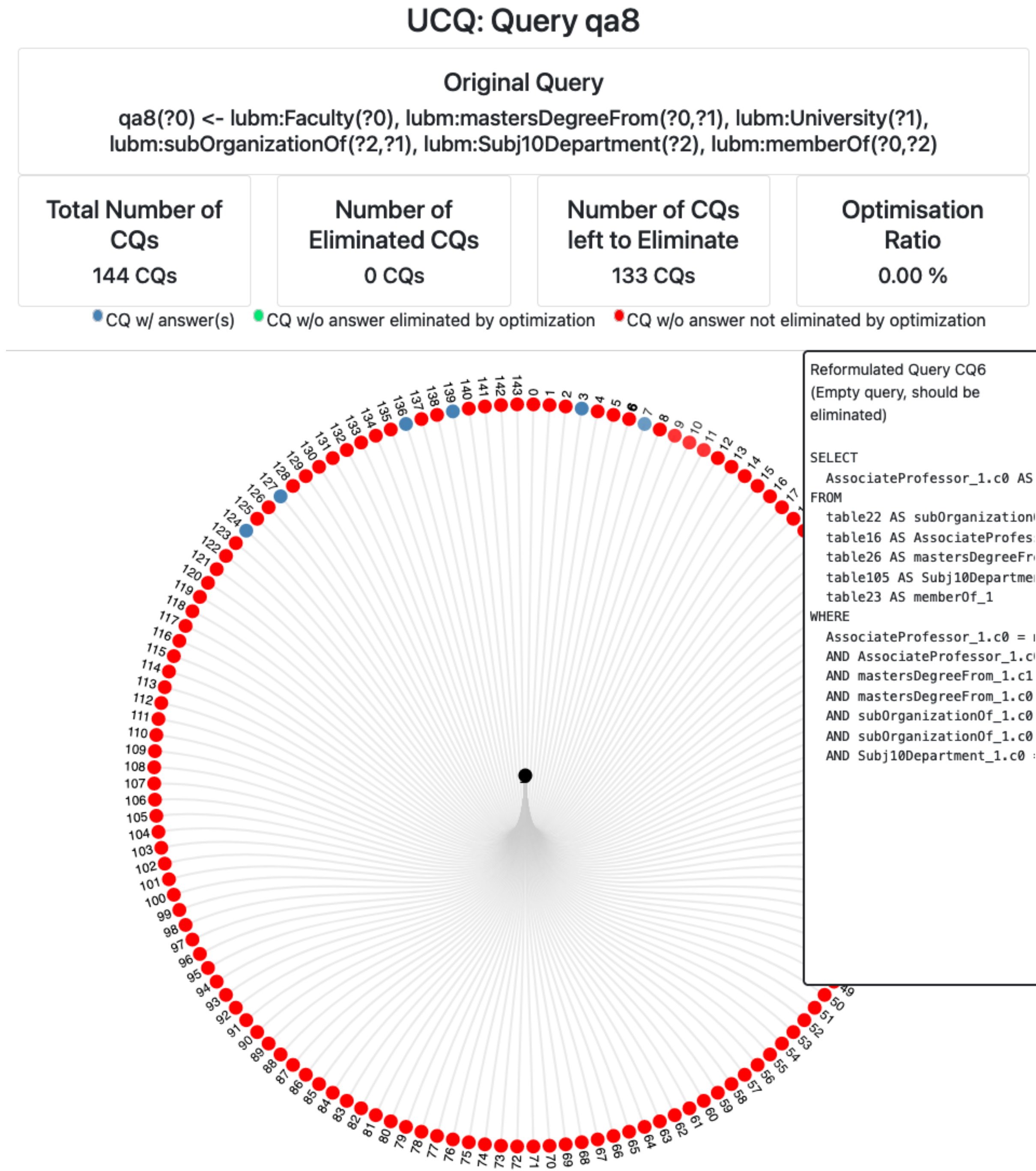


Figure 4.4 – UCQ/REF Reformulation Inspection of QA8

color coded in *blue*. Selecting any of the nodes allows the user to see the SQL query it corresponds to, as illustrated by the figure.

UCQ: Query qa8

Original Query

qa8(?0) <- lubm:Faculty(?0), lubm:mastersDegreeFrom(?0,?1), lubm:University(?1), lubm:subOrganizationOf(?2,?1), lubm:Subj10Department(?2), lubm:memberOf(?0,?2)

Total Number of CQs	Number of Eliminated CQs	Number of CQs left to Eliminate	Optimisation Ratio
144 CQs	133 CQs	0 CQs	100.00 %

CQ w/ answer(s) · CQ w/o answer eliminated by optimization · CQ w/o answer not eliminated by optimization

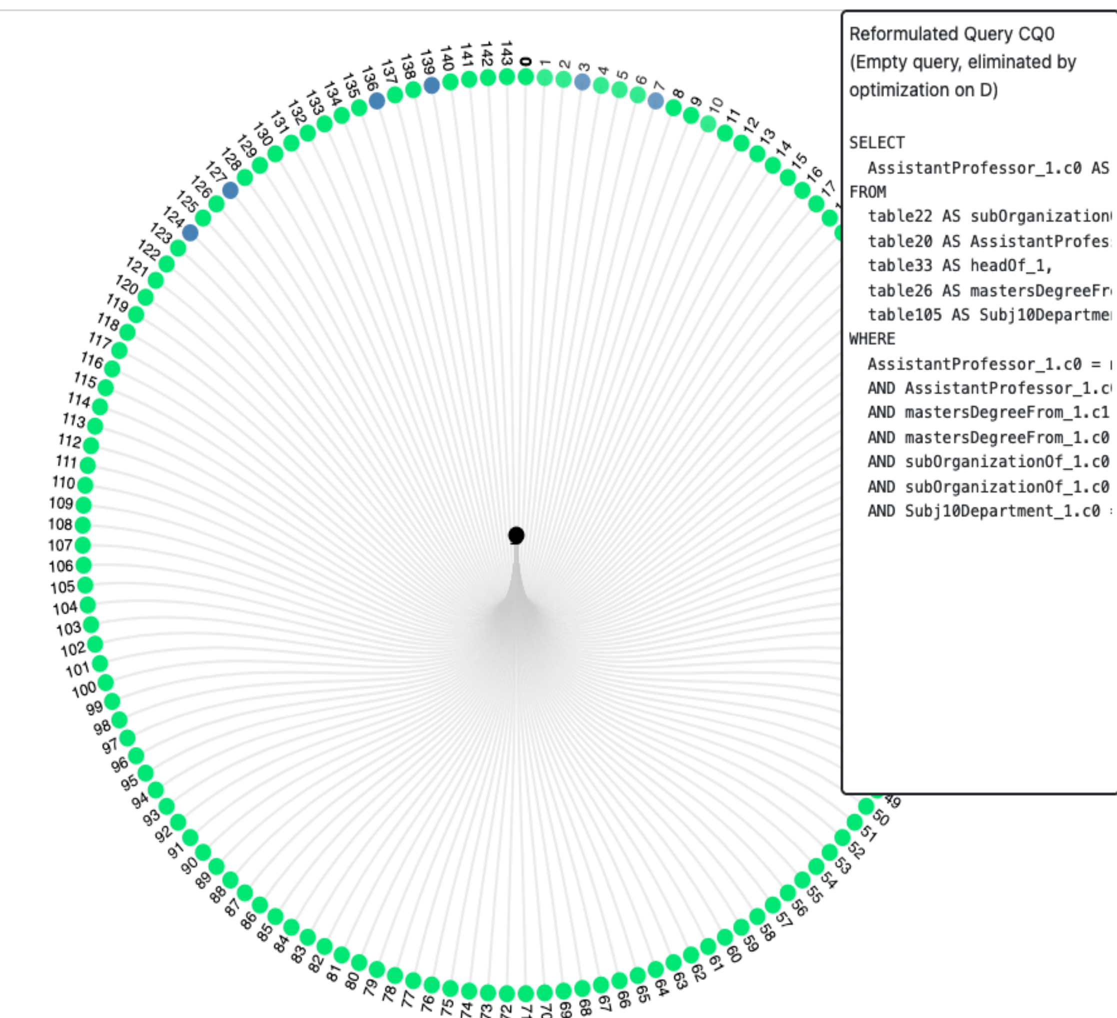


Figure 4.5 – UCQ/DB Reformulation Inspection of QA8

UCQ: Query qa8

Original Query

qa8(?0) <- lubm:Faculty(?0), lubm:mastersDegreeFrom(?0,?1), lubm:University(?1), lubm:subOrganizationOf(?2,?1), lubm:Subj10Department(?2), lubm:memberOf(?0,?2)

Total Number of CQs	Number of Eliminated CQs	Number of CQs left to Eliminate	Optimisation Ratio
144 CQs	93 CQs	40 CQs	69.92 %

● CQ w/ answer(s) ● CQ w/o answer eliminated by optimization ● CQ w/o answer not eliminated by optimization

