PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Doctorado
en Informática

QB4OLAP: Enabling Business Intelligence over Semantic Web Data

Lorena Etcheverry Venturini

2016

QB4OLAP: Enabling Business Intelligence over Semantic Web Data
Lorena Etcheverry Venturini
ISSN 0797-6410
Tesis de Doctorado en Informática
Reporte Técnico RT 16-10
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, 2016

Universidad de la República
Facultad de Ingeniería

QB4OLAP: Enabling Business Intelligence over Semantic Web Data

Tesis presentada al PEDECIBA Informática
y a la Facultad de Ingeniería de la Universidad de la República por

Lorena Etcheverry Venturini

en cumplimiento parcial de los requerimientos
para la obtención del título de Doctor en Informática

Julio 2016

Director de Tesis

Dr.Alejandro Vaisman ..... Instituto Tecnológico de Buenos Aires, Argentina

Tribunal

Dr. Alberto Abelló (Revisor Externo) Universidad Politécnica de Cataluña, España
Dr. Claudio Gutierrez (Revisor Externo) ..... Universidad de Chile, Chile
Dra. María Esther Vidal ..... Universidad Simón Bolívar, Venezuela
Dr. Héctor Cancela ..... Universidad de la República, Uruguay
Dra. Adriana Marotta ..... Universidad de la República, Uruguay

Lorena Etcheverry Venturini: *QB4OLAP: Enabling Business Intelligence over Semantic Web Data,* Publishing and Querying Data Cubes using Semantic Web Technologies
July 2016, Montevideo, Uruguay

A mis padres, Ana María y Omar, por la paciencia, por el amor, y
por alimentar mi curiosidad.

ABSTRACT

The World-Wide Web was initially conceived as a repository of information tailored for human consumption. In the last decade, the idea of transforming the web into a machine-understandable web of data, has gained momentum. To this end, the World Wide Web Consortium (W3C) maintains a set of standards, referred to as the Semantic Web (SW), which allow to openly share data and metadata. Among these is the Resource Description Framework (RDF), which represents data as graphs, RDF-S and OWL to describe the data structure via ontologies or *vocabularies*, and SPARQL, the RDF query language. On top of the RDF data model, standards and recommendations can be built to represent data that adheres to other models. The *multidimensional* (MD) model views data in an n-dimensional space, usually called a *data cube*, composed of dimensions and facts. The former reflect the perspectives from which data are viewed, and the latter correspond to points in this space, associated with (usually) quantitative data (also known as measures). Facts can be aggregated, disaggregated, and filtered using the dimensions. This process is called Online Analytical Processing (OLAP).

Despite the RDF Data Cube Vocabulary (QB) is the W3C standard to represent statistical data, which resembles MD data, it does not include key features needed for OLAP analysis, like dimension hierarchies, dimension level attributes, and aggregate functions. To enable this kind of analysis over SW data cubes, in this thesis we propose the QB4OLAP vocabulary, an extension of QB.

A problem remains, however: writing efficient analytical queries over SW data cubes requires a deep knowledge of RDF and SPARQL, unlikely to be found in typical OLAP users. We address this problem in this thesis. Our approach is based on allowing analytical users to write queries using what they know best: OLAP operations over data cubes, without dealing with SW technicalities. For this, we devised CQL, a simple, high-level query language over data cubes. Then we make use of the structural metadata provided by QB4OLAP to translate CQL queries into SPARQL ones. We adapt general-purpose SPARQL query optimization techniques, and propose query improvement strategies to produce efficient SPARQL queries. We evaluate our implementation tailoring the well known Star-Schema benchmark, which allows us to compare our proposal against existing ones in a fair way. We show that our approach outperforms other ones. Finally, as another result, our experiments allow us to study which combinations of improvement strategies fits better to an analytical scenario.

Key words: Semantic Web, OLAP, Multidimensional data.

RESUMEN

La *World-Wide Web* fue concebida como un repositorio de información a ser procesada y consumida por humanos. Pero en la última década ha ganado impulso la idea de transformar a la Web en una gran base de datos procesables por máquinas. Con este fin, el *World Wide Web Consortium* (W3C) ha establecido una serie de estándares también conocidos como estándares para la Web Semántica (WS), los cuales permiten compartir datos y metadatos en formatos abiertos. Entre estos estándares se destacan: el *Resource Description Framework* (RDF), un modelo de datos basado en grafos para representar datos y relaciones entre ellos, RDF-S y OWL que permiten describir la estructura y el significado de los datos por medio de ontologías o vocabularios, y el lenguaje de consultas SPARQL. Estos estándares pueden ser utilizados para construir representaciones de otros modelos de datos, por ejemplo datos tabulares o datos relacionales.

El modelo de datos multidimensional (MD) representa a los datos dentro de un espacio n-dimensional, usualmente denominado cubo de datos, que se compone de dimensiones y hechos. Las primeras reflejan las perspectivas desde las cuales interesa analizar los datos, mientras que las segundas corresponden a puntos en este espacio n-dimensional, a los cuales se asocian valores usualmente numéricos, conocidos como medidas. Los hechos pueden ser agregados y resumidos, desagregados, y filtrados utilizando las dimensiones. Este proceso es conocido como Online Analytical Processing (OLAP).

Pese a que la W3C ha establecido un estándar que puede ser utilizado para publicación de datos multidimensionales, conocido como el RDF Data Cube Vocabulary (QB), éste no incluye algunos aspectos del modelo MD que son imprescindibles para realizar análisis tipo OLAP como son las jerarquías de dimensión, los atributos en los niveles de dimensión, y las funciones de agregación para resumir valores de medidas. Para permitir este tipo de análisis sobre cubos en la SW, en esta tesis se propone un vocabulario que extiende el vocabulario QB denominado QB4OLAP.

Sin embargo, para realizar análisis tipo OLAP en forma eficiente sobre cubos QB4OLAP es necesario un conocimiento profundo de RDF y SPARQL, los cuales distan de ser populares entre los usuarios OLAP típicos. Esta tesis también aborda este problema. Nuestro enfoque consiste en brindar un conjunto de operaciones clásicas para los usuarios OLAP, y luego realizar la traducción en forma automática de estas operaciones en consultas SPARQL. Comenzamos definiendo un lenguaje de consultas para cubos en alto nivel: Cube Query Language (CQL), y luego explotamos la metadata representada mediante QB4OLAP para realizar la traducción a SPARQL. Asimismo, mejoramos el rendimiento de las consultas obtenidas, adaptando y aplicando técnicas existentes de optimización de consultas SPARQL. Para evaluar nuestra propuesta adaptamos a los estándares de la SW

el *Star Schema benchmark*, el cual es el estándar para la evaluación de sistemas tipo OLAP. Esto permite comparar nuestro enfoque con otras propuestas existentes, asi como evaluar el impacto de nuestras estrategias de mejoras de consultas SPARQL. De esta comparación podemos concluir que nuestro enfoque supera a otras propuestas existentes, y que nuestras técnicas de mejoras logran incrementar en 10 veces el rendimiento del sistema.

Palabras clave: Web Semántica, OLAP, Datos Multidimensionales

PUBLICATIONS

Some ideas and figures contained in this document have already appeared in previous publications. Chapter 4 has partially appeared in [3] and [4]. The ideas in Chapter 5 were presented in [1] and [6] . The formal model described in Chapter 6 was presented in [2], while the querying approach was described in [5]. In [8] we have presented the web application that implements our querying approach (detailed in Section 6.7). Finally, we contributed to sections related to the QB4OLAP vocabulary (Introduction, Preliminaries, Representing Multidimensional Data in RDF, and Related Work) in [7].

[1] Mauricio Bouza, Brian Elliot, Lorena Etcheverry, and Alejandro A. Vaisman. "Publishing and Querying Government Multidimensional Data Using QB4OLAP." In: *9th Latin American Web Congress, LA-WEB 2014, Ouro Preto, Minas Gerais, Brazil, 22-24 October, 2014*. 2014, pp. 82–90 (cit. on p. ix).

[2] Lorena Etcheverry, Silvia A. Gómez, and Alejandro A. Vaisman. "Modeling and Querying Data Cubes on the Semantic Web." In: *CoRR* abs/1512.06080 (2015). URL: http://arxiv.org/abs/1512.06080 (cit. on p. ix).

[3] Lorena Etcheverry and Alejandro A. Vaisman. "Enhancing OLAP Analysis with Web Cubes." In: *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*. Vol. 7295. 2012, pp. 469–483 (cit. on p. ix).

[4] Lorena Etcheverry and Alejandro A. Vaisman. "QB4OLAP: A Vocabulary for OLAP Cubes on the Semantic Web." In: *Proceedings of the Third International Workshop on Consuming Linked Data, COLD 2012, Boston, MA, USA, November 12, 2012*. Vol. 905. 2012 (cit. on p. ix).

[5] Lorena Etcheverry and Alejandro A. Vaisman. "Querying Semantic Web Data Cubes." In: *Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Panama City, Panama, May 8-10, 2016*. 2016 (cit. on p. ix).

[6] Lorena Etcheverry, Alejandro A. Vaisman, and Esteban Zimányi. "Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP." In: *Data Warehousing and Knowledge Discovery - 16th International Conference, DaWaK 2014, Munich, Germany, September 2-4, 2014. Proceedings*. Vol. 8646. 2014, pp. 45–56 (cit. on p. ix).

[7] Jovan Varga, Alejandro A. Vaisman, Oscar Romero, Lorena Etcheverry, Torben Bach Pedersen, and Christian Thomsen. "Dimensional Enrichment of Statistical Linked Open Data." In: (). Submitted to the Journal of Web Semantics (December 2015) (cit. on p. ix).

[8] Jovan Varga, Lorena Etcheverry, Alejandro Vaisman, Oscar Romero, Torben Pedersen, and Christian Thomsen. "QB2OLAP: Enabling OLAP on Statistical Linked Open Data." In: *32nd IEEE International Conference on Data Engineering ICDE 2016, Helsinki, Finland, May 16-20, 2016. Demo paper* (cit. on p. ix).

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

Donald E. Knuth

ACKNOWLEDGMENTS

Pursuing a doctoral degree is certainly like running a marathon, where lots of stamina and endurance are needed. I'm very thankful to all the people that supported me during this journey.

First, I would like to thank the guidance of my doctoral advisor, Alejandro Vaisman, for his commitment, insights, and suggestions.

My gratitude extends to my colleagues at InCo for their valuable advice, and time, in particular to those with whom I've shared many hours of mutual support, and insightful discussions.

I also want to thank Esteban Zimányi, Oscar Romero, Torben B. Pedersen, Christian Thomsen, and Jovan Varga, for all the fruitful discussions on Multidimensional Data modeling, and Oscar Corcho for his comments on QB4OLAP vocabulary design. Thank you very much!

My research interest in Linked Data and the Semantic Web is thanks to Mariano Consens, who introduced me to these topics during a tutorial he gave at Montevideo in 2009. Thanks Mariano!

I am honored and thankful to my thesis committee members: Alberto Abelló, Claudio Gutierrez, María Esther Vidal, Héctor Cancela, and Adriana Marotta. It is a privilege for me that they accepted to be part of this committee.

Last, but not least, thanks to my family and friends, for listening when I'm angry, hugging me when I'm sad, and laughing with or at me whenever possible. In particular, thank you Edu for being by my side throughout the most crucial parts of this journey. Your constant support, patience, and love are essential to me.

This dissertation would not have been possible without funding from the Comisión Académica de Posgrado (CAP), Universidad de la República, Uruguay.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

"The White Rabbit put on his spectacles.
'Where shall I begin, please your Majesty?' he asked.
'Begin at the beginning,' the King said gravely,
'and go on till you come to the end: then stop.' "

Lewis Carroll, Alice In Wonderland

The World-Wide Web was initially conceived as a repository of information tailored for human consumption. In the last decade, the idea of transforming the web into a machine-understandable web of data, has gained momentum. The World Wide Web Consortium (W3C)[1] has developed a set of standards, referred to as the *Semantic Web* (SW), to assist in the transformation of the traditional web of Documents into a web of data. Among these standards is the Resource Description Framework (RDF) [33], which represents data as graphs, and is the data model of the SW. The W3C has also defined SPARQL [13] as the RDF query language for RDF, and the ontology languages RDF-S [5] and OWL [15] to describe the data structure via ontologies or *vocabularies*. The interoperability between heterogeneous data sets is achieved by using common vocabularies to describe data, which help in the integration of published data sets.

The RDF data model represents data as sets of triples of the form $(\texttt{subject}, \texttt{predicate}, \texttt{object})$, which can be seen as a graph, where `subject` and `object` represent nodes, and `predicate` represents and edge. A set of standards or good practices are needed to specify how RDF data must be represented. For example, the *Linked Data* paradigm is a set of best practices for publishing and interlinking structured data on the web using SW standards, in particular RDF [14]. Recent studies report that the amount of open data available as Linked Data is approximately 90 billion triples in over 3,300 data sets, most of them freely-accessible via SPARQL query endpoints.[2] However, these general recommendations focus on the representation of tabular or relational data, but they do not suffice to represent other data models, in particular multidimensional data.

The *multidimensional* (MD) model views data in an n-dimensional space, usually called a *data cube*, composed of *dimensions* and *facts*. The former reflect the perspectives from which data are viewed, and the latter correspond to points in this space, associated with (usually) quantitative data (also known as *measures*). There is a large number of MD models in the literature based on the data cube metaphor [11, 16, 45]. *Statistical databases* are closely related to the MD data model, since they also organize data in hypercubes, but the underlying data model shows some differences [37]. Online Analytical Processing (OLAP) is

1 http://www.w3.org/
2 http://www.stats.lod2.eu/

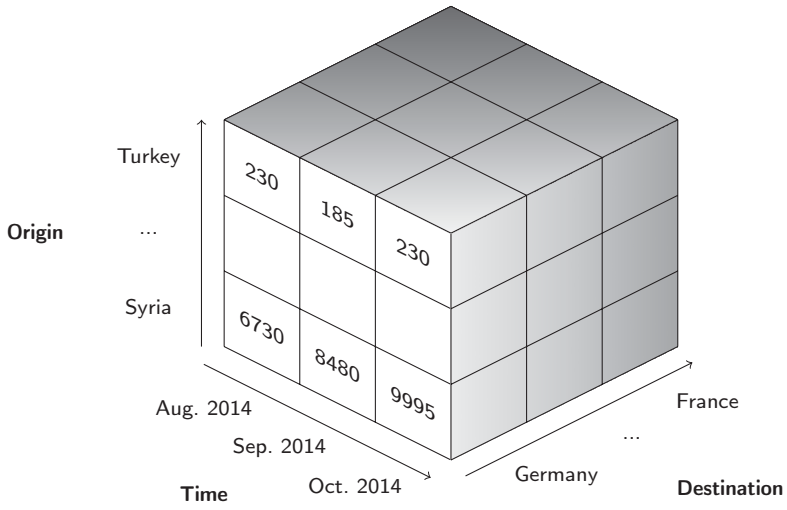


Figure 1: Sample data from the asylum applications data cube.

a well-established approach for MD data analysis to support decision making, and is one of the techniques used in Business Intelligence (BI) processes. OLAP operations allow to aggregate, disaggregate, and filter facts using the dimensions.

As an example of MD data, consider the evolution over time of asylum applications in Europe. These data can be modeled as an OLAP cube, with dimensions Origin, Destination, and Time, as shown in Figure 1. Dimension instances are composed of *members*. For example, 'Syria' is a member of the dimension Origin. A cell in this cube represents a *fact*, and is of the form ('Syria', 'Germany', '201410', 9995), meaning that 9995 asylum applications were issued to Germany by Syrian citizens in October, 2014. The first three elements in the tuple represent the dimension members (the cube coordinates), while the last one is the measure that quantifies the fact. This kind of representation allows the end user to analyze data in a very simple way. Thus, we can see in Figure 1 that the number of asylum applications to Germany, issued by Syrian citizens, has steadily increased during 2014, while the applications to Germany, issued by Turkish citizens, is quite stable in the same period.

Aggregations can be performed along dimension hierarchies. For example, months can be organized into years, such that, for instance, '201408' and '201410' belong to the Month level and are related to '2014' at the Year level. OLAP operations allow us, for instance, to aggregate and disaggregate data. For example, aggregating the factual data in Figure 1 up to the Year level, we can see that the total asylum applications to Germany, issued by Syrian and Turkish citizens, amount to 25205 and 745, respectively. Figure 2 shows a representation of the aggregated cube. Although many different OLAP operators can be found in the literature, the most common ones are:

ROLL-UP: Aggregates measures along a dimension hierarchy, using an aggregate function, producing measures at a coarser granularity.

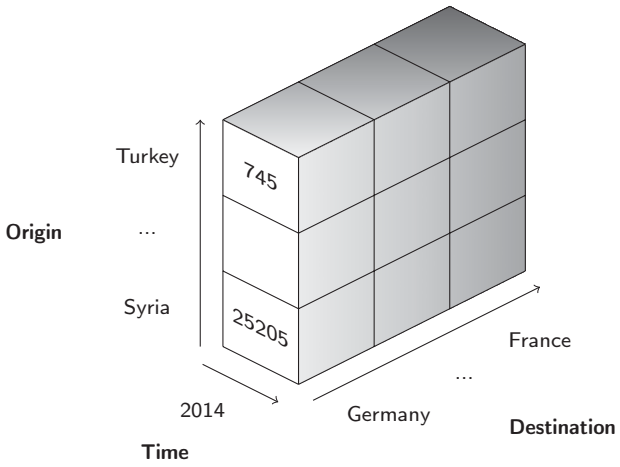


Figure 2: Sample data from Figure 1 aggregated to the Year level.

DRILL-DOWN: Disaggregates previously summarized data, and can be considered the inverse of Roll-Up.

SLICE: Removes a dimension from a cube.

DICE: Selects a subset of the cells of a cube, that satisfy a Boolean formula ϕ over level members or measure values.

Typically, the MD data model and OLAP operations relate to Data Warehouse (DW) systems, and have been historically used as techniques for data analysis *within* organizations using, mostly, commercial tools with proprietary formats. In the context of the web of data, the need to publish, share, and analyze data cubes did not take long to arise. In particular, initiatives such as Open Data[3] and Open Government Data[4], are encouraging organizations to publish statistical and MD data. In this new scenario, the BI community faces several challenges. We want to emphasize on two of them. First, there is a need for instruments to represent MD data and metadata (e.g., dimensional structure, which is essential to adequately interpret and reuse data), using SW standards. Second, it is necessary to provide mechanisms to analyze SW data *á la* OLAP.

To cope with the first challenge, the W3C defined the *RDF Data Cube Vocabulary*[8] (QB), as an standard for the representation of statistical data. Although statistical data and MD data for OLAP are quite similar, they are not the same. As a consequence, QB does not include key features needed for OLAP analysis, such as the possibility to represent aggregation hierarchies, and aggregate functions. Further, QB only supports dimensions with one aggregation level. These metadata are needed to automate the translation of OLAP operations into the underlying technology storing the MD data. For example, DWs have been typically implemented using relational technology and the definition of a *well-formed* MD schema allows the automatic translation of OLAP operations into SQL queries. In Chapter 8 we discuss QB, and show that it fails to adequately represent MD data for OLAP.

3 http://www.okfn.org/opendata/
4 http://www.opengovdata.org/

Concerning OLAP analysis of SW data (the second challenge), two main approaches can be found in the literature. The first one consists in extracting MD data from the web, and loading them into traditional data management systems for OLAP analysis. The second one explores data models and tools that allow publishing and performing OLAP analysis directly over MD data, using SW standards. Our work positions in the latter scenario, paying particular attention to: (1) the use of SW technologies to model, manipulate, and share MD data; (2) the specific characteristics of SW data (e.g: amount of data, distributed storage, etc.) that determine the need for novel data analysis techniques; and (3) the extraction of MD data from the SW, identifying dimensions and facts that can be then analyzed using traditional OLAP analysis tools. All of these requires the definition of a precise vocabulary for representing adequately the BI (in particular, OLAP) data on the SW. Over these vocabulary, MD models and OLAP operators can be defined. To address this need, in this thesis we propose an extension to QB, denoted QB4OLAP.

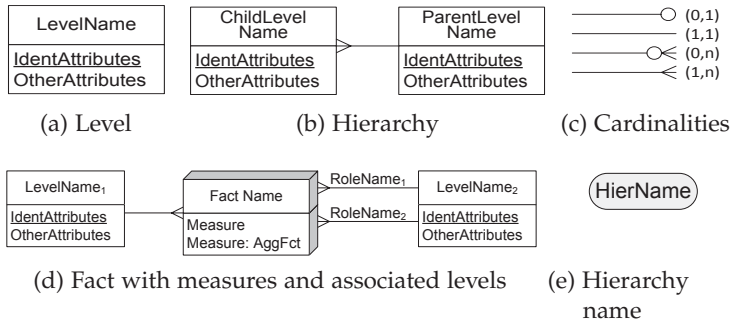
The main goal of this thesis is to provide a representation of MD data and metadata using SW standards, such as RDF and RDF-S, and to study the problem of performing OLAP operations over this representation.

1.1 CASE STUDIES

We motivate our work with two case studies, which we use throughout this thesis. The first one corresponds to a well-known fictitious case study, namely the NorthWind DW. The second one corresponds to the analysis of real-world data about asylum applications in Europe, originally published by the European Union Statistics Agency (Eurostat).[5] We use the *MultiDim* model [29, 42] to present the conceptual schema of each case. Of course, any conceptual model could be used instead. To make this document self-contained we next sketch the main components of the MultiDim model.

THE MULTIDIM CONCEPTUAL MODEL The main components of Multidim are depicted in Figure 3. A *schema* is composed of a set of dimensions and a set of facts. A *dimension* is composed of either one *level*, or one or more hierarchies. Instances of a level are called *members*. A level has a set of *attributes* that describe the characteristics of their members (Figure 3a), and one or more *identifiers*, each identifier being composed of one or several attributes. A *hierarchy* is composed of a set of levels (Figure 3b). Given two related levels in a hierarchy, the lower one is called the *child* and the higher one, the *parent*; the relationships between them are called *parent-child relationships*, whose *cardinalities* are shown in Figure 3c. A dimension may contain several hierarchies identified by a *hierarchy name* (Figure 3e). The name of the leaf level in a hierarchy defines the dimension name, except when the same level participates several times in a fact, in which case the role name defines the dimension name. These are called *role-playing dimen-*

5 http://www.ec.europa.eu/eurostat/web/products-datasets/-/migr_asyappctzm



(a) Level (b) Hierarchy (c) Cardinalities

(d) Fact with measures and associated levels (e) Hierarchy name

Figure 3: Notation of the MultiDim model

sions. A *fact* (Figure 3d) relates several levels. Instances of a fact are called *fact members*. A fact may contain attributes called *measures*. The aggregation function associated to a measure can be specified next to the measure name (Figure 3d), the default being the SUM function.

It is worth noting that *MultiDim* does not have a graphical element to represent a dimension. Thus, a dimension is depicted by means of its constituent elements, namely, levels and hierarchies. For example, the conceptual model in Figure 4 shows six dimensions. The bottom level of each dimension is connected to the fact, and the same level can participate several times in a fact, playing different roles. Each role is identified by a name and is represented by a separate link between the corresponding level and the fact. We will assume that the name of the bottom level is the name of its corresponding dimension. If a level participates with different roles in a fact, we will consider each participation as a different dimension, whose name is the name of the role itself. For example, in Figure 4, the Time dimension participates in the data cube with two different roles: the OrderDate and the DueDate of the sale.

1.1.1 *The NorthWind DW*

The NorthWind DW is a well-known example that represents sales from a retail company. Figure 4 depicts a simplified version of this data set (adapted from [42]) that we will use as one of our case studies. The Sales fact contains three measures: Quantity, which represents the number of items purchased in each sale; UnitPrice, which represents the price of each unit; and SalesAmount, which represents the total amount of each sale. These measures can be analyzed according to six analysis dimensions: the Product that has been purchased, the Employee that performed the sale, the Customer that purchased the items, the Supplier that supplies the items, and finally the Time dimension that participates with two different roles that correspond to the OrderDate and the DueDate of the sale.

We have chosen this example as a case study, because its conceptual model includes features that go beyond the typical MD modeling constructs. For example, the geographical dimension presents a *ragged hierarchy*, the supervision hierarchy is a *recursive hierarchy*, and

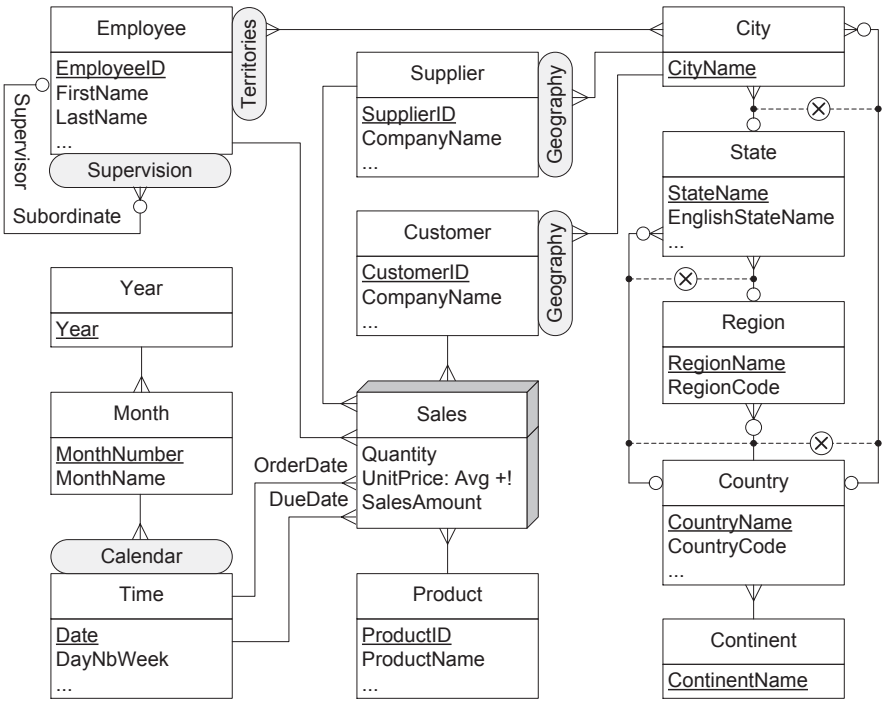


Figure 4: Conceptual schema of the NorthwindDW (adapted from [42])

the time dimension participates with more than one role in the cube. This modeling constructs are presented in Section 2.1.

1.1.2 *Asylum Applications Data from Eurostat*

Our second case study represents asylum applications to countries in the European Union, and contains information about the number of asylum applicants per month, age, sex, country of citizenship, application type, and country that receives the application. This data set is published in RDF using the QB vocabulary, in the Eurostat - Linked Data dataspace.[6]

QB data sets are composed of a set of *observations* representing data instances according to a *data structure definition*, which describes the schema of the data cube. As an example, and given that we will explain QB later in this work, in Table 1 we show some observations in the original data cube, in tabular format. The first row lists the dimensions in the cube, and the second row lists the dimension level that corresponds to the observation.

We wanted to enrich the original data set in order to enhance the analysis possibilities. Thus, we made use of the features of QB4OLAP, the vocabulary we propose in this work, to reuse the published observations, and create new dimension hierarchies. We discuss how this extension is performed in Section 5.3. Conceptually, we created three new dimension levels. In the geographical dimension we added two new levels: one, to represent Continents, and another one to represent the Government type of each country. In the time dimension

6 http://www.eurostat.linked-statistics.org/

SEX	AGE	TIME	APPLICATION TYPE	CITIZENSHIP	DESTINATION	MEASURES
Sex	*Age*	*Month*	*Application type*	*Country*	*Country*	*#applications*
F	18 to 34	201409, September 2014	new applicant	SY, Syria	DE, Germany	425
M	18 to 34	201409, September 2014	new applicant	SY, Syria	DE, Germany	1680
M	18 to 34	201409, September 2014	new applicant	SY, Syria	FR, France	95

Table 1: Tabular representation of sample observations in the asylum applications data cube.

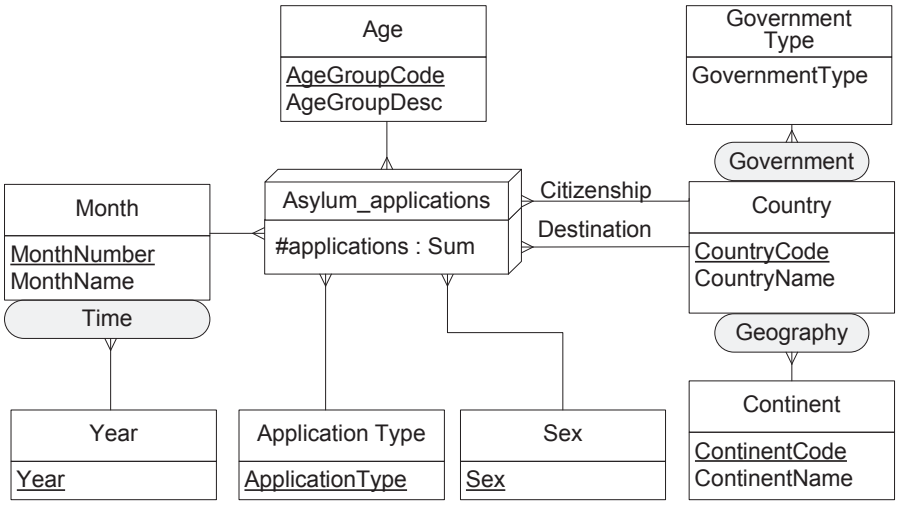


Figure 5: Conceptual schema of the Asylum Applications cube

we added the Year level to organize months. To implement this, we enriched the existent data set with DBpedia[7] data to populate the new levels in the geographical dimension, and manipulated data on dates to populate the Year level. Figure 5 shows the resulting conceptual schema of the data cube. The asylum_applications fact contains a measure (#applications) that represents the number of applications. This measure can be analyzed according to six analysis dimensions: the sex of the applicant, age which organizes applicants according to their age group, month which represents the time of the application and consists of two levels (month and year), application_type that tells if the applicant is a first-time applicant or a returning applicant, and a geographical dimension that organizes countries into continents (Geography hierarchy) or according to its government type (Government hierarchy). This geographical dimension participates in the cube with two different roles: the citizenship of the asylum applicant, and the destination country of the application.

Thus, over the new cube in Figure 5 we can pose queries such as "*Total asylum applications per year*", or "*Total asylum applications per year submitted by Asian citizens to France or United Kingdom, where this number is higher than 5,000*", which we discuss later in this paper.

7 http://www.dbpedia.org

1.2 CONTRIBUTIONS

As a *first contribution*, we define QB4OLAP vocabulary, an extension to the QB, that supports most of the concepts defined in classic MD models for OLAP, and which cannot be represented in QB, among other ones: dimension hierarchies, aggregate functions, and level attributes.

As a *second contribution*, we study how to represent existing data cubes using QB4OLAP. We propose and implement algorithms to translate traditional data cubes (stored in relational databases) into RDF using QB4OLAP. We also propose how to extend existing data cubes, represented using the QB vocabulary.

As our *third contribution*, we formalize a MD data model, and show that a data cube represented using this model, can be represented using the QB4OLAP vocabulary.

As our *fourth contribution*, we propose a query language for OLAP (denoted CQL) where the main object is the data cube. CQL provides the operators needed to manipulate the data cube. This way, the user just queries data cubes, independently of the underlying data representation. We use our data model to clearly define the semantics of these operators. The idea is that an OLAP user, without any knowledge of SPARQL or SW concepts, can write QL queries over a conceptual MD model, regardless the underlying data model and data types. Thus, users can exploit MD data directly over the web, without the need of exporting these data to a relational repository.

As our *fifth contribution* we: (1) present a high-level query simplification heuristic strategy for CQL queries; (2) propose algorithms to automatically translate CQL queries into equivalent SPARQL ones over QB4OLAP data cubes; (3) propose an heuristic-based strategy to improve the performance of the SPARQL queries produced in (2); (4) introduce a benchmark, based on TPC-H Star-Schema benchmark, to evaluate the performance of the SPARQL queries; we show that our improvement procedure substantially speeds up the query evaluation process, and outperforms other proposals; (5) present the *QB4OLAP toolkit*, a web application that allows exploring and querying QB4OLAP cubes using the machinery described in this document.

1.3 OVERVIEW

In Chapter 2 we summarize preliminary concepts that are used along this thesis, while in Chapter 8 we present the context and related work. The rest of the document is organized in three parts: the first part deals with the *representation of data cubes using SW Standards*, the second deals with *OLAP analysis of SW data cubes*, and the third presents the conclusions of our work in both directions.

Part i is organized in two chapters. In Chapter 4 we present our proposal to represent data cubes on the SW, based on the QB4OLAP vocabulary. Then, in Chapter 5, we discuss how to create QB4OLAP data cubes. In particular, we show how existing QB cubes can be

extended using QB4OLAP, and also show how to produce QB4OLAP cubes from relational cubes.

Part ii is also organized in two chapters. In Chapter 6 we present our approach to OLAP analysis over SW data cubes, based on a high level language and its implementation as SPARQL queries. Then, in Chapter 7 we present the evaluation of our approach. Part iii contains our conclusions.

2

PRELIMINARY CONCEPTS

"Those are my principles,
and if you don't like them...well I have others."

Groucho Marx

In this section we present the concepts on MD data modeling, statistical databases, and SW standards that are relevant to this work.

2.1 MULTIDIMENSIONAL DATA MODELING

As already mentioned, the MD model views data in an n-dimensional space, usually called a *data cube*, where *dimensions* represent the perspectives from which data are viewed, and *facts* represent points in this space and are associated with usually quantitative data (also known as *measures*). Although there is a large number of MD models in the literature based on the data cube metaphor [11, 16, 45], most of them agree on considering that dimensions are organized in *hierarchies*, which allow representing the data under analysis at different abstraction levels. Dimension hierarchies can be classified according to their characteristics. We now present several kinds of hierarchies, following [42].

BALANCED HIERARCHIES These are the simplest kind of hierarchies, where at the conceptual level there is only one path. At the instance level, all parent members have at least one child member, and each child member has exactly one corresponding parent member. The Month dimension, in Figure 5, is an example of this.

UNBALANCED AND RECURSIVE HIERARCHIES Unbalanced hierarchies despite having only one path at the conceptual level, not enforce all parent members to have children. Recursive hierarchies are considered a special case of unbalanced hierarchies, where a level plays the role of parent and child in a parent-child relationship. As an example of this consider the Supervision hierarchy in the Employee dimension from Figure 4. It is easy to see that this hierarchy is unbalanced, since employees with no subordinates do not have corresponding child members.

GENERALIZED AND RAGGED HIERARCHIES Generalized hierarchies contain multiple **exclusive** paths, that share at least the leaf level. At the instance level, each member belongs to only one path. These hierarchies are useful to represent, at the same level, members of different types. In Multidim, the symbol $\otimes$ is used to indicate exclusive paths. Ragged hierarchies are a special case of generalized hierarchies, where alternative paths are obtained by skipping one or

more intermediate levels. As an example, consider the Geography hierarchy in the Supplier dimension from Figure 4. This hierarchy allows a flexible representation of geographical administrative organization. For example some countries may be organized in states, while others use regions and states. The hierarchy is flexible enough to allow some cities to be directly related to a country, while other should be related to the state they belong.

ALTERNATIVE AND PARALLEL HIERARCHIES Alternative hierarchies also contain multiple paths that share at least the leaf level, but these paths are **non exclusive**. At the instance level, each member of the leaf level belongs to all the paths, which represent different analysis criteria that the user may choose. Parallel hierarchies occur when more than one hierarchy can be defined for a dimension.

NONSTRICT HIERARCHIES Usually, parent-child relationships cardinality is one-to-many. We refer to these as **strict** hierarchies. In opposition, those that have at least one many-to-many parent child relationship are called **nonstrict**. As an example, consider the parent-child relationship between Employee and City levels, in the Territories hierarchy from Figure 4, representing that an employee can work in more than one city. Nonstrict hierarchies introduce the problem of measure double counting in aggregations. This means that, in our example, the quantity items sold by each employee will add-up to each of the cities where the employee works. Different strategies can be found in the literature to cope with this problem.

2.2 STATISTICAL DATABASES

Statistical Data Bases (SDB) also organize data as hypercubes whose axes are *dimensions*, and dimensions are structured in *classification hierarchies* that allow analysis at different levels of aggregation. Each point in this MD space is mapped through *observations* into one or more spaces of *measures*. The Statistical Data and Metadata eXchange inititative (SDMX)[1] propose several standards for the publication, exchange and processing of statistical data. In particular, an information model is defined [35] from which we summarize some concepts that are relevant for the remainder.

A *Dimension* denotes a metadata concept used to classify a statistical series, e.g., a statistical concept indicating a certain economic activity or a geographical reference area. Two particular dimensions are identified: the *TimeDimension*, used to convey the time period of the observation in a data set; and the *MeasureDimension*, whose purpose is to specify formally the meaning of the measures. Dimensions, measures, and attributes are generally referred as *Components*. *Codelists* enumerate a set of values to be used in the representation of dimensions, attributes, and other structural parts of SDMX. They can be supplemented by other structural metadata which indicates how

1 http://SDMX.org

OLAP	SDB
Roll up	S-aggregation
Drill down	S-disaggregation
Slice	S-projection
Dice	S-selection
Drill across	S-union

Table 2: Correspondence between OLAP and SDB operators [37]

codes are organized into hierarchies. A *Data Set* denotes a set of observations that share the same dimensionality, which is specified by a set of unique components (Dimension, *MeasureDimension*, *TimeDimension*), together with associated *AttributeValues* that define specific characteristics about the artifact to which it is attached. Each data set has a set of structural metadata. These descriptions are referred to in SDMX as *Data Structure Definitions* (DSD). The DSD includes information about how concepts are associated with the measures, dimensions, and attributes of a data 'cube' along with information about the representation of data and related identifying and descriptive (structural) metadata.

Several operators are defined over SDBs, and a correspondence can be defined between these and OLAP ones. Table 2 presents a correspondence taken from the classic work by Shoshani [37]. The SDMX standard does not define operators over data sets, but provides a mechanism to restrict the values within a data set via *constraints. CubeRegions* or *Slices* are a particular kind of constraint that allow to specify a set of component values, defining a subset of the total range of the content of a data structure. The application of a *Slice* constraint results on a subset of the original data set, fixing values for some components (e.g: selecting some years in a TimeDimension) but not reducing its dimensionality. Is it worth noting that the name *Slice* may be misleading, since the result of the application of a *CubeRegion* constraint resembles a *dicing* OLAP operation rather than a *slicing* one.

2.3 SEMANTIC WEB STANDARDS

In this section we summarize the W3C standards used in this thesis.

2.3.1 *RDF*

RDF [33] is the base data model and language of the SW. The basic construct of RDF is a triple, of the (s, p, o) form, where s stands for *subject*, p for *predicate*, and o for *object*. In general, s, p, and o are *resources*, identified with internationalized resource identifiers (IRIs). An object can also be a data value, denoted a *literal* in RDF, or a *blank node*, typically used to represent anonymous resources. Subjects can also be represented by blank nodes. A set of RDF triples or *RDF data*

set is a directed graph whose nodes are subjects or objects, and whose arcs represent predicates. IRIs are commonly written as a *prefix* label and a local part, separated by a colon ":". These are turned into IRIs by concatenating the IRI associated with the prefix and the local part. Usually, triples representing schema and instance data coexist in RDF data sets. A collection of reserved words defined in RDF Schema [5] (called the RDF-S vocabulary) is used to define classes, properties, and hierarchical relationships. For example, the triple (*r*, `rdf:type`, *c*) explicitly states that *r* is an instance of *c*, and it also implicitly states that object *c* is an instance of `rdfs:Class`.