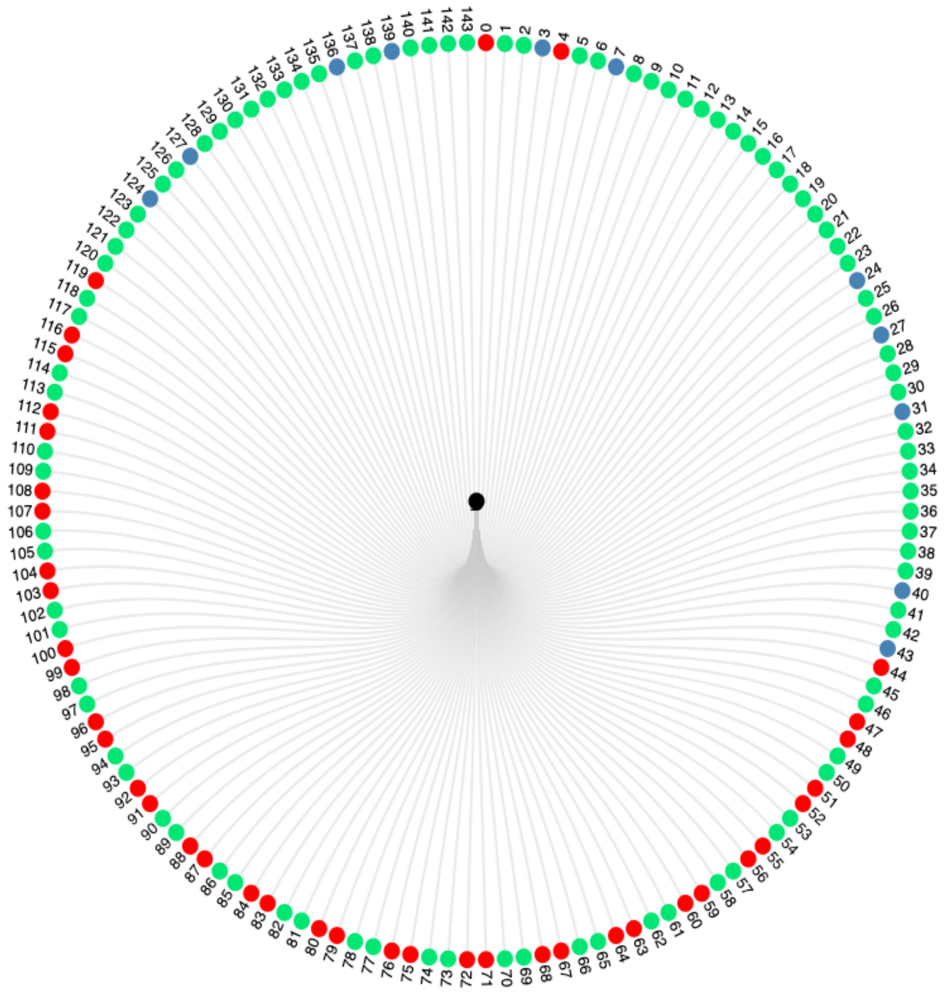


Figure 4.6 – UCQ/S Reformulation Inspection of QA8

If we were to optimize this UCQ w.r.t. the database, all of the red nodes will be eliminated and we can see an improvement in the performance time (second UCQ bar in figure 4.3). The resulting query is illustrated in figure 4.5, where all CQs represented by green nodes are eliminated before evaluating the final query, represented by the blue nodes on the database.

It is interesting, however, to note that the time necessary to evaluate this query after optimization is relatively small while the time necessary to optimize it is relatively high: the orange portion of the second UCQ bar in figure 4.3 is a clear representation of how costly it is to optimize this query using the database.

Finally, by clicking on the third UCQ bar from figure 4.3, we can see the UCQ inspection illustrated in figure 4.6. The *green*, nodes in this figure represent the CQs we were able to correctly identify as empty using our summary; they represent 69.92% of the total number of empty CQs. This optimization corresponds to a significant improvement in performance time as can be seen in figure 4.3.

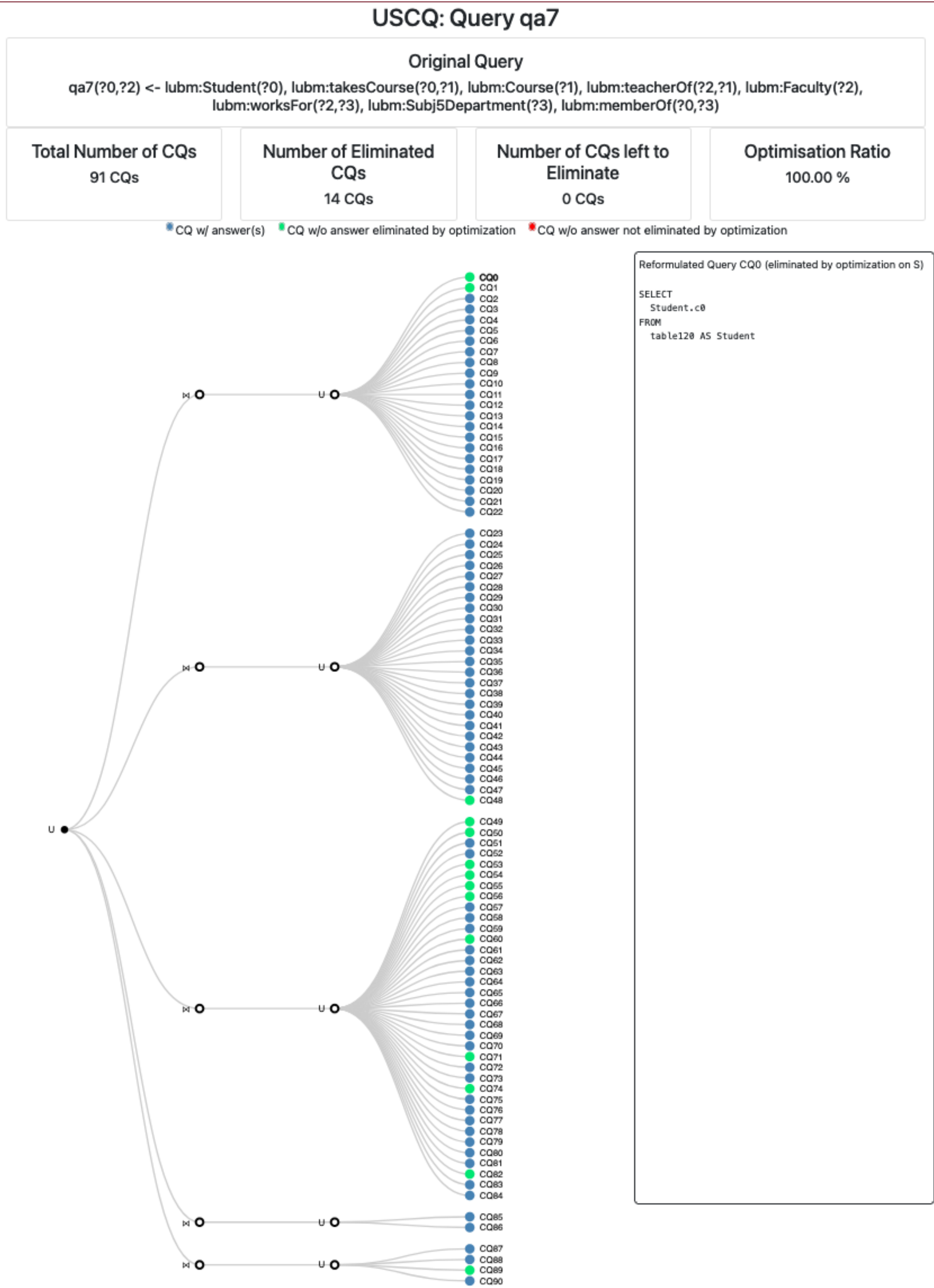


Figure 4.7 – USCQ/S Reformulation Inspection of QA7

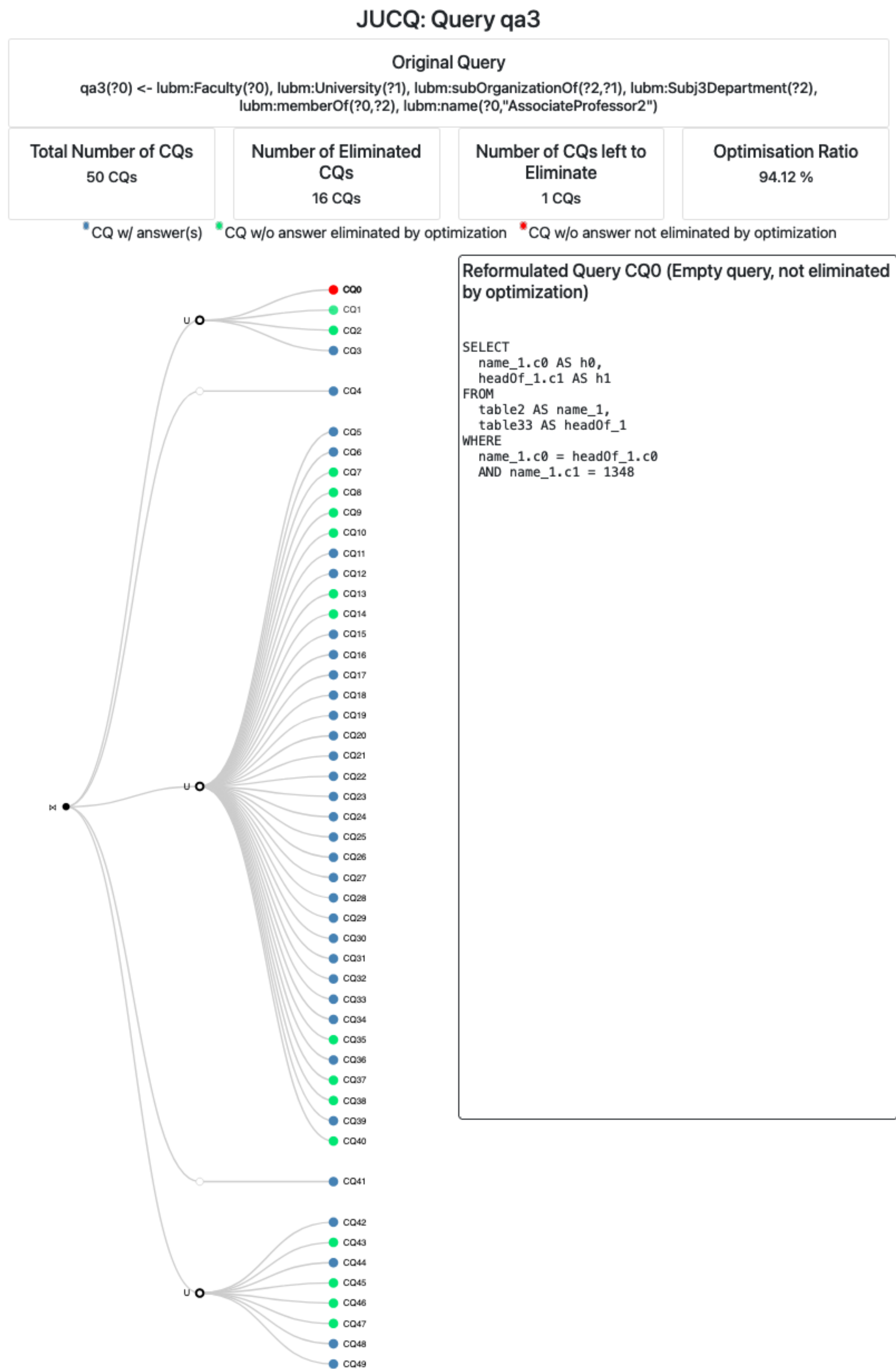


Figure 4.8 – JUCQ/S Reformulation Inspection of QA3

Similarly, we created tree-style representations that allow users to visually inspect both USCQs and JUCQs. Figure 4.7, p. 82, is a USCQ representation of query QA7 and Figure 4.8, p. 83), is a JUCQ representation of query QA3. We notice that in general, the nature of both USCQs and JUCQs are such that they contain a smaller number of irrelevant subCQs compared to UCQs.

In general, using either the summary or the database with Ω, we are able to discard all irrelevant subCQs from both JUCQs and USCQs. Using the database however requires more time to process than using the summary.

The key take-outs a user should have from using the OptiRef tool to analyze the FO-rewriting languages as well as their optimizations are the following:

- A visual inspection of the query forms clearly shows the number of irrelevant queries they contain. UCQs in particular, generally contain a large number of irrelevant subCQs: we observed over our experiments that less than 0.1% of the subCQs in a UCQ contribute to producing the final answer. Meanwhile, at least 65% of subCQs in a USCQ or a JUCQ contribute to producing the final answer.
- Optimizing FO-rewriting queries w.r.t. to the database produces a query reformulation with no irrelevant subCQs; i.e., the most optimized version of the reformulation, but it is costly to optimize using the database in practice.
- The summary is usually very small in size: less than 8% of the database size.
- Optimizing query reformulations w.r.t. to the summary significantly reduces the number of empty subCQs. And, although it does not produce the most optimal form of a query optimization, it manages to significantly improve query performance overall.

4.4 Conclusion

In this chapter we went over the implementation of Optiref: it is a framework for optimizing FO-rewriting as well as a GUI that allows users to interact and analyze results. Its key benefits is that it allows us to better understand the performance improvement brought forth by our work on FO-rewriting optimization.

In the next chapter we revisit the work done in the state of the art and position ourselves w.r.t. it.

Chapter 5

Related work

Contents

5.1 **OMQA via FO-rewriting** 85

5.2 **Database Summaries** 86

5.3 **Summary Maintenance** 87

5.4 **Conclusion** 87

In this chapter, we go over the work done in the literature related to this this. Namely, we consider OMQA via FO-rewriting (Section 5.1), next we focus on database summaries (Section 5.2), before finally discussing summary maintenance (Section 5.3).

5.1 OMQA via FO-rewriting

In this thesis, we focus on OMQA via the FO-rewriting technique with the goal of optimizing it. We devised a novel optimization framework, which is complementary to, and capitalizes on, the optimizations that have been proposed so far in the literature, e.g., [24, 46, 40, 60, 13, 14]. These optimizations are both ontology-dependent and data-independent. They exploit the ontology's rules to find equivalent query reformulations that can be evaluated faster on relational databases, similarly to semantic query optimization that exploits the rules of deductive databases, e.g., [22]. In particular, the optimizations for FO-rewriting has focused on studying logically minimal (i.e., non-redundant) query reformulations (e.g., [24, 46]), syntactically compact query reformulations (e.g., [40, 60]) and query reformulations built according to their estimated evaluation cost on an RDBMS (e.g., [13, 14]).

The novelty of our framework is to add a complementary data-dependent optimization step to query reformulations produced by state-of-the-art FO-rewriting tools, e.g., [24, 60, 7, 61, 62, 55, 54, 14, 13]. To the best of our knowledge, data-dependent optimization of OMQA via FO-rewriting has not been considered in the literature. This idea of data-

dependent optimization originates from the preliminary work [10] that studies data layouts for RDF data in order to efficiency evaluate the reformulations of conjunctive SPARQL queries of [11]. In particular, [10] suggests the use of data summaries to speed up query reformulation evaluation [12, 44].

Our framework is general enough to apply to many FO-rewriting settings, in particular those in Table 2, and it guarantees the correctness of OMQA on the queried KBs. For the FO-rewriting settings in which it was evaluated, it significantly improves OMQA time performance for the prominently-adopted UCQ query reformulations, e.g., [38, 39, 46, 53, 40, 19, 54, 24, 61, 55, 37, 11], and for the JUCQ ones of [14, 13].

Finally, an originality of our framework is that it builds on the Ω optimization function that rewrites a query reformulation into a simpler contained one, by pruning away subqueries that are useless to its evaluation on a given database. Notably, useless subqueries are identified rapidly by using database summaries, which we devised for this particular purpose by adapting the quotient operation [41] to databases.

5.2 Database Summaries

Database summaries are compact representations of databases. They have been widely studied in the area of graph databases [50] and semantic web databases [21], where they serve several purposes such as data exploration, data visualization, data compression, and data management optimization.

The quotient operation has been used to summarize RDF KBs for the purpose of data exploration, e.g., [36], and closer to our work, to summarize description logic databases a.k.a. ABoxes for the purpose of data management optimization [34, 26, 27].

In particular, for the $\mathcal{SHIN}$ description logic, summaries have been used to conduct consistency checking on ABoxes in [34] as well as to perform instance checking in [26], which is a particular case of query answering; the SHER reasoner [27] implements these results. We point out that the summaries of [34, 26, 27] have been built to perform the particular tasks of consistency checking and query answering directly on them.

The summaries of this thesis depart from the above-mentioned ones because we devised our summaries for a specific and novel task. In particular, we adapted the quotient operation to relational databases and we defined the new equivalence relation $\equiv_\Omega$ for the task of sound and fast identification of CQs with no answer on a database. $\equiv_\Omega$ departs

from prior equivalence relations by being based on the instances of concepts that KB's databases describe with n-ary relationships between them, and not on bisimulation [43], e.g., [52, 33], or cooccurrence of relationships [35, 36]. To the best of our knowledge, summaries have not been used for the optimization of OMQA via FO-rewriting.

Finally, we note that the semantics of applying the quotient operation to ABoxes has been characterized [32], regardless of particular description logic and equivalence relation, in terms of the classical notions of *subsumption*, i.e., containment, and *most specific concept* in description logics [6].

5.3 Summary Maintenance

Although the computation of our summaries is linear in the size of databases, the summarization times we reported in our experiments show that it would be prohibitive to redo full summarization upon updates. We therefore rely on incremental summary maintenance.

We remark that the need for incremental maintenance is shared with the two other OMQA techniques, materialization, e.g., [37, 49] and combined approach, e.g., [48, 47, 51], though we need to maintain a summary that is a small and simple homomorphic approximation of the KB's database, while materialization and combined approach need to maintain a large and complex (approximation of a) chase of the KB's database, i.e., database plus entailed facts.

The maintenance of our summaries has been studied in the context of a Research Master's *(Master Recherche)* internship [5]. By definition of a summary built with $\equiv_\Omega$, in the worst case, an insertion fuses two equivalence classes and a deletion splits an equivalence class into several ones. Maintenance rewrites the affected summary facts, i.e., in which some term moves from an equivalence class to another, based on the updated homomorphism σ modeled with a union-find data structure (recall Section 3) that also supports the delete operation in optimal constant amortized time complexity [4].

5.4 Conclusion

In this chapter, we pointed out how the contributions of this thesis build on, and compare to the work done in the literature. Particularly, we went over the literature of OMQA, database summaries and summary maintenance.