We now introduce a graphical representation for RDF graphs that we use in this thesis. Ellipses represent IRI-identified resources, rectangles represent literals, and empty circles represent blank nodes, Labeled arcs represent predicates. Figure 6 uses this representation to show data about Germany, retrieved from DBPedia[2] and GeoNames[3]. All the RDF prefixes used in this work are defined in Appendix A.

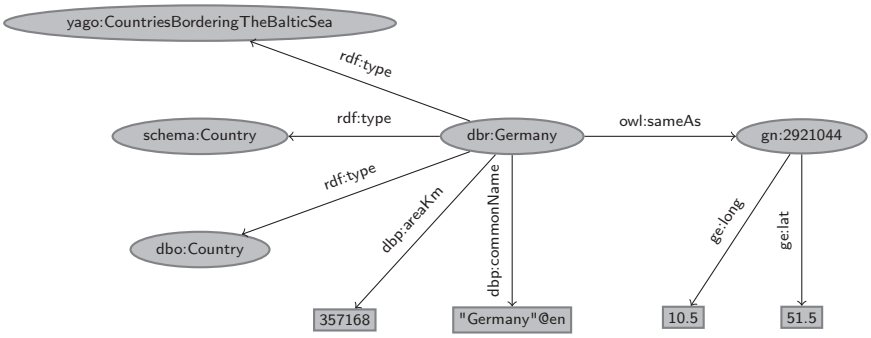


Figure 6: Sample RDF data.

Finally, many formats for RDF serialization exist, and in this thesis we use Turtle [4]. In this notation, triples are separated by '.' . The ';' symbol is used to repeat the subject of triples that vary only in predicate and object, while the ',' symbol is used to repeat the subject and predicate of triples that only differ in the object. The special predicate `rdf:type` may be abbreviated as 'a'. Example 2.3.1 shows the triples represented in Figure 6 using this notation.

Example 2.3.1. The triples below correspond to the data depicted in Figure 6

```
dbr:Germany rdf:type dbo:Country .
dbr:Germany rdf:type yago:CountriesBorderingTheBalticSea ,
                     schema:Country .

dbr:Germany owl:sameAs gn:2921044 ;
            dbp:areaKm 357168 ;
            dbp:commonName "Germany"@en .

gn:2921044 ge:lat 51.5; ge:long 10.5 .
```

□

2 Germany on DBpedia http://dbpedia.org/page/Germany
3 Germany on GeoNames http://www.geonames.org/2921044/federal-republic-of-germany.html

2.3.2 *SPARQL*

SPARQL 1.1 [13] is the W3C standard query language for RDF at the moment of writing this thesis. The query evaluation mechanism of SPARQL is based on subgraph matching: RDF triples are interpreted as nodes and edges of directed graphs, and the query graph is matched to the RDF data graph, instantiating the variables in the query graph definition. The selection criteria is expressed as a graph pattern in the `WHERE` clause, composed by *basic graph patterns (BGP)*. The '.' operator represents the conjunction of graph patterns. Relevant to our study, SPARQL 1.1 supports aggregate functions and the `GROUP BY` clause. Example 2.3.2 shows an example.

Example 2.3.2. The following SPARQL query contains two BGPs. The first one retrieves triples with `dbr:Germany` as subject, `owl:sameAs` as predicate, and anything as object (variable ?s). The second BGP matches any triple in the graph. The conjunction of these BGPs retrieves, for each node connected to `dbr:Germany` via `owl:sameAs`, all the outgoing edges and connected nodes.

```
SELECT ?s ?p ?o
WHERE {dbr:Germany owl:sameAs ?s .
       ?s ?p ?o}
```

The results of running the query above over the graph in Example 2.3.1 are the following:

```
gn:2921044 ge:lat 51.5 .
gn:2921044 ge:long 10.5 .
```

□

2.3.3 *R2RML*

R2RML [9] is a language to express mappings from relational databases to RDF data sets, allowing representing relational data in RDF using a customized structure and vocabulary. Both, R2RML mapping documents (written in Turtle syntax) and mapping results, are RDF graphs. The main object of an R2RML mapping is the *triples map*, a collection of triples composed of a *logical table*, a *subject map*, and one or more *predicate object maps*. A logical table is either a base table or a view (using the predicate `rr:tableName`), or an SQL query (using the predicate `rr:sqlQuery`). A predicate object map is composed of a predicate map and an object map. Subject maps, predicate maps, and object maps are either constants (`rr:constant`), column-based maps (`rr:column`), or template-based maps (`rr:template`). Templates use column names as placeholders. Foreign keys are handled referencing object maps, which use the subjects of another triples map as the objects generated by a predicate-object map. A set of R2RML mappings can either be used to generate a static set of triples that represent the underlying relational data (*data materialization*) or to provide a non-materialized RDF view of the relational data (*on-demand mapping*). In this thesis we use R2RML to translate existent OLAP cubes, stored in relational databases, into QB4OLAP cubes (see Section 5.2)

2.4 SUMMARY

In this chapter we have presented the concepts on MD data modeling, statistical databases, and SW standards that are relevant to this work.

3

MULTIDIMENSIONAL DATA REPRESENTATION ON THE SEMANTIC WEB

"The idea of 10 dimensions might sound exciting,
but they would cause real problems
if you forget where you parked your car."

Stephen Hawking, The Grand Design

The problems studied in this thesis are relevant in the broader context of the so-called BI 2.0, an evolution of traditional BI and OLAP applications. In this chapter, we first describe this scenario. Then, we present the RDF Data Cube vocabulary (QB), the W3C recommendation to publish statistical data and metadata in RDF following the Linked Data principles. Although appropriate to represent and publish statistical data, QB has a set of shortcomings when it comes to represent a MD model for OLAP. In this chapter we also elaborate on these limitations

3.1 THE BIG PICTURE

Since the mid 90's, DWs and traditional OLAP applications have been built to consolidate enterprise business data, allowing taking timely and informed decisions based on up-to-date consolidated data. However, the availability of enormous amounts of data from different domains is calling for a shift in the way DW and BI practices are being carried out. It is becoming clear that, for certain kinds of BI applications, the traditional approach, where day-to-day business data produced in an organization is collected in a huge common repository for data analysis, needs to be revised, to account for efficiently handling large scale data. Moreover, in the emerging domains where BI practices are gaining acceptance, massive-scale data sources are becoming common, posing new challenges to the DW research community [41].

Recently, terms like *self-service BI* [1], *Situational BI* [28], and *Exploratory OLAP* [2], have emerged to refer to the capability of incorporating situational data into the decision process in the context of the so-called BI 2.0, with little or no intervention of programmers or designers. The web, and in particular the SW, is considered as a large source of data that could enrich decision processes (e.g., sentiment analysis data on user reviews about a product). Abello et al. [1] present an envisioned framework to support self-service BI, based on the notion of *fusion cubes*, i.e., MD cubes that can be dynamically extended both in their schema and instances, and in which data and metadata are associated with quality and provenance annotations. Figure 7 presents a functional perspective on self-service BI, where the user starts by posing an OLAP-like query that cannot be answered using only existent data. Then, the system finds potentially relevant

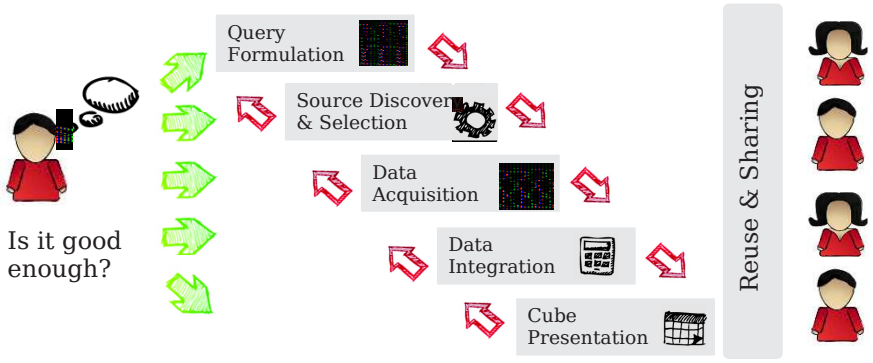


Figure 7: A functional perspective on self-service BI (adapted from [1])

data sources, gets data from them, curates and integrates fetched data, and finally presents the results to the user, who can then store and share the results. This envisioned scenario is supervised by the user, who evaluates the outcome of each step and may decide to intervene in previous steps to get better results.

But the SW is not only a source of data in this scenario, it is also a powerful set of tools that can play a key role by modeling data semantics. Abello et al. [2] exhaustively survey and discuss the possibilities of using SW technologies in the context of Exploratory OLAP, concluding that they can help to discover, acquire, integrate, and query new external data. Along these lines, Ibrahimov et al. [17] present a framework for Exploratory BI over Linked Open Data. Their goal is to semi-automatically derive MD schemas and instances, from already published Linked Data. This proposed framework uses the QB4OLAP vocabulary that we present in this thesis, to represent the discovered OLAP schemas, while the VoID vocabulary is used to link the schema with available SPARQL endpoints that can be used to populate the schemas. Varga et al. [43] present a methodology to semi-automatically discover hierarchical dimensions in Linked Data, starting from data cubes represented in QB. This work also proposes a semantic-based method to define which aggregate function is suitable for each metric. The resulting cube schema and instances are also represented using the QB4OLAP vocabulary.

3.2 THE RDF DATA CUBE VOCABULARY (QB)

As mentioned above, the RDF Data Cube Vocabulary (from now on we will use the term QB), is the W3C recommendation to publish statistical data and metadata in RDF, following the Linked Data principles. QB is based on the main components of the SDMX information model explained above. Figure 8 depicts the QB vocabulary. Capitalized terms represent RDF classes and non-capitalized terms represent RDF properties. Capitalized terms in italics represent "abstract" classes (i.e., classes with no instances). An arrow with black triangle head from class A to class B, labeled `rel` means that `rel` is an RDF property with domain A and range B. White triangles represent subclasses or sub-properties. For better comprehension, we next use the

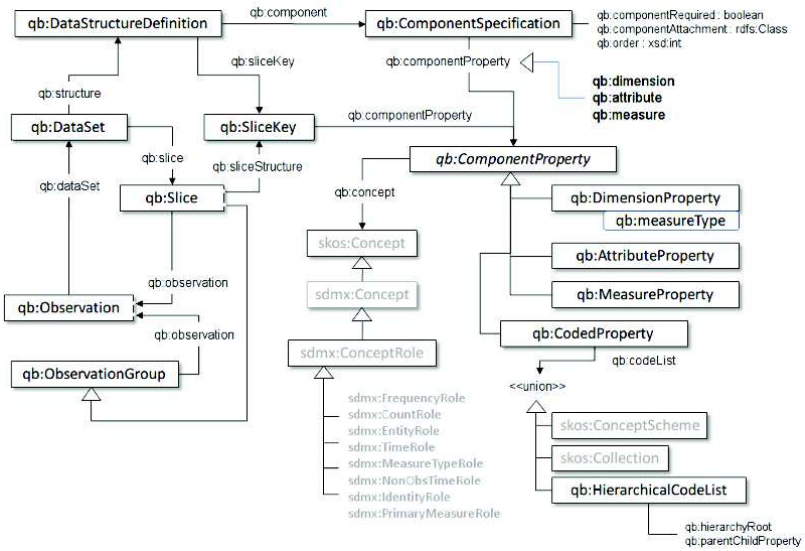


Figure 8: QB (cf. [8]) vocabulary

asylum applications dataset published by Eurostat, presented in Section 1.1.2, to explain the QB elements.

Following SDMX, the schema of a QB data set is specified by means of the *data structure definition* (DSD). This concept is modelled as an instance of `qb:DataStructureDefinition` class, and is formed by a set of *component* properties, which are instances of subclasses of the `qb:ComponentProperty` class that represent *dimensions*, *measures*, and *attributes*. Component properties are not directly related to the DSD: the `qb:ComponentSpecification` class is an intermediate class, typically instantiated as RDF blank nodes, that allows specifying additional attributes for a component in a DSD (e.g., a component may be tagged as *required* (i.e., mandatory), using the `qb:componentRequired` property). Different parts of a component specification are linked using properties that depend on the kind of component: `qb:dimension` for dimensions, `qb:measure` for measures, and `qb:attribute` for attributes. The `qb:component` property DSDs with component specifications. Note that a DSD can be shared by different QB data sets (and each QB data set is linked to its DSD) by means of the `qb:structure` property. Figure 9 shows a graphical representation of the asylum applications data cube schema, where the node `dsd:migr_asyappctzm` represents the DSD of the cube.

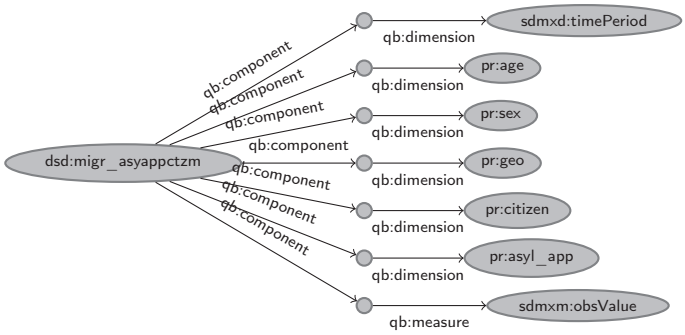


Figure 9: QB representation of the asylum applications data cube schema.

Example 3.2.1 below shows the triples that represent the DSD of the asylum applications case study. Prefix definitions are omitted in all the examples, please refer to Appendix A to expand them

Example 3.2.1. The DSD of the asylum applications example is defined in the file `migr_asyappctzm.ttl`[1] and looks as follows.

```
1 dsd:migr_asyappctzm a qb:DataStructureDefinition ;
2   qb:component [ qb:dimension sdmxd:timePeriod] ;
3   qb:component [ qb:dimension pr:age] ;
4   qb:component [ qb:dimension pr:sex ] ;
5   qb:component [ qb:dimension pr:geo] ;
6   qb:component [ qb:dimension pr:citizen] ;
7   qb:component [ qb:dimension pr:asyl_app] ;
8   qb:component [ qb:dimension sdmxm:obsValue] ;
9   skos:notation "migr_asyappctzm_DSD" .
```

This DSD is composed of six dimensions: `sdmxd:timePeriod` represents the month in which the application was issued, `pr:age` is the age group of the applicant, `pr:sex` is the sex of the applicant, `pr:citizen`, the applicant's country of origin, `pr:geo` is the applicant's country of destination, and, finally, `pr:asyl_app` represents the type of application (e.g., a new or returning asylum applicant). The measure of the data set is the generic `sdmxm:obsValue` predicate (line 8). □

Each DSD describes a MD data space indexed by *dimensions*, and measured values are associated with points in this data space. The associations between coordinates and measured values are called *observations* in the QB vocabulary (in OLAP terminology, *facts*). These are represented as instances of the `qb:Observation` class, which are organized in *data sets* (instances of the `qb:DataSet` class), through the `qb:dataSet` property. As already mentioned, each `qb:DataSet` is associated with the DSD that describes it, via the `qb:structure` property.

Every observation in a data set should be linked to a value in each dimension of its corresponding DSD using properties, which are instances of the `qb:DimensionProperty` class; analogously, values for each observation are associated with measures via instances of the `qb:MeasureProperty` class. Instances of `qb:AttributeProperty` represent observation attributes. Figure 10 shows an observation from the asylum applications case study, which states that 425 new asylum applications were issued by female persons in the age group 18 to 34 from Syria to Germany in March 2014. This observation corresponds to the third row of Table 1.

1 http://eurostat.linked-statistics.org/dsd/migr_asyappctzm.ttl

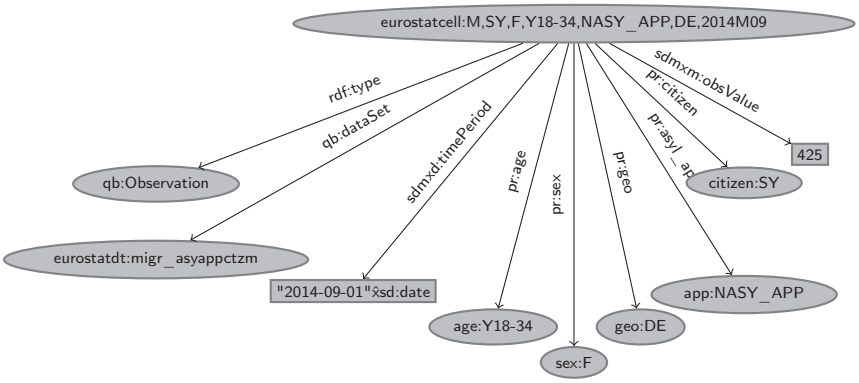


Figure 10: An observation from the asylum applications data set.

Example 3.2.2 below shows triples that represent the observation in Figure 10.

Example 3.2.2. The following triples represent the definition of a data set that contains all the observations of the asylum applications data cube (lines 1-3), followed by the triples that represent an observation in this dataset, stating that 425 new asylum applications were issued to Germany by female Syrian citizens in the age group 18 to 34 in March 2014 (lines 5-14). This observation corresponds to the third row of Table 1.

```
1  eurostatdt:migr_asyappctzm
2    rdf:type qb:DataSet;
3    qb:structure dsd:migr_asyappctzm .
4
5  eurostatcell:M,SY,F,Y18-34,NASY_APP,DE,2014M09
6    rdf:type qb:Observation
7    qb:dataSet eurostatdt:migr_asyappctzm ;
8    sdmxd:timePeriod "2014-09-01"^xsd:date ;
9    pr:age age:Y18-34 ;
10   pr:sex sex:F ;
11   pr:geo geo:DE ;
12   pr:citizen citizen:SY ;
13   pr:asyl_app app:NASY_APP ;
14   sdmxm:obsValue 425 .
```

Notice that each of the RDF properties defined as components in the DSD (Example 3.2.1) are used here to link the observation with either dimension members or measure values. In particular, at line 8, the `sdmxd:timePeriod` predicate links the observation with a literal value that represents a date. This imposes restrictions on the analysis capacities, because RDF triples cannot have literals as subject, and therefore, it is not possible to link a date modeled as a literal with further information (for example, the year). We discuss on this topic in Chapter 5

In order to allow reusing the concepts defined in the SDMX guidelines [34], QB provides the `qb:concept` property which links components to the general concepts they represent. The latter are modeled using the `skos:Concept` class defined in the SKOS vocabulary.[2] For example, a dimension property representing time, may be linked to the `sdmxc:timePeriod`.[3]

2 http://www.w3.org/TR/skos-reference/
3 http://purl.org/linked-data/sdmx/2009/concept

Although QB can define the structure of a fact (via the DSD), it does not provide a mechanism to represent an OLAP dimension structure (i.e., the dimension levels and the relationships between levels). However, QB allows representing hierarchical relationships between level *members* in the *dimension instances*. The QB specification describes three possible scenarios with respect to the organization of dimensions: (a) If there is no need to define hierarchical relationships between dimension members, QB recommends to represent the members using `skos:Concept` and the set of admissible values using `skos:ConceptScheme` or `skos:Collection`; (b) To represent hierarchical relationships, the `skos:narrower` predicate should be used, with the following meaning: If two concepts A and B are such that `A skos:narrower B`, B represents a narrower concept than A (e.g., `animals skos:narrower mammals`). The `skos:hasTopConcept` property allows linking a concept scheme, with the most general concepts it contains (note that this implies that hierarchies of values in QB should be traversed from coarser granularity concepts down to finer granularity concepts, while OLAP navigation usually traverses dimension hierarchies the other way round); (c) If publishers want to reuse existing data as their codelists, where hierarchical relationships are already defined using specific properties, QB provides the class `qb:HierarchicalCodeList`.

Finally, QB *slices* have the same semantics than SDMX slices, meaning that they are not operators over an existing cube, but new structures and new instances (observations) in which one or more values of dimension members are fixed. The structure of a slice is defined using a new DSD and an instance of the `qb:SliceKey` class. The class `qb:Slice` allows to group the observations that correspond to a particular slice (using the `qb:observation` property) and the structure of each slice is attached using the `qb:sliceStructure` property.

3.3 IS THE QB VOCABULARY ENOUGH TO MODEL MULTIDIMENSIONAL DATA FOR OLAP?

The QB vocabulary has been designed to publish statistical data. As already discussed in Section 2.2, statistical data are similar to MD data, but present some differences which impose limitations on the capability of QB for representing MD data suitable for OLAP. these limitations are discussed next. The first and main limitation we encounter in the QB vocabulary is its **lack of support for an OLAP dimension structure**. Although QB allows representing hierarchical relationships between *level members* in the *dimension instances*, it does not provide a mechanism to represent an OLAP dimension schema (i.e., the dimension levels and the relationships between levels)[4]. That means, QB allows stating that France is a concept of a finer granularity than Europe, but not that France is a Country, Europe is a Continent, and that countries aggregate over continents. QB proposes two mechanisms to represent hierarchical relationships between di-

4 http://www.w3.org/TR/vocab-data-cube/#schemes

mension members. The first one consists in using the semantic relationship `skos:narrower`, with the following meaning: If two concepts `A` and `B` are such that `A skos:narrower B`, `B` represents a narrower concept than `A` (e.g., `continent skos:narrower country`). Although this mechanism is similar to OLAP relationships between dimension members (called *rollup* relationships, as we will explain later) it presents two important differences. First, it allows navigating a hierarchy of dimension members from a top concept to more specific concepts, where rollup relationships navigate in the opposite direction. Second, it is usually assumed that rollup relationships in OLAP are transitive, while, according to the SKOS documentation[5], the `skos:narrower` property is not transitive. The second mechanism avoids the use of the `skos:narrower` property, and allows to associate each dimension with a custom property that implements parent-child relationships, but again proposes the navigation of the hierarchy from top concepts to more specific ones.

As a second limitation, **QB does not provide native support to represent aggregate functions**. Many OLAP operations change the granularity level of the data represented in a data cube (e.g., a roll-up operation over the Time dimension from the Month level up to the Year level). This involves aggregating measure values along dimensions, using the aggregate function defined for each measure. These aggregate functions depend on the nature of the measure (i.e., additive, semi additive, non additive [42]). The ability to link each measure with an aggregate function is therefore crucial and, although present in OLAP tools, it is not considered in QB.

Finally, **QB does not support level descriptive attributes**. In the MD model, each dimension level usually groups a set of real-world objects or concepts with similar characteristics. Thus, each level is associated with a set of *attributes* that describe the characteristics of their members. For example, the level Country may have the attributes countryName, population, etc., and one or more *identifiers* [42]. The latter uniquely identifies a level member, and the former describe its characteristics. QB does not provide a mechanism to associate a set of attributes with a dimension level. This makes it difficult to perform a *Dice* operation. For example, to produce a data cube that only refers to a country (e.g., France), we would need to filter the cube cells using the *IRI* representing that country, rather than the country's name. This would be not only unnatural to use, but also highly inefficient. Further, it would not be possible to define conditions that allow to keep cells that refer to countries with a population higher than a certain value.

3.4 SUMMARY

In this chapter we have presented the general framework of our work, positioning it in the field of self-service BI, aimed at incorporating situational data into the decision process, with the least possible in-

5 ttp://www.w3.org/TR/2009/NOTE-skos-primer-20090818/#sechierarchy

tervention of programmers or designers. We then discussed QB, the current W3C standard for publication of statistical data on the web, and addressed its limitations when it comes to representing MD data for OLAP analysis.

Part I

MODELING OLAP DATA CUBES ON THE SEMANTIC WEB

4

QB4OLAP

"To present a whole world that doesn't exist
and make it seem real,
we have to more or less pretend we're polymaths.
That's just the act of all good writing".

William Gibson

In Section 3.3 we have discussed the limitations of the QB vocabulary for representing MD data for OLAP. Despite this, QB remains, of course, suitable to represent statistical data. In addition, there is already a considerable amount of data published using it. This motivates us to extend QB in order to support the missing concepts defined in classic MD models for OLAP, rather than designing a new vocabulary from scratch. We denoted this extension as QB4OLAP.[1] In this chapter we introduce this new vocabulary.

To design the QB4OLAP vocabulary, the following requirements where considered.

- **Expressiveness:** QB4OLAP must be able to represent the most common features of the MD model. The features considered are based on the MultiDim model [42].
- **Interoperability:** QB4OLAP must allow to operate over already published observations, which conform to DSDs defined in QB, without the need of rewriting the existing observations. Note that in a typical MD model, observations are the largest part of the data while dimensions are usually orders of magnitude smaller. In Section 5.3 we will show how data cubes in QB can be extended and analyzed using QB4OLAP.
- **OLAP automation:** QB4OLAP must include all the metadata needed to automatically generate SPARQL queries that implement OLAP operations. The idea is that OLAP users should not need to be proficient in SPARQL. Even wrappers for OLAP tools can be developed to query RDF data sets directly. We elaborate on this topic in Chapter 6.

In short, QB4OLAP adds the following concepts to the QB vocabulary:

- **Dimension structure:** The structure of a dimension is defined in terms of *levels*, which compose *hierarchies*. The structure of each level is defined in terms of a set of *level attributes*.

1 http://www.purl.org/qb4olap/cubes

- **Dimension instances:** Level instances are called *level members*, and there is a relation between level members in consecutive levels. In typical OLAP these are called *rollup* relations (or, most usually, functions). In QB4OLAP these relationships between level members (from most specific to more general concepts) are modeled using *custom rollup properties* (this is motivated by the fact that more than one relation may hold between two dimension levels). Also, level members are associated with values for each attribute.

- **Aggregate functions:** Aggregate functions are used to compute measure aggregate values when performing OLAP operations (e.g:Roll-Up). QB4OLAP allows us to represent aggregate functions, and to associate them with measures in the data cube schema.

Figure 11 depicts the QB4OLAP vocabulary. Original QB terms are prefixed with "`qb:`", while QB4OLAP terms are prefixed with "`qb4o:`" and displayed in gray background. Capitalized terms represent RDF classes, non-capitalized terms represent RDF properties, and terms within ovals represent class instances. An arrow from class A to class B, labeled `rel` means that `rel` is an RDF property with domain A and range B. White triangles represent sub-class or sub-property relationships. Black diamonds represent `rdf:type` relationships (instances). We now present the main features of QB4OLAP, using our asylum applications case study.

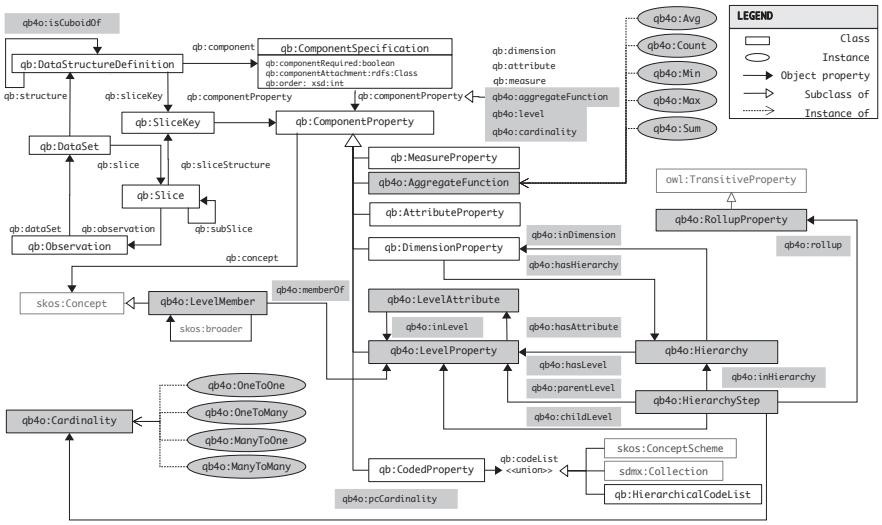


Figure 11: The QB4OLAP vocabulary

4.1 DATA CUBE STRUCTURE

Like in QB, in QB4OLAP, the data cube structure is an instance of the class `qb:DataStructureDefinition`. However, instead of using *dimensions* and measures to define a DSD, in QB4OLAP we use *dimension levels* and measures. This allows us to be specific about the granular-

ity level considered for each dimension. To represent *dimension levels*, in QB4OLAP we propose to use the same mechanism used in QB to represent dimensions: as classes of properties. For this, we introduced the class `qb4o:LevelProperty` (a sub-class of `qb:ComponentProperty`) for representing dimension levels. Instances of this class are used to specify the schema of the cube in terms of dimension levels, using `qb:DataStructureDefinition`.

In order to represent *aggregate functions*, we introduced the class `qb4o:AggregateFunction` and the `qb4o:aggregateFunction` property. The later associates measures with aggregate functions, and, together with the concept of component sets, allows a given measure to be associated with different aggregate functions in distinct cubes.

It is also worth noting that, in common MD data models, each fact is related with at most one level member for each level that participates in the fact. But sometimes this restriction has to be skipped, yielding so-called many-to-many dimensions [42]. To support these, we added the `qb4o:cardinality` property, which can be used to state the cardinality of the relationship between a fact and a level. Note that these constraints are purely declarative and not enforced by the model.

We now present the representation of the asylum applications data cube schema, using the QB4OLAP vocabulary. Example 4.1.1 shows the triples that define the DSD, while Figure 12 shows a graphical representation of these triples. For the sake of clarity, the definitions of the prefixes are omitted here. We refer to Appendix A to expand them. This case is derived from the existing QB specification of the cube, presented in Example 3.2.1. In Chapter 5 we will explain how we created this schema, that enables OLAP analysis of existing QB observations.

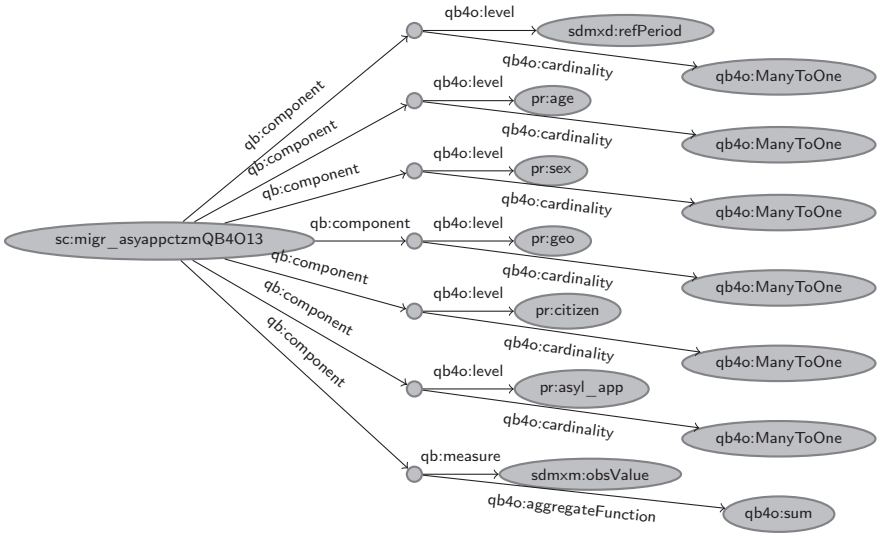


Figure 12: QB4OLAP representation of the asylum applications data cube schema.

Example 4.1.1. The triples below, define the new DSD for the asylum applications data cube defined using QB4OLAP.

```
sc:migr_asyappctzmQB4O13 a qb:DataStructureDefinition ;
 qb:component [ qb:measure sdmxm:obsValue ;
  qb4o:aggregateFunction qb4o:sum ] ;
 qb:component [ qb4o:level pr:age ;
  qb4o:cardinality qb4o:ManyToOne ] ;
 qb:component [ qb4o:level sdmxd:refPeriod ;
  qb4o:cardinality qb4o:ManyToOne ] ;
 qb:component [ qb4o:level pr:sex ;
  qb4o:cardinality qb4o:ManyToOne ] ;
 qb:component [ qb4o:level pr:geo ; qb4o:cardinality qb4o:ManyToOne ] ;
 qb:component [ qb4o:level pr:citizen ; qb4o:cardinality qb4o:ManyToOne ] ;
  qb:component [ qb4o:level pr:asyl_app ; qb4o:cardinality qb4o:ManyToOne] ;
 dc:conformsTo <http://purl.org/qb4olap/cubes> .

#states that eurostat instances are described by the schema defined in QB4OLAP
eurostatdt:migr_asyappctzm qb:structure sc:migr_asyappctzmQB4O13 ;
 dc:title "Asylum and first time asylum applicants to European countries by
 citizenship, age and sex Monthly data"@en ;
 dc:source <http://appsso.eurostat.ec.europa.eu/nui/show.do?
              dataset=migr_asyappctzm> .
```

Note that the dimension properties of the original QB cube are now used as *level properties*, and considered the lowest levels in each dimension hierarchy. This enables the analysis of existing observations without the need of rewriting them. We give further details on this mechanism in Chapter 5. □

4.2 REPRESENTATION OF DIMENSION HIERARCHIES

As already mentioned, dimension hierarchies and levels are first-class citizens in classic MD models for OLAP. Therefore, QB4OLAP focuses on their representation, and several classes and properties are introduced to this end. Before presenting them, we list a set of requirements that the QB4OLAP representation of hierarchies must meet, in order to model the different kinds of hierarchies presented in Section 2.1.

Requirement 1: Each dimension may be allowed to have more than one hierarchy;

Requirement 2: Each level may be allowed to belong to more than one hierarchy;

Requirement 3: In each hierarchy, a level may be allowed to have a different set of parent levels;

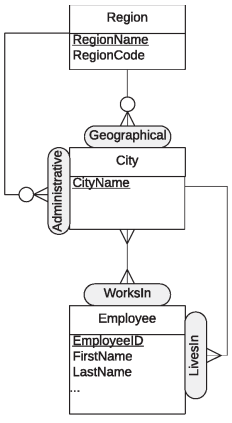
Requirement 4: Cardinality constraints must be associated to each parent-child relationship;

Requirement 5: For each parent-child relationship (at schema level), the property representing the rollup relationship between level members (at the instance level) must be defined.

To represent *dimension hierarchies*, the class `qb4o:Hierarchy` was introduced. The relationship between dimensions and hierarchies is represented via the property, `qb4o:hasHierarchy`. We can also use the 'inverse' property, `qb4o:inDimension`. A hierarchy is modeled as a *collection of pairs of levels*. Each pair is represented as an instance of

the `qb4o:HierarchyStep` class, which represents the reification of the parent-child relationship between two levels in a hierarchy. This reification allows linking each pair of levels with: (a) its two component levels, using properties `qb4o:childLevel` and `qb4o:parentLevel`, respectively; (b) the hierarchy it belongs to, using the `qb4o:inHierarchy` property; (c) the cardinality of the parent-child relationship, via property `qb4o:pcCardinality`, and instances of `qb4o:Cardinality` class; and (d) the property that implements the parent-child relationship at the instance level, using the `qb4o:rollup` property. As usual in RDF, blank nodes are used to implement such reification.

Our strategy for representing *rollup relationships* (in what follows, RUPs) between levels, deserves some comments. A näive approach would be to represent all the RUPs in a hierarchy with the same RDF property. Although this solution is enough for most kind of hierarchies presented in Section 2.1, it does not suffice to represent, at the instance level, dimensions with more than one RUP between the same pair of levels, usually denoted as *parallel dependent hierarchies* [42]. As an example of this situation, consider the dimension presented in Figure 13a. The Employee and City levels participate in two hierarchies: one that represents the city where the employee lives (LivesIn), and another that represents the city where the employee works (WorksIn). Also, the City and Region levels participate in two hierarchies: one that represents the administrative region that corresponds to a city (Administrative), and another that represents the geographical region (Geographical). Figure 13b presents, in tabular format, a possible set of instances for the RUPs in this schema.



Employee	LivesIn	WorksIn
employee1	city1	city2
employee2	city2	city1
employee3	city2	city2

City	Administrative	Geographical
city1	region1	region1
city2	region1	region2

(a) Conceptual model (b) Sample rollup relationship instances

Figure 13: Having multiple rollup relationships between levels.

In Example 4.2.1 we show the triples that represent these instances. We can see that, in order to distinguish the city where employee1 lives (city1), from the city where she works (city2), different properties are needed (X and Y in our example). Therefore, we propose to associate each hierarchy step, with the RDF property that represents the RUP relationship at the instance level. To model the fam-

ily of all rollup relationships between level members, we introduced the class `qb4o:RollupProperty`, and to emphasize that rollup relationships are transitive, this new class is declared as sub-class of `owl:TransitiveProperty`[2]

Example 4.2.1. Multiple rollup relationships between a pair of levels.

```
:employee a qb4o:LevelProperty .
:city a qb4o:LevelProperty .
:region a qb4o:LevelProperty .

:employee1 a qb4o:LevelMember; qb4o:memberOf :employee .
:employee2 a qb4o:LevelMember; qb4o:memberOf :employee .
:employee3 a qb4o:LevelMember; qb4o:memberOf :employee .
:city1 a qb4o:LevelMember; qb4o:memberOf :city .
:city2 a qb4o:LevelMember; qb4o:memberOf :city .
:region1 a qb4o:LevelMember; qb4o:memberOf :region .
:region2 a qb4o:LevelMember; qb4o:memberOf :region .
:employee1 X :city1; Y :city2 .
:employee2 X :city2; Y :city1 .
:employee3 X :city2; Y :city2 .
:city1 W :region1; Z :region1 .
:city2 W :region1; Z :region2 .
```

□

Finally, to represent *level attributes*, we introduced the class of properties `qb4o:LevelAttribute`. When defining the structure of a dimension, each level (an instance of `qb4o:LevelProperty`) is associated with a set of properties that are instances of `qb4o:LevelAttribute` via the `qb4o:hasAttribute` property. Later, these attribute properties will be used to link level members with the values of their attributes (see Example 4.3.1).

We conclude this section showing the representation of the Citizenship dimension schema (from Figure 5) using QB4OLAP. Figure 14 shows graphically a portion of the representation of this dimension, while Example 4.2.2 presents the triples that define this dimension.

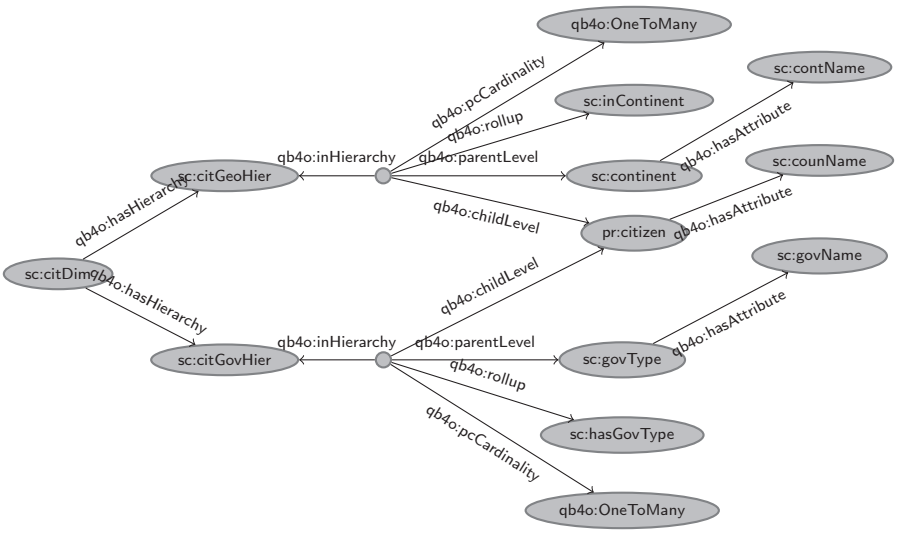


Figure 14: An excerpt of the Citizenship dimension schema

2 http://www.w3.org/TR/2004/REC-owl-ref-20040210/#TransitiveProperty-def

Example 4.2.2. To define the Citizenship dimension, we first declare a resource that represents the dimension, and relate it with its hierarchies. The conceptual model for this dimension is depicted in Figure 5.

```
sc:citDim a qb:DimensionProperty ;
  rdfs:label "Asylum geographical origin (citizenship) dimension"@en ;
  qb4o:hasHierarchy sc:citGeoHier, sc:citGovHier .
```

We now define each hierarchy, declare to which dimension it belongs, and which levels it traverses.

```
sc:citGeoHier a qb4o:Hierarchy ;
  rdfs:label "Asylum origin Geographical Hierarchy"@en ;
  qb4o:inDimension sc:citDim ;
  qb4o:hasLevel pr:citizen, sc:continent .

sc:citGovHier a qb4o:Hierarchy ;
  rdfs:label "Asylum origin Government Hierarchy"@en ;
  qb4o:inDimension sc:citDim ;
  qb4o:hasLevel pr:citizen, sc:govType .
```

Next, we define the base level in this dimension, that means, the one whose instances compose the observations (in other words, the finest granularity), and the upper levels in each hierarchy. In this case, since we are building a schema to analyze existent observations already published using QB, we reuse the properties that link observations with members (with type `qb:DimensionProperty`), and declare these properties as levels (with type `qb4o:LevelProperty`) By doing this, we avoid the cost of rewriting the observations.

```
# Base levels
pr:citizen a qb4o:LevelProperty ;
  rdfs:label "Country of origin when issuing an asylum application"@en ;
  qb4o:hasAttribute sc:counName .

#Upper hierarchy levels
sc:continent a qb4o:LevelProperty ;
  rdfs:label "Continent"@en ;
  qb4o:hasAttribute sc:contName .

sc:govType a qb4o:LevelProperty ;
  rdfs:label "Government Type"@en ;
  qb4o:hasAttribute sc:govName .

#Level attributes
sc:counName a qb4o:LevelAttribute ;
  rdfs:label "Country name"@en ;
  rdfs:range xsd:string .

sc:contName a qb4o:LevelAttribute ;
  rdfs:label "Continent name"@en ;
  rdfs:range xsd:string .

sc:govName a qb4o:LevelAttribute ;
  rdfs:label "Government type name"@en ;
  rdfs:range xsd:string .
```

Finally, the hierarchy steps (i.e., parent-child relationships) are defined. Each hierarchy step is associated with the property that will be used to implement the rollup relationship between level members at the instance level, which are also defined here.

```
#hierarchy steps
_:ih1 a qb4o:HierarchyStep ;
  qb4o:inHierarchy schema:citGeoHier ;
  qb4o:childLevel pr:citizen ;
```

```
  qb4o:parentLevel sc:continent ;
  qb4o:pcCardinality qb4o:OneToMany ;
  qb4o:rollup sc:inContinent .

_:ih2 a qb4o:HierarchyStep ;
  qb4o:inHierarchy sc:citGovHier ;
  qb4o:childLevel pr:citizen ;
  qb4o:parentLevel sc:govType ;
  qb4o:pcCardinality qb4o:OneToMany ;
  qb4o:rollup sc:hasGovType .

#rollup relationships
sc:inContinent a qb4o:RollupProperty .
sc:hasGovType a qb4o:RollupProperty .
```

□

4.3 DIMENSION MEMBERS REPRESENTATION

To represent *dimension level members* at the instance level, we introduced the class `qb4o:LevelMember`. Members can be attached to the levels they belong to, by using the property `qb4o:memberOf`, which resembles the semantics of `skos:member`. As already discussed, rollup relationships between members are expressed using custom properties defined in the dimension structure, conveying the idea that hierarchies of level members should be navigated from finer granularity concepts up to coarser granularity concepts. Figure 15 shows an excerpt of the representation of the Citizenship dimension schema and a sample dimension instance (on the right hand side of the figure).

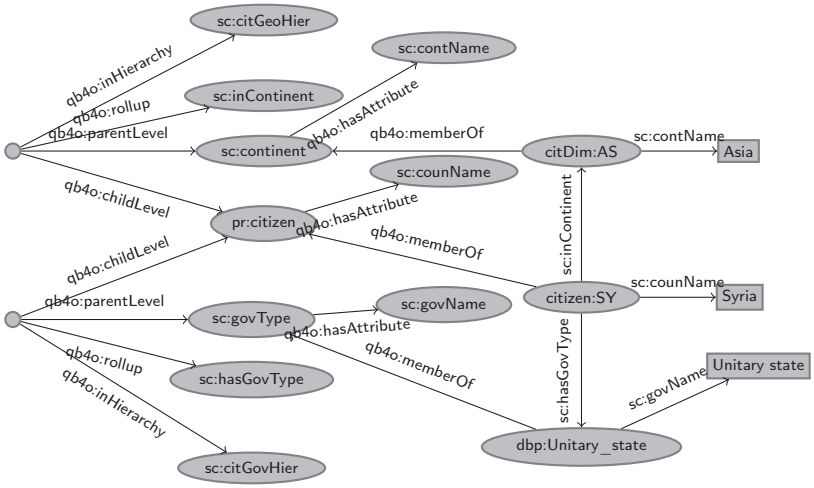


Figure 15: Citizenship dimension: schema and sample instance.

Note the relationship between schema and instance. For example, the property `sc:contName` is declared to be an attribute of the Continent level (`sc:continent`), and it is used to link a member of this level (Asia represented by the node `citDim:AS`), with the literal that represents its name. We can also see that the property `qb4o:memberOf` is used to tell that Asia (`citDim:AS`) is a member of the dimension level Continent. The triples represented in Figure 15 are detailed in Example 4.3.1.

Example 4.3.1. Definition of the dimension members corresponding to Syria.

```
1  citizen:SY qb4o:memberOf pr:citizen ;
2  sc:counName "Syria"@en ;
3  sc:inContinent citDim:AS ;
4  sc:hasGovType dbpedia:Unitary_state .
5
6  citDim:AS
7  qb4o:memberOf sc:continent ;
8  sc:contName "Asia" ;
9  skos:notation "AS" ;
10 skos:prefLabel "Asia"@en .
11
12 dbpedia:Unitary_state
13 qb4o:memberOf sc:governmentType ;
14 sc:govName "Unitary state"@en .
```

Line 1 declares that the resource that represents Syria is a member of the Country level (represented by `pr:citizen`). Lines 3 and 4 state that Syria rolls up to Asia (represented by `citDim:AS`) and to Unitary state (represented by `dbpedia:Unitary_state`), which are defined as members of the levels Continent and Government Type in lines 7 and 13 respectively. The properties that represent rollup relationships were previously defined in the dimension structure (see Example 4.2.2). Finally, the Country level has only one attribute: the Country Name (represented by `schema:countryName`). Line 2 tells that the value of this attribute, for the resource that represents Syria, is the string "Syria"@en. □

4.4 SUMMARY

In this chapter we have introduced the QB4OLAP vocabulary, with focus on the rationale of its design decisions. We also provided simple examples, showing how the main components of the MD model are represented, at the schema and instance levels. In the next chapter we will show how this vocabulary can be used to represent the most common features of the model, as well as complex dimension hierarchies.

5

QB4OLAP IN USE

"The street finds its own uses for things."

William Gibson, Burning Chrome

In the previous chapter we have introduced the QB4OLAP vocabulary through simple examples. In this chapter we address the problem of creating QB4OLAP data cubes. The idea is to show the three typical ways in which QB4OLAP can be used: (1) To create data cubes from scratch; (2) To export a relational DW into RDF format; (3) To enrich with structural metadata, existing cubes published in QB; In all the cases, we need a conceptual MD model to guide the design of the cube. In the second case, we also need to be able to export the relational DW *instance* into RDF, to populate the cube. In this chapter, we first show the expressiveness of QB4OLAP, by providing guidelines and examples on how to represent advanced concepts in conceptual MD modeling, and an algorithm to translate a conceptual model into RDF. We then discuss how to obtain QB4OLAP data cubes from a relational representation (ROLAP) of a DW. Finally, we elaborate on how to create a QB4OLAP schema to exploit a data set published using the QB vocabulary, addressing the third use case.

5.1 FROM CONCEPTUAL MULTIDIMENSIONAL MODELS TO QB4OLAP DATA CUBES

As discussed in Section 2.1, conceptual MD models, and dimension hierarchies in particular, can be quite complex in real-world situations. In the previous chapter we explain how the main components of the MD model are represented in QB4OLAP. In this section we provide guidelines and examples on how to represent more advanced MD design features. For each case, we show how to represent the schema and the instances. All the examples provided are taken from the Northwind case study (Figure 4). We conclude with an algorithm that translates a MultiDim conceptual model into a QB4OLAP schema. We start with flat dimensions (i.e., dimensions with just one level), and then increase the degree of difficulty.

5.1.1 *Flat Dimensions*

Flat dimensions are dimensions with only one level. As an example, consider the Product dimension in the Northwind case study. This kind of dimension can be represented in QB4OLAP via a hierarchy with no steps, i.e., with just a single level. Example 5.1.1 shows how to declare the schema of flat dimensions, while instances are defined as usual.

Example 5.1.1. To represent flat dimensions, we define hierarchies without steps.

```
nw:productDim a qb:DimensionProperty ;
  rdfs:label "Product"@en ;
  qb4o:hasHierarchy nw:productHier .

nw:productHier a qb4o:Hierarchy ;
  rdfs:label "Product Hierarchy"@en ;
  qb4o:inDimension nw:productDim ;
  qb4o:hasLevel nw:product .

nw:product a qb4o:LevelProperty ;
  rdfs:label "Product"@en ;
  qb4o:hasAttribute nw:productName .

nw:productName a qb4o:LevelAttribute ;
  rdfs:label "Product name"@en ;
  rdfs:range xsd:string .
```

□

5.1.2 *Shared Levels*

Sometimes, levels are shared between two or more dimensions. As an example, consider the levels City, State, Region, Country, and Continent in the Northwind case study (Figure 4), that participate in the dimensions Supplier and Customer. Note that, when levels are shared, rollup relationships between level members are also shared, and have the same semantics. As a consequence, in the QB4OLAP representation, hierarchy steps can be shared between different hierarchies. At the schema level, we propose to define each shared level only once, and use these levels in the definition of the hierarchy steps. The properties that will represent the rollup relationships at the instance level, are also shared by the different hierarchies. Example 5.1.2 shows how to declare the schema of two dimensions that share levels, while Example 5.1.3 shows how to declare their instances.

Example 5.1.2. To represent the dimensions Supplier and Customer, we first define each dimension and its corresponding hierarchy. Observe that the levels in each hierarchy are all shared, with the exception of the bottom level in each hierarchy.

```
# -- Customer dimension definition
nw:customerDim a rdf:Property , qb:DimensionProperty ;
  rdfs:label "Customer Dimension"@en ;
  qb4o:hasHierarchy nw:customerGeo .

# -- Customer Geography hierarchy
nw:customerGeo a qb4o:Hierarchy ;
  rdfs:label "Customer Geography Hierarchy"@en ;
  qb4o:inDimension nw:customerDim ;
  qb4o:hasLevel nw:customer, nw:city, nw:state ;
  qb4o:hasLevel nw:region, nw:country, nw:continent .

# -- Supplier dimension definition
nw:supplierDim a rdf:Property , qb:DimensionProperty ;
  rdfs:label "Supplier Dimension"@en ;
  qb4o:hasHierarchy nw:supplierGeo .

# -- Supplier Geography hierarchy
nw:supplierGeo a qb4o:Hierarchy ;
```

```
 rdfs:label "Supplier Geography Hierarchy"@en ;
 qb4o:inDimension nw:supplierDim ;
 qb4o:hasLevel nw:supplier, nw:city, nw:state ;
 qb4o:hasLevel nw:region, nw:country, nw:continent .
```

Then we define each level, the rollup relationships, and finally the structure of each hierarchy in terms of hierarchy steps. We only include the definition of the levels Customer, Supplier ,City, and State.

```
# -- Levels
nw:supplier a qb4o:LevelProperty .
nw:customer a qb4o:LevelProperty .
nw:city a qb4o:LevelProperty .
nw:state a qb4o:LevelProperty .

# -- Rollup relationships (shared in all hierarchies)
nw:inCity a qb4o:RollupProperty .
nw:inState a qb4o:RollupProperty .

_:hs11 a a qb4o:HierarchyStep ;
 qb4o:inHierarchy nw:customerGeo ;
 qb4o:childLevel nw:customer ; qb4o:parentLevel nw:city ;
 qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:inCity .

_:hs21 a a qb4o:HierarchyStep ;
 qb4o:inHierarchy nw:supplierGeo;
 qb4o:childLevel nw:supplier ; qb4o:parentLevel nw:city ;
 qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:inCity .

_:hs3 a a qb4o:HierarchyStep ;
 qb4o:inHierarchy nw:customerGeo, nw:supplierGeo;
 qb4o:childLevel nw:city; qb4o:parentLevel nw:state ;
 qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:inState .
```

Finally, the bottom levels of these dimensions are used in the definition of a data cube, as follows.

```
nw:Northwind a qb:DataStructureDefinition ;
# -- Lowest level for each dimension in the cube
 qb:component [qb4o:level nw:customer ; qb4o:cardinality qb4o:ManyToOne ] ;
 qb:component [qb4o:level nw:supplier ; qb4o:cardinality qb4o:ManyToOne ] ;
 [...]
```

□

Example 5.1.3. Instances of shared levels are defined as usual. The triples bellow represent a customer and a supplier who live in the same city.

```
nwc:343 qb4o:memberOf nw:customer
  nw:companyName "Paris spécialités" ;
  nw:inCity nwci:35 ;
  [...]

nws:18 qb4o:memberOf nw:supplier ;
 nw:companyName "Aux joyeux ecclésiastiques" ;
 nw:inCity nwci:35 ;
 [...]

nwci:35 qb4o:memberOf nw:city ;
 nw:cityName "Paris" ;
 nw:inState nwst:224 ;
 [...]

nwst:224 qb4o:memberOf nw:state ;
 nw:stateName "Paris" ;
 [...]
```

□

5.1.3 *Role-playing Dimensions*

In some cases, the same dimension is used more than once in a data cube, and therefore a mechanism is needed to distinguish the role that each dimension plays in it. These are called *role-playing dimensions* (introduced in Section 1.1). As an example, consider the Time dimension, that participates in the Northwind case study (Figure 4) with two different roles that correspond to the OrderDate and the DueDate of the sale.

To represent role-playing dimensions in QB4OLAP, we propose to define, at the schema level, one dimension per role. These dimensions share all the levels, with the exception of the bottom one, which needs to be different for each dimension in order to distinguish them in each observation (recall that each observation is linked to the level members that participate in it using the properties defined as levels). At the instance level, members and rollup relationships are declared only once, and each level member is linked to all the corresponding levels. The mechanism is similar to the one described for shared levels. Example 5.1.4 shows how to declare the schema of role-playing dimensions, while Example 5.1.5 shows how to declare their instances.

Example 5.1.4. To represent the role-playing dimensions OrderDate and DueDate, we define one dimension for each role, and one hierarchy for each dimension; hierarchy steps are shared between hierarchies. We include a portion of the definition of each level.

```
# -- OrderDate dimension definition
nw:orderDateDim a rdf:Property , qb:DimensionProperty ;
  rdfs:label "Order Date Dimension"@en ;
  rdfs:subPropertyOf sdmx-dimension:refPeriod ;
  qb4o:hasHierarchy nw:calendarOrderDate .

# -- DueDate dimension definition
nw:dueDateDim a rdf:Property , qb:DimensionProperty ;
  rdfs:label "Due Date Dimension"@en ;
  rdfs:subPropertyOf sdmx-dimension:refPeriod ;
  qb4o:hasHierarchy nw:calendarDueDate .

# -- Rollup relationships (shared in all hierarchies)
nw:inMonth a qb4o:RollupProperty .
nw:inYear a qb4o:RollupProperty .

# -- Order Date Calendar hierarchy
nw:calendarOrderDate a qb4o:Hierarchy ;
  rdfs:label "Calendar Hierarchy"@en ;
  qb4o:inDimension nw:orderDateDim ;
  qb4o:hasLevel nw:orderDate, nw:month , nw:year .

# -- DueDate Calendar hierarchy
nw:calendarDueDate a qb4o:Hierarchy ;
  rdfs:label "Calendar Hierarchy"@en ;
  qb4o:inDimension nw:dueDateDim ;
  qb4o:hasLevel nw:dueDate, nw:month, nw:year .

_:hs11 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:calendarOrderDate ;
  qb4o:childLevel nw:orderDate ; qb4o:parentLevel nw:month ;
  qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:inMonth .

_:hs21 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:calendarDueDate ;
  qb4o:childLevel nw:dueDate ; qb4o:parentLevel nw:month ;
```

```
  qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:inMonth .

_:hs2 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:calendarOrderDate, nw:calendarDueDate ;
  qb4o:childLevel nw:month ; qb4o:parentLevel nw:year ;
  qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:inYear .

# -- Levels definition (bottom levels have the same attributes)
nw:orderDate a qb4o:LevelProperty ;
  rdfs:label "Time Level"@en ;
  qb4o:hasAttribute nw:date ;
  qb4o:hasAttribute nw:dayNoWeek ;
  qb4o:hasAttribute nw:dayNameWeek ;
  qb4o:hasAttribute nw:dayNoMonth ;
  qb4o:hasAttribute nw:dayNoYear ;
  qb4o:hasAttribute nw:weekNoYear .

nw:dueDate a qb4o:LevelProperty ;
  rdfs:label "Time Level"@en ;
  qb4o:hasAttribute nw:date ;
  qb4o:hasAttribute nw:dayNoWeek ;
  qb4o:hasAttribute nw:dayNameWeek ;
  qb4o:hasAttribute nw:dayNoMonth ;
  qb4o:hasAttribute nw:dayNoYear ;
  qb4o:hasAttribute nw:weekNoYear .
```

Finally, we show how the bottom levels of these dimensions are used in the definition of a data cube.

```
nw:Northwind a qb:DataStructureDefinition ;
# -- Lowest level for each dimension in the cube
  qb:component [qb4o:level nw:orderDate ; qb4o:cardinality qb4o:ManyToOne ] ;
  qb:component [qb4o:level nw:dueDate ; qb4o:cardinality qb4o:ManyToOne ] ;
  [...]
```

□

Example 5.1.5. To represent instances of role-playing dimensions, we define instances of the bottom level only once, and link it to the bottom level of each role-playing dimension. This example represents the date "1998-06-06".

```
nwt:19980606
  nw:dayNameWeek "Sunday" ;
  nw:dayNoMonth 6 ;
  nw:date "1998-06-06" ;
  nw:dayNoWeek 7 ;
  nw:weekNoWeek 23 ;
  nw:dayNoYear 157 ;
  nw:inMonth nwm:June1998 ;
  qb4o:memberOf nw:orderDate,nw:dueDate .

nwm:June1998
  nw:monthName "June" ;
  nw:monthNo 6 ;
  nw:inYear nwy:1998 ;
  qb4o:memberOf nw:month .

nwy:1998
  nw:yearNo 1998 ;
  qb4o:memberOf nw:year .
```

□

5.1.4 *Recursive Hierarchies*

Recursive hierarchies (introduced in Section 2.1) are a special case of unbalanced hierarchies, where the same level plays the role of parent and child. As an example, consider the Supervision hierarchy in the Employee dimension of the Northwind case study (Figure 4), where subordinate employees are related to their supervisors, which are also members of the Employee level. QB4OLAP supports this kind of hierarchy, since it just suffices to use the same level as child and parent in a hierarchy step. Example 5.1.6 shows how to declare the schema of a recursive hierarchy, while instances of recursive hierarchies are represented as usual.

Example 5.1.6. To represent recursive hierarchies, we define a step that declares the same level as parent and child.

```
nw:employeeDim a rdf:Property , qb:DimensionProperty ;
  rdfs:label "Employee Dimension"@en ;
  qb4o:hasHierarchy nw:supervision .

# -- Supervision hierarchy
nw:supervision a qb4o:Hierarchy ; rdfs:label "Supervision Hierarchy"@en ;
  qb4o:inDimension nw:employeeDim ;
  qb4o:hasLevel nw:employee .

# -- Roll up relationship
nw:supervisor a qb4o:RollupProperty .

_:hs1 a qb4o:HierarchyStep ; qb4o:inHierarchy nw:supervision ;
  qb4o:childLevel nw:employee ; qb4o:parentLevel nw:employee ;
  qb4o:pcCardinality qb4o:ManyToOne ; qb4o:rollup nw:supervisor .
```

□

5.1.5 *Parallel Dependent Hierarchies*

This case occurs when a dimension has more than one hierarchy, and these hierarchies may share levels. As an example of this, consider the dimension presented in Figure 13a, discussed in Section 4.2. The Employee and City levels participate in two hierarchies: one that represents the city where the employee lives (LivesIn), and another that represents the city where the employee works (WorksIn). Also, the City and Region levels participate in two hierarchies: one that represents the administrative region that corresponds to a city (Administrative), and another that represents the geographical region (Geographical). We propose to define one hierarchy for each possible path from the bottom level up to the top, and one rollup property per parent-child relationship. This allows us to specify, via the rollup property, which path should be used to aggregate data. In this case, hierarchy steps can also be shared between hierarchies. Example 5.1.7 shows how to declare the schema of parallel dependent hierarchies, while Example 5.1.8 shows how to declare their instances.

Example 5.1.7. To represent parallel dependent hierarchies LivesIn, WorksIn, Geographical, and Administrative we first define four hierarchies, and four rollup properties (one per parent-child relationship).

```
:employeeDim a rdf:Property , qb:DimensionProperty ;
qb4o:hasHierarchy :empLivesAdmin , :empLivesGeo , :empWorksAdmin , :empWorksGeo .

# -- Dimension levels
:employee a qb4o:LevelProperty .
:city a qb4o:LevelProperty .
:region a qb4o:LevelProperty .

# -- Rollup relationships
:worksin a qb4o:RollupProperty .
:livesin a qb4o:RollupProperty .
:administrative a qb4o:RollupProperty .
:geographical a qb4o:RollupProperty .

# -- Hierarchies
# -- One hierarchy for each path from bottom to top
:empLivesGeo a qb4o:Hierarchy ;
  qb4o:inDimension :employeeDim ;
  qb4o:hasLevel :employee , :city , :region .

:empLivesAdmin a qb4o:Hierarchy ;
  qb4o:inDimension :employeeDim ;
  qb4o:hasLevel :employee , :city , :region .

:empWorksGeo a qb4o:Hierarchy ;
  qb4o:inDimension :employeeDim ;
  qb4o:hasLevel :employee , :city , :region .

:empWorksAdmin a qb4o:Hierarchy ;
  qb4o:inDimension :employeeDim ;
  qb4o:hasLevel :employee , :city , :region .
```

We then define each step, and link it with the dimensions it belongs to.

```
# -- Hierarchy step from employee to city via livesin
_:hs1 a qb4o:HierarchyStep;
  qb4o:childLevel :employee ; qb4o:parentLevel :city ;
  qb4o:rollup :livesin ;
  qb4o:inHierarchy :empLivesGeo , :empLivesAdmin ;
  qb4o:pcCardinality qb4o:ManyToOne .

# -- Hierarchy step from employee to city via worksin
_:hs2 a qb4o:HierarchyStep ;
  qb4o:childLevel :employee ; qb4o:parentLevel :city ;
  qb4o:rollup :worksin ;
  qb4o:inHierarchy :empWorksGeo , :empWorksAdmin ;
  qb4o:pcCardinality qb4o:ManyToOne .

# -- Hierarchy step from city to region via administrative
_:hs3 a qb4o:HierarchyStep ;
  qb4o:childLevel :city ; qb4o:parentLevel :region ;
  qb4o:rollup :administrative ;
  qb4o:inHierarchy :empLivesAdmin , :empWorksAdmin ;
  qb4o:pcCardinality qb4o:ManyToOne .

# -- Hierarchy step from city to region via geographical
_:hs4 a qb4o:HierarchyStep ;
  qb4o:childLevel :city ; qb4o:parentLevel :region ;
  qb4o:rollup :geographical ;
  qb4o:inHierarchy :empLivesGeo , :empWorksGeo ;
  qb4o:pcCardinality qb4o:ManyToOne .
```

□

Example 5.1.8. This example shows the triples that represent the instances in Figure 13b.

```
:employee1 a qb4o:LevelMember ; qb4o:memberOf :employee .
:employee2 a qb4o:LevelMember ; qb4o:memberOf :employee .
:employee3 a qb4o:LevelMember ; qb4o:memberOf :employee .

:city1 a qb4o:LevelMember ; qb4o:memberOf :city.
:city2 a qb4o:LevelMember ; qb4o:memberOf :city.

:region1 a qb4o:LevelMember ; qb4o:memberOf :region.
:region2 a qb4o:LevelMember ; qb4o:memberOf :region.

:employee1 :livesin :city1, :worksin :city2 .
:employee2 :livesin :city2, :worksin :city1 .
:employee3 :livesin :city2, :worksin :city2 .
:city1 :administrative :region1 ; :geographical :region1 .
:city2 :administrative :region1 ; :geographical :region2 .
```

□

5.1.6 *Translating a MultiDim Model to QB4OLAP*

We wrap-up the above, presenting an algorithm that, starting from a conceptual (MultiDim) schema, produces a QB4OLAP representation of the cube schema. We assume that there is a conceptual schema that represents a cube C, with a fact F composed of a set M of measures, a set D of dimensions, and a set RP of role-playing dimensions. Each role-playing dimension $rp \in RP$ is a pair $(d, role)$, where $d \in D$. Each dimension $d \in D$ is composed of a set L of levels, organized in hierarchies $h \in H$. Each level $l \in L$ is described by a set of attributes A. The algorithm comprises seven steps described next. We call CS_{RDF} the RDF graph that represents the cube schema, which is built incrementally. We will illustrate the steps in this algorithm, with the Northwind DW case study introduced in Section 1.1.1, and depicted in Figure 4.

Step 1 (Dimensions) For each dimension $d \in D$, if d is a role-playing dimension, for each role $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple stating that dr_{RDF} is a resource of type `qb:DimensionProperty`. Else, $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple stating that d_{RDF} is a resource of type `qb:DimensionProperty`. Triples indicating the name of each dimension can be added using property `rdfs:label`.

The triples below show how some dimensions in Figure 4 are represented (`@en` indicates that the names are in English, and `nw:` is a prefix for the cube schema graph). Triples in lines 2 and 3 correspond to the Time dimension participating with the roles OrderDate and DueDate.

```
nw:employeeDim a qb:DimensionProperty ; rdfs:label "Employee Dimension"@en .
nw:orderDateDim a qb:DimensionProperty ; rdfs:label "OrderDate Dimension"@en .
nw:dueDateDim a qb:DimensionProperty ; rdfs:label "DueDate Dimension"@en .
nw:productDim a qb:DimensionProperty ; rdfs:label "Product Dimension"@en .
nw:orderDim a qb:DimensionProperty ; rdfs:label "Order Dimension"@en .
nw:shipperDim a qb:DimensionProperty ; rdfs:label "Shipper Dimension"@en .
nw:customerDim a qb:DimensionProperty ; rdfs:label "Customer Dimension"@en .
nw:supplierDim a qb:DimensionProperty ; rdfs:label "Supplier Dimension"@en.
```

Step 2 (Hierarchies) For each hierarchy $h \in H$, $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple stating that h_{RDF} has type `qb4o:Hierarchy`. Triples indicating the name of each hierarchy can be added using property `rdfs:label`.

Applying Step 2 to the hierarchies in the Employee dimension we obtain:

```
nw:supervision a qb4o:Hierarchy ; rdfs:label "Employee Supervision Hierarchy"@en .
nw:territories a qb4o:Hierarchy ; rdfs:label "Employee Territories Hierarchy"@en .
```

Step 3 (Levels and Attributes) For each level $l \in L$, $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple telling that l_{RDF} is a resource of type `qb4o:LevelProperty`. For each attribute $a \in A$, add to CS_{RDF} a triple stating that a_{RDF} has type `qb4o:AttributeProperty`. Finally, add triples relating a level l_{RDF} with its corresponding attribute a_{RDF}, using the property `qb4o:hasAttribute`. Triples indicating the names of levels and attributes can be added using property `rdfs:label`.

Applying Step 3 to level Employee in the Employee dimension we obtain:

```
nw:employee a qb4o:LevelProperty ; rdfs:label "Employee Level"@en ;
  qb4o:hasAttribute nw:firstName ; qb4o:hasAttribute nw:lastName .
nw:firstName a qb:AttributeProperty ; rdfs:label "First Name"@en .
nw:lastName a qb:AttributeProperty ; rdfs:label "Last Name"@en .

nw:city a qb4o:LevelProperty ;
  rdfs:label "City Level"@en ;
  qb4o:hasAttribute nw:cityName .
nw:cityName a qb:AttributeProperty ; rdfs:label "City Name"@en .
```

Step 4 (Dimension-Hierarchy Relationships) For each $h \in H$ in $d \in D$, relate d_{RDF} in CS_{RDF} to h_{RDF}, and h_{RDF} to d_{RDF}. Then, $CS_{RDF} = CS_{RDF} \cup \{d_{RDF}\}$ `qb4o:hasHierarchy` h_{RDF} and $CS_{RDF} = CS_{RDF} \cup \{h_{RDF}\}$ `qb4o:inDimension` d_{RDF}.

Applying Step 4 to the Employee dimension and its hierarchies we obtain:

```
nw:employeeDim qb4o:hasHierarchy nw:Supervision ;
  qb4o:hasHierarchy nw:territories .
nw:supervision qb4o:inDimension nw:employeeDim .
nw:territories qb4o:inDimension nw:employeeDim .
```

Step 5 (Hierarchy Structure) For each hierarchy $h \in H$ composed of a level $l \in L$, relate h_{RDF} in CS_{RDF} to l_{RDF}. Then $CS_{RDF} = CS_{RDF} \cup \{h_{RDF}\}$ `qb4o:hasLevel` l_{RDF}. Let (l, l') be a pair of levels in the C, such that $l, l' \in h$, and $parentLevel(l, h) = l'$ with cardinality car. Also, let l_{RDF}, l'_{RDF}, and h_{RDF} be the representations of l, l' and h in CS_{RDF}. Then add to CS_{RDF} a blank node hs_{RDF} of type `qb4o:HierarchyStep`, and a triple to connect the hierarchy step with the hierarchies it belongs to (hs_{RDF} `qb4o:inHierarchy` h_{RDF}). Add triples hs_{RDF} `qb4o:childLevel` lh_{RDF}, hs_{RDF} `qb4o:parentLevel` lh'_{RDF}. Also add to CS_{RDF} a node rup_{RDF} of type `qb4o:RollupProperty` that represents the rollup relationship for that step, and then a triple hs_{RDF} `qb4o:rollup` rup_{RDF}. Finally, add the triple hs_{RDF} `qb4o:pcCardinality` car_{RDF}, to represent the cardinality of the relationship.

A part of the Employee and Supplier dimension structure obtained is shown below. Notice that these dimensions have hierarchies that share levels, and this is reflected by sharing hierarchy steps. Lines 28 to 33 and 34 to 39 show the support of a ragged hierarchy.

```
1 nw:supervision qb4o:hasLevel nw:employee .
2 nw:territories qb4o:hasLevel nw:employee , nw:city , nw:state .
```

```
nw:territories qb4o:hasLevel nw:country , nw:continent .

nw:supplierGeo qb4o:hasLevel nw:supplier , nw:city , nw:state .
nw:supplierGeo qb4o:hasLevel nw:region , nw:country , nw:continent .

nw:supervises a qb4o:RollupProperty .
nw:worksIn a qb4o:RollupProperty .
nw:inState a qb4o:RollupProperty .
nw:inCountry a qb4o:RollupProperty .

# -- nw:supervision hierarchy structure
_:pl1 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:supervision ;
  qb4o:childLevel nw:employee ;
  qb4o:parentLevel nw:employee ;
  qb4o:rollup nw:supervises ;
  qb4o:cardinality qb4o:OneToMany .

# -- nw:territories hierarchy structure
_:pl2 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:territories ;
  qb4o:childLevel nw_employee ;
  qb4o:parentLevel nw:city ;
  qb4o:rollup nw:worksIn ;
  qb4o:cardinality qb4o:ManyToMany .
_:pl3 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:territories,nw:supplierGeo;
  qb4o:childLevel nw:city ;
  qb4o:parentLevel nw:state ;
  qb4o:rollup nw:inState ;
  qb4o:cardinality qb4o:OneToMany .
_:pl4 a qb4o:HierarchyStep ;
  qb4o:inHierarchy nw:territories,nw:supplierGeo ;
  qb4o:childLevel nw:city ;
  qb4o:parentLevel nw:country ;
  qb4o:rollup nw:inCountry ;
  qb4o:cardinality qb4o:OneToMany .
```

Step 6 (Measures) For each measure $m \in M$, $CS_{RDF} = CS_{RDF} \cup \{t\}$, such that t is a triple that states that m_{RDF} is a resource with type `qb4o:MeasureProperty`. The range of each m_{RDF} can be defined using the rdfs:range predicate.

The following triples are the result of the application of Step 6 to our example.

```
nw:quantity a qb:MeasureProperty ;
  rdfs:label "Quantity"@en ;
  rdfs:range xsd:integer .
nw:unitPrice a qb:MeasureProperty ;
  rdfs:label "UnitPrice"@en ;
  rdfs:range xsd:decimal .
nw:salesAmount a qb:MeasureProperty ;
  rdfs:label "SalesAmount"@en ;
  rdfs:range xsd:decimal .
```

Step 7 (Cube) For each fact F, $CS_{RDF} = CS_{RDF} \cup \{t\}$, such that t is a triple stating that c_{RDF} has type `qb:DataStructureDefinition`. For each measure $m \in M$, add to CS_{RDF} the triples c_{RDF} qb:component [qb:measure m_{RDF}; qb4o:aggregateFunction f_{RDF}], where f_{RDF} is an aggregation function defined in QB4OLAP. Also, for each of the levels $l \in L$ related to a fact F in the schema, $CS_{RDF} = CS_{RDF} \cup \{c_{RDF}$ `qb:component` [`qb:level` l_{RDF}; `qb4o:cardinality` car_{RDF}]}, where car_{RDF} represents the cardinality of the relationship between facts and level members, and is one of cardinality restrictions defined in

QB4OLAP (`qb4o:OneToOne`, `qb4o:OneToMany`, `qb4o:ManyToOne`, `qb4o:ManyToMany`).

The following triples are the result of the application of Step 7.

```
# -- Cube definition (Data structure)
nw:Northwind a qb:DataStructureDefinition ;
# Lowest level for each dimension in the cube
qb:component [qb4o:level nw:employee ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:orderDate ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:dueDate ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:supplier ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:customer ; qb4o:cardinality qb4o:ManyToOne] ;
# -- Measures in the cube
qb:component [qb:measure nw:quantity ; qb4o:aggregateFunction qb4o:sum] ;
qb:component [qb:measure nw:unitPrice ; qb4o:aggregateFunction qb4o:avg] ;
qb:component [qb:measure nw:salesAmount ; qb4o:aggregateFunction qb4o:sum] .
```

5.2 FROM RELATIONAL DWS TO QB4OLAP

Traditional DW systems usually store MD data in relational databases. These systems, also known as Relational OLAP or ROLAP, implement conceptual models at the logical level as a collection of tables organized in specialized structures known as *star* and *snowflake* schemas, which relate a fact table to several dimension tables through foreign keys. In a *star schema*, a fact table is linked through foreign keys to one or more *denormalized* dimension tables. In a *snowflake schema*, dimension tables are *normalized*, and a dimension is represented as a collection of tables linked to each other through foreign keys. A *starflake schema* is a mixed approach, which includes normalized and denormalized dimension tables in the same model. Since QB4OLAP is aimed at publishing new *and* existing MD data, we would like to be able to translate a relational DW (i.e., the actual instance of the DW) into QB4OLAP, as automatically as possible. We already showed how to create a cube schema in QB4OLAP, starting from a conceptual schema. In this section, we first present a procedure to translate an existing ROLAP DW, into its RDF representation, using QB4OLAP. In the second part of the section, we show an implementation of this procedure. Since there is no standard in ROLAP to represent the cube schema (i.e. dimensions, structure, hierarchies, levels, etc.), we assume that we have the conceptual model associated with the relational DW.

5.2.1 *ROLAP to QB4OLAP*

We now present a procedure to obtain a QB4OLAP implementation from a relational cube instance, starting from: (a) the cube conceptual schema C; (b) CS_{RDF}, the RDF representation of the schema of C; (c) the relational implementation of the cube C, which we denote CI_{ROLAP}. The procedure generates a mapping file CI_{RDF} that generates an RDF representation of the data stored in CI_{ROLAP}, using the schema CS_{RDF}. This mapping file is expressed using R2RML, the W3C standard for expressing mappings between relational databases and RDF data, which is based on templates (see Section 2.3.3). The

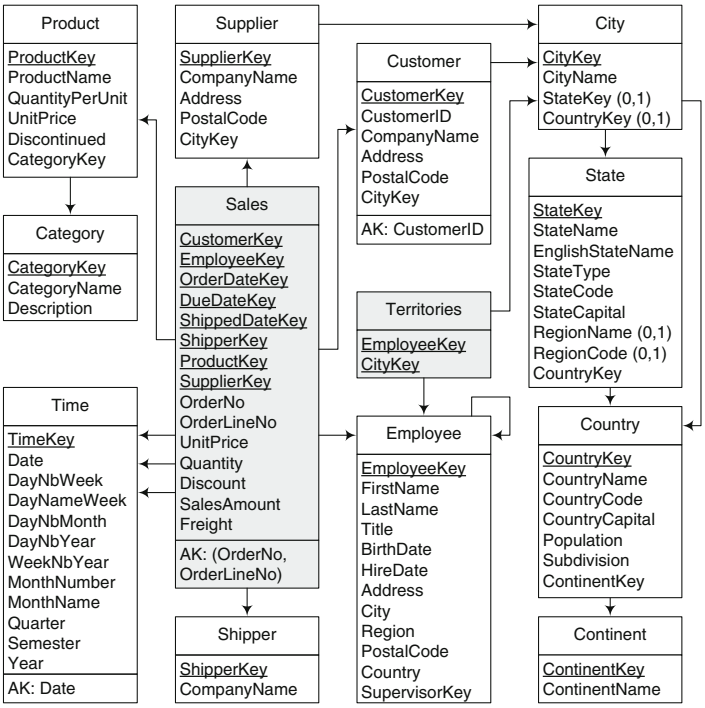


Figure 16: Relational representation of the Northwind DW (from [42])

obtained R2RML mappings can either be used to generate a static set of triples that represent the underlying relational data (data materialization), or to provide a non-materialized RDF view of the relational data (on-demand mapping). A relevant problem here is how to generate adequate IRIs. In this case, a collection of IRI-safe strings P are used to generate unique level members IRIs with R2RML rr:template, based on the identifiers found in the relational data, such that each level member has its corresponding $p_i \in P$. Analogously, f is used to generate unique IRIs for fact instances.

We organize the procedure in two parts: (1) Define mappings to generate level members; (2) Define mappings to generate facts (observations). Since the typical MD model constructs, and in particular dimension hierarchies, may have different relational representations, the logical design of the underlying relational database determines which mappings should be generated.

The procedure we present next, addresses the most common kinds of dimension hierarchies discussed, and their relational representation as presented in [42], showing, for each case, how mappings can be generated. The hierarchies addressed are: balanced, ragged, recursive, and nonstrict. This covers a wide range of problems. Other particular cases can be addressed based on these cases. Of course, we also address the generation of facts (observations).