Conclusion

Summary

In this thesis, we tackled the problem of improving the performance of OMQA, a.k.a. query answering on knowledge bases, made of a database and an ontology. In the literature, OMQA is performed using several methods; *materialization* which embeds ontological knowledge into the database producing a saturated database, *FO-rewriting* which embeds ontological knowledge into the query producing a reformulation of it, and a combination of the two.

We focused on FO-rewriting for which we identified an essential weakness: it is ontology-dependent and data independent. Being data independent means that FO-rewriting produces query reformulations that are general and do not take into consideration a given instance of a database. In practice, these generic reformulations are costly for RDBMSs to evaluate, and can be significantly improved.

We therefore devised a novel optimization framework for FO-rewriting, in order to make it both ontology-dependent and data-dependent. Our framework builds on our Ω optimization function that rewrites a query reformulation into a contained query with the same answers on the queried KB's database, which can be evaluated faster. For that, Ω uses a summary of the database, which is a homomorphic approximation of it, computed as its quotient w.r.t. the $\equiv_{\Omega}$ equivalence relation between database terms that we defined.

Using the LUBM$^{\exists}$ benchmark, we empirically showed that our framework significantly improves OMQA time performance in general for the widely-adopted UCQ query reformulations, e.g., [38, 39, 46, 53, 40, 19, 54, 24, 61, 55, 37, 11], and frequently for the JUCQ ones of [14, 13], while performance is marginally affected for the USCQ ones of [60]. We provided insight for that.

Finally, we implemented our proposed optimizing framework for OMQA via FO-rewriting into a tool called OptiRef, which we used to conduct our experiments. We additionally designed an intuitive and interactive Graphical User Interface (GUI) enabling users to visually explore both our query performance results and their associated logical plans. This tool serves as a means to highlight the advantages inherent in our

proposed OMQA optimization framework.

Perspectives

Our work on optimizing FO-rewriting can be pursued in several ways.

A first approach would be to experimentally evaluate the effect of our optimization framework beyond DL-lite$_{\mathcal{R}}$. We envision conducting more experiments by using our framework on different benchmarks that are not necessarily restricted by the expressivity of DL-lite$_{\mathcal{R}}$, which we recall underpins the W3C's standard for OMQA on large data volume: the QL profile of OWL2.

Additionally, in this thesis we considered summaries that are homomorphic approximations of databases. They are based on the quotient operation adapted from graph databases. A perspective would be to study alternative database summaries for our framework, which could further improve OMQA time performance: summaries could be obtained either via the quotient operation and other equivalence relations than $\equiv_{\Omega}$, or with other procedures than the quotient operation.

Finally, in this thesis we focused on optimizing OMQA via FO-rewriting after query reformulations were produced by algorithms from the literature. A perspective would be to investigate how data-dependent query optimization can be done during the query reformulation algorithms, as opposed to after it. The idea would be to integrate optimization w.r.t. database summaries within query reformulation algorithms in order to directly produce an optimized query reformulations.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of Databases*, Addison-Wesley, 1995, ISBN: 0-201-53771-0, URL: `http://webdam.inria.fr/Alice/`.

[2] Sergio Abriola, Pablo Barceló, Diego Figueira, and Santiago Figueira, « Bisimulations on Data Graphs », *in*: *J. Artif. Intell. Res.* **61** (2018), page(s): 171–213.

[3] Afnan G. Alhazmi, Tom Blount, and George Konstantinidis, « ForBackBench: A Benchmark for Chasing vs. Query-Rewriting », *in*: *Proc. VLDB Endow.* **15**.*8* (2022), page(s): 1519–1532.

[4] Stephen Alstrup, Mikkel Thorup, Inge Li Gørtz, Theis Rauhe, and Uri Zwick, « Union-Find with Constant Time Deletions », *in*: *ACM Trans. Algorithms* **11**.*1* (2014), page(s): 6:1–6:28.

[5] Adel Aly, « Data Summary Maintenance for Query Optimization », *in*: *Master Science Informatique*, Université de Rennes, 2023.

[6] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds., *The Description Logic Handbook*, Cambridge University Press, 2003, ISBN: 0-521-78176-0.

[7] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, Swan Rocher, and Clément Sipieter, « Graal: A Toolkit for Query Answering with Existential Rules », *in*: *RuleML*, vol. 9202, 2015, page(s): 328–344.

[8] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat, « On rules with existential variables: Walking the decidability line », *in*: *Artificial Intelligence* **175**.*9-10* (2011), page(s): 1620–1654.

[9] Meghyn Bienvenu, « Ontology-Mediated Query Answering: Harnessing Knowledge to Get More from Data », *in*: *IJCAI*, 2016, page(s): 4058–4061.

[10] Maxime Buron, François Goasdoué, Ioana Manolescu, Tayeb Merabti, and Marie-Laure Mugnier, « Revisiting RDF storage layouts for efficient query answering », *in*: *ISWC Workshop on Scalable Semantic Web Knowledge Base Systems*, CEUR Workshop Proceedings, 2020, page(s): 17–32.

[11] Maxime Buron, François Goasdoué, Ioana Manolescu, and Marie-Laure Mugnier, « Reformulation-Based Query Answering for RDF Graphs with RDFS Ontologies », *in*: *ESWC*, vol. 11503, Springer, 2019, page(s): 19–35.

[12] Maxime Buron, Cheikh Brahim El Vaigh, and François Goasdoué, « Towards Faster Reformulation-based Query Answering on RDF graphs with RDFS ontologies. Poster paper », *in*: *ISWC*, 2021.

[13] Damian Bursztyn, François Goasdoué, and Ioana Manolescu, « Optimizing Reformulation-based Query Answering in RDF », *in*: *EDBT*, 2015, page(s): 265–276.

[14] Damian Bursztyn, François Goasdoué, and Ioana Manolescu, « Teaching an RDBMS about ontological constraints », *in*: *PVLDB* (2016), page(s): 1161–1172.

[15] Andrea Calı, Georg Gottlob, and Thomas Lukasiewicz, « A general datalog-based framework for tractable query answering over ontologies », *in*: *PODS*, 2009, page(s): 77–86.

[16] Andrea Calı, Georg Gottlob, and Thomas Lukasiewicz, « Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints », *in*: *ICDT*, 2009, page(s): 14–30.

[17] Andrea Calı, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris, « Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications », *in*: *LICS*, 2010, page(s): 228–242.

[18] Andrea Calı, Georg Gottlob, Thomas Lukasiewicz, and Andreas Pieris, « A logical toolbox for ontological reasoning », *in*: *SIGMOD Rec.* **40**.*3* (2011), page(s): 5–14.

[19] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati, « Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family », *in*: *J. Autom. Reasoning* (2007), page(s): 385–429.

[20] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati, « Data complexity of query answering in description logics », *in*: *Artif. Intell.* **195** (2013), page(s): 335–360.

[21] Sejla Cebiric, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika, « Summarizing semantic graphs: a survey », *in*: *VLDB J.* **28**.*3* (2019), page(s): 295–327.

[22] Upen S. Chakravarthy, John Grant, and Jack Minker, « Logic-Based Approach to Semantic Query Optimization », *in*: *ACM Trans. Database Syst.* **15**.*2* (1990), page(s): 162–207.

[23] Michel Chein and Marie-Laure Mugnier, *Graph-based Knowledge Representation - Computational Foundations of Conceptual Graphs*, Advanced Information and Knowledge Processing, Springer, 2009.

[24] Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou, « Optimized Query Rewriting for OWL2QL », *in*: *CADE*, 2011, page(s): 192–206.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms, 3rd Edition*, MIT Press, 2009, ISBN: 978-0-262-03384-8.

[26] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma, « Scalable Semantic Retrieval through Summarization and Refinement », *in*: *AAAI*, 2007, page(s): 299–304.

[27] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Edith Schonberg, and Kavitha Srinivas, « Scalable Highly Expressive Reasoner », *in*: *J. Web Semant.* **7**.*4* (2009), page(s): 357–361.

[28] Wafaa El Husseini, Cheikh Brahim El Vaigh, François Goasdoué, and Hélène Jaudoin, « Gestion de données efficace dans les bases de connaissances », *in*: *BDA 2022 - 38èmes journées de la conférence BDA "Gestion de Données – Principes, Technologies et Applications"*, Clermont-Ferrand, France, Oct. 2022, URL: `https://inria.hal.science/hal-03812670`.

[29] Wafaa El Husseini, Cheikh Brahim El Vaigh, François Goasdoué, and Hélène Jaudoin, « OptiRef : optimisation pour la gestion de données dans les bases de connaissances », *in*: *BDA 2022 - 38èmes journées de la conférence BDA "Gestion de Données – Principes, Technologies et Applications"*, Clermont-Ferrand, France, Oct. 2022, URL: `https://inria.hal.science/hal-03812666`.

[30] Wafaa El Husseini, Cheikh Brahim El Vaigh, François Goasdoué, and Hélène Jaudoin, « OptiRef: Query Optimization for Knowledge Bases », *in*: *WWW 2023 - The International World Wide Web Conference 2023*, Austin, United States, Apr. 2023, DOI: `10.1145/3543873.3587342`, URL: `https://inria.hal.science/hal-04023665`.

[31] Wafaa El Husseini, Cheikh Brahim El Vaigh, François Goasdoué, and Hélène Jaudoin, « Query Optimization for Ontology-Mediated Query Answering », *in*: *The ACM Web Conference (WWW)*, Singapore, Singapore, May 2024, DOI: 10.1145/3589334.3645567, URL: https://hal.science/hal-04470002.