Figure 16 shows the relational representation of the conceptual model presented in Figure 4. We use this representation to illustrate the concepts in this section. Underlined attributes represent primary keys, while arrows represent foreign key constraints. Optional attributes are denoted placing a cardinality constraint (0,1) next to them.

Step 1 (**Balanced hierarchies**) These hierarchies can be represented as *snowflake schemas* or as *star schemas*.

If $h \in H$ is a balanced hierarchy composed of a set of levels L, represented as a *snowflake schema*, there exists a set of tables $T_h \in CI_{ROLAP}$, where each table $t_i \in T_h$ represents a level in $l_i \in L$, and contains a key attribute pk_i and one attribute a_i for each level attribute $at_i \in l_i$. For each pair $l_i, l_{i+1} \in h$, represented as $t_i, t_{i+1} \in T_h$, such that $parentLevel(l_i, h) = l_{i+1}$, there exists a foreign key attribute $fk_i \in t_i$ referencing $pk_{i+1} \in t_{i+1}$.

To generate the instances of a *balanced hierarchy represented as a snowflake schema*, for each level $l_i \in h$, $CI_{RDF} = CI_{RDF} \cup \{t\}$, where t is an R2RML `rr:TripleMap` that generates the members of l_i. The components of t are: the `rr:logicalTable` t_i, a `rr:subjectMap` which is an IRI built using the rr:template $p_i\{pk_i\}$, and one or more predicate object maps (`rr:predicateObjectMap`)that express: (1) To which level $l_{i_{RDF}} \in CS_{RDF}$ the members generated by t belong; (2) The value of each attribute $a_{RDF} \in l_{i_{RDF}}$, which is obtained from the attributes in t_i, specified using `rr:column`; (3) The associated members in other levels l_j, using the predicate specified by `qb4o:rollup` for each step and the `rr:template` $p_j\{fk_j\}$.

As an example, the Product dimension, obtained from Figure 4, is a balanced hierarchy represented as a snowflake schema (Figure 16). We next show the R2RML mapping that generates the triples representing the members in level Product using data from the table Product. The mapping in lines 10 to 15 implements the RUP relationship between members of the Product and Category levels (represented in the relational data via the attribute CategoryKey in Product table, and a foreign key to the Category table). We only present the mappings corresponding to the attribute ProductName, the other ones are analogous.

```
1  <#TriplesMapProduct > a rr:TriplesMap ;
2  rr:logicalTable [ rr:tableName "Product" ] ;
3  rr:subjectMap [
4  rr:termType rr:IRI ;
5  rr:template
6  "http://www.fing.edu.uy/inco/cubes/instances/northwind/Product#{ProductKey}";];
7  rr:predicateObjectMap [ rr:predicate qb4o:memberOf ; rr:object nw:product ; ] ;
8  rr:predicateObjectMap [ rr:predicate nw:productName ;
9  rr:objectMap [ rr:column "ProductName" ];] ;
10 rr:predicateObjectMap [ rr:predicate nw:inCategory;
11 rr:objectMap [
12 rr:termType rr:IRI ;
13 rr:template
14 "http://www.fing.edu.uy/inco/cubes/instances/northwind/Category#{CategoryKey}"];].
15
```

The triples shown below, are generated by the mapping above. They represent the product "Ravioli Angelo" in the category "Grains/Cereals".

```
nwp:57 qb4o:memberOf nw:product;
  nw:productName "Ravioli Angelo";
  nw:inCategory nwca:5 .
```

If $h \in H$ is a balanced hierarchy composed of a set of levels L, represented as a *star schema*, there exists a table $t_h \in CI_{ROLAP}$ representing all levels in $l_i \in L$. For each l_i there exists an attribute $pk_i \in t_h$ which

identifies each level member and for each level attribute $at_i \in l_i$ there exists an attribute $a_i \in t_h$.

The mapping for a *balanced hierarchy represented as a star schema* is similar to the one in Step 1, except that the `rr:logicalTable` is the same for all levels. According to Figure 16, the Time dimension from Figure 4, is a balanced hierarchy represented as a star schema. The R2RML mapping that produces the members in levels Month and Year is the following:

```
<#TriplesMapMonth> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName "Time" ] ;
rr:subjectMap [
 rr:termType rr:IRI ;
 rr:template "http://www.fing.edu.uy/instances/nw/Month#{MonthName}{Year}";];
rr:predicateObjectMap [
 rr:predicate qb4o:memberOf ;
 rr:object nw:month; ] ;
rr:predicateObjectMap [
 rr:predicate nw:monthNumber ;
 rr:objectMap [ rr:column "MonthNumber" ] ; ] ;
rr:predicateObjectMap [
 rr:predicate nw:monthName ;
 rr:objectMap [ rr:column "MonthName" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:inYear;
  rr:objectMap [
    rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Year#{Year}" ] ;] .
<#TriplesMapYear> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName "Time" ] ;
rr:subjectMap [
 rr:termType rr:IRI ;
 rr:template "http://www.fing.edu.uy/instances/nw/Year#{Year}" ; ] ;
rr:predicateObjectMap [ rr:predicate qb4o:memberOf ; rr:object nw:year ; ] ;
rr:predicateObjectMap [ rr:predicate nw:yearNumber ;
                        rr:objectMap [rr:column "Year"];].
```

The R2RML mapping above, generates triples like:

```
nwm:201101 qb4o:memberOf nw:month ;
 nw:monthNumber 1; nw:monthName "January"; nw:inYear nwy:2011 .
nwy:2011 qb4o:memberOf nw:year ; nw:yearNumber 2011 .
```

Step 2 (**Ragged hierarchies**) These hierarchies can be represented as *snowflake schemas* or as *star schemas*.

If $h \in H$ is a ragged hierarchy composed of a set of levels L, represented as a *snowflake schema*, there exists a set of tables $T_h \in CI_{ROLAP}$, where each table $t_i \in T_h$ represents a level in $l_i \in L$, and contains a key attribute pk_i, and one attribute a_i for each level attribute $at_i \in l_i$. For each pair $l_i, l_{i+1} \in h$, represented as $t_i, t_{i+1} \in T_h$, such that $parentLevel(l_i, h) = l_{i+1}$, there exists a optional foreign key attribute $fk_i \in t_i$ referencing $pk_{i+1} \in t_{i+1}$. Note that these foreign key attributes may contain null values. Given that R2RML template-valued term mappings only produce results when the attributes used in the template are not null, to generate instances we can apply the same strategy than the one used in balanced hierarchies represented as a snowflake schema.

As an example, the Geography dimension, obtained from Figure 4, is a ragged hierarchy represented as a snowflake schema (Figure 16). We next show the R2RML mapping that produces the members in level City using data from the table City. Members in this level may have a corresponding one in the State or Country level, but not in both.

The mapping in lines 10 to 14 implements the RUP relationship between members of the City and State levels (represented in the relational model via a value different than null in attribute StateKey in the City table, and a foreign key to the State table). Analogously, the mapping to the Country level is presented in lines 10 to 14.

```
1  <#TriplesMapCity> a rr:TriplesMap ;
2  rr:logicalTable [ rr:tableName "City" ] ;
3  rr:subjectMap [
4  rr:termType rr:IRI ;
5  rr:template
6  "http://www.fing.edu.uy/inco/cubes/instances/northwind/City#{CityKey}";];
7  rr:predicateObjectMap [ rr:predicate qb4o:memberOf ; rr:object nw:city ; ] ;
8  rr:predicateObjectMap [ rr:predicate nw:cityName ;
9  rr:objectMap [ rr:column "CityName" ];] ;
10 rr:predicateObjectMap [ rr:predicate nw:inState;
11 rr:objectMap [
12 rr:termType rr:IRI ;
13 rr:template
14 "http://www.fing.edu.uy/inco/cubes/instances/northwind/State#{StateKey}"];].
15 rr:predicateObjectMap [ rr:predicate nw:inCountry;
16 rr:objectMap [
17 rr:termType rr:IRI ;
18 rr:template
19 "http://www.fing.edu.uy/inco/cubes/instances/northwind/Country#{CountryKey}"];].
```

The triples shown below, are generated by the mapping above. They represent the "Vatican" city in the "Vatican" country, and "Salvador" city in the Brazilian state of "Bahia".

```
nwci:Vatican qb4o:memberOf nw:city;
  nw:cityName "Vatican";
  nw:inCountry nwco:1025.
nwci:Salvador qb4o:memberOf nw:city;
  nw:cityName "Salvador";
nw:inState nwst:926;
```

If $h \in H$ is a ragged hierarchy composed of a set of levels L, represented as a *star schema*, we can also apply the strategy for balanced hierarchies.

Step 3 (**Recursive hierarchies**) These hierarchies are represented as a table containing all attributes in a level, and a foreign key to the same table, relating children members to their parents. If $h \in H$ is a parent-child hierarchy, composed of a pair of levels $l_i, l_{i+1} \in h$ such that $parentLevel(l_i, h) = l_{i+1}$, there exists a table $t_h \in CI_{ROLAP}$ which contains a key attribute pk_i that identifies the members of l_i and an attribute $fk_i \in t_h$, that identifies the members of l_{i+1} and is a foreign key referencing $pk_i \in t_h$.
The mapping for level members in a *recursive* hierarchy is similar to the one presented for a star representation of a balanced hierarchy, since all hierarchy levels are populated from the same table (`rr:logicalTable`). The Supervision hierarchy in Figure 4 is an example of this. Its relational representation is depicted in the Employee table in Figure 16. For this hierarchy we have the following.

```
<#TriplesMapEmployee> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName "Employee" ] ;
rr:subjectMap [
  rr:termType rr:IRI ;
  rr:template "http://www.fing.edu.uy/instances/nw/Employee#{EmployeeKey}" ; ];
rr:predicateObjectMap [ rr:predicate qb4o:memberOf ; rr:object nw:employee ; ] ;
rr:predicateObjectMap [ rr:predicate nw:firstName ;
```

```
                          rr:objectMap [ rr:column "FirstName" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:lastName ;
                          rr:objectMap [ rr:column "LastName" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:supervises;
 rr:objectMap [
  rr:termType rr:IRI ;
  rr:template
   "http://www.fing.edu.uy/instances/nw/Employee#{SupervisorKey}"];].
```

We show next some triples generated by the R2RML mapping.

```
nwe:5 qb4o:memberOf nw:employee ;
  nw:firstName "Steven" ; nw:lastName "Buchanan" ; nw:supervises nwe:2 .
```

Step 4 (**Nonstrict hierarchies**). Here, each level is represented in a different table, and a *bridge table* is used to represent the many-to-many relationship between level members. If $h \in H$ is a nonstrict hierarchy, composed of a set of levels L, there exists a set of tables $T_h \in CI_{ROLAP}$, one table $t_i \in T_h$ with a key attribute pk_i, for each level $l_i \in L$. For each pair of levels $l_i, l_{i+1} \in h$, represented as $t_i, t_{i+1} \in T$, such that $parentLevel(l_i, h) = l_{i+1}$ and members of l_i have exactly one associated member in l_{i+1}, the mapping is the same as for the snowflake representation of balanced hierarchies. If members of l_i have more than one associated member in l_{i+1}, there exists a bridge table $b_i \in T$ that contains two attributes fk_i, fk_{i+1} referencing $pk_i \in t_i$ and $pk_{i+1} \in t_{i+1}$ respectively. Thus, each pair of levels is populated by three triple maps `rr:TriplesMap`: two of them generate level members, while the third one uses the bridge table as `rr:logicalTable` to generate parent-child relationships between level members.

The Territories hierarchy in Figure 4, represented in the relational model of Figure 16, is nonstrict. The R2RML mapping that generates the parent-child relationship between members in the Employees and City levels, in the Territories hierarchy is given next.

```
<#TriplesMapTerritories>
rr:logicalTable [ rr:tableName "Territories" ] ;
  rr:subjectMap [
    rr:template "http://www.fing.edu.uy/instances/nw/Employee#{EmployeeKey}";];
rr:predicateObjectMap [ rr:predicate nw:worksIn ;
  rr:objectMap [
   rr:termType rr:IRI ;
   rr:template "http://www.fing.edu.uy/instances/nw/City#{CityKey}" ] ; ] .
```

The R2RML mapping generates triples like the following ones.

```
nwe:5  nw:worksIn nwci:98, nwci:101, nwci:115, nwci:117, nwci:124 .
```

Step 5 (**Facts**) For each fact F, $CI_{RDF} = CI_{RDF} \cup t$; t is an R2RML `rr:TripleMap` that generates fact instances (observations). The components of t are as follows: a `rr:logicalTable`, a `rr:subjectMap`, which is an IRI built using the rr:template $f\{F_{KEY}\}$, a `rr:predicateObjectMap` stating the observations dataset, a `rr:predicateObjectMap` for each level related to the fact, and one `rr:predicateObjectMap` for each measure. F_{KEY} provides a unique value for each fact, and can be obtained from a fact table column, or concatenating the keys of all the level members that participate in the fact.

The R2RML mapping that generates the members in the Sales facts is shown below. Note that, in this case, the concatenation of the attributes OrderLine and OrderLineNo from the table Sales can be used to identify each fact. Also note the representation of the role-playing dimensions, and the key in rr:template.

```
<#TriplesMapSales> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName "Sales" ] ;
rr:subjectMap [
 rr:termType rr:IRI ;
 rr:template "http://www.fing.edu.uy/instances/nw/Sale#{OrderNo}_{OrderLineNo}";
 rr:class qb:Observation ; ] ;
rr:predicateObjectMap [ rr:predicate qb:dataSet ; rr:object nwi:dataset1 ; ] ;
rr:predicateObjectMap [ rr:predicate nw:customer ;
  rr:objectMap [
    rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Customer#{CustomerKey}"];] ;
rr:predicateObjectMap [ rr:predicate nw:employee ;
  rr:objectMap [
    rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Employee#{EmployeeKey}"];];
rr:predicateObjectMap [ rr:predicate nw:orderDate ;
  rr:objectMap [
    rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Time#{OrderDateKey}" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:dueDate ;
  rr:objectMap [
    rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Time#{DueDateKey}" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:supplier ;
  rr:objectMap [
    rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Supplier#{SupplierKey}"];];
rr:predicateObjectMap [ rr:predicate nw:quantity ;
                        rr:objectMap [ rr:column "Quantity" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:unitPrice ;
                        rr:objectMap [ rr:column "UnitPrice" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:salesAmount ;
                        rr:objectMap [ rr:column "SalesAmount" ] ; ] .
```

The triples shown next, are produced by the R2RML mapping above.

```
@prefix nws: <http://www.fing.edu.uy/inco/cubes/instances/northwind/Sale#>
nws:10248_1 a qb:Observation,
  qb:dataSet <http://www.fing.edu.uy/inco/cubes/instances/northwind#dataset1> ;
  nw:customer nwc:357 ;
  nw:employee nwe:5 ;
  nw:orderDate nwt:4 ;
  nw:dueDate nwt:32 ;
  nw:supplier nws:5 ;
  nw:quantity 12 ; nw:unitPrice 14 ; nw:salesAmount 168 .
```

5.2.2 *Implementation*

We have implemented the approach in Section 5.2.1. We denoted such implementation as the *QB4OLAP Engine*. This engine takes as input the specification of a MD data cube, and its relational implementation (a set of relational tables), and produces two RDF graphs that use the QB4OLAP vocabulary. One of these graphs represents the *schema of the cube* and the other one an *instance of the cube*. The graphs are stored in an RDF triple store, which also implements an SPARQL

endpoint. This allows publishing the cubes on the web, and offering the capability of performing queries over them.

Given that there is no machine-processable and standard format to represent the cube schema in traditional ROLAP systems, in practice, this information is usually stored in proprietary formats which are not interchangeable between different vendors. One exception is the case of the Pentaho Mondrian open-source OLAP server.[1] The Mondrian schema[2] is an XML document that defines a MD database. It contains a *logical model*, consisting of cubes, dimensions, hierarchies, and members, and a *mapping* of this model into a physical one. The physical model is the source of the data which is presented through the logical model, typically, a star schema, implemented as a set of tables in a relational database. Mondrian supports star, snowflake, and starflake schemas.

QB4OLAP Engine is composed of several modules that tackle each one of the extraction and transformation processes. Figure 17 depicts the data and control flows between these modules, and the interaction with external components.

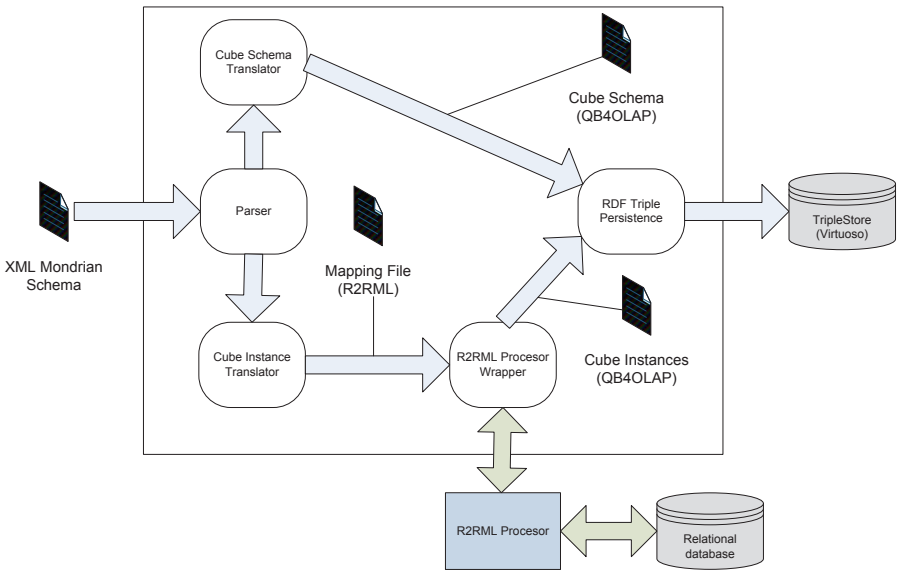


Figure 17: QB4OLAP Engine architecture.

The *parser* component first validates the input XML file that contains the Mondrian schema against its DTD. Then, it extracts the logical model of the cube, and the mappings onto the physical model, creating an in-memory representation of this information that will be used throughout all the transformation processes. The *cube schema translator* is responsible for translating the logical model of the data cube into RDF, producing as output a set of RDF triples that represent the schema of the cube using QB4OLAP.

The *cube instance translator* is responsible for translating the data in the relational database into RDF triples, that represent instances of the data cube. That means, it generates triples, using QB4OLAP,

1 http://www.community.pentaho.com/projects/mondrian/
2 http://www.mondrian.pentaho.com/documentation/schema.php

and the schema of the cube, namely: level members with their corresponding attribute values, parent-child relationships between level members, and fact instances. The task at hand corresponds to the more general problem of providing an RDF view over relational data. Instead of directly generating triples that represent the instances, this component produces a set of R2RML mappings, that encodes how to produce these instances from the data stored in the physical model of the cube. Then, this set of R2RML mappings can either be used to generate a static set of triples that represent the underlying relational data (*data materialization*) or to provide a non-materialized RDF view of the relational data (*on-demand mapping*). Each of these strategies has a set of well-known advantages and disadvantages [10]. In our current implementation we have chosen to materialize the triples that represent the instance. This decision is mainly based on the static nature of the underlying data (there in, in general, no need for updates in a DW). After the mappings are obtained, the *R2RML processor wrapper* interacts with an R2RML processor, which actually builds the RDF triples. Finally, the *RDF triples persistence* module stores all the triples into a triplestore.

5.3 ENRICHING QB DATASETS USING QB4OLAP

The QB4OLAP vocabulary is compatible with QB, in the sense that QB4OLAP cube schemas can be built on top of data cube instances (observations) already published using QB. Existing applications, or applications that do not require OLAP style-analysis, can still use the QB schema and instances. Therefore, the cost of adding OLAP capabilities to existing datasets is the cost of building the new schema, in other words, the cost of building the analysis dimensions while the cube instances remain untouched. We call this process *data enrichment*.

As already mentioned, one of the main differences between QB and QB4OLAP is the possibility of the latter of specifying a dimension hierarchy. On the contrary, QB allows only to define dimension members to be hierarchically organized into a `skos:ConceptScheme` structure using the `skos:narrower` property or its inverse `skos:broader`. This concept scheme represents a hierarchy of level *members*. Therefore, generating a QB4OLAP cube from a QB cube requires producing several pieces of information, not present in a typical QB data set, among them:

- The hierarchy of levels (structural metadata).
- At the dimension instance level, the association between members and dimension levels, and RUP functions between members.
- The association between measures and aggregate functions.

This information can be inferred from internal and/or external data, either (semi-)automatically or manually (e.g., by a curator). Exploiting the existing QB semantics (i.e., metadata) and the analysis of the

data set instances (i.e., data), enables the automatic discovery of potentially new metadata concepts (e.g., new dimension levels), which could be suggested to the user, or required by her. In addition, the user may help the (semi-)automatic procedures in the taks of adding missing semantics and conflict resolution [43]. If the structural information could not be produced or obtained, we must assume a hierarchy of only one level, which can be created from the QB data straightforwardly.

Another problem to be addressed when producing a QB4OLAP data set from a QB one, refers to the generation of IRIs. Recall that observations in QB are expressed using *dimensions*, while QB4OLAP requires observations to be expressed in terms of *dimension levels*. To avoid rewriting observations, we propose to reuse the IRIs that represent dimensions in the QB DSD, to represent the bottom level of each dimension in the generated QB4OLAP DSD.

Algorithm 1 receives a QB cube schema, and producesa QB4OLAP cube schema. The new schema must be linked to the dataset containing the (existing) observations. The algorithm creates and populates the dimension structure, and creates a new DSD. We assume that, for each dimension, a hierarchy of levels is known, and that we also know how to populate each level.

Algorithm 1 Creating a cube in QB4OLAP from a cube in QB

Input: dsd_1,the data structure definition of a data cube c_1 in QB; D_1 the set of dimensions in c_1; M, the set of pairs (m_i, ag_i) where m_i is a measure and ag_i is its corresponding aggregate function. For each $d_i \in D_1$, at least one hierarchy of levels h_i is known. For each level $l_i \in h_i$ a set of level members lm_i is known.
Output: dsd_2, the data structure definition of a data cube c_2 in QB4OLAP, obtained from the QB data set.

for all $d_i \in D_1$ (d_i a `qb:DimensionProperty`) **do**
  Create a new dimension d_j (d_j a `qb:DimensionProperty`)
  Let H_i be the set of known hierarchies for d_i
  for all $h_i \in H_i$ **do**
    Add a triple (h_i a `qb4o:Hierarchy`)
    Add triples (h_i `qb4o:inDimension` d_j) and (d_j `qb4o:hasHierarchy` h_i)
    for all $l_i \in h_i$ **do**
      Add triples (l_i a `qb4o:LevelProperty`) and (h_i `qb4o:hasLevel` l_i)
      Let lm_i be the set of know member levels of l_i
      for all $me_i \in lm_i$ **do**
        Add a triple (me_i `qb4o:memberOf` l_i).
      end for
    end for
    for all $(l_j, l_k) \in h_i$ such that $l_j \rightarrow l_k$ **do**
      Add triples ($\_hs_{jk}$ a `qb4o:HierarchyStep`), ($\_hs_{jk}$ `qb4o:inHierarchy` h_i)
      Add triples ($\_hs_{jk}$ `qb4o:childLevel` l_j), ($\_hs_{jk}$ `qb4o:parentLevel` l_k)
      Add a triple ($\_hs_{jk}$ `qb4o:rollup` rup_{jk}), being rup_{jk} the RDF property that implements $\rightarrow$
    end for
    if l_i is the bottom level in h_i **then**
      Add a triple (dsd_2 `qb:component` [`qb4o:levell`$_i$])
    end if
  end for
end for
for all m_i such that (dsd_1 `qb:component` [`qb:measure` m_i]) **do**
  Add a triple (dsd_2 `qb:component` [`qb:measure` m_i;`qb:hasAggregateFunction` ag_i])
end for

The cube structure definition we presented in Example 4.1.1 is the result of applying Algorithm 1 to the cube structure in Example 3.2.1.

To conclude, we analyze the complexity of Algorithm 1 with respect to the size of the components of the cube.

COMPLEXITY. Let $c = (x, D, M)$ be a cube schema, where D is the set of dimensions, and M is the set of measures. For each $d_i \in D$, we denote G_{d_i} the directed acyclic graph (DAG) representing the dimension schema. For each level $l_i \in G_{d_i}$, LM_{l_i} is the set of level members. Let $LM_{d_i} = \bigcup_{i=1}^{i=|L_{d_i}|} LM_{l_i}$.

Proposition 5.3.1. The upper complexity bound for Algorithm 1 is given by $O(\sum_{i=0}^{|D|} (|nodes(G_{d_i})| + |edges(G_{d_i})| + |LM_{d_i}|) + |M|)$. The upper bound of $|edges(L_{d_i})|$ is given by $\sum_{j=1}^{k} |n_{j-1}|.|n_j|$, where n_j is the set of nodes in G_{d_i} at distance j from the bottom node in G_{d_i}, and k is the distance between the bottom and top (i.e., *All*) nodes.

□

Proposition 5.3.1 shows that the main source of complexity is the size of the set of dimension level members, which is usually small, compared to the set of observations (facts).

5.3.1 *Implementation*

In a joint work with other researchers in the field [44], we have produced QB2OLAP, a tool to semi-automatically enrich QB data sets with QB4OLAP semantics. By exploring the data set, the system defines dimension levels and hierarchies, and generates the corresponding QB4OLAP triples. The workflow of the Enrichment process is presented in Figure 18.

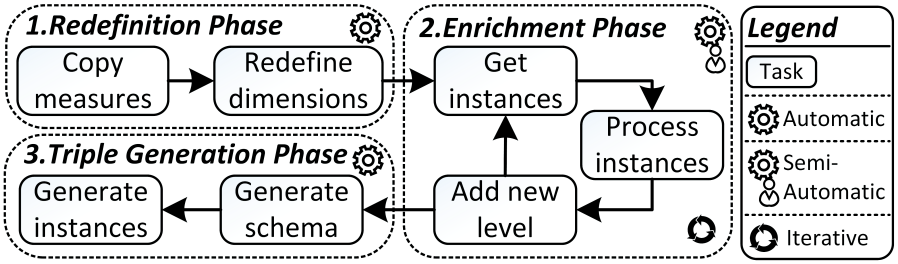


Figure 18: QB2OLAP enrichment workflow [44]

First, we have the *Redefinition Phase*, where elements of the input QB cube are redefined according to QB4OLAP semantics, i.e., dimensions are redefined as levels, while measures are copied and an aggregate function is assigned to them. Starting from the levels of this redefined schema, the *Enrichment Phase* collects level instances and their properties, discovering candidate hierarchies based on functional dependencies. Finally, QB4OLAP triples are produced in the *Triple Generation Phase*. The tool is implemented in Java 8, and the Jena 2.13.0 library[3] is used to manipulate RDF. QB and QB4OLAP graphs, and Virtuoso Open Source (version 7)[4] is used as triplestore. The module GUI is implemented in SWT.

3 https://jena.apache.org/
4 http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main

5.4 SUMMARY

In this chapter we have shown different strategies to create QB4OLAP cubes, illustrating the three main ways in which QB4OLAP is intended to be used: creating cubes from scratch (using a MD model), exporting an existing relational DW, and enriching existing QB data sets at the minimum possible cost. First, we showed how advanced MD design features can be represented in QB4OLAP. Then, we provided algorithms and tools to obtain QB4OLAP data cubes from a ROLAP DW. Finally, we discussed how to create a QB4OLAP schema to enrich a data set published using the QB vocabulary.

Part II

QUERYING SEMANTIC WEB DATA CUBES

6

QUERYING QB4OLAP DATA CUBES

"The Answer to the Great Question...
Of Life, the Universe and Everything...
Is... Forty-two,' said Deep Thought,
with infinite majesty and calm."

Douglas Adams, The Hitchhiker's Guide to the Galaxy

One of the goals of our work is to promote publishing and querying QB4OLAP cubes. Typically, users in a self-BI environment, or even traditional OLAP users, are hardly aware of SW models and languages, like RDF or SPARQL. However, they will easily capture the idea of languages dealing with cube operations. In this chapter we motivate this idea, and explain our approach to enable OLAP analysis of QB4OLAP cubes, by users non-expert on SW technologies.

6.1 MOTIVATION AND GENERAL SCHEME

To promote OLAP users to publish and analyze statistical and MD data on the SW, we must make their lives easy. That means, they should not be required to learn new languages or models that are far from their field of expertise. To accomplish this, we believe that the best way to proceed is to allow them to manipulate the object that they know the best, namely, *the data cube*, and make them forget about SW technicalities. This is why in this chapter we propose a high-level language, denoted CQL, based on an algebra for OLAP. The idea is that the user will write her queries at the *conceptual level*, and we provide the mechanisms to translate CQL queries into SPARQL ones over the QB4OLAP-based RDF representation (at the *logical level*). The main advantage of this approach is that it allows users to perform OLAP queries over QB4OLAP cubes, without dealing with RDF or SPARQL. Also, SPARQL optimization tips and best practices can be incorporated in the CQL to SPARQL translation process, producing efficient SPARQL queries that would be hard to obtain for an average user.

The query processing pipeline is shown in Figure 19. The process starts with a CQL query that is first simplified. This stage aims at rewriting the query to eliminate unnecessary operations, and reordering operations written in a sequence that is probably not the best one. We remark that, in a self-service BI environment, users may not be experts, even to write queries in simple languages like CQL. The second step translates the simplified CQL query into a single SPARQL expression, following a *naïve* approach. Finally, we apply SPARQL optimization heuristics to improve the performance of the *naïve* queries, taking into consideration SPARQL performance improvement strategies and physical data organization.

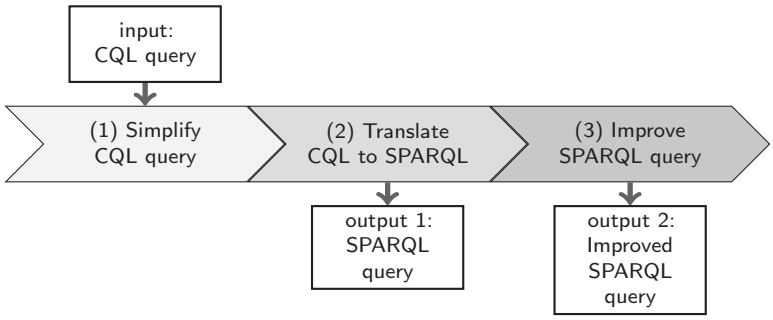


Figure 19: Our Query processing pipeline takes as input a CQL query and produces SPARQL queries.

In the remainder of this chapter, we will explain in detail each step in this pipeline. First, in Section 6.2, we present a formal data model for OLAP data cubes, also showing that it is possible to represent data cubes that adhere to this formal model using QB4OLAP. Then, in Section 6.3 we present CQL, giving a precise semantics to the OLAP operators that conform this language. In Section 6.4 we describe our approach to CQL queries simplification. The last two sections are dedicated to SPARQL implementation of CQL queries. In Section 6.5 we introduce the algorithms that produce naïve SPARQL implementations of CQL queries, while we discuss how these queries can be improved in Section 6.6.

6.2 DATA MODEL

Several data models for MD data are found in the literature. The data model we use in the sequel is inspired on the user-centric conceptual data model proposed by Ciferri et al. [6], where a thorough analysis of MD data models and query languages is presented. We now introduce the formal model for data cubes upon which we build our query language. The asylum applications case, presented in Section 1.1.2, is used to exemplify each formal concept.

Definition 6.2.1. (**Dimension schema**). A *dimension schema* is a tuple $\langle d, \mathcal{L}, \rightarrow, \mathcal{H} \rangle$ where: (a) d is the name of the dimension; (b) $\mathcal{L}$ is a set of pairs $\langle l, A_l \rangle$, called *levels*, where l identifies a level in $\mathcal{L}$, and $A_l = \langle a_1, \ldots, a_n \rangle$ is a tuple of level *attributes*. Each attribute a_i has a domain $\mathrm{Dom}(a_i)$; (c) '$\rightarrow$' is a partial order over the levels in $\mathcal{L}$, with a unique bottom level and a unique top level (All); (d) $\mathcal{H}$ is a set of pairs $\langle h_n, L_h \rangle$, called *hierarchies*, where h_n identifies the hierarchy, $L_h \subseteq \mathcal{L}$, and there is at least one path between the bottom level in d, and the top level All that contains only the levels in L_h. □

Example 6.2.1. (Citizenship dimension schema) The schema of the Citizenship dimension in Figure 5 is defined as:
$\langle$Citizenship $\mathcal{L} = \{\langle$Country, $\langle$countryCode, countryName$\rangle\rangle$,
$\langle$Continent, $\langle$continentCode, continentName$\rangle\rangle$,
$\langle$GovernmentType, $\langle$governmentType$\rangle\rangle$,
$\langle$All, $\langle$all$\rangle\rangle\}$;
' $\rightarrow$ ' $= \{$Country $\rightarrow$ Continent, Country $\rightarrow$ GovernmentType,

$\mathsf{Continent} \rightarrow \mathsf{All}, \mathsf{GovernmentType} \rightarrow \mathsf{All}\}$;
$\mathcal{H} = \{\langle \mathsf{Geography}, \{\mathsf{Country}, \mathsf{Continent}, \mathsf{All}\}\rangle,$
$\langle \mathsf{Government}, \{\mathsf{Country}, \mathsf{GovernmentType}, \mathsf{All}\}\rangle\}$

□

Definition 6.2.2. (**Dimension instance**). Given a dimension schema $\langle d, \mathcal{L}, \rightarrow, \mathcal{H}\rangle$, a *dimension instance* I_d is a tuple $\langle\langle d, \mathcal{L}, \rightarrow, \mathcal{H}\rangle, \mathcal{T}_l, \mathcal{R}\rangle$ where: (a) $\mathcal{T}_l$ is a finite set of tuples of the form $\langle v_1, v_2, \ldots, v_n\rangle \forall l$ such that $L = \langle l, \langle a_1, \ldots, a_n\rangle\rangle \in \mathcal{L}$, and $\forall i, i = 1, \ldots, n, v_i \in \mathrm{Dom}(a_i)$; (b) $\mathcal{R}$ is a finite set of relations $\mathrm{RUP}^{L_j}_{L_i}, L_i, L_j \in L$, and where $L_i \rightarrow L_j \ \in$ '$\rightarrow$', called *rollup relationships*. □

Remark 1. Most MD models assume that relations between parent and child levels are actually functions, allowing only dimensions where each member in the child level has exactly one associated member in the parent level. Despite Definition 6.2.2 allows rollup relations, in the next sections, and to simplify the presentation, we will work with rollup functions.

Example 6.2.2. (Citizenship dimension instance) A possible instance of the Citizenship dimension in Figure 5 is:
$\mathcal{T}_{\mathsf{Country}} = \{\langle$ 'AD', 'Andorra' $\rangle, \ldots, \langle$'ZW', 'Zimbabwe'$\rangle\}$
$\mathcal{T}_{\mathsf{Continent}} = \{\langle$'AF', 'Africa'$\rangle, \ldots, \langle$'OC', 'Oceania'$\rangle\}$
$\mathcal{T}_{\mathsf{GovernmentType}} = \{\langle$'Republic'$\rangle, \ldots, \langle$'Unitary state'$\rangle\}$
$\mathcal{T}_{\mathsf{All}} = \{\langle$ 'all'$\rangle\}$
$\mathcal{R} = \{\mathrm{RUP}^{\mathsf{Continent}}_{\mathsf{Country}}, \mathrm{RUP}^{\mathsf{All}}_{\mathsf{Continent}}, \mathrm{RUP}^{\mathsf{GovernmentType}}_{\mathsf{Country}}, \mathrm{RUP}^{\mathsf{All}}_{\mathsf{GovernmentType}}\}$,
with $\mathrm{RUP}^{\mathsf{Continent}}_{\mathsf{Country}} = \{(\langle$ 'AD', 'Andorra'$\rangle$, $\langle$'EU', 'Europe'$\rangle), \ldots, (\langle$ 'ZW', 'Zimbabwe'$\rangle$, $\langle$'AF', 'Africa'$\rangle)\}$;
$\mathrm{RUP}^{\mathsf{GovernmentType}}_{\mathsf{Country}} = \{(\langle$ 'AD', 'Andorra'$\rangle$, $\langle$'Unitary state'$\rangle), \ldots, (\langle$ 'ZW', 'Zimbabwe'$\rangle$, $\langle$'Presidential system'$\rangle)\}$;
$\mathrm{RUP}^{\mathsf{All}}_{\mathsf{Continent}} = \{(\langle x\rangle, \langle \mathsf{all}\rangle) \mid x \in \mathcal{T}_{\mathsf{Continent}}\}$;
$\mathrm{RUP}^{\mathsf{All}}_{\mathsf{GovernmentType}} = \{(\langle x\rangle, \langle \mathsf{all}\rangle) \mid x \in \mathcal{T}_{\mathsf{GovernmentType}}\}$.

□

Definition 6.2.3. (**Cube schema**). Assume that there is a set $\mathcal{A}$ of aggregate functions (at this time we consider the typical SQL functions Sum, Count, Avg, Max, Min, the ones addressed in [25]).

A *cube schema* is a tuple $\langle C_n, \mathcal{D}, \mathcal{M}, \mathcal{F}\rangle$ where: (a) C_n is the name of the cube; (b) $\mathcal{D}$ is a finite set of dimension schemas (Definition 6.2.1); (c) $\mathcal{M}$ is a finite set of attributes, where each $m \in \mathcal{M}$, called *measure*, has domain $\mathrm{Dom}(m)$; (d) $\mathcal{F} : \mathcal{M} \rightarrow \mathcal{A}$ is a function that maps measures in $\mathcal{M}$ to an aggregate function in $\mathcal{A}$. □

Example 6.2.3. (Asylum_application cube schema) We define the cube schema, presented in Figure 5 as:
$\langle$ Asylum_application, {Sex,Age,Time,Application_type, Citizenship, Destination}, {#applications},{#applications,Sum} $\rangle$, where dimension Citizenship is defined as in Example 6.2.1. We omit the definition of the other dimensions, for the sake of brevity and to avoid redundancy. □

To define a cube instance we need to introduce the notion of *cuboid*.

Definition 6.2.4. (**Cuboid instance**). Given: (a) A cube schema $\langle C_n, \mathcal{D}, \mathcal{M}, \mathcal{F}\rangle$, where $|\mathcal{D}| = r$ and $|\mathcal{M}| = p$, (b) A dimension instance I_{d_i} for each $d_i \in \mathcal{D}, i = 1, \ldots, r$; and (c) A set of levels $\mathcal{V}_{Cb} = \{L_1, L_2, \ldots, L_D\}$ where $L_j \in \mathcal{L}_j$ in $d_i, i = 1, \ldots, r$, such that not two levels belong to the same dimension, a *cuboid instance* Cb is a partial function $Cb : \mathcal{T}_{L_1} \times \cdots \times \mathcal{T}_{L_D} \to Dom(m_1) \times \cdots \times Dom(m_M)$, where $m_k \in \mathcal{M}, \forall k, k = 1, \ldots, p$. The elements in the domain of Cb are called *cells*, and $\mathcal{V}_{Cb}$ it the *set of levels* of the cuboid. □

Example 6.2.4. (Cuboid instance) Consider the cube schema defined in Example 6.2.3. A possible instance of the cuboid Cb_1, where $\mathcal{V}_{Cb_1}$ ={Sex, Age, Month, Application_type, Country, Country} is presented in Figure 20, using a tabular representation, where the first row lists the dimensions in the cube schema, and the second row lists the level corresponding to this cuboid. □

Figure 20: Tabular representation of a cuboid instance of the asylum_application cube schema

Sex	Age	Time	Application_type	Citizenship	Destination	Measures
Sex	*Age*	*Month*	*Application_type*	*Country*	*Country*	*#applications*
F	18 to 34	201408, August 2014	new applicant	SY, Syria	DE, Germany	330
F	18 to 34	201410, October 2014	new applicant	SY, Syria	DE, Germany	490
M	18 to 34	201410, October 2014	new applicant	SY, Syria	DE, Germany	2050
M	35 to 64	201408, August 2014	new applicant	SY, Syria	DE, Germany	495
M	35 to 64	201410, October 2014	new applicant	SY, Syria	DE, Germany	795

We can now define a lattice of cuboids referring to the same cube schema, provided that we define an order between cuboids. We do this next.