[32] Cheikh-Brahim El Vaigh and François Goasdoué, « A Well-founded Graph-based Summarization Framework for Description Logics », *in*: *Description Logics*, 2021.

[33] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu, « Query preserving graph compression », *in*: *SIGMOD*, 2012, page(s): 157–168.

[34] Achille Fokoue, Aaron Kershenbaum, Li Ma, Edith Schonberg, and Kavitha Srinivas, « The Summary Abox: Cutting Ontologies Down to Size », *in*: *ISWC*, 2006, page(s): 343–356.

[35] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu, « Incremental structural summarization of RDF graphs », *in*: *EDBT*, 2019, page(s): 566–569.

[36] François Goasdoué, Pawel Guzewicz, and Ioana Manolescu, « RDF graph summarization for first-sight structure discovery », *in*: *VLDB J.* **29**.*5* (2020), page(s): 1191–1218.

[37] François Goasdoué, Ioana Manolescu, and Alexandra Roatis, « Efficient query answering against dynamic RDF databases », *in*: *EDBT*, 2013, page(s): 299–310.

[38] Georg Gottlob, Giorgio Orsi, and Andreas Pieris, « Ontological queries: Rewriting and optimization », *in*: *ICDE*, 2011, page(s): 2–13.

[39] Georg Gottlob, Giorgio Orsi, and Andreas Pieris, « Query Rewriting and Optimization for Ontological Databases », *in*: *ACM Trans. Database Syst.* **39**.*3* (2014), page(s): 25:1–25:46.

[40] Georg Gottlob and Thomas Schwentick, « Rewriting Ontological Queries into Small Nonrecursive Datalog Programs », *in*: *KR*, 2012.

[41] Jonathan L. Gross, Jay Yellen, and Ping Zhang, eds., *Handbook of Graph Theory*, Discrete Mathematics and Its Applications, Chapman & Hall / CRC Press, Taylor & Francis, 2013, ISBN: 978-1-58488-090-5.

[42] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin, « LUBM: A benchmark for OWL knowledge base systems », *in*: *J. Web Semant.* **3**.*2-3* (2005), page(s): 158–182.

[43] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke, « Computing Simulations on Finite and Infinite Graphs », *in*: *FOCS*, 1995, page(s): 453–462.

[44] Wafaa El Husseini, Cheikh Brahim El Vaigh, François Goasdoué, and Hélène Jaudoin, « Ontology-mediated query answering: performance challenges and advances. Demo paper », *in*: *ISWC*, 2021.

[45] Tomasz Imielinski and Witold Lipski Jr., « Incomplete Information in Relational Databases », *in*: *J. ACM* **31**.*4* (1984), page(s): 761–791.

[46] Mélanie König, Michel Leclère, Marie-Laure Mugnier, and Michaël Thomazo, « Sound, complete and minimal UCQ-rewriting for existential rules », *in*: *Semantic Web* (2015), page(s): 451–475.

[47] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev, « The Combined Approach to Ontology-Based Data Access », *in*: *IJCAI*, 2011, page(s): 2656–2661.

[48] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev, « The Combined Approach to Query Answering in DL-Lite », *in*: *KR*, 2010.

[49] Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri, « Fast Query Answering over Existential Rules », *in*: *ACM Trans. Comput. Log.* **20**.*2* (2019), page(s): 12:1–12:48.

[50] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra, « Graph Summarization Methods and Applications: A Survey », *in*: *ACM Comput. Surv.* **51**.*3* (2018), page(s): 62:1–62:34.

[51] Carsten Lutz, Inanç Seylan, David Toman, and Frank Wolter, « The Combined Approach to OBDA: Taming Role Hierarchies Using Filters », *in*: *ISWC*, 2013, page(s): 314–330.

[52] Tova Milo and Dan Suciu, « Index Structures for Path Expressions », *in*: *ICDT*, 1999, page(s): 277–295.

[53] Giorgio Orsi and Andreas Pieris, « Optimizing Query Answering under Ontological Constraints », *in*: *PVLDB* **4**.*11* (2011), page(s): 1004–1015.

[54] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik, « Efficient Query Answering for OWL 2 », *in*: *ISWC*, 2009, page(s): 489–504.

[55] Riccardo Rosati and Alessandro Almatelli, « Improving Query Answering over DL-Lite Ontologies », *in*: *KR*, 2010.

[56] Avi Silberschatz, Henry F. Korth, and S. Sudarshan, *Database System Concepts, Seventh Edition*, McGraw-Hill Book Company, 2020.

[57] Robert Endre Tarjan, « Amortized Computational Complexity », *in*: *SIAM Journal on Algebraic Discrete Methods* **6**.*2* (1985), page(s): 306–318.

[58] Robert Endre Tarjan and Jan van Leeuwen, « Worst-case Analysis of Set Union Algorithms », *in*: *J. ACM* **31**.*2* (1984), page(s): 245–281.

[59] GraphIK team, *GRAAL owl2dlgp*, 2015, URL: `https://graphik-team.github.io/graal/downloads/owl2dlgp` (visited on 2023).

[60] Michaël Thomazo, « Compact Rewritings for Existential Rules », *in*: *IJCAI*, 2013, page(s): 1125–1131.

[61] Tassos Venetis, Giorgos Stoilos, and Giorgos B. Stamou, « Incremental Query Rewriting for OWL 2 QL », *in*: *DL*, 2012.

[62] Roberto De Virgilio, Giorgio Orsi, Letizia Tanca, and Riccardo Torlone, « NYAYA: A System Supporting the Uniform Management of Large Sets of Semantic Data », *in*: *ICDE*, 2012, page(s): 1309–1312.

[63] W3C, *OWL 2 Web Ontology Language Document Overview (Second Edition)*, Dec. 11, 2012, URL: `https://www.w3.org/TR/owl2-overview/`.

Appendix A

Appendix

A.1 Queries

A.1.1 Query Answering Queries

Table A.1 – List of queries used to evaluate query answering and their description.

Name	Query	Description
QA0	*qa0(?0,?2) <- lubm:Student(?0), lubm:takesCourse(?0,?1), lubm:Subj1Course(?1), lubm:teacherOf(?2,?1), lubm:Professor(?2), lubm:headOf(?2,?3), lubm:Subj1Department(?3), lubm:memberOf(?0,?3)*	A similar query to QA which specifies a type for the *Courses* and a type for the *Department*
QA1	*qa1(?0) <- lubm:Person(?0), lubm:worksFor(?0,?1), lubm:Department(?1), lubm:takesCourse(?0,?2), lubm:Course(?2)*	A query that looks for people (?0) that work in a department (?1) and take some courses(?2).
QA2	*qa2(?0) <- lubm:Student(?0), lubm:publicationAuthor(?1,?0), lubm:Publication(?1), lubm:teachingAssistantOf(?0,?2), lubm:Course(?2)*	A query that looks for students (?0) that are authors of a publication(?1) and are teaching assistants of some courses (?2).

Continued on next page

Table A.1 List of queries used to evaluate query answering and their description. (continued)

Name	**Query**	**Description**
QA3	*qa3(?0) <- lubm:Faculty(?0), lubm:University(?1), lubm:subOrganizationOf(?2,?1), lubm:Subj3Department(?2), lubm:memberOf(?0,?2), lubm:name(?0,"AssociateProfessor2")*	A query which looks for Faculty members (?0) whose name is "AssociateProfessor2" and who are a member a department (?2) of type "Subj3Department". This department is a sub-organization of a university (?1).
QA4	*qa4(?0) <- lubm:Faculty(?0), lubm:degreeFrom(?0,?1), lubm:University(?1), lubm:subOrganizationOf(?2,?1), lubm:Subj10Department(?2), lubm:memberOf(?0,?2)*	A query which looks for Faculty members (?0) having a degree from a certain university (?1) and who are a members of a department (of type "Subj10Department") that is a sub-organization of that same university.
QA5	*qa5(?1,?4) <- lubm:Subj3Department(?1), lubm:Subj4Department(?4), lubm:Subj10Professor(?0), lubm:memberOf(?0,?1), lubm:publicationAuthor(?2,?0), lubm:Professor(?3), lubm:memberOf(?3,?4), lubm:publicationAuthor(?2,?3)*	A query which looks for pairs of departments (?1) and (?4) that share at least one publication with authors who are professors from each of the respective departments.
QA6	*qa6(?0,?1,?2) <- lubm:Professor(?0), lubm:teacherOf(?0,?1), lubm:worksFor(?0,?2), lubm:degreeFrom(?0, "http://www.University870.edu"), lubm:researchInterest(?0,"Research21"), lubm:name(?0,"AssociateProfessor2"), lubm:emailAddress(?0, "AssociateProfessor2@Department1.University0.edu"), lubm:telephone(?0,"xxx-xxx-xxxx")*	A query which looks for a list of professors (?0), classes (?1) and departments (2) s.t. (?0) are teachers of (?1) and (?0) work for (?2). The degree, research interest, name, email address and telephone number of (?0) are specified.