Definition 6.2.5. (**Adjacent Cuboids**). Two cuboids Cb_1 and Cb_2, that refer to the same cube schema, are *adjacent* if their corresponding level sets $\mathcal{V}_{Cb_1}$ and $\mathcal{V}_{Cb_2}$ differ in exactly one level, i.e., $|\mathcal{V}_{Cb_1} - \mathcal{V}_{Cb_2}| = |\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1}| = 1$. □

Example 6.2.5. (Adjacent cuboids) Consider the cube schema defined in Example 6.2.3, and the cuboids Cb_1, Cb_2, and Cb_3 given by $\mathcal{V}_{Cb_1}$ = {Sex, Age, Month, Application_type, Country, Country}, $\mathcal{V}_{Cb_2}$ ={All, Age, Month, Application_type, Country, Country}, and $\mathcal{V}_{Cb_3}$ = {All, Age, Year, Application_type, Country, Country}. According to Definition 6.2.5, Cb_1 is adjacent to Cb_2, and Cb_2 is adjacent to Cb_3, but Cb_1 is not adjacent to Cb_3. □

Definition 6.2.6. (**Order between adjacent cuboids**).
Given two adjacent cuboids Cb_1 and Cb_2, such that $\mathcal{V}_{Cb_1} - \mathcal{V}_{Cb_2} = \{L_c\}$ and $\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1} = \{L_p\}$, and L_p and L_c are levels in a dimension d_k such that $L_c \to L_p$; then, we define the order $Cb_1 \preceq Cb_2$ between both cuboids. Moreover, for each pair of adjacent cuboids $Cb_1 \preceq Cb_2$,

Sex	Age	Time	Applica-tion_type	Citizenship	Destination	Measures
All	*Age*	*Month*	*Application_type*	*Country*	*Country*	*#applica-tions*
all	18 to 34	201408	new applicant	SY, Syria	DE, Germany	330
all	18 to 34	201410	new applicant	SY, Syria	DE, Germany	2540
all	35 to 64	201408	new applicant	SY, Syria	DE, Germany	495
all	35 to 64	201410	new applicant	SY, Syria	DE, Germany	795

(a) Cuboid Cb_2

Sex	Age	Time	Applica-tion_type	Citizenship	Destination	Measures
All	*Age*	*Year*	*Application_type*	*Country*	*Country*	*#applica-tions*
all	18 to 34	2014	new applicant	SY, Syria	DE, Germany	2870
all	35 to 64	2014	new applicant	SY, Syria	DE, Germany	1290

(b) Cuboid Cb_3

Figure 21: Two cuboid instances of the asylum_application cube schema

each cell $c = (c_1, \ldots, c_{k-1}, c_k, c_{k+1}, \ldots, c_n, m_1, m_2, \ldots m_s) \in Cb_2$ can be obtained from the cells in Cb_1 as follows. Let $(c_1, \ldots, c_{k-1}, b_{k1}, c_{k+1}, \ldots, c_n, m_{1,1}, m_{2,1}, \ldots m_{s,1})$, $(c_1, \ldots, c_{k-1}, b_{k2}, c_{k+1}, \ldots, c_n, m_{1,2}, m_{2,2}, \ldots, m_{s,2}), \ldots, (c_1, \ldots, c_{k-1}, b_{kq}, c_{k+1}, \ldots, c_n, m_{1,p}, m_{2,p}, \ldots m_{s,p})$ be all the cells in Cb_1 where $(b_{k_i}, c_k) \in RUP^{Lp}_{L_c}, i = 1 \ldots q$. Measures in $c \in Cb_2$ are computed as $m_i = AGG_i(m_{i,1}, \ldots, m_{i,j})$, $j = 1..q$, where AGG_i is the aggregate function related to m_i. □

Example 6.2.6. (Order between cuboids) Consider cuboids Cb_1, Cb_2, and Cb_3 in Example 6.2.5. Then $Cb_1 \preceq Cb_2$, because Month $\rightarrow$ Year holds, and $Cb_2 \preceq Cb_3$, because Country $\rightarrow$ Continent holds. □

Finally we define a *cube instance* as the lattice of all possible cuboids that share the same cube schema.

Definition 6.2.7. (**Cube Instance**). Given a cube schema $\langle C_n, \mathcal{D}, \mathcal{M}, \mathcal{F} \rangle$ where $|\mathcal{D}| = D$ and $|\mathcal{M}| = M$, and a dimension instance I_i for each $D_i \in \mathcal{D}, i = 1, \ldots, D$, a cube instance CI is the lattice $\{CB, \preceq\}$ where CB is the set of all possible cuboids, and $\preceq$ is the order between adjacent cuboids in CB. □

A *Cube Instance* is the lattice composed of all cuboids that share the same cube schema, defined over the $\preceq$ order relation. The bottom of this lattice is the original cube, and the top is the cuboid with just the All level for all the dimensions in the cube. If Cb_i and Cb_j are two cuboids in the lattice, such that there is a path from Cb_i to Cb_j, we say that $Cb_i \preceq^* Cb_j$.

Example 6.2.7. (Cuboids of the asylum_application cube) Consider the cube schema defined in Example 6.2.3. All possible combinations of the levels in the six dimensions of the cube, lead to 216 cuboids, which are organized in a lattice. Assuming the instance of cuboid Cb_1 in Figure 20, Figure 21a and Figure 21b present tabular representations of instances of cuboids Cb_2, and Cb_3 given by $\mathcal{V}_{Cb_2}$ ={All, Age, Month, Application_type, Continent, Country}, and $\mathcal{V}_{Cb_3}$ ={All, Age, Year, Application_type, Continent,Country}. □

6.2.1 *Data cubes in QB4OLAP*

We have shown that all the concepts in the formal model introduced above, can be represented in QB4OLAP. This includes dimension schemas (with all of their features, like attributes, levels, and hierarchies), dimension instances, cube schemas, cube instance. The only concept whose QB4OLAP representation we have not addressed so far, is the concept of *cuboids*. We next define this notion, since we need it in the sequel.

Definition 6.2.8. (**Cuboid instance in QB4OLAP**) A cuboid instance in QB4OLAP is a set of `qb:Observations` organized in a `qb:DataSet`. An instance of the `qb:DataStructureDefinition` class, represents the set of levels in the cuboid and the property `qb:structure` is used to relate them. To state the fact that a cuboid instance adheres to a specific cube schema we use the property `qb4o:isCuboidOf`. □

Before presenting an example, we define the representation of a cube schema (according to Definition 6.2.3).

Definition 6.2.9. (**Cube schema in QB4OLAP**) A cube schema in QB4OLAP is an instance of the `qb:DataStructureDefinition` class, that represents the set of **dimensions** and measures in the cube. □

Example 6.2.8. (Cube Schema in QB4OLAP) The QB4OLAP representation of the asylum_application cube schema is defined as follows.

```
sc:migr_asyappctzmCUBE
rdf:type qb:DataStructureDefinition ;
qb:component [ qb:measure sdmxm:obsValue ; qb4o:aggregateFunction qb4o:sum] ;
qb:component [ qb:dimension sc:sexDim ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [ qb:dimension sc:ageDim ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [ qb:dimension sc:timeDim ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [ qb:dimension sc:asylappDim ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [ qb:dimension sc:citizenshipDim ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [ qb:dimension sc:destinationDim ; qb4o:cardinality qb4o:ManyToOne] ;
skos:notation "migr_asyappctzmCUBE" .
```

□

Example 6.2.9. (Cuboid instance in QB4OLAP) Over the cube schema in Example 6.2.8, we can define a cuboid that contains the lowest level for each dimension in the cube (i.e., the bottom of the cuboid lattice). Below we show the QB4OLAP representation of the set of levels of the cuboid, the definition of a data set that represents the cuboid instance, and also a cell in this cuboid (`qb:Observation`), which corresponds to the last row in Figure 20.

```
sc:migr_asyappctzmBOTTOM
 rdf:type qb:DataStructureDefinition ;
 qb4o: isCuboidOf sc:migr_asyappctzmCUBE;
 qb:component [ qb:measure sdmxm:obsValue; qb4o:aggregateFunction qb4o:sum ] ;
 qb:component [ qb4o:level pr:sex ] ;
 qb:component [ qb4o:level pr:age ] ;
 qb:component [ qb4o:level sdmxd:refPeriod ] ;
 qb:component [ qb4o:level pr:asyl_app ] ;
 qb:component [ qb4o:level pr:citizen] ;
 qb:component [ qb4o:level pr:geo ] ;
 skos:notation "migr_asyappctzmBOTTOM" .
```

```
eurostatdt:migr_asyappctzm
 rdf:type qb:DataSet ;
 qb:structure sc:migr_asyappctzmBOTTOM .

eurostatcell:M,CD,F,Y18-34,NASY_APP,BE,2013M03
 rdf:type qb:Observation ;
 qb:dataSet eurostat: migr_asyappctzm ;
 pr:citizen citizen:CD ;
 pr:sex sex:F ;
 pr:age age:Y18-34 ;
 pr:asyl_app asyl_app:NASY_APP ;
 pr:geo geo:BE ;
 measure:obsValue 30 ;
 sdmxd:refPeriod time:201303 .
```

□

In Section 6.5 we discuss in detail how to compute the cuboids in the lattice.

6.3 THE CQL LANGUAGE

We now introduce a high-level query language, which we denote CQL. The language is based on an algebra for OLAP, and deals only with cubes, independently of the underlying data types, or of how the cube is actually implemented. Further, the data model introduced in Section 6.2 allows use to define a precise semantics for the OLAP operations supported by CQL. We present the language next.

CQL follows the ideas introduced by Ciferri et al. [6], where a clear separation between the conceptual and the logical levels is made, allowing users to manipulate cubes regardless of their underlying representation. In that paper, an algebra, denoted Cube Algebra, is sketched. CQL is a subset of such algebra, and we chose it mainly due to two reasons: (i) It includes the most common OLAP operations; (ii) The correctness of these algebra has been proven in [25], where such a proof is given for the first time. The semantics we define for our operations is equivalent to the one given in [25].

6.3.1 *CQL Operations*

The CQL language is composed of the OLAP operations presented in Chapter 1: ROLL-UP, DRILL-DOWN, SLICE, and DICE. In this section we formalize them, in terms of our data model, and the notion of cuboid lattice, which allows us to give the operations an elegant and precise semantics.

We consider the following sets: $\mathbb{C}$ is the set of all the cuboids in a cube instance, $\mathbb{D}$ is the set of dimensions, $\mathbb{M}$ is the set of measures, $\mathbb{L}$ is the set of dimension levels, and $\mathbb{B}$ is the set of boolean expressions over level attributes and measures. For clarity, and to simplify the definitions, we assume that the aggregate function associated to the measures is SUM, so we drop $\mathcal{F}$ from the cube schema definition.

The ROLL-UP operation is a function ROLL-UP: $\mathbb{C} \times \mathbb{D} \times \mathbb{L} \rightarrow \mathbb{C}$ that summarizes data at a higher level in a dimension hierarchy. It is defined as follows:

Definition 6.3.1. (**ROLL-UP operation**) Given a cube instance CI with schema $\langle C_n, \mathcal{D}_{in}, \mathcal{M}_{in}\rangle$, a cuboid $C_{in} \in \mathsf{CI}$ with its corresponding set of levels $\mathcal{V}_{C_{in}}$, a dimension $d \in \mathcal{D}_{in}$ with schema $\langle d, \mathcal{L}, \rightarrow\rangle$, and two levels L_{in}, L_{out} in $\mathcal{D}_{in}$ such that: $L_{in} \in \mathcal{V}_{C_{in}}$ and $L_{in} \rightarrow^* L_{out}$, then ROLL-UP$(C_{in}, \mathcal{D}_{in}, L_{out})$ returns a cuboid $C_{out} \in \mathsf{CI}$ such that $\mathcal{V}_{C_{out}} = (\mathcal{V}_{C_{in}} - \{L_{in}\}) \cup \{L_{out}\}$. Notice that $C_{in} \prec^* C_{out}$ in the lattice CI. □

The DRILL-DOWN operation is a function DRILL-DOWN: $C \times D \times L \rightarrow C$ that disaggregates data down to a specific level in a dimension hierarchy.

Definition 6.3.2. (**DRILL-DOWN operation**) Given a cube instance CI with schema $\langle C_n, \mathcal{D}_{in}, \mathcal{M}_{in}\rangle$, a cuboid $C_{in} \in \mathsf{CI}$ with its corresponding set of levels $\mathcal{V}_{C_{in}}$, a dimension $d \in \mathcal{D}_{in}$ with schema $\langle d, \mathcal{L}, \rightarrow\rangle$, and two levels L_{in}, L_{out} in $\mathcal{D}_{in}$ such that: $L_{in} \in \mathcal{V}_{C_{in}}$ and $L_{out} \rightarrow^* L_{in}$, then DRILL-DOWN$(C_{in}, \mathcal{D}_{in}, l_{out})$ returns a cuboid $C_{out} \in \mathsf{CI}$ such that $\mathcal{V}_{C_{out}} = (\mathcal{V}_{C_{in}} - \{L_{in}\}) \cup \{L_{out}\}$. Notice that $C_{out} \prec C_{in}$ in the lattice CI. □

It is straightforward to show, using the lattice of cuboids, that the cuboid produced by a DRILL-DOWN on a dimension D is always reachable from the bottom of the lattice, so it can also be obtained performing a ROLL-UP over the same dimension D from the bottom cuboid. We will use this result in the sequel.

The DICE operation is a function DICE: $C \times B \rightarrow C$ that selects the values in dimension levels and measures that satisfy a Boolean condition. It resembles the SELECTION (σ) operation in relational algebra.

Definition 6.3.3. (**DICE operation**) Given a cube instance CI with schema $\langle C_n, \mathcal{D}_{in}, \mathcal{M}_{in}\rangle$, a cuboid $C_{in} \in \mathsf{CI}$ with its corresponding set of levels $\mathcal{V}_{C_{in}}$, and a Boolean condition ϕ over the measures in $\mathcal{M}_{in}$ and/or the attributes of the levels in $\mathcal{V}_{C_{in}}$, DICE(C_{in}, ϕ) returns a cuboid $C_{out} \in C$ as follows:
(a) $c_i = (c_{i_1}, \ldots, c_{i_n}, m_{i_1}, \ldots m_{i_s}) \in C_{out}$
if $\exists c_j = (c_{j_1}, \ldots, c_{j_n}, m_{j_1}, \ldots m_{j_s}) \in C_{in}$ and $c_{i_p} = c_{j_p}\ \forall p, p = 1, \ldots, n$ $m_{i_q} = m_{j_q}\ \forall q, q = 1, \ldots, s$, and c_j satisfies ϕ;
(b) $\mathcal{V}_{C_{out}} = \mathcal{V}_{C_{in}}$ □

The SLICE operation is a function SLICE: $C \times (D \cup M) \rightarrow C$ that reduces the dimensionality of a cube by removing one of its dimensions or measures. In the case of eliminating a dimension, the ROLL-UP operation is applied to this dimension in the cuboid before removing it.

Definition 6.3.4. (**SLICE operation**) Given a cube instance CI with schema $\langle C_n, \mathcal{D}_{in}, \mathcal{M}_{in}\rangle$, where $|\ \mathcal{D}_{in}\ | > 1$ or where $|\ \mathcal{M}_{in}\ | > 1$ a cuboid $C_{in} \in \mathsf{CI}$ with its corresponding set of levels $\mathcal{V}_{C_{in}}$, and a dimension $d \in \mathcal{D}_{in}$, or a measure $M \in \mathcal{M}_{in}$, depending on the input parameters. (1) SLICE(C_{in}, d) returns a cuboid $C_{out} \in C$ as follows:
(a) $c_i = (c_{i_1}, \ldots, c_{i_{k-1}}, c_{i_{k+1}}, m_{i_1}, \ldots m_{i_s}) \in C_{out}$
if $\exists c_j = (c_{j_1}, \ldots, c_{i_{k-1}}, all, c_{i_{k+1}} \ldots, c_{j_n}, m_{j_1}, \ldots m_{j_s})$ and $c_j \in$ ROLL-UP(C_{in}, d, All) and $c_{i_p} = c_{j_p}\ \forall p, p = 1, \ldots, n$, $p \neq k$, and $m_{i_q} = m_{j_q}$

$\forall q, q = 1, \ldots, s$;
(b) $\mathcal{V}_{C_{out}} = \mathcal{V}_{C_{in}} - \{l_d\}$, where l_d is the level corresponding to dimension d in C_{in}.
(2) SLICE(C_{in}, M) returns a cuboid $C_{out} \in C$ as follows:
(a) $c_i = (c_{i_1}, \ldots, c_{i_n}, m_{i_1}, \ldots, m_{i_{k-1}}, m_{i_{k+1}}, \ldots, m_{i_s}) \in C_{out}$
if $\exists c_j = (c_{j_1}, \ldots, c_{j_n}, m_{j_1}, \ldots, m_{j_{k-1}}, m_{j_k}, m_{j_{k+1}}, \ldots, m_{j_s})$ and $c_j \in C_{in}$
and $c_{i_p} = c_{j_p}$ $\forall p, p = 1, \ldots, n$, and $m_{i_q} = m_{j_q}$ $\forall q, q = 1, \ldots, s$, $q \neq k$;
(b) $\mathcal{V}_{C_{out}} = \mathcal{V}_{C_{in}}$ □

From Definitions 6.3.1 through 6.3.4, it follows that ROLL-UP and DRILL-DOWN only imply a navigation across a lattice (and do not modify it). Thus, we denote them *Instance Preserving Operations* (IPO). On the other hand, we denote DICE and SLICE as *Instance Generating Operations* (IGO), since they induce a new lattice (because they reduce the number of cells in the cuboid, or reduce the dimensionality of the cube, respectively), whose bottom cuboid is the result of the corresponding operation.

In the sequel we assume that the starting point in the navigation path representing a CQL query, is a cuboid which is at the bottom of a certain cube instance. Also, we will use the following properties. The proofs can be found in [12].

Property 6.3.1. (ROLL-UP/DRILL-DOWN commutativity) A sequence of two consecutive ROLL-UP (DRILL-DOWN) operations over different dimensions is commutative. □

Property 6.3.2. (ROLL-UP/DRILL-DOWN composition) A sequence of consecutive ROLL-UP and DRILL-DOWN operations over the same dimension D, is equivalent to a ROLL-UP from the bottom level of D, to the level reached by the last operation in the sequence. □

Property 6.3.3. (ROLL-UP/DRILL-DOWN identity) The application of the ROLL-UP or DRILL-DOWN operation over a dimension D from a level L to itself is equivalent to not applying the operation at all. □

Property 6.3.4. (Slicing ROLL-UP and DRILL-DOWN) Doing a SLICE operation over a dimension D after a sequence of ROLL-UP and DRILL-DOWN operations over D, is equivalent to apply only the SLICE operation. □

6.3.2 *CQL syntax*

We now show the syntax of CQL with an example over the asylum applications case study. The complete CQL syntax diagrams are included in Appendix B. Example 6.3.1 presents the CQL expression for Query 1 below.

> **Query 1:** Total asylum applications submitted by African citizens to France in 2013, (by sex, time, age, and citizenship country)

Example 6.3.1. (CQL queries syntax) The following CQL query produces a cuboid that answers Query 1.

```
$C1:= ROLLUP (migr_asyappctzm, timeDim, year);
$C2:= ROLLUP ($C1,citizenshipDim,continent);
$C3:= DICE ($C2,(citizenshipDim|continent|continentName = "Africa"));
$C4:= DICE ($C3,(destinationDim|geo|countryName = "France" AND
       timeDim|year|yearNum = 2013));
$C5:= DRILLDOWN ($C4,citizenshipDim,citizenship);
$C6:= SLICE ($C5,asylappDim);
$C7:= SLICE ($C6,destinationDim);
```

First, a ROLL-UP operation aggregates measures up to the Year level in the Time dimension. To keep only the cells that correspond to African citizens, a ROLL-UP is performed over the Citizenship dimension, up to the Continent level; then a DICE operation keeps cells corresponding to members of this level, that satisfy the condition over the contName attribute. Another DICE operator restricts the results to cells that correspond to France and to the year 2013. Then, a DRILL-DOWN is applied to go back to the Citizenship level (the applicant's country). Finally, dimensions Application Type and Destination are sliced out since we do not want them in the result. We use the notation dimension | level | attribute in the DICE expressions. □

6.3.2.1 *Valid CQL Queries*

Not all possible combination of CQL operations can be considered a valid input to our CQL simplification process. We define *well-formed* CQL queries as follows.

Definition 6.3.5. (**Well-formed CQL query**). A *well-formed* query q is a sequence of CQL operations that satisfies the following conditions: (i) There is at most one SLICE operation over each dimension D or measure M; (ii) Every DRILL-DOWN operation over a dimension D is preceded by at least one ROLL-UP operation over the same dimension; (iii) There is no DICE operation including conditions over measure values, in-between a ROLL-UP and a DRILL-DOWN. □

The reason why we prevent DICE operations including conditions over measure values in-between a ROLL-UP and/or DRILL-DOWN, is that we want to avoid storing additional information, in particular the computation trace. We illustrate this situation in Example 6.3.2.

Example 6.3.2. (Condition (iii) in Definition 6.3.5) Consider the query:

> **Query 2:** Total asylum applications per month by sex, time, age, citizenship, destination, and application type, only for years where the total amount of applications is less than 100.

The program below answers Query 2.

```
$C1:=ROLLUP(migr_asyapp, timeDim, year);
$C2:=DICE($C1, obsValue < 100);
$C3:=DRILLDOWN($C2,timeDim, month);
```

First, a ROLL-UP aggregates measures up to the Year level on the Time dimension. A DICE is then applied to keep cells that satisfy the restriction over the measure value (that is, a *previously aggregated* measure). As results are expected at the Month level, we would need to keep track of the cells in the cuboid at the Month level, that roll up to

the years that satisfy the DICE condition at the Year level. Condition (iii) in Definition 6.3.5 prevents this. □

To summarize, the following patterns define the valid CQL queries, using regular expression notation. DICE_l and DICE_m denote DICE operations applied only over level attribute or measure values, respectively.

P1: $(\text{SLICE}^*|\text{DICE}^*|\text{ROLL-UP}^*)^+$

P2: $(\text{SLICE}^*|\text{ROLL-UP}^+|\text{DRILL-DOWN}^+|\text{DICE}_l^+)^+$

P3: $(\text{SLICE}^*|\text{ROLL-UP}^+|\text{DRILL-DOWN}^+|\text{DICE}_l^*)^+\text{DICE}_m^+$

6.4 CQL SIMPLIFICATION

As we have already mentioned, CQL is aimed at being used by non-experts. Thus, even well-formed CQL queries may include unnecessary operations that should be eliminated. Further, operations can be reordered to reduce the size of the cuboid as early as possible. Based on the properties defined in Section 6.3.1, we define the following set of rewriting rules (we also indicate the properties on which the rules are based).

Rule 1. Remove all the ROLL-UP or DRILL-DOWN operations with the same origin and target level (Property 6.3.3).

Rule 2. Find sequences of ROLL-UP and/or DRILL-DOWN operations over the same dimension d that do not have a DICE_L operation in-between, where L is a level in d. Find the last level L_d in the sequence. If L_d differs from the bottom level of d (call it $L_{0\,d}$), replace the group of operations with a single ROLL-UP from $L_{0\,d}$ to L_d. Otherwise, remove all the operations in the group (Properties 6.3.1, 6.3.2, and 6.3.3).

Rule 3. If there is a SLICE operation over a dimension d, and no DICE operation that considers level members of d, move the SLICE operation to the beginning of the query; otherwise move it to the end.

Rule 4. If there is a SLICE operation over a measure M, and no DICE operation that mentions M, move the SLICE to the beginning of the query; otherwise move it to the end.

Rule 5. If there is a SLICE operation over a dimension d, a sequence of ROLL-UP and/or DRILL-DOWN operations over d, and there is no DICE operation that mentions levels of d, remove all the ROLL-UP and DRILL-DOWN operations, and keep only the SLICE operation (Property 6.3.4).

Let q_{in} and q_{out} be the CQL query before and after the simplification process, respectively. Then, q_{out} satisfies the following properties (proofs available in Appendix B).

Property 6.4.1. If there is no DICE operation in q_{in}, there is at most one ROLL-UP, and no DRILL-DOWN operation, for each Dimension d in q_{out}.

Property 6.4.2. SLICE operations are either at the beginning or at the end of q_{out}, but not in the middle.

We conclude this section presenting an example of the simplification process.

Example 6.4.1. (CQL simplification)

> **Query 3:** Total asylum applications per year (by sex, time, age, destination, and application type)

The following CQL query answers Query 3.

```
$C1:= ROLLUP (migr_asyappctzm, timeDim, year);
$C2:= ROLLUP ($C1,destinationDim,government);
$C3:= ROLLUP ($C2,citizenshipDim,continent);
$C4:= DRILLDOWN ($C3,destinationDim,country);
$C5:= SLICE ($C4,citizenshipDim);
```

The application of Rule 2 to $C2 and $C4 replaces them with a single Roll-up on dimension Destination, from level Country to itself, so it can be removed, according with Rule 1. By Rule 3, operation $C5 is moved to the beginning of the query. Finally, by Rule 5, $C3 can be removed, as operation $C5 performs a Slice over the same dimension. The result of the process is:

```
$C1:= SLICE (migr_asyappctzm,citizenshipDim);
$C2:= ROLLUP ($C1, timeDim, year);
```

□

6.5 FROM CQL TO SPARQL OVER QB4OLAP

The next step in the process is the translation of CQL queries into SPARQL expressions over QB4OLAP cubes. We start by showing that in a QB4OLAP representation, adjacent cuboids that satisfy the order relation presented in Definition 6.2.6, can be computed using SPARQL queries. Combining this rationale with the semantics of each CQL operation presented in Section 6.3, we are able to provide algorithms that implement each operation as SPARQL queries over QB4OLAP-based RDF data cubes.

Before presenting the algorithms, we discuss on the structure of the generated SPARQL queries, and introduce auxiliary functions. SPARQL queries may return results in different formats. In particular `SELECT` queries return a table of values, while `CONSTRUCT` queries return a graph (i.e., a set of triples). Since each operation returns a cuboid in a certain cube instance, and since cuboids in QB4OLAP are RDF graphs, it is clear that algebra operations should be implemented in SPARQL using `CONSTRUCT` queries. However, since SPARQL 1.1 does not allow us to compute aggregations in a `CONSTRUCT` query, we will use subqueries for this computation, thus producing two queries: (a) an inner `SELECT` query to compute aggregations, and (b) an outer `CONSTRUCT` query that generates the graph using the computed results. Notice that the inner query is responsible for the actual computation of values, while the outer query just generates the output as a graph.

Also, to improve the clarity of the presentation of all the algorithms in this section, we use the auxiliary functions defined in Table 3. We

also use an abstract representation of a SPARQL query, where for each query: (a) *queryType* can be `SELECT` or `CONSTRUCT`; (b) *resultFormat* represents the set of variables and expressions included in the `SELECT` clause, or the set of BGPs included in the `CONSTRUCT` clause, depending on the type of the query; (c) *grPatterns* represents the set of graph patterns in the `WHERE` clause; (d) *subQueries* represents the set of subqueries in the `WHERE` clause; (e) *filter* represents a `FILTER` clause; (f) and *groupBy* represents the set of variables included in the `GROUP BY` clause. We assume that each of these parts can be accessed and modified, and we use the dot notation (".") to access them. We also consider a function $add()$, such that $add(s)$ appends s to a particular part of a query. For example, given a query q such that $q.queryType$ ==="SELECT", $q.resultFormat.add(v)$ adds the variable v to the `SELECT` clause of q.

Table 3: Auxiliary functions

Function signature	Description
newVarName()	Generates and returns a unique SPARQL variable name.
val(v)	Returns the value stored in variable v
levels(s)	Returns all the levels in a schema s (i.e., all the values of ?l that satisfy s `qb:component` ?c. ?c `qb4o:level` ?l)
getLevel(s,d)	Returns the only level l that corresponds to dimension d in the schema s (i.e. the only value of ?l that satisfies s `qb:component` ?c. ?c `qb4o:level` ?l. ?h `qb4o:hasLevel` ?l. ?h `qb4o:inDimension` ?d)
levelsPath(l_o,l_d,d)	Returns a path of levels from level l_o to l_d in dimension d
getRollup(l_c,l_p,d)	Returns the predicate that implements the RUP function from level l_c to level l_p in dimension d
measures(s)	Returns all the measures in a schema s (all the values of ?m that satisfy s `qb:component` ?c. ?c `qb:measure` ?m)
aggFunction(m,s)	Returns the aggregation function of measure m (all the values of ?f that satisfy s `qb:component` ?c. ?c `qb:measure` ?m ;`qb4o:aggregateFunction` ?f).

6.5.1 *Computing Adjacent Cuboids*

Algorithm 2 performs the computation of adjacent cuboids as presented in Definition 6.2.6. It takes as input a cuboid instance Cb_1 represented in QB4OLAP, and a level L_p such that $L_p \notin \mathcal{V}_{Cb_1}$ and $\exists L_c \in \mathcal{V}_{Cb_1}$, where $L_c \rightarrow L_p$ holds in a dimension D_k; the algorithm produces a SPARQL query that computes a cuboid instance Cb_2 that satisfies $Cb_1 \preceq Cb_2$ (i.e. $\mathcal{V}_{Cb_1} - \mathcal{V}_{Cb_2} = \{L_c\}$ and $\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1} = \{L_p\}$).

We now present the algorithms that implement each CQL operation as a SPARQL query over QB4OLAP-based RDF data cubes.

6.5.2 *IPO Operations as SPARQL Queries*

According to Definition 6.2.7, a cube instance is the lattice $\{CB, \preceq\}$ where CB is the set of all possible cuboids that adhere to a cube schema, and $\preceq$ is the order relation between adjacent cuboids in CB. As stated by Definition 6.2.5, for each pair of adjacent cuboids $Cb_1 \preceq Cb_2$, each cell in Cb_2 can be computed from the cells in Cb_1.

Algorithm 2 Generates a SPARQL query that computes Cb_2, such that $Cb_1 \preceq Cb_2$

Input: Cb_1, a cuboid instance in QB4OLAP; L_p, a level such that $L_c \in \mathcal{V}_{Cb_1}$ and $L_c \rightarrow L_p$ in the lattice $(\mathcal{L}, \rightarrow)$ of a dimension D_k
Output: $query$, a SPARQL CONSTRUCT query that represents a cuboid instance Cb_2 that satisfies $Cb_1 \preceq Cb_2$

```
1: function CreateAdjacentCuboidInstance
2:     L_c ← getLevel(D_k)
3:     for all L ∈ Levels = levels(Cb_1) do
4:         if L ≠ L_c then
5:             newVar(l_i)
6:             query.resultFormat.add(?id , l, val(L_i))
7:             sq.resultFormat.add(val(L_i))
8:             sq.grPattern.add(?i , l, val(L_i))
9:             sq.groupBy.add(val(L_i))
10:        end if
11:    end for
12:    for all m ∈ M = measures(Cb_1) do
13:        f = aggFunction(m)
14:        newVar(m_i); newVar(ag_i)
15:        query.resultFormat.add(?id , m, val(ag_i))
16:        sq.resultFormat.add(f(val(m_i)) AS ag_i)
17:        sq.grPattern.add(?i , m, val(m_i))
18:    end for
19:    newVar(lm_i),newVar(plm_i)
20:    rup ← getRollup(L_c, L_p, D_k)
21:    sq.grPattern.add(?i , val(L_c), val(lm_i))
22:    sq.grPattern.add(val(plm_i), qb4o:memberOf, val(L_p))
23:    sq.grPattern.add(val(lm_i),rup,val(plm_i))
24:    sq.groupBy.add(plm_i)
25:    sq.resulFormat.add(plm_i)
26:    query.resultFormat.add(?id , L_p, val(plm_i))
27:    query.grPattern.set(sq)
28:    return query
29: end function
```

Therefore, starting from the bottom cuboid in the lattice, which is the cuboid instance whose cells are members of the bottom levels in each dimension of the schema, all the possible cuboids that form the cube instance can be computed incrementally.

To compute the ROLL-UP operation over a cuboid Cb_{in} and a dimension d, it suffices to start at Cb_{in}, and navigate the cube lattice visiting adjacent cuboids that differ only in the level associated to dimension d, until we reach a cuboid Cb_{out} that has the desired level in that dimension (note that this path is unique).

Remark 2. It is not necessary to compute all the cuboids in the path, it suffices to compute the target cuboid. To do so, we must add all the triples needed to traverse the dimension hierarchy up to the target level, and aggregate measure values up to this level. □

As already mentioned, two SPARQL queries are needed: an inner query q_{in} that traverses the dimension hierarchy and computes aggregate values using `GROUP BY`, and an outer one, called q_{out} that builds triples based on the values computed in q_{in}.

Algorithm 3 builds both queries simultaneously, using the add function. Lines 2 and 3 state the query type for each query. Line 4 states that generated observations belong to the dataset $newDS$. Lines 6 through 12 project the members of each level in the schema into the result of both queries, also adding triples to the `WHERE` clause of the

inner query, and adding the variables that represent the level members to the GROUP BY clause, also in the inner query. Lines 13 through 19 do the same for measures. In Lines 14 and 18, f represents the SPARQL function corresponding to the aggregate function for each measure, and $f(val(m_i))$ is the string that should be included to calculate the aggregated value (e.g SUM(?m) if $val(m_i)$ =?m). Lines 20 to 33 add the triples needed to navigate the dimension hierarchy. Line 22 retrieves the RDF property that implements the RUP relation for each step in the path. Line 24 adds to the inner query, a triple that associates the level member with the observation (only for the base level L_c in dimension d); Line 27 adds a triple that allows us to state to which level the level member belongs, and line 29 retrieves the parent level member of the current level applying the RUP function obtained in Line 22 (this is done for all the levels in the path except for the target level L_{out}. When level L_{out} is reached Lines 31 and 32 add the target level to the GROUP BY and SELECT clauses of the inner query, respectively, while Line 33 adds the target level to the outer query result. Finally, Line 34 sets the inner query as a subquery within the WHERE clause of the outer query, which is returned in Line 35. For clarity, we have omitted the clause that generates the expression that binds variable ?*newObs* to a dynamically generated IRI from the values in the observation.

Example 6.5.1. The SPARQL query generated by Algorithm 3 for RollUp(Asylum_application, Citizenship, Continent) is:

```
CONSTRUCT {
  ?newObs a qb:Observation .
  ?newObs qb:dataSet queries:ejRollup.
  ?newObs sdmxd:refPeriod ?time .
  ?newObs pr:sex ?sex .
  ?newObs pr:geo ?geo.
  ?newObs pr:age ?age.
  ?newObs pr:asyl_app ?apptype.
  ?newObs sc:continent ?citContinent .
  ?newObs sdmxm:obsValue ?sumApp }
FROM <http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp_clean>
  FROM <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyappctzmQB4O13>
WHERE {
  SELECT ?newObs ?time ?sex ?geo ?age ?apptype ?citContinent
           (SUM(xsd:integer(?m)) AS ?sumApp)
  WHERE{
    ?obs qb:dataSet eurostatdt:migr_asyappctzm.
    ?obs sdmxd:refPeriod ?time .
    ?obs pr:sex ?sex ; pr:geo ?geo.
    ?obs pr:age ?age; pr:asyl_app ?apptype.
    ?obs sdmxm:obsValue ?m .
    ?obs pr:citizen ?citizen .
    ?citizen qb4o:memberOf pr:citizen.
    ?citizen sc:inContinent ?citContinent.
    ?citContinent qb4o:memberOf sc:continent.
    bind (iri(concat('http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp',
    md5(concat(str(?time), str(?sex), str(?geo), str(?age), str(?apptype),
    str(?citContinent))))) as ?newObs)
  }
  GROUP BY ?newObs ?time ?sex ?geo ?age ?apptype ?citContinent
}
```

□

Algorithm 3 Generates a SPARQL query that implements a ROLL-UP in QB4OLAP

Input: C_{in}, cuboid instance in QB4OLAP, where $\mathcal{V}_{C_{in}}$ is the set of levels of the cuboid, and d_r is the data set that represents the cuboid instance; L_{out} is a level such that $L_c \in \mathcal{V}_{C_{in}}$ and $L_c \rightarrow^* L_{out}$ in the lattice $(\mathcal{L}, \rightarrow)$ of a dimension $\mathcal{D}$.

Output: q_{out}, a SPARQL CONSTRUCT query that represents a cuboid instance, C_{out} = ROLL-UP($C_{in}, \mathcal{D}, L_{out}$).

```
1: function CreateRollUpQuery(C_in, D, L_out)
2:     q_out.queryType = 'CONSTRUCT'
3:     q_in.queryType = 'SELECT'
4:     q_out.grPatterns.add(?newObs , qb:dataSet , newDS)
5:     L_c ← getLevel(C_in, d)
6:     for all L ∈ Levels = levels(C_in) do
7:         if L ≠ L_c then
8:             L_i ←newVar()
9:             q_in.grPatterns.add(?obs , l, val(L_i))
10:            q_in.groupBy.add(val(L_i))
11:            q_in.resultFormat.add(val(L_i))
12:            q_out.resultFormat.add(?newObs , l, val(L_i))
13:        end if
14:    end for
15:    for all m ∈ M = measures(C_in) do
16:        f ← aggFunction(m)
17:        m_i ←newVar()
18:        ag_i ←newVar()
19:        q_in.grPatterns.add(?obs , m, val(m_i))
20:        q_in.resultFormat.add(f(val(m_i)) AS ag_i)
21:        q_out.resultFormat.add(?newObs , m, val(ag_i))
22:    end for
23:    for all (L_i, L_j) ∈ path = levelsPath(L_c, L_out, D) do
24:        lm_i ←newVar()
25:        rup ← getRollup(L_i, L_j, D)
26:        if L_i = L_c then
27:            q_in.grPatterns.add(?obs , val(L_i), val(lm_i))
28:        else
29:            plm_i ←newVar()
30:            q_in.grPatterns.add(val(plm_i), qb4o:memberOf, val(L_i))
31:            if L_i ≠ L_out then
32:                q_in.grPatterns.add(val(lm_i),rup,val(plm_i))
33:            else
34:                q_in.groupBy.add(plm_i)
35:                q_in.resulFormat.add(plm_i)
36:                q_out.resultFormat.add(?newObs , l_p, val(plm_i))
37:            end if
38:        end if
39:    end for
40:    q_out.subqueries.add(q_in)
41:    return q_out
42: end function
```

In Section 6.4 we have discussed that it is possible to transform a DRILL-DOWN operation into a ROLL-UP, therefore there is no need to provide an specific implementation for DRILL-DOWN in QB4OLAP.