Continued on next page

Table A.1 List of queries used to evaluate query answering and their description. (continued)

Name	Query	Description
QA7	*qa7(?0,?2) <- lubm:Student(?0), lubm:takesCourse(?0,?1), lubm:Course(?1), lubm:teacherOf(?2,?1), lubm:Faculty(?2), lubm:worksFor(?2,?3), lubm:Subj5Department(?3), lubm:memberOf(?0,?3)*	A query which looks for pairs of students (?0) and faculty members (?2) where the students are enrolled in courses taught by the faculty members who work for the same department that the students are members of.
QA8	*qa8(?0) <- lubm:Faculty(?0), lubm:mastersDegreeFrom(?0,?1), lubm:University(?1), lubm:subOrganizationOf(?2,?1), lubm:Subj10Department(?2), lubm:memberOf(?0,?2)*	A query which looks for a list faculty (?0) that are members of a department which is a sub-organization of the university from which they obtained their Masters degree.
QA9	*qa9(?1,?4) <- lubm:Subj3Department(?1), lubm:Subj4Department(?4), lubm:Subj3Professor(?0), lubm:memberOf(?0,?1), lubm:publicationAuthor(?2,?0), lubm:Subj5Professor(?3), lubm:memberOf(?3,?4), lubm:publicationAuthor(?2,?3)*	A query which looks for pairs of departments (?1) and (?4) that share at least one publication with authors who are professors from each of the respective departments. (A similar query to QA5, but more specialized)

A.1.2 Consistency Checking Queries

Name	**Query**	**Description**
QC0	*qc0(?0) <- lubm:Person(?0), lubm:Organization(?0)*	A Person which is also an Organization
QC1	*qc1(?0) <- lubm:Organization(?0), lubm:Student(?0)*	An Organization which is also a Student
QC2	*qc2(?0) <- lubm:Organization(?0), lubm:Publication(?0)*	An Organization which is also a Publication
QC3	*qc3(?0) <- lubm:Professor(?0), lubm:Department(?0)*	A Professor which is also a Department
QC4	*qc4(?0) <- lubm:Professor(?0), lubm:Publication(?0)*	A Professor which is also a Publication

Table A.2 – List of queries used to evaluate consistency checking and their description.

A.2 Query Characteristics

In this section, we detail the computation of the optimization ratio for each of the reformulation languages, for both QA and CC queries.

A.2.1 UCQ Queries

Here, we provide a breakdown of the numbers used to compute the optimization ratio for QA queries using the UCQ language.

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,760	1,950	1,702	1,152	720	496	300	184	144	32
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
# CQs with an answer on $\mathcal{D}$	6	0	40	16	29	0	3	2	11	0
# CQs with NO answer on $\mathcal{D}$	2,754	1,950	1,662	1,136	691	496	297	182	133	32
# CQs with an answer on $\mathcal{S}$	504	0	42	200	153	100	144	42	51	8
# CQs with NO answer on $\mathcal{S}$	2,256	1,950	1,660	952	567	396	156	142	93	24
optim. ratio for UCQ/S	81.92	100	99.88	83.8	82.05	79.84	52.53	78.02	69.92	75

Table A.3 – QA queries characteristics - UCQ ($\mathcal{O}$ = LUBM$^{\exists}_{20}$ and $\mathcal{D}$ = LUBM150M)

Query	C0c|C0i	C1c|C1i	C2c|C2i	C3c|C3i	C4c|C4i
#atoms	2	2	2	2	2
#contained CQs w.r.t. $\mathcal{O}$	3,645	1,035	765	630	510
# answers in $ans(q, \mathcal{K})$	0|1	0|1	0|1	0|1	0|1
# CQs with an answer on $\mathcal{D}$	0|5	0|5	0|4	0|2	0|2
# CQs with NO answer on $\mathcal{D}$	3,645|3,640	1,035|1,030	765|761	630|628	510|508
# CQs with an answer on $\mathcal{S}$	0|5	0|5	0|10	0|21	0|2
# CQs with NO answer on $\mathcal{S}$	3,645|3,640	1,035|1,030	765|755	630|609	510|508
compl. ratio for UCQ/S	100|100	100|100	100|99.21	100|96.97	100|100

Table A.4 – CC queries characteristics - UCQ ($\mathcal{O}$ = LUBM$^{\exists}_{20}$ and $\mathcal{D}$ = LUBM150M)

A.2.2 USCQ Queries

Here, we provide a breakdown of the numbers used to compute the optimization ratio for QA queries using the USCQ language.

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,759	1,949	1,701	1,151	719	495	299	183	143	31
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
# contained CQs w.r.t. $\mathcal{O}$ (USCQ)	57	203	254	48	53	112	37	91	48	52
# CQs with an answer on $\mathcal{D}$	48	176	194	33	36	86	29	77	33	38
# CQs with NO answer on $\mathcal{D}$	9	27	60	15	17	26	8	14	15	14
# CQs with an answer on $\mathcal{S}$	48	176	194	33	36	86	29	77	33	38
# CQs with NO answer on $\mathcal{S}$	9	27	60	15	17	26	8	14	15	14
optim. ratio for USCQ/S	100	100	100	100	100	100	100	100	100	100

Table A.5 – QA queries characteristics - USCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$)

Query	C0c|C0i	C1c|C1i	C2c|C2i	C3c|C3i	C4c|C4i
#atoms	2	2	2	2	2
#contained CQs w.r.t. $\mathcal{O}$	3,645	1,035	765	630	510
# answers in $ans(q, \mathcal{K})$	0|1	0|1	0|1	0|1	0|1
# contained CQs w.r.t. $\mathcal{O}$ (USCQ)	130	68	62	51	47
# CQs with an answer on $\mathcal{D}$	92|93	52|53	33|33	45|46	26|27
# CQs with NO answer on $\mathcal{D}$	38|37	16|15	29|29	6|5	21|20
# CQs with an answer on $\mathcal{S}$	92|93	52|53	33|33	45|46	26|27
# CQs with NO answer on $\mathcal{S}$	38|37	16|15	29|29	6|5	21|20
compl. ratio for USCQ/S	100|100	100|100	100|100	100|100	100|100

Table A.6 – CC queries characteristics - USCQ ($\mathcal{O} = \text{LUBM}^{\exists}_{20}$ and $\mathcal{D} = \text{LUBM150M}$)

A.2.3 JUCQ Queries

Here, we provide a breakdown of the numbers used to compute the optimization ratio for QA queries using the JUCQ language. It is worth noting that queries reformulated using the JUCQ language correspond to UCQ reformulations in the context of CC queries.

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,759	1,949	1,701	1,151	719	495	299	183	143	31
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
# contained CQs w.r.t. $\mathcal{O}$ (JUCQ)	59	50	68	50	65	41	39	184	144	12
# CQs with an answer on $\mathcal{D}$	53	108	74	33	39	96	31	4	29	10
# CQs with NO answer on $\mathcal{D}$	9	23	13	17	27	150	9	180	12	3
# CQs with an answer on $\mathcal{S}$	53	108	74	34	39	100	32	42	29	10
# CQs with NO answer on $\mathcal{S}$	9	23	13	16	27	146	8	142	12	3
optim. ratio for JUCQ/S	100	100	100	94.12	100	97.33	88.89	78.89	100	100

Table A.7 – QA queries characteristics - JUCQ ($\mathcal{O}$ = LUBM$^{\exists}_{20}$ and $\mathcal{D}$ = LUBM150M)

A.3 Execution Time

In this section, we provide a detailed breakdown of query evaluation times, broken into: reformulation time, optimization time and execution time. We report on the values for all queries using all RDBMSs.

A.3.1 UCQ Queries

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,759	1,949	1,701	1,151	719	495	299	183	143	31
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
PSQL Reformulation Time	1,038	501	320	135	62	96	388	88	22	28
PSQL Optimization Time	0	0	0	0	0	0	0	0	0	0
PSQL Execution Time	533,708.4	129,597	146,732	175,980.8	120,090.4	101,129.6	3,702.8	39,818.2	14,139	1,655.6
PSQL Reformulation Time /DB	1,042	452	292	108	41	119	63	93	27	27
PSQL Optimization Time /DB	42,517.4	63,298.8	71,068.4	17,874.8	25,114.2	9,874.6	3,436.4	14,965.6	12,220.4	1,176.6
PSQL Execution Time /DB	1,329.2	0	7,901.8	3,367.6	1,150.4	0	44.2	268.6	371.2	0
PSQL Reformulation Time /S	427	964	322	170	79	689	73	92	48	39
PSQL Optimization Time /S	11,235	1.4	2.6	78	307.8	510.4	2,678	187.4	29.4	21.4
PSQL Execution Time/S	17,310.4	0	9,634.4	22,281.2	4,501	5,509.6	1,784.6	20,435.6	1,122	181.6
MySQL Reformulation Time	324	794	339	129	40	105	69	88	33	39
MySQL Optimization Time	0	0	0	0	0	0	0	0	0	0
MySQL Execution Time	42,892.2	177,954.4	203,834.4	33,732.6	31,421.4	18,437.4	135	69,991.6	10,295.8	783.6
MySQL Reformulation Time /DB	269	523	295	139	46	106	64	97	32	26
MySQL Optimization Time /DB	39,110.8	177,528.4	199,927.2	28,287.4	29,786.4	12,304	0	57,844.6	10,045	3,304.2
MySQL Execution Time /DB	1,138	0	38,875.6	4,612.4	1,719.6	0	1	639.6	619	0
MySQL Reformulation Time /S	275	412	290	110	41	113	73	98	26	27
MySQL Optimization Time /S	21.4	1.2	0	3.8	3.2	12.8	1	2.4	0.8	0.4
MySQL Execution Time/S	19,918.4	0	41,193	20,855.4	5,147.8	6,076.4	61.8	61,873	1,707.8	752.6
DB2 Reformulation Time	433	709	591	169	44	146	91	349	37	31
DB2 Optimization Time	0	0	0	0	0	0	0	0	0	0
DB2 Execution Time	-2	-2	-2	139,339.4	55,867.2	6,051.2	8.8	117,779.8	14,685.4	524.8
DB2 Reformulation Time /DB	417	1,081	472	167	70	144	80	489	38	35
DB2 Optimization Time /DB	108,840.6	123,661.6	85,351.4	3,520.4	68,278.4	8,574	1.2	16,926.2	40,727.6	1,484.2
DB2 Execution Time /DB	1,497.6	0	31,576	412.2	23,282.6	0	0	11,816.2	10,559.8	0
DB2 Reformulation Time /S	513	705	370	189	41	199	67	520	42	34
DB2 Optimization Time /S	4.8	0.8	2	39	0	1.6	0	2	0	0
DB2 Execution Time/S	15,037,8	0	36,676.2	2,099.6	11,590	791	4	108,200.8	16,012.2	296.2