6.5.3 *IGO Operations as SPARQL Queries*

IGO operations take as input a cuboid in a cube instance, induce a new cube instance, and return a cuboid over this newly induced lattice of cuboids. In some cases (e.g. SLICE) the operations also affect the schema of the cube before producing the cube instance.

The DICE operation takes as input a cuboid in a cube instance, and a boolean expression ϕ over measure values and/or attribute values, and returns a cuboid in a new cube instance keeping only the cells from the input cuboid that satisfy ϕ. The implementation of this operation in SPARQL selects the `qb:Observations` that satisfy ϕ. Since measures and attributes are literals, conditions over them can be implemented as `FILTER` clauses. Also, conditions that only involve equality can be efficiently implemented via graph patterns, restricting the result of the query to observations that are related to a particular level member. Inequalities over level members are represented as `FILTER` clauses. Algorithm 4 generates the SPARQL implementation of the DICE operation, which is also based in an inner query which performs the filter, and an outer query that produces the results.

Example 6.5.2. (SPARQL query that implements the DICE operation) The operation:
DICE (Asylum_application, ((201303<=month<=201307) $\vee$ (#applications >80) $\wedge$ Destination.country.countryName = Belgium)) is implemented in SPARQL as follows.

```
CONSTRUCT { ?obs ?p ?v}
FROM <http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp_clean>
  FROM <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyappctzmQB4013>
WHERE{
  SELECT ?o ?p ?v
  WHERE{
     ?obs a qb:Observation;
     ?obs qb:dataSet eurostatdt:migr_asyappctzm.
     ?obs sdmxd:refPeriod ?time.
     ?obs sdmxm:obsValue ?m.
     ?obs pr:geo ?lm1 .
     ?lm1 sc:countryName "Belgium"@en .
     ?time sc:yearMonthNum ?timeMonthNum.
     ?obs ?p ?v.
  FILTER (?timeMonthNum >= 201303 && ?timeMonthNum <= 201307&& xsd:integer(?m)>80)
  }
}
```

□

The SLICE operation comes in two flavors. In one case, it takes as input a cuboid instance and a dimension. In the other, it takes a cuboid instance and a measure. In both cases the implementation of this operation in QB4OLAP requires the creation of a new schema, where the input dimension or the measure are removed. In the case where the operation receives a dimension (SLICE(C_{in}, D)), the new cube instance

Algorithm 4 Generates a SPARQL query that implements a DICE in QB4OLAP

Input: C_{in}, a cuboid instance in QB4OLAP, with its corresponding set of levels $\mathcal{V}_{C_{in}}$, a boolean condition ϕ over the measures in $\mathcal{M}_{in}$, and/or over the attributes of the levels in $\mathcal{V}_{C_{in}}$; d_r is the data set that represents the cuboid instance.
Output: q_{out}, a SPARQL CONSTRUCT query that represents a cuboid instance. C_{out} = DICE(C_{in}, ϕ)

1: **function** CREATEDICEQUERY(C_{in}, ϕ)
2: q_{out}.queryType = 'CONSTRUCT'
3: q_{in}.queryType = 'SELECT'
4: l_{vars} = []
5: m_{vars} = []
6: $bgps_{filter}$ = []
7: q_{out}.grPatterns.add(?newObs , qb:dataSet , $newDS$)
8: **for all** $L \in Levels = levels(C_{in})$ **do**
9: $L_i \leftarrow$newVar()
10: $l_{vars}[l] = L_i$
11: q_{in}.grPatterns.add(?obs , l, val(L_i))
12: q_{in}.resultFormat.add(val(L_i))
13: q_{out}.resultFormat.add(?newObs , l, val(L_i))
14: **end for**
15: **for all** $m \in M = measures(C_{in})$ **do**
16: $m_i \leftarrow$newVar()
17: $m_{vars}[m] = m_i$
18: q_{in}.grPatterns.add(?obs , m, val(m_i))
19: q_{in}.resultFormat.add((val(m_i))
20: q_{out}.resultFormat.add(?newObs , m, val(ag_i))
21: **end for**
22: $treeCond \leftarrow$parseCondition(ϕ)
23: procCondition($treeCond, l_{vars}, m_{vars}, bgps_{filter}, cond_{filter}$)
24: **for all** $bgp \in bgps_{filter}$ **do**
25: q_{in}.grPatterns.add(bgp)
26: **end for**
27: q_{in}.filter.add($cond_{filter}$)
28: q_{out}.subqueries.add(q_{in})
29: **return** q_{out}
30: **end function**

Input: $tree$ is a bin tree representing a boolean condition ϕ. Internal nodes represent operators (AND,OR,NOT). Leaves represent conditions over level attributes or measure values.l_{vars} is the set of level members, m_{vars} is the set of measure values
Output: q_{out} is a SPARQL CONSTRUCT query that represents a cuboid instance C_{out} = DICE(C_{in}, ϕ)

31: **function** PROCCONDITION($tree, l_{vars}, m_{vars}, bgps, filter$)
32: **if** $tree = leaf$ **then**
33: **if** $tree.type$ = "LEVEL" **then**
34: $v \leftarrow$findVariable($tree.element, l_{vars}$)
35: $la \leftarrow$newVar()
36: bgps.add(v , $tree.level$, la)
37: filter = (la, $tree.oper$, $tree.value$)
38: **else**
39: $v \leftarrow$findVariable($tree.element, m_{vars}$)
40: filter = (v, $tree.oper$, $tree.value$)
41: **end if**
42: **else**
43: procCondition($tree.left, l_{vars}, m_{vars}, bgps_{left}, filter_{left}$)
44: procCondition($tree.right, l_{vars}, m_{vars}, bgps_{right}, filter_{right}$)
45: bgps.add($bgps_{left}$)
46: bgps.add($bgps_{right}$)
47: filter.add($filter_{left}$,$tree.oper$,$filter_{right}$)
48: **end if**
49: **end function**

C_{out} is computed as (ROLL-UP(C_{in}, D, All)). The SPARQL query generation algorithm is straightforward, and we omit it here to avoid redundancy.

6.5.4 *Putting all Together*

By combining the algorithms presented above, we can produce a single SPARQL expression that implements a CQL query.

> **Query 4:** Total asylum applications per year submitted by Asian citizens to France or United Kingdom, where applications count > 5000 (by sex, time, age, citizenship country, and destination country)

```
$C1 := ROLLUP (migr_asyappctzm, citizenshipDim,continent);
$C2 := ROLLUP ($C1, timeDim, year);
$C3 := DICE ($C2, (citizenshipDim|continent|continentName = "Asia"));
$C4 := DICE ($C3, ( obsValue > 5000 AND
                  (destinationDim|country|countryName = "France")
                  OR (destinationDim|country|countryName = "United_Kingdom")));
```

Example 6.5.3. (CQL to SPARQL translation)

The SPARQL query below, produced by our translation algorithms, implements Query 4. It contains a subquery, where aggregated values are computed, and an outer query where the `FILTER` conditions that implement the DICE operations are applied. Lines 10 through 12 implement the first roll-up (C1). Variable ?lm1 will be instantiated with each member of the Country level in the Citizen dimension hierarchy, related to an observation ?o (lines 10 and 11). Then, we navigate the hierarchy up to the level Continent, using the rollup property `sc:inContinent` (line 12). The variable ?plm1 will contain the continent corresponding to the country that instantiates ?lm1. It is placed in the `SELECT` clause of the inner query (line 5), in the `GROUP BY` clause of the inner query (line 25), and in the result of the outer query (line 1). Analogously, the navigation that corresponds to the ROLLUP in C2 is performed in lines 13 through 15. Lines 16 to 19 instantiate the level members of the remaining dimensions in the cube, which are also added to the `GROUP BY` clause, and to the `SELECT` clause of the inner and outer query. Line 9 retrieves the value of the measure in each observation, and the SUM aggregate function computes ?ag1 in line 5. The aggregated value is added to result of the outer query (line 1). In this case, measure values are converted to integer before applying the SUM function due to format restrictions of Eurostat data. Finally, to implement the DICE operation in statement C3, we need to obtain the name of each continent (line 20) and then use a `FILTER` clause to keep only the cells that correspond to "Asia" (line 22). The DICE operation in statement C4 is split. The restriction on country names is implemented adding lines 23 and 24 to the `FILTER` clause (country names are retrieved in line 21), while the restriction on the measure values must be performed *after* the aggregation, and is implemented by the `FILTER` clause of the outer query (line 26). The `REGEX` clause evaluates a regular expression. □

```
SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 ?ag1
FROM loc-ins:migr_asyapp_clean
FROM loc-sch:migr_asyappctzmQB4O13
WHERE {
 { SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 (SUM(xsd:integer(?m1)) as ?ag1)
 WHERE {
   ?o a qb:Observation .
   ?o qb:dataSet data:migr_asyappctzm .
   ?o sdmx-meas:obsValue ?m1 .
   ?o pr:citizen ?lm1 .
   ?lm1 qb4o:memberOf pr:citizen . ?lm1 sc:inContinent ?plm1 .
   ?plm1 qb4o:memberOf sc:continent .
   ?o sdmx-dim:refPeriod ?lm2 .
   ?lm2 qb4o:memberOf sdmx-dim:refPeriod . ?lm2 sc:inYear ?plm2 .
   ?plm2 qb4o:memberOf sc:year .
   ?o pr:geo ?lm3 .
   ?o pr:sex ?lm4 .
   ?o pr:age ?lm5 .
   ?o pr:asyl_app ?lm6 .
   ?plm1 sc:continentName ?plm11 .
   ?lm3 sc:countryName ?lm31 .
   FILTER ( REGEX (?plm11,"Asia" , "i") &&
         (REGEX (?lm31,"France" , "i") ||
          REGEX (?lm31,"United_Kingdom" , "i")) ) }
GROUP BY ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
} FILTER ( ?ag1 > 5000) }
```

6.6 SPARQL QUERIES IMPROVEMENT

We have shown that we can automatically produce naïveSPARQL queries to implement CQL queries over QB4OLAP. To improve the performance of such queries, we adapted existing techniques to the characteristics of MD data in general, and of QB4OLAP representation. In this section we first provide a brief background on SPARQL queries optimization, and then present how we adapted improvement techniques to our setting.

6.6.1 *SPARQL Queries Optimization*

In [26] the authors focus on the static analysis of SPARQL queries, in particular those that contain the `OPTIONAL` operator, which is not used in our SPARQL queries. Tsialimanis et. al [40] propose a heuristic approach to the optimization for SPARQL joins, based on the selectivity of graph patterns. Loizou et al.[27] propose a set of heuristics to improve the performance of SPARQL queries, while Vesse [46] collects recommendations on how to improve the performance of queries over Jena triplestore.

All of these are general-purpose studies. On the contrary, we can take advantage of the characteristics of our data model (e.g., the OLAP operators, and the information provided by QB4OLAP metadata) to define improvement rules that may not apply to a more generic scenario.

6.6.2 *SPARQL Improvement Strategy*

First, we adapted to our setting the heuristics proposed by Loizou et al.[27] to improve the performance of SPARQL queries. We next indicate the heuristics, and how we use some of them.

H1 - Minimize optional graph patterns. This heuristic is based on the fact that the introduction of `OPTIONAL` clauses leads to PSPACE-completeness of the SPARQL evaluation problem[32]. Note that the SPARQL queries we produce do not use the OPTIONAL operator.

H2 - Use named graphs to localize SPARQL graph patterns. This heuristic is based on the fact that there is a trivial positive correlation between the performance of a query and the number of triples it is evaluated against. To take advantage of this, we organize QB4OLAP data into two named graphs. The *schema* graph stores the schema and dimension members, while the *instances* graph stores only observations. Due to MD data nature, in most cases the size of the instances graph will be considerably bigger than the schema graph. With this organization we can ensure a bound on the number of graph patterns over the instance graph, which will be at most $2+|D|+|M|$, being D the set of dimensions, and M the set of measures.

H3 - Reduce intermediate results. This proposes to reduce intermediate results, replacing connected triple patterns with path expressions. This kind of patterns do not occur in our queries, and therefore this heuristic cannot be applied. Recall from Section 4.2, that QB4OLAP proposes to use a different predicate to represent each RUP relationship between level members, instead of using a single predicate like `skos:narrower` in QB.

H4 - Reduce the impact of cartesian products. This only applies when rows in the result differ at most in one value. In those cases, it is suggested to collapse sets of almost identical rows in one, and to use aggregate functions. Since in the result of an OLAP query, each row represents exactly one point in the space (so there is no redundancy), this heuristic cannot be applied to our problem.

H5 - Specifying alternative URIs. Proposes to transform `FILTER` clauses with disjunction (`||`) of equality constraints, using either the `UNION` of patterns, or a `VALUES` expression. In Example 6.6.1 we show these transformations. Since the reported results are not conclusive on which of these strategies leads to better performant queries, we decided to try them both (see Chapter 7).

Example 6.6.1. (Rewriting `FILTER` clauses with disjunction of equality constraints) The queries below show how `FILTER` clauses with disjunction of equality constraints can be replaced using **H5**. □

```
SELECT ?a
WHERE {
  ?a <predicate> ?b .
  FILTER (?b = value1 || ?b = value2)}
#rewriting FILTER using UNION
SELECT ?a
WHERE {
  { ?a <predicate> value1 }
  UNION
```

```
  { ?a <predicate> value2 } }
#rewriting FILTER using VALUES
SELECT ?a
WHERE {
  ?a <predicate> ?b .
  VALUES ?b (value1 value2)}
```

As our second strategy, we considered the recommendations in [46], namely: (i) Split conjunctive FILTER equality constraints into a cascade of FILTER equality constraints; (ii) Replace a FILTER equality constraint that compares a variable and a constant, with a graph pattern. The first recommendation may help the query processor to push FILTER constraints down in the query tree, while the second allows the query processor to use indexes to select the patterns that match the criteria.

Example 6.6.2. (Improving FILTERs) The first expression below, returns the values of ?a that are associated via <predicate> with values greater that 'value1'. We then apply the strategies mentioned above, i.e., splitting and rewriting.

```
SELECT ?a
WHERE {
  ?a ?b ?c .
  FILTER (?b = <predicate> && ?c > value1)}
#splitting FILTER conjunction
SELECT ?a
WHERE {
  ?a ?b ?c .
  FILTER (?b = <predicate>)
  FILTER (?c > value1)}
#replacing FILTER equality constraints over constants by BGPs
SELECT ?a
WHERE {
  ?a <predicate> ?c.
  FILTER (?c > value1)}
```

□

The next example shows the result of applying the improvement strategies to the query of Example 6.5.3.

Example 6.6.3. (SPARQL queries improvement) The application of **H2** organizes graph patterns in the inner query in two GRAPH clauses: one that corresponds to patterns on the instance graph (lines 9 to 19), and another on the schema graph (lines 20 to 30). Applying **H5**, the FILTER clause on country names is replaced by a VALUES clause (line 29). Filter clauses are split, and the FILTER clause on continent name is replaced by a graph pattern (line 24). □

```
SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 ?ag1
FROM NAMED loc-ins:migr_asyapp_clean
FROM NAMED loc-sch:migr_asyappctzmQB4013
WHERE {
  { SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 (SUM(xsd:integer(?m1)) as ?ag1)

  WHERE {
    {GRAPH loc-ins:migr_asyapp_clean
      {?o a qb:Observation .
      ?o qb:dataSet eurostatdt:migr_asyappctzm .
      ?o sdmx-meas:obsValue ?m1 .
      ?o pr:citizen ?lm1 .
```

```
    ?o sdmx-dim:refPeriod ?lm2 .
    ?o pr:geo ?lm3 .
    ?o pr:sex ?lm4 .
    ?o pr:age ?lm5 .
    ?o pr:asyl_app ?lm6 .
    }}.
   {GRAPH loc-sch:migr_asyappctzmQB4O13
    {?lm1 qb4o:memberOf pr:citizen .
    ?lm1 sc:inContinent ?plm1 .
    ?plm1 qb4o:memberOf sc:continent .
    ?plm1 sc:continentName "Asia" .
    ?lm2 qb4o:memberOf sdmx-dim:refPeriod .
    ?lm2 sc:inYear ?plm2 .
    ?plm2 qb4o:memberOf sc:year .
    ?lm3 sc:countryName ?lm31 .
    VALUES ?lm31 {"France"@en "United_Kingdom"@en}
  }}}
  GROUP BY ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
} FILTER (?ag1 > 5000) }
```

Our third, and final, strategy, is based on Stocker et. al [38]. This optimization is based on graph pattern selectivity. The idea behind this approach is to reduce intermediate results by first applying the most selective patterns. This requires to keep estimates on the selectivity of each pattern. In our case, we take advantage of MD data characteristics to estimate the selectivity of patterns beforehand: Since typically, RUP relationships between level members are functions, each level member has exactly one parent on the level immediately above. Thus, for each pair of levels L_i and L_j such that $L_i \rightarrow L_j$ in a hierarchy H, $|L_i| \geqslant |L_j|$. Moreover, in most cases $|L_i| > |L_j|$ holds. Based on the above, we define alternative *ordering criteria* (OC) for the graph patterns.

- **Ordering Criteria 1 (OC1)** - For each dimension appearing in the query, apply first the patterns that corresponds to higher levels.
- **Ordering Criteria 2 (OC2)** - For each dimension, apply OC1. Then, reorder dimensions using the following criteria: First consider dimensions with conditions that fix a certain member, then dimensions with conditions that restrain to a range of members, and then the other dimensions.
- **Ordering Criteria 3 (OC3)** - For each dimension apply OC1. Then, reorder dimensions according to OC2. If more than one dimension satisfy any of the criteria in OC2, then use the number of members in the highest level reached for each dimension to decide the relative order between these dimensions. For example: If dimension A and dimension B fix members a and b at levels l_A and l_B respectively, and $|l_A| \geqslant |l_B|$, then dimension A goes before dimension B.

Example 6.6.4. (Reordering triple patterns) We show the result of applying OC2 to reorder the triple patterns on the schema graph in Example 6.6.3. Triples in lines 2 through 5 correspond to the Citizenship dimension, lines 7 and 8 correspond to Destination dimension, and lines 10 through 12 correspond to the Time dimension. For each dimension, the graph patterns are ordered from higher levels in the hierarchy to lower ones. Then, the relative position of each dimension in the query is altered with respect to the naive query. The Citizenship dimension is considered first since a member of the dimension

is fixed to "Asia". Then we consider the Destination dimension because there is a restriction on members of this dimension ("France" or "United Kingdom"). □

```
GRAPH loc-sch:migr_asyappctzmQB4013 {
 ?plm1 sc:continentName "Asia" .
 ?plm1 qb4o:memberOf sc:continent .
 ?lm1 sc:inContinent ?plm1 .
 ?lm1 qb4o:memberOf pr:citizen .

 ?lm3 sc:countryName ?lm31 .
 VALUES ?lm31 {"France"@en "United_Kingdom"@en}

 ?plm2 qb4o:memberOf sc:year .
 ?lm2 sc:inYear ?plm2 .
 ?lm2 qb4o:memberOf sdmx-dim:refPeriod .}
```

COMPLEXITY: We end this section with some complexity remarks. It has been proved that the evaluation of a SPARQL 1.0 query is NP-complete for the AND-FILTER-UNION fragment of the language[32]. Moreover, the evaluation of queries that only contain UNION, AND operators is already NP-complete, as proved in [36]. Perez et. al [32] also proved that the main source of complexity in SPARQL 1.0 queries is the introduction of the OPTIONAL, which leads to PSPACE-completeness of the evaluation problem. The SPARQL queries we produce, both naïve and improved, avoid the OPTIONAL operator but make an intensive use of two features added to SPARQL in version 1.1: The computation of aggregates (GROUP BY clauses), and subqueries. To the best of our knowledge there are still no theoretical results on the complexity of such queries, and a study of this issue is beyond the scope of this work.

6.7 IMPLEMENTATION

The *QB4OLAP toolkit* is a web application that implements our approach, allowing to explore and query QB4OLAP cubes. It is composed of two modules. The *Explorer module* enables the user to navigate the cube schema, and visualize dimension instances stored in a SPARQL endpoint. Figure 22 presents a screenshot of the explorer module.

The *Querying module* implements the querying processing pipeline presented in Figure 19, where the user starts by writing a CQL query. Then, the application simplifies this CQL query, displaying the simplified version to the user. Then, the user may choose to generate either a naïve SPARQL query or an improved SPARQL one. The query produced is presented to the user and executed. Results are presented in tabular format. Figure 23 presents a screenshot of the querying module. The QB4OLAP toolkit is available online.[1]

The QB4OLAP toolkit has been entirely developed in Java Script over the Node.js platform[2] using Express[3] web framework. Handle-

1 https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/
2 https://nodejs.org/en/
3 http://expressjs.com/

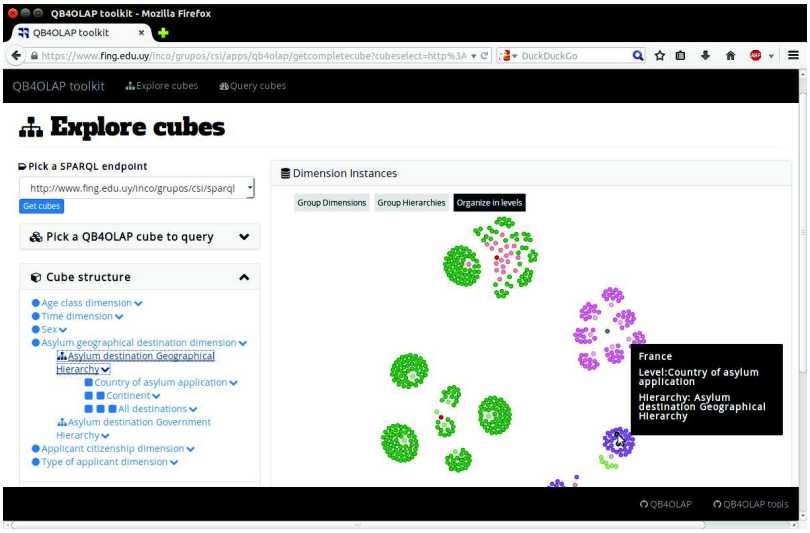


Figure 22: QB4OLAP toolkit: Explorer module

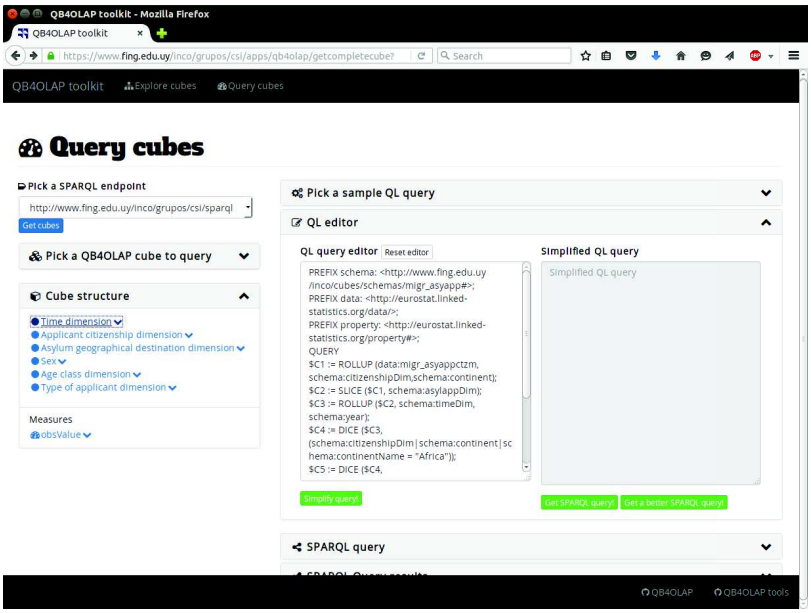


Figure 23: QB4OLAP toolkit: Query module

bars[4], jQuery[5], and D3.js[6] are used to implement the front-end. Virtuoso Open Source version 7 is used for RDF storage and SPARQL backend. The communication with Virtuoso is implemented via HTTP and using JSON format to exchange data. Figure 24 presents the technology stack of QB4OLAP toolkit. Source code is available at GitHub.[7]

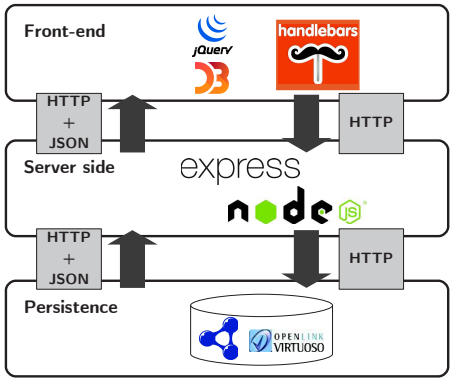


Figure 24: QB4OLAP toolkit: technology stack

6.8 SUMMARY

In this chapter we have presented our approach to perform OLAP queries over SW cubes, which is based on a high-level query language, called CQL. To provide clear semantics, we formalized a MD data model, and showed that this data model can be implemented using QB4OLAP. We present a high-level query simplification heuristic strategy for CQL queries, and proposed algorithms to automatically translate CQL queries into equivalent SPARQL ones over QB4OLAP data cubes. Finally, we presented a set of heuristics to improve the performance of the produced SPARQL queries.

4 http://handlebarsjs.com/
5 http://jquery.com/
6 https://d3js.org/
7 https://github.com/lorenae/qb4olap-tools

7

EVALUATION

"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts".

Arthur Conan Doyle, Sherlock Holmes

In this chapter we report and discuss experimental results aimed at showing that our approach allows to write complex OLAP queries in a high-level query language, which can then be automatically translated into efficient SPARQL queries. Our evaluation goal is twofold: On the one hand, we want to compare our approach against other one(s) that are aimed at querying OLAP cubes on the web. On the other hand, we would like to measure the impact of optimization strategies, and choose combinations that yield to better performance results. We want to remark that to facilitate the replication of the benchmark and experiments presented in this chapter, we provide a virtual machine with the complete experimental environment [1].

7.1 THE SSB-QB4OLAP BENCHMARK

In order to compare our approach against other ones that also propose to query OLAP cubes on the web (e.g., [21], which we discuss in Section 8.2), we need a baseline.

The SSB-QB benchmark [21] is an adaptation of the Star Schema Benchmark (SSB)[30] to evaluate OLAP queries on SW data cubes. It consists of: (i) A representation of the SSB cube schema and dimension instances *using QB and other related vocabularies*; (ii) A representation of SSB facts as QB observations; and (iii) a set of thirteen SPARQL queries over these data. These queries are equivalent to the SSB queries, and aim at representing the most common types of star schema queries in an OLAP setting.

Based on SSB-QB, we built the **SSB-QB4OLAP benchmark**, which consists of: (i) A representation of the SSB cube schema and dimension instances *using QB4OLAP*; (ii) The *same observations than in SSB-QB*; and (iii) A set of thirteen *CQL queries* that are *equivalent to the SSB-QB* queries (and also to the SSB queries). Thus, the SSB-QB4OLAP benchmark allows us to compare our approach against [21] (our first evaluation goal), and also to measure the impact of our improvement strategies (our second evaluation goal). For this, we first translated the CQL queries into SPARQL ones, using the naïve approach described in Section 6.5, and then explored which combination of strategies yields the best query performance, based on several metrics. Next, we introduce the SSB-QB4OLAP Benchmark in detail.

1 https://github.com/lorenae/ssb-qb4olap

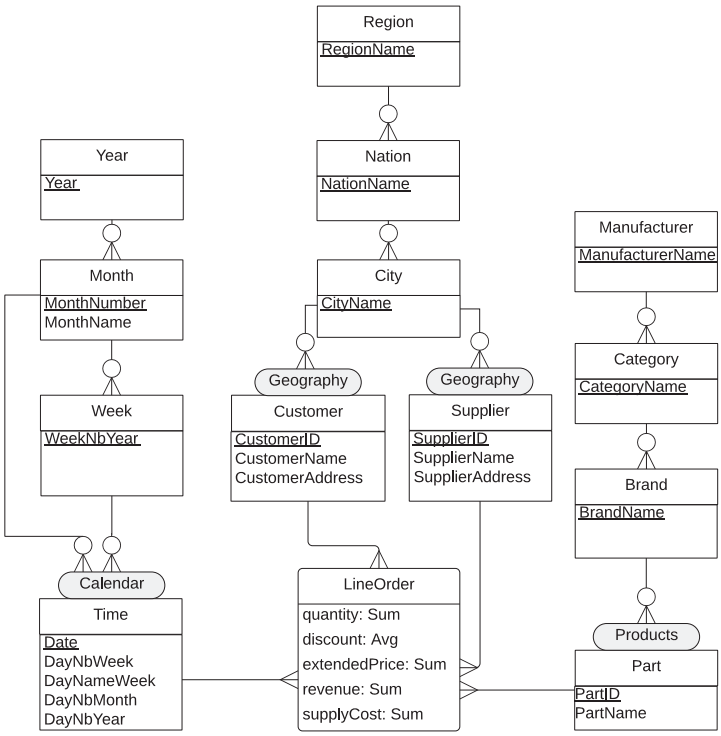


Figure 25: Conceptual schema of the SSB-QB4OLAP cube

7.1.1 *SSB-QB4OLAP Data*

SSB-QB4OLAP represents the SSB data cube at Scale 1, and is organized in three sets of triples that represent the cube schema, the facts (observations), and the dimension instances (i.e., level members, attribute values, and RUP relationships). The cube schema corresponds to the conceptual MD model presented in Figure 25. Each line order contains five measures (quantity, discount, extended price, revenue, and supply cost), which can be analyzed along four dimensions: Time, Part, Customer, and Supplier.

The representation of this conceptual model in QB4OLAP, obtained applying the techniques presented in Section 5.1, consists of about 250 RDF triples. The set of observations is the same as in SSB-QB, and consists of about 132 Million RDF triples, representing 6 Million line orders. Finally, a set of about 2.8 Million RDF triples represent level members, attribute values, and RUP relationships. Table 4 shows the number of members in each level, while Table 5 summarizes data volumes. All these RDF triples are available for querying, at our SPARQL endpoint[2].

7.1.2 *SSB-QB4OLAP queries*

Queries are organized in four so-called *query flights*, which represent different types of usual star-schema queries (functional coverage), and to access different portions of the set of line orders (selectivity coverage). The **first query flight (QF1)** is composed of three queries

2 https://www.fing.edu.uy/inco/grupos/csi/sparql

Dim.	Level	#members	Dim.	Level	#members
Time	Time	2556	Part	Part	2000000
	Week	371		Brand	1000
	Month	84		Cat.	25
	Year	7		Manuf.	5
Custom.	Custom.	30000	Supp.	Supplier	2000
	City	250		City	250
	Nation	25		Nation	25
	Region	5		Region	5

Table 4: SSB-QB4OLAP dataset statistics: dimension members

CONCEPT	SIZE(ITEMS)	SIZE(TRIPLES)	SIZE(MB)
cube schema	4 dimensions, 5 measures	~ 250	0,013
observations	6 Million line orders	132 Million	4430
level members, attribute values, and RUP relationships	see Table 4	2,8 Million	127

Table 5: SSB-QB4OLAP dataset statistics: size

(Q_1-Q_3) that impose restrictions on only one dimension, and quantify the revenue increase that would have resulted from eliminating certain company-wide discounts in a range of products in a certain year. The three queries in the **second query flight (QF2)** (Q_4-Q_6) impose restrictions on two dimensions, and compare revenue for some product classes, for suppliers in a certain region, grouped by more restrictive product classes, along all years. The **third query flight (QF3)** has four queries (Q_7-Q_{10}) that impose restrictions on three dimensions, and aims at providing revenue volume for line order transactions by customer nation, supplier nation, and year within a given region, in a certain time period. The **fourth query flight (QF4)** has three queries (Q_{11}-Q_{13}) and restrictions over four dimensions. It represents a "what if" sequence of operations analyzing the profit for customers and suppliers from America on specific product classes over all years. The 13 queries are presented as CQL queries in Section C.1.

7.2 EXPERIMENTAL SETUP

We ran our evaluation on an Ubuntu Server 14.04.1 LTS, 8x Intel(R) Xeon(R) E5620 @2.40GHz CPU with 4 cores, 32 GB RAM, and 500 GB for local data storage. We use Virtuoso Open source (V 07.20.3214) as RDF store.

The BIBM tool[3] was used to perform *TPC-H power tests*, and in each test suit, a mix of thirteen queries was used with scale 1 and 2 client streams. We also ran a test suit using the query mix from the SSB-QB. We measured the average response time for each query and the following TPC-H metrics for each query mix:

3 http://sourceforge.net/projects/bibm/

- *TPC-H Power*, which measures the query processing power in queries per hour (QphH);
- *TPC-H Throughput* (QphH): the total number of queries executed over the length of the measurement interval;
- *TPC-H Composite*, which is the geometric mean of the previous metrics.

The last metric reflects the query processing power when queries are submitted in a single stream, and the query throughput for queries submitted by multiple concurrent users [39]. In Section 7.2.1 we report the experiments performed to evaluate the SPARQL improvement strategies, and Section 7.2.2 compares our approach with SSB-QB.

7.2.1 *Evaluation of improvement strategies*

We measured the impact on performance, of the improvement strategies presented in Section 6.6, in order to find out which combination of strategies results more beneficial. The strategies are summarized in Table 6, while Table 7 indicates which strategies can be applied to each of the thirteen queries in the benchmark.

S1: Use named graphs to reduce the search space [27]
S2: Replace FILTER equality constraints that compare a variable and a constant with BGPs [46]
S3: Split FILTER clauses with CONJUNCTION of constraints into a cascade of FILTER clauses with atomic constraints [46]
S4: Replace FILTER clauses with DISJUNCTION of equality constraints using UNION or VALUES[27]
S5: Reorder triple patterns applying most restrictive patterns for each dimension first (using criteria OC1, OC2, or OC3)

Table 6: Strategies used to improve query performance

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13
S1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S2	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓
S3	✓	✓	✓		✓		✓	✓	✓	✓		✓	✓
S4									✓	✓	✓	✓	
S5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 7: Applicability of each improvement strategy to SSB-QB4OLAP queries.

The combination of all possible strategies defines a space from which we chose a subset, based on the applicability of the strategies to the different queries. Thus, we devised a space of *Evaluation Scenarios (ES)*, where each scenario represents the application of a sequence of improvement strategies to the naïve SPARQL queries. Figure 26 shows the space of evaluation scenarios as a tree. Each node represents an ES, and labels on edges represent the improvement strategy

applied to transform a parent ES into a child ES. We can see that S1 and S2 were chosen to belong to all evaluation scenarios, since they apply to most queries. Then we consider the question of applying S3 (ES3) or not. For S4 we consider both flavours: either replacing FILTER conjunction with UNION or VALUES clauses. Finally, we consider the triples-reordering strategy (S5) using each of the ordering criteria discussed in Section 6.6. As an example, ES11 is the result of applying improvement strategies S1, S2, S4 (VALUES) and S5 (OC1), to naïve SPARQL queries. See Section C.3 for details on all the generated SPARQL queries (247 in total).

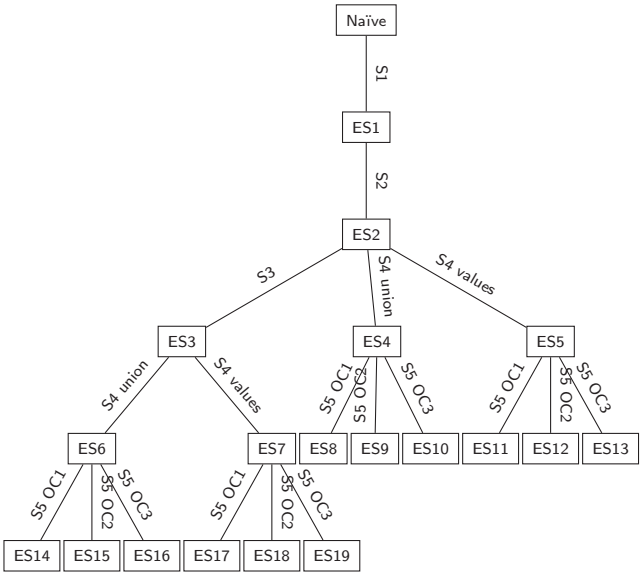


Figure 26: Improvement Strategies Evaluation Scenarios

Table 8 reports results for the naïve approach and all the evaluation scenarios. ES7 and ES11 are the scenarios with better performance. Figure 27 reports the average execution time for each query at the best improvement scenarios.

7.2.2 *Comparison against SSB-QB*

One of goal of our experiments was to compare the queries produced by our naïve approach, and the best and worst cases of the improved queries, against the SSB-QB queries. Thus, we implemented the SSB-QB queries in our experimental setting, and ran them. Table 9 shows the results obtained for each TPC-H metric, and Figure 28 presents a detailed comparison on the execution time for each query. We compare SSB-QB best case (the minimum execution time) against the naïve SSB-QB4OLAP worst case (the maximum execution time). See Appendix C for details on the queries.

7.3 DISCUSSION

Regarding the improvement scenarios our results show that, for the TPC-H Composite metric, scenario ES11 outperforms the other ones,

	Power (QpH)	Throughput (QpH)	Composite (QpH)	Interval (sec)
Naïve	63.8	75.6	69.5	1237.6
ES1	253.1	293.3	272.4	319.2
ES2	402.4	361.2	381.2	259.1
ES3	326.7	353.9	340.0	264.5
ES6	354.5	108.3	196.0	864.2
ES14	217.3	148.9	179.9	628.7
ES15	257.4	198.7	226.2	471.0
ES16	415.5	254.0	324.9	368.4
ES7	706.8	561.9	630.2	166.6
ES17	427.2	368.4	396.7	254.1
ES18	427.6	339.4	381.0	275.8
ES19	456.6	379.6	416.4	246.6
ES4	375.8	215.9	284.9	433.4
ES8	253.6	171.5	208.6	545.7
ES9	227.0	146.5	182.4	638.8
ES10	214.7	148.0	178.2	632.6
ES5	490.8	418.6	453.3	223.6
ES11	693.1	750.1	721.0	124.8
ES12	472.4	368.9	417.5	253.7
ES13	380.2	327.2	352.7	286.1

Table 8: TPC-H metrics: improvement evaluation

	Power (QpH)	Throughput (QpH)	Composite (QpH)	Interval (sec)
SSB-QB[21]	69.9	17.2	34.7	5447.0
SSB-QB4OLAP Naïve	63.8	75.6	69.5	1237.6
SSB-QB4OLAP ES14 (worst case)	217.3	148.9	179.9	628.7
SSB-QB4OLAP ES11 (best case)	693.1	750.1	721.0	124.8

Table 9: TPC-H metrics comparison

with a 10X improvement with respect to the naïve scenario (see Table 8), and a 10X speed-up in the execution time for the query mix. The second best scenario is ES7, with a 9X improvement on TPC-H Composite with respect to the naïve scenario and a 9X speed-up. However, the average execution time per query is similar in both scenarios, except for queries Q7 (where ES7 outperforms ES11) and Q12 (where ES11 outperforms ES7). Both scenarios apply S1, S2, and S4 (with `VALUES` splitting of `FILTER` conditions), but ES7 applies S3, while ES11 applies S5 with OC1 reordering (Figure 26).

About the impact of each improvement strategy (Table 8), strategies S1 and S2 combined yield a 5.5X improvement with respect to naïve queries. However, we cannot be conclusive on the impact of strategy S3. Note that the pairs of scenarios (ES6, ES4) and (ES7, ES5) only differ on the application of this strategy. In the first case, the scenario where S3 is applied performs worse (ES6), while in the second case the scenario where S3 is applied performs better (ES7). For S4, our results show that, replacing `FILTER` disjunctive conditions with `VALUES` clauses, improves performance (ES3 vs. ES7 and ES2 vs. ES5), while `UNION` downgrades the performance (ES3 vs ES6 and ES2 vs. ES4).