Table A.8 – QA queries performance per RDBMS - UCQ ($\mathcal{O}$ = LUBM$^{\exists}_{20}$ and $\mathcal{D}$ = LUBM150M)

A.3.2 USCQ Queries

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,759	1,949	1,701	1,151	719	495	299	183	143	31
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
PSQL Reformulation Time	18	24	21	22	13	28	26	94	14	21
PSQL Optimization Time	0	0	0	0	0	0	0	0	0	0
PSQL Execution Time	14,194.8	47,356.8	82,776.4	9,039.4	11,823.4	23,662.8	2,104	72,965.2	9,308.4	15,026.8
PSQL Reformulation Time /DB	18	25	25	45	39	41	27	91	16	21
PSQL Optimization Time /DB	5,945.2	27,577.2	43,531.2	4,443.2	5,524	10,722	1,006.4	13,207.2	4,113.4	10,338.6
PSQL Execution Time /DB	14,224.2	47,029.2	93,601.4	8,919.2	11,872.6	23,939.8	2,038.2	72,783.8	9,583.2	14,997.4
PSQL Reformulation Time /S	18	27	22	25	14	46	24	86	14	23
PSQL Optimization Time /S	0	2,071.6	0	0	0	0	0	0	0	0
PSQL Execution Time/S	14,189.6	47,215.2	82,394.2	9,048.4	11,956.6	23,795	2,116.4	72,839	9,360.2	15,119.8
MySQL Reformulation Time	24	28	38	21	17	40	23	85	17	22
MySQL Optimization Time	0	0	0	0	0	0	0	0	0	0
MySQL Execution Time	59,745.2	187,456.4	223,490.8	223,186.2	42,258.8	60,565.2	9,332.2	122,053.8	36,010.4	62,064.8
MySQL Reformulation Time /DB	19	23	36	25	22	41	24	83	15	18
MySQL Optimization Time /DB	9,297.2	41,813.6	55,983.2	6,406.6	7,257	18,578.8	1,487	19,034	6,210.6	14,742
MySQL Execution Time /DB	59,725	187,263.4	222,905	226,038.2	42,208.8	59,693.4	9,359.4	120,912.6	36,099.2	61,837.6
MySQL Reformulation Time /S	18	26	44	23	16	38	23	88	23	19
MySQL Optimization Time /S	0	5,848.2	67.4	0	0	0	0	1.6	0	0.2
MySQL Execution Time/S	59,328.2	188,644.8	223,124.8	224,998.4	42,425.8	60,256.4	9,358.2	121,379	35,794.2	61,440
DB2 Reformulation Time	20	28	25	21	16	42	24	482	17	29
DB2 Optimization Time	0	0	0	0	0	0	0	0	0	0
DB2 Execution Time	5,326.2	11,838	89,883	3,055.4	3,942.6	53438,6	525.8	16,232.8	4,468.2	4.422.6
DB2 Reformulation Time /DB	18	24	21	25	13	34	28	345	14	19
DB2 Optimization Time /DB	0	0.8	1.6	0	0	33.8	0	0.4	0	0
DB2 Execution Time /DB	5,338	11,924.2	90,473	2,997.6	3,991.2	53,594.4	508	16,213.8	4,496.4	4,403
DB2 Reformulation Time /S	20	26	27	26	15	42	29	546	15	22
DB2 Optimization Time /S	0	0	0	0	0	0	0	1.4	0	0
DB2 Execution Time/S	5,376.2	11,806.4	90,738.8	3,000.2	3,960.6	51,832.8	523.8	16,156.6	4,437.4	4,371.2

Table A.9 – QA queries performance per RDBMS - USCQ ($\mathcal{O}$ = LUBM$^{\exists}_{20}$ and $\mathcal{D}$ = LUBM150M)

A.3.3 JUCQ Queries

Query	QA0	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9
# atoms	8	5	5	6	6	8	8	8	6	8
# contained CQs w.r.t. $\mathcal{O}$	2,759	1,949	1,701	1,151	719	495	299	183	143	31
# answers in $ans(q, \mathcal{K})$	36,922	0	521,041	1,076	102	0	2	1,309,926	21	0
PSQL Reformulation Time	33	21	25	26	24	64	210	91	11	28
PSQL Optimization Time	0	0	0	0	0	0	0	0	0	0
PSQL Execution Time	12,243.4	37,223.6	36,680.8	2,036.6	15,869.6	58,147.6	16	40,008.2	14,407.8	209
PSQL Reformulation Time /DB	54	37	34	26	24	40	202	93	18	27
PSQL Optimization Time /DB	5,936.6	12,004	16,563.6	1,287.2	11,840.8	27,125	472.4	16,970.2	12,331	5,475.8
PSQL Execution Time /DB	12,526	36,889.8	36,213.4	1,834.8	8484.4	45,134.2	14	8,916	370.8	512
PSQL Reformulation Time /S	49	34	33	26	23	41	199	88	18	27
PSQL Optimization Time /S	1.2	0	0	0	0	0	0.4	189.8	5.2	0
PSQL Execution Time/S	12,879.6	36,818.4	36,150.6	1,893.6	8,527	45,290.2	15.6	20,008.8	1,132	518.4
MySQL Reformulation Time	53	34	19	28	21	66	46	90	16	38
MySQL Optimization Time	0	0	0	0	0	0	0	0	0	0
MySQL Execution Time	58,873.4	112,386.2	120,772.6	12,518.8	39,366.2	210.6	7,717.2	66,205.6	285,077.8	2,489.8
MySQL Reformulation Time /DB	53	38	32	27	41	41	58	87	27	37
MySQL Optimization Time /DB	18,963.8	40,666.8	22,890.6	3,876	19,814	329,418	5,439	65,890.2	65,366.4	13,474.4
MySQL Execution Time /DB	58,730.2	113,225.2	120,841.4	12,224.2	36,473.4	450.4	7,804	16,437.4	292,771.4	2,751.8
MySQL Reformulation Time /S	52	37	18	26	40	114	55	108	19	58
MySQL Optimization Time /S	0	17.2	0	0	0	27.2	0	62.8	0	0.4
MySQL Execution Time/S	58,255.4	111,879.2	120,323	12,456.8	35,516.6	271.4	7,787.4	62,202.8	290,946.8	2,538.2
DB2 Reformulation Time	93	56	43	43	41	148	58	497	43	92
DB2 Optimization Time	0	0	0	0	0	0	0	0	0	0
DB2 Execution Time	5,518.4	3,689.6	17,324.6	3,110.6	3,980.8	3,325.8	529	117,732.6	18,111.6	1,118.2
DB2 Reformulation Time /DB	89	54	33	42	42	151	52	528	44	84
DB2 Optimization Time /DB	0	2.4	8.6	0.6	0	709.6	0	16,799.6	2,453	0
DB2 Execution Time /DB	5,269.6	3,616.6	17,744.2	3,028.8	3,879.8	3,260.4	510	11,627.4	18,059.8	1101.6
DB2 Reformulation Time /S	86	35	34	42	41	160	54	273	26	56
DB2 Optimization Time /S	0	0	0	0	0	0.2	0	61.2	0	0
DB2 Execution Time/S	5,312.8	3,618.4	17,315.4	3,033.2	3,970.4	3,318.6	517	106,222.4	17,976.2	1,099.6

Table A.10 – QA queries performance per RDBMS - JUCQ ($\mathcal{O}$ = LUBM$^{\exists}_{20}$ and $\mathcal{D}$ = LUBM150M)