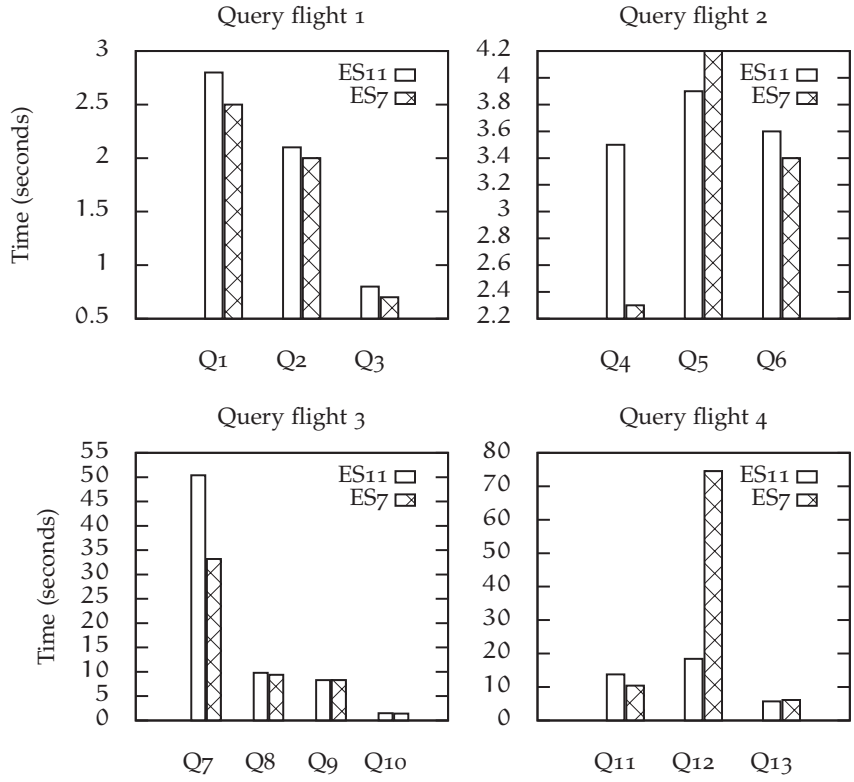


Figure 27: Naïve vs. improved queries response time

Finally, we cannot be conclusive on the impact of reordering graph patterns.

As presented in Section 7.2, we only evaluate our improvement strategies on Virtuoso Open Source triplestore. We cannot conclude that these results can be generalized across different SPARQL engines, although some of the implemented strategies seem promising according to the results reported in [27].

Comparing our approach with SSB-QB, although the values for TPC-H Power metric are very similar, values for TPC-H composite show that even our naïve approach represents a 2X improvement with respect to SSB-QB (Table 9). Considering our less improved scenario (ES14), we get a 5X enhancement, and 20X if we consider our best improved scenario (ES11). A detailed analysis on the execution time of each query (see Figure 28) shows that our approach outperforms SSB-QB for Q1, Q4, Q7, Q11, and Q12.

We next further analyze the reasons why our naïve approach has better performance than the SSB-QB queries.

First, SSB-QB queries include an ORDER BY clause to order results, while our queries do not. From our point of view, ordering triples is only advisable for visualization purposes and is not required for computing the result of the query. Second, as a consequence of the absence of level attributes, SSB-QB queries use string comparison on IRIs to fix level members, while we can use comparison over other data types, like, for example, numeric values. It is well-known that

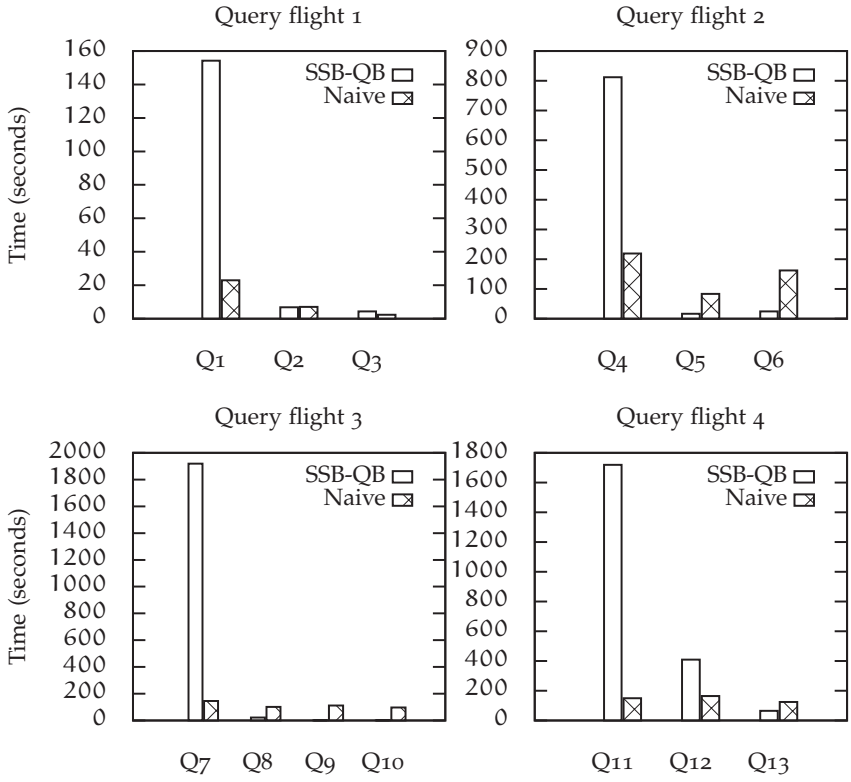


Figure 28: SSB-QB and Naïve SSB-QB4OLAP queries response time

string comparison is usually slower that integer comparison. Finally, we observe that the BGPs used to traverse hierarchies in SSB-QB may not take advantage of Virtuoso indexes, while the BGPs used in our approach are designed to use the indexes.

To illustrate this last point we give some insight on Virtuoso, and then present an example. Virtuoso triple store uses a relational data base to store data. In particular, all the triples are stored in a single table with four columns named graph (G), subject (S), predicate (P), and object (O). According to Virtuoso documentation [4], the following indexes are implemented on this table :

- PSOG - primary key index
- POGS - bitmap index for lookups on object value.
- SP - partial index for cases where only S is specified.
- OP - partial index for cases where only O is specified.
- GS - partial index for cases where only G is specified.

Since the primary key is PSOG data are physically ordered on this criteria. Our strategy takes advantage of this index, while SSB-QB does not. As an example, consider the representation of Q8 in SSB-QB (Figure 29) and in our naïve approach (Figure 30), and in particular the

4 Virtuoso technical documentation http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtRDFPerformanceTuning

BGPs that perform the ROLL-UP operation over the Time dimension (lines 7-11 in Figure 29 and lines 8-11 in Figure 30). Even though our approach uses more BGPs, at the time of the evaluation of each BGP, only the object of the triple is unknown, while in SSB-QB, subjects are unknown.

```
1  SELECT ?c_city ?s_city ?d_year sum(?rdfh_lo_revenue) as ?lo_revenue
2  FROM <http://lod2.eu/schemas/rdfh-inst#ssb1_ttl_qb>
3  FROM <http://lod2.eu/schemas/rdfh#ssb1_ttl_dsd>
4  FROM <http://lod2.eu/schemas/rdfh#ssb1_ttl_levels>
5  WHERE {
6    ?obs qb:dataSet rdfh-inst:ds.
7    ?obs rdfh:lo_orderdate ?d_date.
8    ?d_yearmonthnum skos:narrower ?d_date.
9    ?d_yearmonth skos:narrower ?d_yearmonthnum.
10   ?d_year skos:narrower ?d_yearmonth.
11   rdfh:lo_orderdateYearLevel skos:member ?d_year.
12   ?obs rdfh:lo_custkey ?c_customer.
13   ?c_city skos:narrower ?c_customer.
14   ?c_nation skos:narrower ?c_city.
15   ?c_region skos:narrower ?c_nation.
16   rdfh:lo_custkeyRegionLevel skos:member ?c_region.
17   ?obs rdfh:lo_suppkey ?s_supplier.
18   ?s_city skos:narrower ?s_supplier.
19   ?s_nation skos:narrower ?s_city.
20   ?s_region skos:narrower ?s_nation.
21   rdfh:lo_suppkeyRegionLevel skos:member ?s_region.
22   ?obs rdfh:lo_revenue ?rdfh_lo_revenue.
23   FILTER(?c_nation = rdfh:lo_custkeyNationUNITED-STATES ).
24   FILTER(?s_nation = rdfh:lo_suppkeyNationUNITED-STATES ).
25   FILTER(str(?d_year) >= "http://lod2.eu/schemas/rdfh#lo_orderdateYear1992" and
26        str(?d_year) <= "http://lod2.eu/schemas/rdfh#lo_orderdateYear1997").
27 }
28 GROUP BY ?d_year ?c_city ?s_city
29 ORDER BY ASC(?d_year) DESC(?lo_revenue)
```

Figure 29: Query 8 (SSB-QB)

```
SELECT ?plm2 ?plm3 ?plm5 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
  ?plm2 qb4o:memberOf schema:year .
  ?o rdfh:lo_custkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
  ?plm3 qb4o:memberOf schema:city . ?plm3 schema:inNation ?plm4 .
  ?plm4 qb4o:memberOf schema:nation .
  ?o rdfh:lo_partkey ?lm3 .
  ?o rdfh:lo_suppkey ?lm4 .
  ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm5 .
  ?plm5 qb4o:memberOf schema:city . ?plm5 schema:inNation ?plm6 .
  ?plm6 qb4o:memberOf schema:nation .
  ?plm4 schema:nationName> ?plm41 .
  ?plm6 schema:nationName> ?plm61 .
  ?plm2 schema:yearNum ?plm21 .
  FILTER (REGEX (?plm41,"UNITED_STATES" , "i")) &&
       (REGEX (?plm61,"UNITED_STATES" , "i")) &&
       (?plm21 >= 1992) && (?plm21 <= 1997)
  }
GROUP BY ?plm2 ?plm3 ?plm5
```

Figure 30: Query 8 (SSB-QB4OLAP naïve)

7.4 SUMMARY

In this chapter we presented the experiments performed to evaluate the concepts and algorithms presented in this thesis. We first presented the SSB-QB4OLAP benchmark, which consists of a set of high-level queries expressed in CQL and a data cube represented in QB4OLAP. Then, we used this benchmark to measure the impact of different SPARQL improvement strategies, with respect to naïve SPARQL queries, using TPC-H metrics to make this comparison. Finally, we also compared our approach with similar approaches, showing that even our naïve SPARQL queries represent a 2X improvement on query performance.

Part III

CONCLUSION

8

BACKGROUND AND RELATED WORK

"Reading is going toward something that is about to be,
and no one yet knows what it will be."

Italo Calvino, If on a Winter's Night a Traveler

In this chapter we first review the current approaches to the representation and querying of MD data on the SW. In the second part, we compare our proposal against these approaches.

8.1 CURRENT APPROACHES

Current approaches to MD data representation using SW standards can be organized in two categories: (i) Those that use specialized RDF vocabularies to explicitly define data cubes; and (ii) Those that implicitly define a data cube over existing RDF data graphs. Our work followed the explicit approach, and extended the QB vocabulary to include the MD structure.

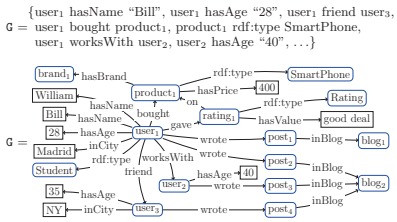
Kämpgen et al. [21, 23] attempt to override the lack of structure in QB using several different vocabularies, in particular QB4OLAP, and an extension to the SKOS vocabulary (named skosclass)[1]. The hierarchical structure of dimensions is described in terms of levels. Hierarchies are instances of `skos:ConceptScheme`, levels are instances of `skosclass:ClassificationLevel`, and levels are linked to the hierarchies they belong to using the `skos:inScheme` predicate. The relative position of each level in a hierarchy is expressed in terms of its depth within it, and this is represented by linking a level with an ordinal number using the `skosclass:depth` datatype property, being the top-most level All the level at depth 0. It is easy to see that, using this representation, levels can only belong to one hierarchy. Moreover, only strict balanced hierarchies are supported (i.e., each level can have at most one parent level). To represent aggregate functions, this approach uses instances defined in the QB4OLAP vocabulary, and the `qb4o:hasAggregateFunction` property lo link measures with them. Finally, the proposal does not consider level attributes, with the drawbacks that this carries, already explained in previous chapters. Regarding dimension instances, the authors propose to link levels with their members using the property `skos:member`.[2] RUPs between level members are represented using the `skos:narrower` property, as suggested by the QB specification.

The OpenCube project[3] aims at developing software tools that facilitate publishing and reusing Linked Statistical Data, strictly using QB. A complete toolkit is provided, to assist in the statistical data

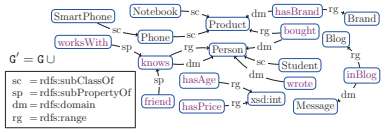
1 http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS
2 https://www.w3.org/TR/2005/WD-swbp-skos-core-spec-20051102/#member
3 http://opencube-project.eu/

life-cycle, including data analytics and visualizations. However, being based purely on QB, the proposal only supports single-level dimensions, reducing the possibilities of preforming OLAP operations.

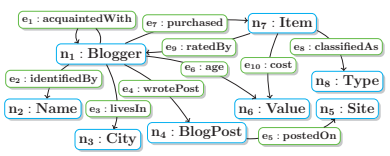
The WaRG project[4] discusses if traditional MD models could be appropriate for analyzing RDF data, and proposes a new analytical model that implicitly defines data cubes over RDF graphs. The core concept here is the Analytical Schema (AnS), a graph that represents an analytical view over existing RDF data, using the classical Global-as-View data integration approach [7]. The nodes in an AnS are RDF classes, while the edges are RDF properties, and the instances of AnSs are intentionally defined using BGPs on the RDF data to be analyzed. Figure 31 presents an example, taken from [7]. In this example, an AnS is defined for analyzing the data graph presented in Figure 31a, considering also the schema of the data, expressed in RDF-S and depicted in Figure 31b. Figure 31c presents the definition of an AnS, where blue labels correspond to classes, and green labels correspond to properties. The instance of this AnS is computed according to the definitions provided for each element in Figure 31d. For example, the class `Blogger` is populated with `Persons` that wrote things published in `blogs`. The authors propose the property `nextLevel` to model hierarchical relationships between dimension levels, which is used to model parent-child relationships between levels in a dimension hierarchy. This strategy only allows to represent strict balanced hierarchies at the schema level, and does not provide a representation for rollup relationships between level members.



(a) Sample RDF data graph.



(b) A schema in RDFS



(c) A sample Analytical Schema (AnS)

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

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

(d) Definition of classes and properties in Figure31c over the graphs in Figures 31a and 31b

Figure 31: An example of an Analytical Schema (AnS) (from [7])

4 https://team.inria.fr/oak/projects/warg/

8.2 ANALYTICAL QUERIES OVER SEMANTIC WEB MULTIDIMENSIONAL DATA

The terms "data analysis" or "analytical queries" are used in the literature to describe a broad kind of approaches to query data for decision-making. We next study and compare OLAP-like approaches that perform this analysis *directly* over SW MD data. To organize the discussion, we define seven categorization criteria: *Query Language Abstraction*, *Query Language Expressiveness*, *Formalization*, *Materialization*, *Reasoning*, *Standardization*, and *Optimization*. In the following section we present each criterion, and then compare existing approaches using this framework.

8.2.1 *Analytical Queries Comparison Criteria*

We define below the criteria we use to compare querying capabilities of different approaches.

QUERY LANGUAGE ABSTRACTION: Refers to the abstraction level of the query language. In particular, we distinguish query languages that operate at the *conceptual level* (i.e., over data cubes), from those that operate at the *logical level* (over the actual representation of the data cube).

QUERY LANGUAGE EXPRESSIVENESS: Accounts for the operations that are supported by the language, and their semantics.

FORMALIZATION: Refers to the definition of a formal model behind the query language, and if the semantics of the operations are provided using this formalization.

MATERIALIZATION: Concerns to the kinds of data that need to be materialized to answer queries. We distinguish between those approaches that only require *base data*, from those to require the materialization of *derived data* (e.g., materialization of aggregate views)

REASONING: Refers to the level of reasoning, if any, involved in the approach.

STANDARDIZATION: Accounts for the SW standards used. In particular we want to distinguish those approaches that entirely rely on SW standards (e.g., RDF, RDF-S, SPARQL), from those that use other mechanisms (e.g., relational databases, ad-hoc query answering machinery).

OPTIMIZATION: Refers to the optimization or improvement strategy, if any, proposed in each approach.

The first three criteria account for the querying capabilities, while the other four, for the machinery required by each approach.

8.2.2 *Comparison*

Kämpgen et al. [20–24] propose a query language at the conceptual level based on the concept of "OLAP queries". Here, an OLAP query is basically a set of operations over a data cube, namely Projection, Slice, Dice, Roll-up/Drill-down, and Drill-across. Although this approach proposes a formal data model for data cubes, only an informal description of each operation semantics is provided, with the exception of Drill-across [24]. Projection and Slice operations modify the schema of the cube. The first one allows to remove measures, while the second one removes dimensions. The Slice operation receives a cube and a dimension, and returns the cube resulting from removing the dimension in the input cube and aggregating over the members of that dimension up to the All level. In this proposal, Dice is not a selection operation, which is the usual semantics in OLAP. On the contrary, it is defined as a combined *filtering and slicing* operation, where filter conditions are restrained to equality conditions over level members. Notice that, in this way, it is not possible to filter cells using conditions over measure values. The Roll-up operation is defined as usual: given a cube, a dimension, and a level in that dimension, this operation changes the granularity of the cube, returning a new cube where measures are aggregated up to the parameter level. The Drill-across operator merges a set of input cubes, and it is only defined for data cubes that share dimensions schema and instances. Regarding materialization, this approach performs queries on base data. In particular it uses facts (observations), the schema of the cube (DSD), and the dimension instances. The approach is based on SW standards, and it provides mechanisms to translate OLAP queries into SPARQL queries over the MD data representation, previously discussed in this chapter. To improve the performance of OLAP queries, the authors explore the materialization of *aggregated views*, a mechanism used in traditional OLAP systems. They propose to avoid the cost of materializing all possible views, by computing and storing just what they call the *closest* view to each OLAP query. However, the approach does not provide a distance function to compute this closeness. Moreover, this improvement approach is only beneficial when queries are known in advance, and only if aggregate views can be reused to answer more than one query. Finally, view maintenance is not addressed in the proposal.

In the context of the WaRG project, Analytical Queries (AnQ) allow us to explore and aggregate data modeled via an Analytical Schema (AnS) [3, 7]. Each AnQ is formed by a *classifier* (a set of BGPs over the AnS that determine the analysis dimensions), a *measure* (also determined by a set of BGPs over the AnS), and an *aggregation function*. Example 8.2.1 shows a sample AnQ from [7]

Example 8.2.1. (Analytical Queries) Consider the data and the Analytical Schema from Figure 31. The following AnQ obtains "the number of sites where each blogger posts, classified by the blogger's age and city":

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

where the classifier and measure queries are defined as:

$$c(x, y_1, y_2) : -x \text{ age } y_1.x \text{ livesIn } y_2$$
$$m(x, z) : -x \text{ wrotePost } y.y \text{ postedOn } z.$$

Then, the answer set of the classifier query is:

$$\{\langle user_1, 28, \text{“Madrid”}\rangle, \langle user_3, 35, \text{“NY”}\rangle\}$$

and the answer set of the measure query is:
$\{\langle user_1, blog_1\rangle, \langle user_1, blog_2\rangle, \langle user_2, blog_2\rangle, \langle user_2, blog_3\rangle\}$.
Finally, the results of the query are:

$$\{\langle 28, \text{“Madrid”}, 2\rangle, \langle 35, \text{“NY”}, 1\rangle\}$$

□

This approach proposes to express some OLAP operations using AnQs. The SLICE operation is defined with its usual semantics. DICE operation restricts a cube to the cells that correspond to certain level members, and only equality conditions over level members are supported. Like in the work by Kämpgen et al, is not possible to filter cells according to measure values. The authors define other operations, like DRILL-IN (adds a dimension), DRILL-OUT (removes a dimension), and ROLL-UP, all sketched via examples, without a precise definition. The answers to an analytical query are computed over the union of the graphs that contain: (a) the data, (b) the schema, and (c) the AnS, but also the triples that can be derived applying RDFS entailment must be considered (e.g., in the graph G′ depicted in Figure 31c, the triple $product_1$ `rdf:type` Phone). The authors explore either the materialization of the AnS instance, or the rewriting of the AnQs in terms of the underlying graphs. The proposed implementation does not use a triple store to handle data. Instead, it stores triples in an in-memory column-based database, called kdb+[5]. Also, AnQs are implemented using the *q* language[6] and SQL, instead of SPARQL. Finally, they explore to improve the efficiency of queries (in particular of OLAP operations) based again in the idea of reusing the results of previous queries[3].

In the context of the OpenCube project, a set of tools were developed (the OpenCube Toolkit), for exploring and analyzing QB data cubes. Users interact with the cube via a GUI, so, for our comparison, we consider that this approach operates at the conceptual level. No formalization is provided. Regarding expressiveness, these tools allow to browse the cube, to DICE the cube, to fix the level members in dimensions, and to perform ROLL-UP and DRILL-DOWN operations over dimensions. Since dimensions have only one level, ROLL-UP operation is equivalent to aggregating cells up to the All level. The result of each of these operations are new cubes, which are materialized as QB slices. Recall from Section 3.2 that QB slices require not only

5 https://kx.com/software.php
6 q is a propietary query language over kdb+ http://kx.com/q/d/q.htm

the definition of a new DSD, but also the materialization of new aggregated observations. Regarding the standards used, these tools are integrated into an existing tool, and use the Sesame triplestore [7] to store the triples and compute aggregates.

Ibrahimov et al. [17] propose to use MDX[8] (a de-facto standard query language for OLAP) as a high-level language to express analytical queries, and then translate MDX into SPARQL queries over several endpoints, using QB4OLAP metadata. Few details are provided on the translation process from MDX to SPARQL. Although *expert OLAP users are likely to know MDX*, in a self-service BI environment most users are unlikely to be so proficient. Therefore, we think that a more intuitive query language, that deals only with cubes, is needed.

Jakobsen et al. [18] study the improvement of SPARQL queries over QB4OLAP data cubes. To reduce the number of joins (BGPs) needed to traverse hierarchies, they propose to generate denormalized representations of data instances called *star patterns* and *denormalized patterns*, which resemble relational representation strategies, for MD data. The idea behind this approach is to directly link facts (observations) with attribute values of related level members. Although preliminary results show an improvement in query performance, this approach prevents level members from being reused and referenced, breaking the Linked Data nature of QB4OLAP data instances. The representation of many-to-many rollup relationships between level members is not discussed in this approach.

Table 10 summarizes the comparison of the querying capabilities of each approach, while Table 11 summarizes the comparison on the machinery required by each approach. In the latter we omit the approach by Ibragimov et al. due to lack of information. We also include the proposal presented in this thesis document.

	ABSTRACTION	EXPRESSIVENESS	FORMALIZATION
Kämpgen et al. [20–24]	conceptual	Slice, Dice, Roll-up, Drill-down, Drill-across	Formal model, no semantics
WaRG [3, 7]	logical	Slice, Dice, Drill-in, Drill-out	Formal model and semantics
OpenCube [19, 31]	conceptual (GUI)	Dice, Roll-up (similar to Slice)	No formal model
Ibrahimov et al. [17]	conceptual	MDX operations	No formal model
Jakobsen et al. [18]	logical	SPARQL queries	does not apply
Our approach (CQL)	conceptual	Slice, Dice, Roll-up, Drill-down	Formal model, semantics grounded on the formal model

Table 10: Comparison summary (part 1)

7 http://rdf4j.org/
8 https://msdn.microsoft.com/en-us/library/ms144785.aspx

	MATERIALIZATION	REASONING	STANDARDS	OPTIMIZATION
Kämpgen et al. [20–24]	schema, instances	none	RDF, RDF-S, SPARQL	aggregate views (materialized)
WaRG [3, 7]	schema, instances, views	RDF-S entailment	RDF, RDF-S, no SPARQL	query answering using views
Open-Cube [19, 31]	schema, instances, aggregated cubes as QB slices	none	RDF, SPARQL	–
Jakobsen et al. [18]	schema, instances, de-normalized cubes	none	RDF, SPARQL	cube denormalization
Our approach	schema, instances	none	RDF, RDF-S, SPARQL	SPARQL improvement

Table 11: Comparison summary (part 2)

8.3 SUMMARY

In this chapter we reviewed the current approaches to the representation of MD data on the SW, and compared our proposal against these approaches according to different criteria. We conclude that the approach by Kämpgen et al. is the one that most resembles to our approach, both on the representation of MD data and OLAP-like analysis.

9

CONCLUSIONS AND FUTURE WORK

"There is no real ending.
It's just the place where you stop the story."

Frank Herbert

In the context of the web of data, applications and users need to publish and share MD data. Existing mechanisms that use SW standards, and in particular QB, the W3C standard, do not represent key features of the MD model, like dimension hierarchies, aggregate functions, or level attributes. Additionally, to perform OLAP operations *directly* over the SW, users have to deal with SW models and languages, like RDF or SPARQL, skills that are hardly mastered by typical analytical users. In this thesis we addressed these problems, providing a vocabulary that allows to represent complex MD models using SW standards, a high-level query language, CQL, where the data cube is a first-class citizen, based on well-known OLAP operations, and the machinery to automatically translate CQL queries into SPARQL. We conclude this document summarizing our contributions and providing ideas for further research.

9.1 CONCLUSIONS

In this thesis we have defined QB4OLAP, an RDF vocabulary to represent MD data using SW standards. This vocabulary is an extension to the current W3C standard, namely QB. We showed that QB4OLAP is expressive enough to represent complex advanced MD design features, like role-playing dimensions, flat hierarchies, parallel hierarchies, and ragged hierarchies, among other ones. These are not supported by other existing approaches. We also developed mechanisms to produce QB4OLAP data cubes. In particular, we presented algorithms and tools to obtain QB4OLAP cubes from data cubes stored in a relational database (ROLAP), and explored how to create rich MD schemas to analyze data already published using the QB vocabulary.

We also studied the problem of querying QB4OLAP data cubes. We proposed a high-level query language, denoted CQL, based on an algebra for OLAP, and we provided a clear semantics for the operations in this algebra, using a formal model and the notion of cuboids. We proposed a mechanism to translate CQL queries into SPARQL. This mechanism comprises three steps: (a) CQL queries simplification, (b) CQL to SPARQL translation, and (c) SPARQL queries improvement. By simplifying CQL queries, we intend to eliminate unnecessary operations, and to reorder operations to reduce the size of the data cube as early as possible in the query evaluation process. We proposed a set of rules, based on properties on the algebra operators, to accomplish these goals. Our CQL to SPARQL translation process (so-called

our naïve approach) is based on a set of algorithms that produce a SPARQL implementation of each operator. These algorithms are grounded on the semantics of the operators and SPARQL. Through the analysis of possible combination of operators (CQL query patterns), we produced SPARQL implementations of CQL queries. Finally, we explored techniques to improve the performance of our naïve queries. For this, we adapted general-purpose SPARQL performance improvement strategies, to the particular characteristics of the MD model, and its QB4OLAP representation.

We have implemented and evaluated our querying approach. The evaluation strategy consisted in the development of a benchmark (SSB-QB4OLAP), which allowed us not only to measure the impact of the performance improvement strategies, but also to compare our approach against the one proposed by Kämpgen et al. (SSB-QB) [20–24]. The comparison, based on the TPC-H Composite metric, showed that our naïve approach represents a 2X improvement with respect to SSB-QB, while our least and best improved scenarios represents a 5X and 20X enhancement, respectively.

Our most relevant conclusion is that *it is possible to represent MD data using SW standards, and also that is feasible to perform OLAP analysis directly over the SW, using standard SPARQL features.* Summing up, we think that the results presented in this thesis can encourage and promote the publication and sharing of MD data on the SW.

9.2 FUTURE WORK

Regarding QB4OLAP evolution, we believe that there are some MD concepts to be incorporated in future versions. So far, the vocabulary allows to declare different cardinality restrictions on RUP relationships, but it does not provide a mechanism to define distributing attributes, which are needed to properly calculate aggregate values in Many-to-Many parent-child relationships. Calculated members, and in particular calculated measures, are also a useful MD construct that can be added to the vocabulary.

Finally, regarding the correct summarizability of measures, QB4OLAP currently supports the definition of one aggregation function per measure in each cube, assuming that this function can be used to aggregate values along any dimension in the cube. This assumption can also be revised in future versions of QB4OLAP. We also plan to extend our querying approach, adding further operators, for example DRILL-ACROSS to allow cube integration.

Additional tools can be implemented to enrich our current toolkit. In this direction, we would like to have a graphical MD conceptual model editor to produce cube schemas in QB4OLAP. Also, to explore the generation of CQL queries via a graphical interface, similar to what is done in traditional OLAP tools.

9.3 OPEN PROBLEMS

One of the main motivations to publish MD data on the SW is to enable its reuse. In this sense, Kalampokis et al.[19] provide an interesting insight on several issues that are present in MD data already published using QB, that represent an obstacle to MD data reuse. Some of them were experienced by us with Eurostat data. In particular, they highlight that a great level of heterogeneity is found, with respect to how the vocabularies are used (and sometimes even misused). This is pointed out mainly as a consequence of the flexibility of the vocabularies, which offer many possibilities to represent MD concepts. We agree on this observation, but we also believe that MD conceptual data modeling is not an easy task. Even assuming that vocabularies are not misused, different publishers may end up with different ways of representing the same conceptual model. Taking this into account, we think that any approach aimed at reusing SW data, and MD data in particular, must deal with heterogeneity since, on the web, heterogeneity is the norm, not the exception. Thus, the development of tools to help users, not only in the publication, but in the modeling process as well, may promote good practices, mitigate errors, and reduce representational heterogeneity.

We identify several open problems regarding CQL algebra. What kind of queries can we write using this algebra? Is the set of operators complete? to answer these questions it seems relevant to study the expressiveness of the currently defined set of operators. Moreover, and regarding the CQL to SPARQL translation, formal verification techniques may be used to prove the equivalence of the obtained queries.

Finally, another key problem is the quality of data and metadata in QB4OLAP. In particular, to enable the reuse of already published multidimensional data, the data cubes should include enough metadata to give meaning to the domain concepts represented (for example, metadata to state that members of the country level in a dimension are countries). In this direction, although the QB vocabulary suggests to link the components of each DSD to the concept that they represent (using the `qb:concept` property), this is rarely used. An appropriate representation of these domain concepts may also allow to use SW reasoning capacities to find relationships between data cubes, to enhance analytical possibilities, and to create knowledge.

Part IV

APPENDIX

A

PREFIXES USED IN THIS THESIS

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX schema: <http://schema.org/>
PREFIX gn: <http://sws.geonames.org/>
PREFIX ge: <http://www.w3.org/2003/01/geo/wgs84_pos#>

PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX qb4o: <http://purl.org/qb4olap/cubes#>
```

Figure 32: RDF prefixes used in this work

```
PREFIX sdmxm: <http://purl.org/linked-data/sdmx/2009/measure#>
PREFIX sdmxd: <http://purl.org/linked-data/sdmx/2009/dimension#>
PREFIX sdmxc: <http://purl.org/linked-data/sdmx/2009/concept#>

PREFIX dsd: <http://eurostat.linked-statistics.org/dsd#>
PREFIX pr: <http://eurostat.linked-statistics.org/property#>
PREFIX citizen: <http://eurostat.linked-statistics.org/dic/citizen#>
PREFIX geo:  <http://eurostat.linked-statistics.org/dic/geo#>
PREFIX age: <http://eurostat.linked-statistics.org/dic/age#>
PREFIX sex: <http://eurostat.linked-statistics.org/dic/sex#>
PREFIX app: <http://eurostat.linked-statistics.org/dic/asyl_app#>
PREFIX eurostatdt: <http://eurostat.linked-statistics.org/data/>
PREFIX eurostatcell: <http://eurostat.linked-statistics.org/data/migr_asyappctzm#>

#Asylum application case study in QB4OLAP
PREFIX loc-ins:<http://www.fing.edu.uy/inco/cubes/instances/>
PREFIX loc-sch:<http://www.fing.edu.uy/inco/cubes/schemas/>
PREFIX sc: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>
PREFIX instances: <http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp>
PREFIX citDim: <http://www.fing.edu.uy/inco/cubes/dims/migr_asyapp/citizen#>
PREFIX time: <http://purl.org/qb4olap/dimensions/time#201409>
PREFIX dbpedia: <http://dbpedia.org/resource/>
#Queries
PREFIX queries: <http://www.fing.edu.uy/inco/cubes/queries/migr_asyapp#>
```

Figure 33: RDF prefixes used in the asylum application case study

```
PREFIX sdmxm: <http://purl.org/linked-data/sdmx/2009/measure#>
PREFIX sdmxd: <http://purl.org/linked-data/sdmx/2009/dimension#>
PREFIX sdmxc: <http://purl.org/linked-data/sdmx/2009/concept#>

PREFIX nw: <http://dwbook.org/cubes/schemas/northwind#> .
PREFIX nwt: <http://dwbook.org/cubes/instances/northwind/Time#>
PREFIX nwm: <http://dwbook.org/cubes/instances/northwind/Month#>
PREFIX nwy: <http://dwbook.org/cubes/instances/northwind/Year#>
PREFIX nws: <http://dwbook.org/cubes/instances/northwind/Supplier#>
PREFIX nwc: <http://dwbook.org/cubes/instances/northwind/Customer#>
PREFIX nwci: <http://dwbook.org/cubes/instances/northwind/City#>
PREFIX nwst: <http://dwbook.org/cubes/instances/northwind/State#>
PREFIX nwco: <http://dwbook.org/cubes/instances/northwind/Country#>
PREFIX nwe: <http://dwbook.org/cubes/instances/northwind/Employee#>
PREFIX nwp: <http://dwbook.org/cubes/instances/northwind/Product#>
PREFIX nwca: <http://dwbook.org/cubes/instances/northwind/Category#>
```

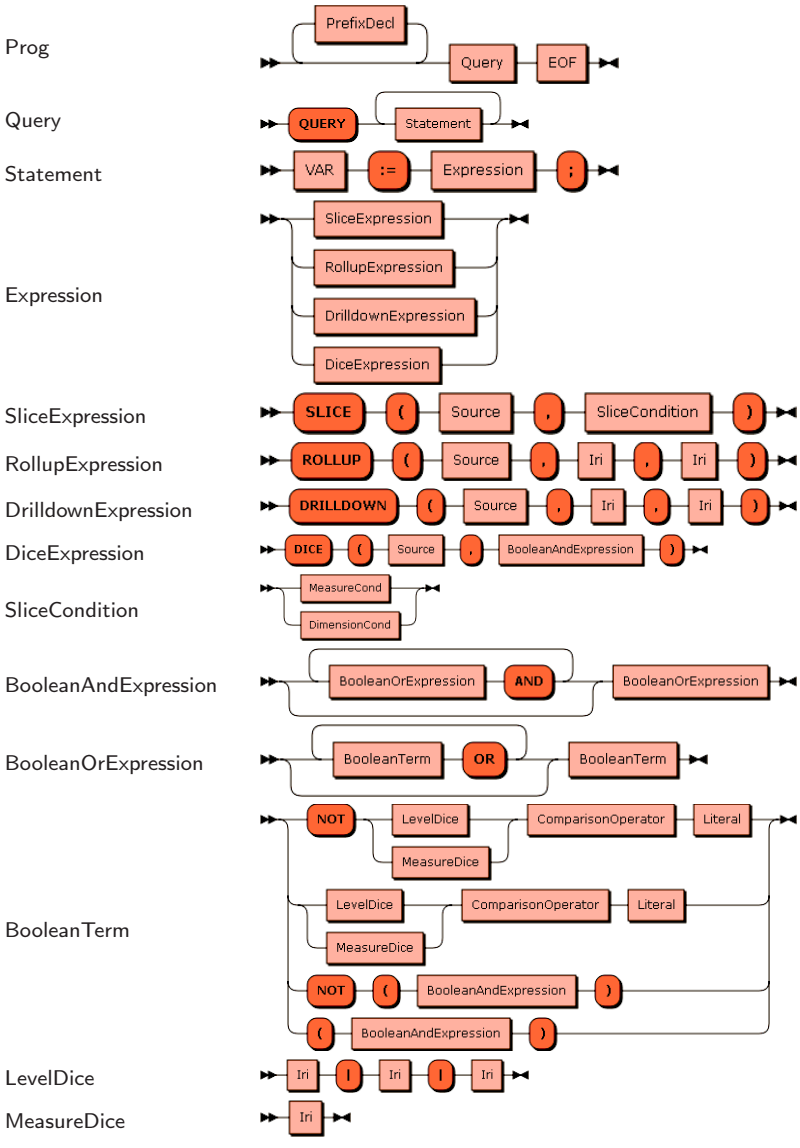
Figure 34: RDF prefixes used in the Northwind case study

B

CQL

B.1 CQL SYNTAX DIAGRAMS

We present CQL syntax diagrams, generated from CQL EBNF language grammar using Railroad Diagram Generator[1].



B.2 CQL SIMPLIFICATION PROOFS

We now present the proofs of the properties satisfied by CQL queries resulting from the simplification presented in Section 6.4. According to Definition 6.3.5, before the simplification process CQL queries satisfy the query patterns P1, P2, or P3. In each proof, we analyze each

1 http://bottlecaps.de/rr/ui

one of these cases. In what follows, we denote q_{in} and q_{out}, the queries before and after the simplification process, respectively.

B.2.1 *Proof of Property 6.4.1*

Property 6.4.1. If there is no DICE operation in q_{in}, there is at most one ROLL-UP, and no DRILL-DOWN operation, for each Dimension d in q_{out}.

Proof. (1) If q_{in} corresponds to P1, and does not include a DICE operation, then $\forall p, operation(p)\ /p \in q_{in}$, either p is a SLICE or a ROLL-UP. For each dimension d in q_{in}, if there exists a sequence of ROLL-UP operations over d, and also a SLICE operation over d, then, after applying Rule 5 (from Section 6.4), the only operation in q_{out} over d is a SLICE . If there exists a sequence of ROLL-UP over d, and there is no SLICE over d, then by the application of Rule 2 (from Section 6.4), the only operation in q_{out} over d is either a ROLLUP (if the target level is different than the origin level), or there is no operation at all (by the application of Rule 1).
(2) If q_{in} corresponds to P2 or P3, and does not include a DICE operation, then $\forall p, operation(p)\ /p \in q_{in}$ either p is a SLICE, a ROLL-UP, or a DRILL-DOWN. For each dimension d in q_{in}, if there exists a sequence of ROLL-UP and DRILL-DOWN operations over d, and also a SLICE operation over d, then, after applying Rule 5 (from Section 6.4), the only operation in q_{out} over d is a SLICE . If there exists a sequence of ROLL-UP over d, and no SLICE over d, then by the application of Rule 2 (from Section 6.4), the only operation in q_{out} over d is either a ROLLUP (if the target level is different than the origin level), or there is no operation at all (by the application of Rule 1). □

B.2.2 *Proof of Property 6.4.2*

Property 6.4.2. SLICE operations are either at the beginning or at the end of q_{out}, but not in the middle.

Proof. We consider two cases: SLICE operations over dimensions, and SLICE operation over measures.
(1) Rule 3 states that for each dimension d in q_{in}, if there exist one or more DICE operations, and a SLICE over d, the SLICE operation will be at the end of q_{out}; and if there is a SLICE and no DICE, then the SLICE operation will be at the beginning of q_{out}. This applies to queries that correspond to all the patterns.
(2) Rule 4 indicates that for each measure m in q_{in}, if there exist one or more DICE operations, and a SLICE over m, the SLICE operation will be at the end of q_{out}; and if there is a SLICE and no DICE, then the SLICE operation will at the beginning of q_{out}. This applies to queries that correspond pattern P3. □

C

SSB-QB4OLAP BENCHMARK

C.1 CQL QUERIES

Figures 36 to 48 show the CQLversion of each query in the benchmark. The prefixes defined in Figure 35 apply to all theses queries.