A.4 Data


```xml
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:ub="http://swat.cse.lehigh.edu/onto/univ-bench.owl#">


<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://swat.cse.lehigh.edu/onto/univ-bench.owl" />
</owl:Ontology>

<ub:University rdf:about="http://www.University0.edu">
   <ub:name>University0</ub:name>
</ub:University>

<ub:Department rdf:about="http://www.Department0.University0.edu">
   <ub:name>Department0</ub:name>
   <ub:subOrganizationOf>
      <ub:University rdf:about="http://www.University0.edu" />   </ub:subOrganizationOf>
</ub:Department>

<ub:Subj18Department rdf:about="http://www.Department0.University0.edu"/>

<ub:FullProfessor rdf:about="http://www.Department0.University0.edu/FullProfessor0">
   <ub:name>FullProfessor0</ub:name>
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/Course0" />
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/GraduateCourse0" />
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/GraduateCourse1" />
   <ub:undergraduateDegreeFrom>
      <ub:University rdf:about="http://www.University888.edu" />   </ub:undergraduateDegreeFrom>
   <ub:mastersDegreeFrom>
      <ub:University rdf:about="http://www.University947.edu" />   </ub:mastersDegreeFrom>
   <ub:doctoralDegreeFrom>
      <ub:University rdf:about="http://www.University260.edu" />   </ub:doctoralDegreeFrom>
   <ub:emailAddress>FullProfessor0@Department0.University0.edu</ub:emailAddress>
   <ub:telephone>xxx-xxx-xxxx</ub:telephone>
   <ub:researchInterest>Research22</ub:researchInterest>
</ub:FullProfessor>

<ub:Subj18Professor rdf:about="http://www.Department0.University0.edu/FullProfessor0"/>

<ub:FullProfessor rdf:about="http://www.Department0.University0.edu/FullProfessor1">
   <ub:name>FullProfessor1</ub:name>
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/Course1" />
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/GraduateCourse2" />
   <ub:undergraduateDegreeFrom>
      <ub:University rdf:about="http://www.University635.edu" />   </ub:undergraduateDegreeFrom>
   <ub:mastersDegreeFrom>
      <ub:University rdf:about="http://www.University375.edu" />   </ub:mastersDegreeFrom>
   <ub:doctoralDegreeFrom>
      <ub:University rdf:about="http://www.University730.edu" />   </ub:doctoralDegreeFrom>
   <ub:worksFor rdf:resource="http://www.Department0.University0.edu" />
   <ub:emailAddress>FullProfessor1@Department0.University0.edu</ub:emailAddress>
   <ub:telephone>xxx-xxx-xxxx</ub:telephone>
   <ub:researchInterest>Research21</ub:researchInterest>
</ub:FullProfessor>

<ub:Subj18Professor rdf:about="http://www.Department0.University0.edu/FullProfessor1"/>

<ub:FullProfessor rdf:about="http://www.Department0.University0.edu/FullProfessor2">
   <ub:name>FullProfessor2</ub:name>
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/Course3" />
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/Course4" />
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/GraduateCourse3" />
   <ub:teacherOf rdf:resource="http://www.Department0.University0.edu/GraduateCourse4" />
   <ub:undergraduateDegreeFrom>
      <ub:University rdf:about="http://www.University747.edu" />   </ub:undergraduateDegreeFrom>
   <ub:mastersDegreeFrom>
      <ub:University rdf:about="http://www.University788.edu" />   </ub:mastersDegreeFrom>
   <ub:doctoralDegreeFrom>
      <ub:University rdf:about="http://www.University258.edu" />   </ub:doctoralDegreeFrom>
   <ub:worksFor rdf:resource="http://www.Department0.University0.edu" />
   <ub:emailAddress>FullProfessor2@Department0.University0.edu</ub:emailAddress>
   <ub:telephone>xxx-xxx-xxxx</ub:telephone>
   <ub:researchInterest>Research27</ub:researchInterest>
</ub:FullProfessor>

<ub:Subj18Professor rdf:about="http://www.Department0.University0.edu/FullProfessor2"/>
```


Figure A.1 – Raw Data in OWL format

```
@prefix ub: <http://swat.cse.lehigh.edu/onto/univ-bench.owl#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
@top <http://www.w3.org/2002/07/owl#Thing>
 ! :- <http://www.w3.org/2002/07/owl#Nothing>(X).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent407>, "UndergraduateStudent407"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent133>, "UndergraduateStudent133"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssociateProfessor13/Publication5>, "Publication5"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/Course1>, "Course1"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent323>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssistantProfessor1>, "AssistantProfessor1"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/FullProfessor6/Publication13>, "Publication13"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/GraduateStudent7>, "GraduateStudent7@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent127>, "UndergraduateStudent127@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent203>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent166>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent165>, "UndergraduateStudent165"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent331>, "UndergraduateStudent331@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent13>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent356>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/AssociateProfessor1>, "AssociateProfessor1@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/FullProfessor8/Publication7>, "Publication7"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/FullProfessor9/Publication10>, "Publication10"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent157>, "UndergraduateStudent157@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/Course28>, "Course28"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/GraduateStudent2>, "GraduateStudent2"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent197>, "UndergraduateStudent197"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/GraduateStudent138>, "GraduateStudent138"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent59>, "UndergraduateStudent59@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent206>, "UndergraduateStudent206@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent286>, "UndergraduateStudent286"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/FullProfessor0>, "FullProfessor0@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/GraduateStudent139>, "GraduateStudent139@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/GraduateStudent118>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent175>, "UndergraduateStudent175@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssistantProfessor0/Publication7>, "Publication7"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent258>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssociateProfessor3/Publication7>, "Publication7"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/FullProfessor5/Publication9>, "Publication9"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/GraduateStudent100>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent373>, "UndergraduateStudent373@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent523>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/GraduateStudent58>, "GraduateStudent58@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent188>, "UndergraduateStudent188@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/GraduateStudent11>, "GraduateStudent11@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent80>, "UndergraduateStudent80@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent148>, "UndergraduateStudent148@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssociateProfessor3/Publication16>, "Publication16"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent115>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/FullProfessor7/Publication1>, "Publication1"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/GraduateCourse30>, "GraduateCourse30"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent330>, "UndergraduateStudent330@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent274>, "UndergraduateStudent274"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent96>, "UndergraduateStudent96"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/AssociateProfessor0>, "AssociateProfessor0@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssociateProfessor10/Publication4>, "Publication4"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent225>, "UndergraduateStudent225@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent364>, "UndergraduateStudent364"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent221>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent78>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent439>, "UndergraduateStudent439@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent343>, "UndergraduateStudent343@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:emailAddress(<http://www.Department0.University0.edu/UndergraduateStudent28>, "UndergraduateStudent28@Department0.University0.edu"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/GraduateStudent45>, "GraduateStudent45"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent218>, "UndergraduateStudent218"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent440>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssistantProfessor3/Publication6>, "Publication6"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/UndergraduateStudent150>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/Course61>, "Course61"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/UndergraduateStudent65>, "UndergraduateStudent65"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/AssociateProfessor6/Publication6>, "Publication6"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/FullProfessor8/Publication8>, "Publication8"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:name(<http://www.Department0.University0.edu/GraduateStudent77>, "GraduateStudent77"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
ub:telephone(<http://www.Department0.University0.edu/GraduateStudent24>, "xxx-xxx-xxxx"^^<http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral>).
```


Figure A.2 – Raw Data in DLP format

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS INFORMATIQUE, SIGNAL SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Gestion efficace de données à l'aide d'ontologies expressives

Mot clés : Gestion de données, bases de connaissances, Optimisation de requêtes

Résumé :
Réponde à des requêtes à l'aide d'ontologies (OMQA) consiste à poser ces requêtes sur des bases de connaissances (KB). Une KB est un ensemble de faits (base de données), qui est décrit par un domaine de connaissance (ontologie). La technique OMQA la plus étudiée est la réécriture FO (FO-rewriting) ; elle consiste à reformuler une requête pour y intégrer les connaissances pertinentes de l'ontologie, avant de poser la sur la base de données. Telles reformulations peuvent alors être complexes et leur optimisation est cruciale pour l'efficacité. Nous élaborons un nouveau cadre d'optimisation pour la FO-rewriting : les requêtes conjonctives (de type select-project-join) posées sur des KBs en datalog$\pm$ et en règles existentielles, logique de description et OWL, ou RDF/S. On optimise les requêtes produites par les algorithmes de la littérature pour la FO-rewriting, en calculant rapidement, à l'aide du résumé de la base de données, des requêtes plus simples (contenues) avec les mêmes réponses et qui sont évaluées plus rapidement par les SGBDs. On montre sur un benchmark OMQA bien établi, que les performances temporelles sont considérablement améliorées par notre cadre d'optimisation, jusqu'à trois ordres de grandeur.

Title: Efficient ontology-based data management

Keywords: Data management, Knowledge bases, Query optimization

Abstract: Ontology-mediated query answering (OMQA) consists in asking database queries on knowledge bases (KBs); a KB is a set of facts called a database, which is described by a domain knowledge called an ontology. A main OMQA technique is FO-rewriting, which reformulates a query asked on a KB w.r.t. to the KB's ontology; query answers are then computed through the relational evaluation of the query reformulation on the KB's database. Essentially, because FO-rewriting compiles the domain knowledge relevant to queries into their reformulations, query reformulations may be complex and their optimization is the crux of efficiency. We devise a novel optimization framework for a large set of OMQA settings that enjoy FO-rewriting: conjunctive queries, i.e., the core select-project-join queries, asked on KBs expressed in datalog$\pm$ and existential rules, description logic and OWL, or RDF/S. We optimize the query reformulations produced by any state-of-the-art algorithm for FO-rewriting by computing rapidly, using a KB's database summary, simpler queries with same answers that can be evaluated faster by DBMSs. We show on a well-established OMQA benchmark that time performance is significantly improved by our optimization framework in general, up to three orders of magnitude.