```
prefix rdfh-inst: <http://lod2.eu/schemas/rdfh-inst#>;
prefix rdfh: <http://lod2.eu/schemas/rdfh#>;
prefix ssb-qb4olap: <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap#>;
```

Figure 35: RDF prefixes used in CQLqueries

```
$C1 := SLICE(rdfh-inst:ds, ssb-qb4olap:customerDim );
$C2 := SLICE($C1, ssb-qb4olap:supplierDim );
$C3 := SLICE($C2, ssb-qb4olap:partsDim );
$C4 := SLICE($C3, MEASURES(rdfh:sum_revenue));
$C5 := SLICE($C4, MEASURES(rdfh:lo_revenue));
$C6 := SLICE($C5, MEASURES(rdfh:lo_supplycost));
$C7 := SLICE($C6, MEASURES(rdfh:sum_profit));
$C8 := DICE ($C7, rdfh:lo_quantity<= 24 );
$C9 := DICE ($C8, rdfh:lo_discount>=1 AND rdfh:lo_discount<=3);
$C10 := SLICE($C9, MEASURES(rdfh:lo_quantity));
$C11 := ROLLUP ($C10, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum = 1993));
$C13 := SLICE($C12, ssb-qb4olap:timeDim );
```

Figure 36: Query 1 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, ssb-qb4olap:customerDim );
$C2 := SLICE($C1, ssb-qb4olap:supplierDim );
$C3 := SLICE($C2, ssb-qb4olap:partsDim );
$C4 := SLICE($C3, MEASURES(rdfh:sum_revenue));
$C5 := SLICE($C4, MEASURES(rdfh:lo_revenue));
$C6 := SLICE($C5, MEASURES(rdfh:lo_supplycost));
$C7 := SLICE($C6, MEASURES(rdfh:sum_profit));
$C8 := DICE ($C7, rdfh:lo_quantity>=26 AND rdfh:lo_quantity<=35);
$C9 := SLICE($C8, MEASURES(rdfh:lo_quantity));
$C10 := DICE ($C9, rdfh:lo_discount>=4 AND rdfh:lo_discount<=6);
$C11 := ROLLUP ($C10, ssb-qb4olap:timeDim, ssb-qb4olap:month);
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:month|
                     ssb-qb4olap:yearmonthnum = 199401));
$C13 := SLICE($C12, ssb-qb4olap:timeDim );
```

Figure 37: Query 2 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, ssb-qb4olap:customerDim );
$C2 := SLICE($C1, ssb-qb4olap:supplierDim );
$C3 := SLICE($C2, ssb-qb4olap:partsDim );
$C4 := SLICE($C3, MEASURES(rdfh:sum_revenue));
$C5 := SLICE($C4, MEASURES(rdfh:lo_revenue));
$C6 := SLICE($C5, MEASURES(rdfh:lo_supplycost));
$C7 := SLICE($C6, MEASURES(rdfh:sum_profit));
$C8 := DICE ($C7, rdfh:lo_quantity>=26 AND rdfh:lo_quantity<=35);
$C9 := SLICE($C8, MEASURES(rdfh:lo_quantity));
$C10 := DICE ($C9, rdfh:lo_discount>=5 AND rdfh:lo_discount<=7);
$C11 := ROLLUP ($C10, ssb-qb4olap:timeDim, ssb-qb4olap:week);
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:week|
                     ssb-qb4olap:yearweeknum = 19946));
$C13 := SLICE($C12, ssb-qb4olap:timeDim );
```

Figure 38: Query 3 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, ssb-qb4olap:customerDim );
$C3 := SLICE($C2, MEASURES(rdfh:lo_extendedprice));
$C4 := SLICE($C3, MEASURES(rdfh:sum_revenue));
$C5 := SLICE($C4, MEASURES(rdfh:lo_supplycost));
$C6 := SLICE($C5, MEASURES(rdfh:sum_profit));
$C7 := SLICE($C6, MEASURES(rdfh:lo_quantity));
$C8 := SLICE($C7, MEASURES(rdfh:lo_discount));
$C9 := ROLLUP ($C8, ssb-qb4olap:supplierDim, ssb-qb4olap:region);
$C10 := ROLLUP ($C9, ssb-qb4olap:partsDim, ssb-qb4olap:category);
$C11 := ROLLUP ($C10, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C12 := DICE ($C11, (ssb-qb4olap:supplierDim|ssb-qb4olap:region|
                     ssb-qb4olap:regionName = "AMERICA"));
$C13 := DICE ($C12, (ssb-qb4olap:partsDim|ssb-qb4olap:category|
                     ssb-qb4olap:categoryName = "MFGR#12"));
$C14:= SLICE($C13, ssb-qb4olap:supplierDim );
```

Figure 39: Query 4 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, ssb-qb4olap:customerDim );
$C3 := SLICE($C2, MEASURES(rdfh:lo_extendedprice));
$C4 := SLICE($C3, MEASURES(rdfh:sum_revenue));
$C5 := SLICE($C4, MEASURES(rdfh:lo_supplycost));
$C6 := SLICE($C5, MEASURES(rdfh:sum_profit));
$C7 := SLICE($C6, MEASURES(rdfh:lo_quantity));
$C8 := SLICE($C7, MEASURES(rdfh:lo_discount));
$C9 := ROLLUP ($C8, ssb-qb4olap:supplierDim, ssb-qb4olap:region);
$C10 := ROLLUP ($C9, ssb-qb4olap:partsDim, ssb-qb4olap:brand);
$C11 := ROLLUP ($C10, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C12 := DICE ($C11, (ssb-qb4olap:supplierDim|ssb-qb4olap:region|
                     ssb-qb4olap:regionName = "ASIA"));
$C13 := DICE ($C12, (ssb-qb4olap:partsDim|ssb-qb4olap:brand|
                     ssb-qb4olap:brandName >= "MFGR#2221"));
$C14 := DICE ($C13, (ssb-qb4olap:partsDim|ssb-qb4olap:brand|
                     ssb-qb4olap:brandName <= "MFGR#2228"));
$C15 := SLICE($C14, ssb-qb4olap:supplierDim );
```

Figure 40: Query 5 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, ssb-qb4olap:customerDim );
$C3 := SLICE($C2, MEASURES(rdfh:lo_extendedprice));
$C4 := SLICE($C3, MEASURES(rdfh:sum_revenue));
$C5 := SLICE($C4, MEASURES(rdfh:lo_supplycost));
$C6 := SLICE($C5, MEASURES(rdfh:sum_profit));
$C7 := SLICE($C6, MEASURES(rdfh:lo_quantity));
$C8 := SLICE($C7, MEASURES(rdfh:lo_discount));
$C9 := ROLLUP ($C8, ssb-qb4olap:supplierDim, ssb-qb4olap:region);
$C10 := ROLLUP ($C9, ssb-qb4olap:partsDim, ssb-qb4olap:brand);
$C11 := ROLLUP ($C10, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C12 := DICE ($C11, (ssb-qb4olap:supplierDim|ssb-qb4olap:region|
                     ssb-qb4olap:regionName = "EUROPE"));
$C13 := DICE ($C12, (ssb-qb4olap:partsDim|ssb-qb4olap:brand|
                     ssb-qb4olap:brandName = "MFGR#2239"));
$C14 := SLICE($C13, ssb-qb4olap:supplierDim );
```

Figure 41: Query 6 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C3 := SLICE($C2, MEASURES(rdfh:lo_supplycost));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:region);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:region);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C10 := DICE ($C9, (ssb-qb4olap:supplierDim|ssb-qb4olap:region|
                    ssb-qb4olap:regionName = "ASIA"));
$C11 := DICE ($C10, (ssb-qb4olap:customerDim|ssb-qb4olap:region|
                     ssb-qb4olap:regionName = "ASIA"));
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum >= 1992));
$C13 := DICE ($C12, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum <= 1997));
$C14 := SLICE ($C13, ssb-qb4olap:partsDim);
```

Figure 42: Query 7 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C3 := SLICE($C2, MEASURES(rdfh:lo_supplycost));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:nation);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:nation);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C10 := DICE ($C9, (ssb-qb4olap:supplierDim|ssb-qb4olap:nation|
                    ssb-qb4olap:nationName = "UNITED_STATES"));
$C11 := DICE ($C10, (ssb-qb4olap:customerDim|ssb-qb4olap:nation|
                     ssb-qb4olap:nationName = "UNITED_STATES"));
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum >= 1992));
$C13 := DICE ($C12, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum <= 1997));
$C14 := SLICE ($C13, ssb-qb4olap:partsDim);
```

Figure 43: Query 8 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C3 := SLICE($C2, MEASURES(rdfh:lo_supplycost));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:city);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:city);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C10 := DICE ($C9, ((ssb-qb4olap:supplierDim|ssb-qb4olap:city|
                     ssb-qb4olap:cityName = "UNITED_KI1")OR
                    (ssb-qb4olap:supplierDim|ssb-qb4olap:city|
                     ssb-qb4olap:cityName = "UNITED_KI5")));
$C11 := DICE ($C10, ((ssb-qb4olap:customerDim|ssb-qb4olap:city|
                      ssb-qb4olap:cityName = "UNITED_KI1")OR
                     (ssb-qb4olap:customerDim|ssb-qb4olap:city|
                     ssb-qb4olap:cityName = "UNITED_KI5")));
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum >= 1992));
$C13 := DICE ($C12, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum <= 1997));
$C14 := SLICE ($C13, ssb-qb4olap:partsDim);
```

Figure 44: Query 9 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C3 := SLICE($C2, MEASURES(rdfh:lo_supplycost));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:city);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:city);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:month);
$C10 := DICE ($C11, ((ssb-qb4olap:supplierDim|ssb-qb4olap:city|
                      ssb-qb4olap:cityName = "UNITED_KI1")OR
                     (ssb-qb4olap:supplierDim|ssb-qb4olap:city|
                      ssb-qb4olap:cityName = "UNITED_KI5")));
$C11 := DICE ($C10, ((ssb-qb4olap:customerDim|ssb-qb4olap:city|
                      ssb-qb4olap:cityName = "UNITED_KI1")OR
                     (ssb-qb4olap:customerDim|ssb-qb4olap:city|
                      ssb-qb4olap:cityName = "UNITED_KI5")));
$C12 := DICE ($C11, (ssb-qb4olap:timeDim|ssb-qb4olap:month|
                     ssb-qb4olap:yearmonthnum = 199712));
$C13 := SLICE ($C14, ssb-qb4olap:partsDim);
```

Figure 45: Query 10 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:region);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:region);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C10 := ROLLUP ($C9, ssb-qb4olap:partsDim, ssb-qb4olap:manufacturer);
$C11 := DICE ($C10, ssb-qb4olap:supplierDim|ssb-qb4olap:region|
                    ssb-qb4olap:regionName = "AMERICA");
$C12 := DICE ($C11, ssb-qb4olap:customerDim|ssb-qb4olap:region|
                    ssb-qb4olap:regionName = "AMERICA");
$C13 := DICE ($C12, ((ssb-qb4olap:partsDim|ssb-qb4olap:manufacturer|
                      ssb-qb4olap:manufacturerName = "MFGR#1")OR
                     (ssb-qb4olap:partsDim|ssb-qb4olap:manufacturer|
                      ssb-qb4olap:manufacturerName = "MFGR#2")));
$C14 := SLICE ($C13, ssb-qb4olap:partsDim);
$C15 := SLICE ($C14, ssb-qb4olap:supplierDim);
```

Figure 46: Query 11 (CQL)

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:region);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:region);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C10 := ROLLUP ($C9, ssb-qb4olap:partsDim, ssb-qb4olap:manufacturer);
$C11 := DICE ($C10, ssb-qb4olap:supplierDim|ssb-qb4olap:region|
                    ssb-qb4olap:regionName = "AMERICA");
$C12 := DICE ($C11, ssb-qb4olap:customerDim|ssb-qb4olap:region|
                    ssb-qb4olap:regionName = "AMERICA");
$C13 := DICE ($C12, ((ssb-qb4olap:partsDim|ssb-qb4olap:manufacturer|
                      ssb-qb4olap:manufacturerName = "MFGR#1")OR
                     (ssb-qb4olap:partsDim|ssb-qb4olap:manufacturer|
                      ssb-qb4olap:manufacturerName = "MFGR#2")));
$C14 := DICE ($C13, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum >= 1997));
$C15 := DICE ($C14, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum <= 1998));
$C16 := SLICE ($C15, ssb-qb4olap:customerDim);
```

Figure 47: Query 12 (CQL)

C.2 NAÏVE SPARQL QUERIES

Figures 50 to 62 show the naïve SPARQL queries that implement each CQLquery presented in Section C.1. The prefixes defined in Figure 49 apply to all theses queries.

```
$C1 := SLICE(rdfh-inst:ds, MEASURES(rdfh:lo_extendedprice));
$C2 := SLICE($C1, MEASURES(rdfh:sum_revenue));
$C4 := SLICE($C3, MEASURES(rdfh:sum_profit));
$C5 := SLICE($C4, MEASURES(rdfh:lo_quantity));
$C6 := SLICE($C5, MEASURES(rdfh:lo_discount));
$C7 := ROLLUP ($C6, ssb-qb4olap:supplierDim, ssb-qb4olap:nation);
$C8 := ROLLUP ($C7, ssb-qb4olap:customerDim, ssb-qb4olap:region);
$C9 := ROLLUP ($C8, ssb-qb4olap:timeDim, ssb-qb4olap:year);
$C10 := ROLLUP ($C9, ssb-qb4olap:partsDim, ssb-qb4olap:category);
$C11 := DICE ($C10, ssb-qb4olap:supplierDim|ssb-qb4olap:nation|
                    ssb-qb4olap:nationName = "UNITED_STATES");
$C12 := DICE ($C11, ssb-qb4olap:customerDim|ssb-qb4olap:region|
                    ssb-qb4olap:regionName = "AMERICA");
$C13 := DICE ($C12, (ssb-qb4olap:partsDim|ssb-qb4olap:category|
                     ssb-qb4olap:categoryName = "MFGR#14"));
$C14 := DICE ($C13, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum >= 1997));
$C15 := DICE ($C14, (ssb-qb4olap:timeDim|ssb-qb4olap:year|
                     ssb-qb4olap:yearNum <= 1998));
$C16 := SLICE ($C15, ssb-qb4olap:customerDim);
```

Figure 48: Query 13 (CQL)

```
prefix qb: <http://purl.org/linked-data/cube#>
prefix qb4o: <http://purl.org/qb4olap/cubes#>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix schema: <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix rdfh: <http://lod2.eu/schemas/rdfh#>
prefix rdfh-inst: <http://lod2.eu/schemas/rdfh-inst#>
```

Figure 49: RDF prefixes used in naïve SPARQL queries

```
SELECT (?ag1 * ?ag2) as ?sum_revenue
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{ ?plm2 schema:yearNum ?plm21 }.
{
 SELECT ?plm2 (SUM(xsd:float(?m2)) as ?ag1) (SUM(xsd:float(?m3)) as ?ag2)
  WHERE {
   ?o a qb:Observation .
   ?o qb:dataSet rdfh-inst:ds .
   ?o rdfh:lo_quantity ?m1 .
   ?o rdfh:lo_discount ?m2 .
   ?o rdfh:lo_extendedprice ?m3 .
   ?o rdfh:lo_orderdate ?lm1 .
   ?lm1 qb4o:memberOf rdfh:lo_orderdate .
   ?lm1 schema:dateInMonth ?plm1 .
   ?plm1 qb4o:memberOf schema:month .
   ?plm1 schema:monthInYear ?plm2 .
   ?plm2 qb4o:memberOf schema:year .
   FILTER (?m1 <= 24) && (?m2 >= 1) && (?m2 <= 3)
 }
 GROUP BY ?plm2 }
FILTER (?plm21=1993) }
```

Figure 50: Query 1 (naïve SPARQL)

```
SELECT (?ag1 * ?ag2) as ?sum_revenue
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{ ?plm1 schema:yearmonthnum> ?plm11 }.
{
 SELECT ?plm1 (SUM(xsd:float(?m2)) as ?ag1)
           (SUM(xsd:float(?m3)) as ?ag2)
 WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_quantity ?m1 .
  ?o rdfh:lo_discount ?m2 .
  ?o rdfh:lo_extendedprice ?m3 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate .
  ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month .
  FILTER (?m1 >= 26) && (?m1 <= 35) && (?m2 >= 4) && (?m2 <= 6)
 }
 GROUP BY ?plm1
 } FILTER (?plm11=199401)
}
```

Figure 51: Query 2 (naïve SPARQL)

```
SELECT (?ag1 * ?ag2) as ?sum_revenue
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{ ?plm1 schema:yearweeknum> ?plm11 }.
{
 SELECT ?plm1 (SUM(xsd:float(?m2)) as ?ag1) (SUM(xsd:float(?m3)) as ?ag2)
 WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_quantity ?m1 .
  ?o rdfh:lo_discount ?m2 .
  ?o rdfh:lo_extendedprice ?m3 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate .
  ?lm1 schema:dateInWeek ?plm1 .
  ?plm1 qb4o:memberOf schema:week .
  FILTER (?m1 >= 26) && (?m1 <= 35) && (?m2 >= 5) && (?m2 <= 7)
 }
 GROUP BY ?plm1
} FILTER (?plm11=19946)
}
```

Figure 52: Query 3 (naïve SPARQL)

```
SELECT ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
 ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
 ?plm2 qb4o:memberOf schema:year .
 ?o rdfh:lo_partkey ?lm2 .
 ?lm2 qb4o:memberOf rdfh:lo_partkey . ?lm2 schema:hasBrand ?plm3 .
 ?plm3 qb4o:memberOf schema:brand . ?plm3 schema:hasCategory ?plm4 .
 ?plm4 qb4o:memberOf schema:category .
 ?o rdfh:lo_suppkey ?lm3 .
 ?lm3 qb4o:memberOf rdfh:lo_suppkey .?lm3 schema:inCity ?plm5 .
 ?plm5 qb4o:memberOf schema:city . ?plm5 schema:inNation ?plm6 .
 ?plm6 qb4o:memberOf schema:nation . ?plm6 schema:inRegion ?plm7 .
 ?plm7 qb4o:memberOf schema:region . ?plm7 schema:inRegion ?plm71 .
 ?plm4 schema:categoryName ?plm41 .
 FILTER (REGEX (?plm71,"AMERICA" , "i"))&&
      (REGEX (?plm41,"MFGR#12" , "i"))
 }
GROUP BY ?plm2 ?plm3
```

Figure 53: Query 4 (naïve SPARQL)

```
SELECT ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_quantity ?m1 .
 ?o rdfh:lo_discount ?m2 .
 ?o rdfh:lo_extendedprice ?m3 .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
 ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
 ?plm2 qb4o:memberOf schema:year .
 ?o rdfh:lo_partkey ?lm2 .
 ?lm2 qb4o:memberOf rdfh:lo_partkey . ?lm2 schema:hasBrand ?plm3 .
 ?plm3 qb4o:memberOf schema:brand .
 ?o rdfh:lo_suppkey ?lm3 .
 ?lm3 qb4o:memberOf rdfh:lo_suppkey . ?lm3 schema:inCity ?plm4 .
 ?plm4 qb4o:memberOf schema:city . ?plm4 schema:inNation ?plm5 .
 ?plm5 qb4o:memberOf schema:nation . ?plm5 schema:inRegion ?plm6 .
 ?plm6 qb4o:memberOf schema:region . ?plm6 schema:inRegion ?plm61 .
 ?plm3 schema:brandName ?plm31 .
 FILTER (REGEX (?plm61,"ASIA" , "i")) &&
      (str(?plm31)>="MFGR#2221") &&
      (str(?plm31)<="MFGR#2228")
 }
GROUP BY ?plm2 ?plm3
```

Figure 54: Query 5 (naïve SPARQL)

```
SELECT ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
  ?plm2 qb4o:memberOf schema:year .
  ?o rdfh:lo_partkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_partkey . ?lm2 schema:hasBrand ?plm3 .
  ?plm3 qb4o:memberOf schema:brand .
  ?o rdfh:lo_suppkey ?lm3 .
  ?lm3 qb4o:memberOf rdfh:lo_suppkey . ?lm3 schema:inCity ?plm4 .
  ?plm4 qb4o:memberOf schema:city . ?plm4 schema:inNation ?plm5 .
  ?plm5 qb4o:memberOf schema:nation . ?plm5 schema:inRegion ?plm6 .
  ?plm6 qb4o:memberOf schema:region . ?plm6 schema:inRegion ?plm61 .
  ?plm3 schema:brandName ?plm31 .
  FILTER (REGEX (?plm61,"EUROPE" , "i")) &&
      (REGEX (?plm31,"MFGR#2239" , "i"))
  }
GROUP BY ?plm2 ?plm3
```

Figure 55: Query 6 (naïve SPARQL)

```
SELECT ?plm2 ?plm4 ?plm7 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
  ?plm2 qb4o:memberOf schema:year .
  ?o rdfh:lo_custkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
  ?plm3 qb4o:memberOf schema:city . ?plm3 schema:inNation ?plm4 .
  ?plm4 qb4o:memberOf schema:nation . ?plm4 schema:inRegion ?plm5 .
  ?plm5 qb4o:memberOf schema:region .
  ?o rdfh:lo_partkey ?lm3 .
  ?o rdfh:lo_suppkey ?lm4 .
  ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm6 .
  ?plm6 qb4o:memberOf schema:city . ?plm6 schema:inNation ?plm7 .
  ?plm7 qb4o:memberOf schema:nation . ?plm7 schema:inRegion ?plm8 .
  ?plm8 qb4o:memberOf schema:region .
  ?plm5 schema:inRegion ?plm51 . ?plm8 schema:inRegion ?plm81 .
  ?plm2 schema:yearNum ?plm21 .
  FILTER (REGEX (?plm51,"ASIA" , "i")) &&
       (REGEX (?plm81,"ASIA" , "i")) &&
       (?plm21 >= 1992) && (?plm21 <= 1997)
  }
GROUP BY ?plm2 ?plm4 ?plm7
```

Figure 56: Query 7 (naïve SPARQL)

```
SELECT ?plm2 ?plm3 ?plm5 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
 ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
 ?plm2 qb4o:memberOf schema:year .
 ?o rdfh:lo_custkey ?lm2 .
 ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
 ?plm3 qb4o:memberOf schema:city . ?plm3 schema:inNation ?plm4 .
 ?plm4 qb4o:memberOf schema:nation .
 ?o rdfh:lo_partkey ?lm3 .
 ?o rdfh:lo_suppkey ?lm4 .
 ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm5 .
 ?plm5 qb4o:memberOf schema:city . ?plm5 schema:inNation ?plm6 .
 ?plm6 qb4o:memberOf schema:nation .
 ?plm4 schema:nationName> ?plm41 .
 ?plm6 schema:nationName> ?plm61 .
 ?plm2 schema:yearNum ?plm21 .
 FILTER (REGEX (?plm41,"UNITED_STATES" , "i")) &&
      (REGEX (?plm61,"UNITED_STATES" , "i")) &&
      (?plm21 >= 1992) && (?plm21 <= 1997)
 }
GROUP BY ?plm2 ?plm3 ?plm5
```

Figure 57: Query 8 (naïve SPARQL)

```
SELECT ?plm2 ?plm3 ?plm4 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
 ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
 ?plm2 qb4o:memberOf schema:year .
 ?o rdfh:lo_custkey ?lm2 .
 ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
 ?plm3 qb4o:memberOf schema:city .
 ?o rdfh:lo_partkey ?lm3 .
 ?o rdfh:lo_suppkey ?lm4 .
 ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm4 .
 ?plm4 qb4o:memberOf schema:city .
 ?plm3 schema:cityName ?plm31 .
 ?plm4 schema:cityName ?plm41 .
 ?plm2 schema:yearNum ?plm21 .
 FILTER (REGEX (?plm31,"UNITED_KI1" , "i") ||
       REGEX (?plm31,"UNITED_KI5" , "i")) &&
      (REGEX (?plm41,"UNITED_KI1" , "i") ||
       REGEX (?plm41,"UNITED_KI5" , "i")) &&
      (?plm21 >= 1992) && (?plm21 <= 1997)
 }
GROUP BY ?plm2 ?plm3 ?plm4
```

Figure 58: Query 9 (naïve SPARQL)

```
SELECT ?plm1 ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month .
  ?o rdfh:lo_custkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm2 .
  ?plm2 qb4o:memberOf schema:city .
  ?o rdfh:lo_partkey ?lm3 .
  ?o rdfh:lo_suppkey ?lm4 .
  ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm3 .
  ?plm3 qb4o:memberOf schema:city . ?plm3 schema:cityName ?plm31 .
  ?plm2 schema:cityName ?plm21 .
  ?plm1 schema:yearmonthnum> ?plm11 .
  FILTER (((REGEX (?plm31,"UNITED_KI1" , "i") ||
         REGEX (?plm31,"UNITED_KI5" , "i"))) &&
        ((REGEX (?plm21,"UNITED_KI1" , "i") ||
         REGEX (?plm21,"UNITED_KI5" , "i"))) &&
        (?plm11 = 199712))
  }
GROUP BY ?plm1 ?plm2 ?plm3
```

Figure 59: Query 10 (naïve SPARQL)

```
SELECT ?plm2 ?plm4 (SUM(xsd:float(?m4)-xsd:float(?m5)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 . ?o rdfh:lo_supplycost ?m5 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
  ?plm2 qb4o:memberOf schema:year .
  ?o rdfh:lo_custkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
  ?plm3 qb4o:memberOf schema:city . ?plm3 schema:inNation ?plm4 .
  ?plm4 qb4o:memberOf schema:nation . ?plm4 schema:inRegion ?plm5 .
  ?plm5 qb4o:memberOf schema:region .
  ?o rdfh:lo_partkey ?lm3 .
  ?lm3 qb4o:memberOf rdfh:lo_partkey . ?lm3 schema:hasBrand ?plm6 .
  ?plm6 qb4o:memberOf schema:brand . ?plm6 schema:hasCategory ?plm7 .
  ?plm7 qb4o:memberOf schema:category . ?plm7 schema:hasManufacturer ?plm8 .
  ?plm8 qb4o:memberOf schema:manufacturer .
  ?o rdfh:lo_suppkey ?lm4 .
  ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm9 .
  ?plm9 qb4o:memberOf schema:city . ?plm9 schema:inNation ?plm10 .
  ?plm10 qb4o:memberOf schema:nation . ?plm10 schema:inRegion ?plm11 .
  ?plm11 qb4o:memberOf schema:region . ?plm11 schema:inRegion ?plm111 .
  ?plm5 schema:inRegion ?plm51 . ?plm8 schema:manufacturerName ?plm81 .
  FILTER (REGEX (?plm111,"AMERICA" , "i")) && (REGEX (?plm51,"AMERICA" , "i")) &&
       ((REGEX (?plm81,"MFGR#1" , "i") || REGEX (?plm81,"MFGR#2" , "i")))
  }
GROUP BY ?plm2 ?plm4
```

Figure 60: Query 11 (naïve SPARQL)

```
SELECT ?plm2 ?plm7 ?plm10 (SUM(xsd:float(?m4)-xsd:float(?m5)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE { ?o a qb:Observation . ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 . ?o rdfh:lo_supplycost ?m5 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
  ?plm2 qb4o:memberOf schema:year .
  ?o rdfh:lo_custkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
  ?plm3 qb4o:memberOf schema:city . ?plm3 schema:inNation ?plm4 .
  ?plm4 qb4o:memberOf schema:nation . ?plm4 schema:inRegion ?plm5 .
  ?plm5 qb4o:memberOf schema:region .
  ?o rdfh:lo_partkey ?lm3 .
  ?lm3 qb4o:memberOf rdfh:lo_partkey . ?lm3 schema:hasBrand ?plm6 .
  ?plm6 qb4o:memberOf schema:brand . ?plm6 schema:hasCategory ?plm7 .
  ?plm7 qb4o:memberOf schema:category . ?plm7 schema:hasManufacturer ?plm8 .
  ?plm8 qb4o:memberOf schema:manufacturer .
  ?o rdfh:lo_suppkey ?lm4 .
  ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm9 .
  ?plm9 qb4o:memberOf schema:city . ?plm9 schema:inNation ?plm10 .
  ?plm10 qb4o:memberOf schema:nation . ?plm10 schema:inRegion ?plm11 .
  ?plm11 qb4o:memberOf schema:region . ?plm11 schema:inRegion ?plm111 .
  ?plm5 schema:inRegion ?plm51 .
  ?plm8 schema:manufacturerName ?plm81 . ?plm2 schema:yearNum ?plm21 .
  FILTER (REGEX (?plm111,"AMERICA" , "i")) && (REGEX (?plm51,"AMERICA" , "i")) &&
       ((REGEX (?plm81,"MFGR#1" , "i") || REGEX (?plm81,"MFGR#2" , "i"))) &&
       (?plm21 >= 1997) && (?plm21 <= 1998)
} GROUP BY ?plm2 ?plm7 ?plm10
```

Figure 61: Query 12 (naïve SPARQL)

```
SELECT ?plm2 ?plm6 ?plm8 (SUM(xsd:float(?m4)-xsd:float(?m5)) as ?ag1)
FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE { ?o a qb:Observation . ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 . ?o rdfh:lo_supplycost ?m5 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?lm1 qb4o:memberOf rdfh:lo_orderdate . ?lm1 schema:dateInMonth ?plm1 .
  ?plm1 qb4o:memberOf schema:month . ?plm1 schema:monthInYear ?plm2 .
  ?plm2 qb4o:memberOf schema:year .
  ?o rdfh:lo_custkey ?lm2 .
  ?lm2 qb4o:memberOf rdfh:lo_custkey . ?lm2 schema:inCity ?plm3 .
  ?plm3 qb4o:memberOf schema:city . ?plm3 schema:inNation ?plm4 .
  ?plm4 qb4o:memberOf schema:nation . ?plm4 schema:inRegion ?plm5 .
  ?plm5 qb4o:memberOf schema:region .
  ?o rdfh:lo_partkey ?lm3 .
  ?lm3 qb4o:memberOf rdfh:lo_partkey . ?lm3 schema:hasBrand ?plm6 .
  ?plm6 qb4o:memberOf schema:brand . ?plm6 schema:hasCategory ?plm7 .
  ?plm7 qb4o:memberOf schema:category .
  ?o rdfh:lo_suppkey ?lm4 .
  ?lm4 qb4o:memberOf rdfh:lo_suppkey . ?lm4 schema:inCity ?plm8 .
  ?plm8 qb4o:memberOf schema:city . ?plm8 schema:inNation ?plm9 .
  ?plm9 qb4o:memberOf schema:nation . ?plm9 schema:nationName> ?plm91 .
  ?plm5 schema:inRegion ?plm51 .
  ?plm7 schema:categoryName ?plm71 . ?plm2 schema:yearNum ?plm21 .
  FILTER (REGEX (?plm91,"UNITED_STATES" , "i")) &&
        (REGEX (?plm51,"AMERICA" , "i")) && (REGEX (?plm71,"MFGR#14" , "i")) &&
        (?plm21 >= 1997) && (?plm21 <= 1998)
} GROUP BY ?plm2 ?plm8 ?plm6
```

Figure 62: Query 13 (naïve SPARQL)

C.3 IMPROVED SPARQL QUERIES

The improved SPARQL queries, for each of the 19 evaluations scenarios, are available online[1]. In this section, we present the queries that correspond to our best scenario (ES11). (Figures 63 to 75). The prefixes used in these queries are the same as in the naïve case (Figure 49).

```
SELECT (?ag1 * ?ag2) as ?sum_revenue
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{ ?plm2 schema:yearNum> 1993 }.
{  SELECT ?plm2 (SUM(xsd:float(?m2)) as ?ag1) (SUM(xsd:float(?m3)) as ?ag2)
 WHERE {
 {GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_quantity ?m1 .
  ?o rdfh:lo_discount ?m2 .
  ?o rdfh:lo_extendedprice ?m3 .
  ?o rdfh:lo_orderdate ?lm1
  FILTER (((?m1 <= 24))&&(((?m2 >= 1) && (?m2 <= 3))))
 }}.
 {GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>{
  ?plm2 qb4o:memberOf schema:year.
  ?plm1 schema:monthInYear ?plm2 .?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 .?lm1 qb4o:memberOf rdfh:lo_orderdate .
 }}}
 GROUP BY ?plm2
}}
```

Figure 63: Query 1 (from improvement scenario ES11)

```
SELECT (?ag1 * ?ag2) as ?sum_revenue
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {{ ?plm1 schema:yearmonthnum 199401 }.
{ SELECT ?plm1 (SUM(xsd:float(?m2)) as ?ag1) (SUM(xsd:float(?m3)) as ?ag2)
WHERE {
 {GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_quantity ?m1 .
  ?o rdfh:lo_discount ?m2 .
  ?o rdfh:lo_extendedprice ?m3 .
  ?o rdfh:lo_orderdate ?lm1 .
  FILTER ((((?m1 >= 26) && (?m1 <= 35)))&&(((?m2 >= 4) && (?m2 <= 6))))
 }}.
 {GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>{
  ?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .
 }}}
GROUP BY ?plm1
}}
```

Figure 64: Query 2 (from improvement scenario ES11)

1 https://github.com/lorenae/ssb-qb4olap/tree/master/ssb_qb4olap

```
SELECT (?ag1 * ?ag2) as ?sum_revenue
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{ ?plm1 schema:yearweeknum> 19946}.
{ SELECT ?plm1 (SUM(xsd:float(?m2)) as ?ag1) (SUM(xsd:float(?m3)) as ?ag2)
 WHERE {
 {GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_quantity ?m1 .
  ?o rdfh:lo_discount ?m2 .
  ?o rdfh:lo_extendedprice ?m3 .
  ?o rdfh:lo_orderdate ?lm1 .
  FILTER ((((?m1 >= 26) && (?m1 <= 35)))&&(((?m2 >= 5) && (?m2 <= 7))))
 }}.
 {GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm1 qb4o:memberOf schema:week .
  ?lm1 schema:dateInWeek ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .
  }}
 }
GROUP BY ?plm1
}}
```

Figure 65: Query 3 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?o rdfh:lo_partkey ?lm2 .
 ?o rdfh:lo_suppkey ?lm3
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
 ?plm2 qb4o:memberOf schema:year .
 ?plm1 schema:monthInYear ?plm2 .?plm1 qb4o:memberOf schema:month .
 ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

 ?plm4 schema:categoryName "MFGR#12" .?plm4 qb4o:memberOf schema:category .
 ?plm3 schema:hasCategory ?plm4 .?plm3 qb4o:memberOf schema:brand .
 ?lm2 schema:hasBrand ?plm3 .?lm2 qb4o:memberOf rdfh:lo_partkey .

 ?plm7 schema:regionName "AMERICA" .?plm7 qb4o:memberOf schema:region .
 ?plm6 schema:inRegion ?plm7 .?plm6 qb4o:memberOf schema:nation .
 ?plm5 schema:inNation ?plm6 .?plm5 qb4o:memberOf schema:city .
 ?lm3 schema:inCity ?plm5 .?lm3 qb4o:memberOf rdfh:lo_suppkey .
}}}
GROUP BY ?plm2 ?plm3
```

Figure 66: Query 4 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?o rdfh:lo_partkey ?lm2 .
  ?o rdfh:lo_suppkey ?lm3
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm2 qb4o:memberOf schema:year .
  ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

  ?plm3 schema:brandName ?plm31 .?plm3 qb4o:memberOf schema:brand .
  ?lm2 schema:hasBrand ?plm3 .?lm2 qb4o:memberOf rdfh:lo_partkey .

  ?plm6 schema:regionName "ASIA" .?plm6 qb4o:memberOf schema:region .
  ?plm5 schema:inRegion ?plm6 .?plm5 qb4o:memberOf schema:nation .
  ?plm4 schema:inNation ?plm5 .?plm4 qb4o:memberOf schema:city .
  ?lm3 schema:inCity ?plm4 . ?lm3 qb4o:memberOf rdfh:lo_suppkey .
  FILTER ((str(?plm31)>="MFGR#2221") && (str(?plm31)<="MFGR#2228"))
}}
}
GROUP BY ?plm2 ?plm3
```

Figure 67: Query 5 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?o rdfh:lo_partkey ?lm2 .
  ?o rdfh:lo_suppkey ?lm3
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm2 qb4o:memberOf schema:year .
  ?plm1 schema:monthInYear ?plm2 .?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

  ?plm3 schema:brandName "MFGR#2239" .?plm3 qb4o:memberOf schema:brand .
  ?lm2 schema:hasBrand ?plm3 .?lm2 qb4o:memberOf rdfh:lo_partkey .

  ?plm6 schema:regionName "EUROPE" .?plm6 qb4o:memberOf schema:region .
  ?plm5 schema:inRegion ?plm6 .?plm5 qb4o:memberOf schema:nation .
  ?plm4 schema:inNation ?plm5 .?plm4 qb4o:memberOf schema:city .
  ?lm3 schema:inCity ?plm4 . ?lm3 qb4o:memberOf rdfh:lo_suppkey .
}}}
GROUP BY ?plm2 ?plm3
```

Figure 68: Query 6 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm4 ?plm7 (SUM(xsd:float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?o rdfh:lo_custkey ?lm2 .
 ?o rdfh:lo_suppkey ?lm4
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
 ?plm2 schema:yearNum> ?plm21 . ?plm2 qb4o:memberOf schema:year .
 ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
 ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

 ?plm5 schema:regionName "ASIA" . ?plm5 qb4o:memberOf schema:region .
 ?plm4 schema:inRegion ?plm5 . ?plm4 qb4o:memberOf schema:nation .
 ?plm3 schema:inNation ?plm4 . ?plm3 qb4o:memberOf schema:city .
 ?lm2 schema:inCity ?plm3 . ?lm2 qb4o:memberOf rdfh:lo_custkey .

 ?plm8 schema:regionName "ASIA" . ?plm8 qb4o:memberOf schema:region .
 ?plm7 schema:inRegion ?plm8 . ?plm7 qb4o:memberOf schema:nation .
 ?plm6 schema:inNation ?plm7 . ?plm6 qb4o:memberOf schema:city .
 ?lm4 schema:inCity ?plm6 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .
 FILTER ((?plm21 >= 1992)&&(?plm21 <= 1997)) }}
 }
GROUP BY ?plm2 ?plm4 ?plm7
```

Figure 69: Query 7 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm3 ?plm5 (SUM(xsd:float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
 ?o a qb:Observation .
 ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 .
 ?o rdfh:lo_orderdate ?lm1 .
 ?o rdfh:lo_custkey ?lm2 .
 ?o rdfh:lo_suppkey ?lm4
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
 ?plm2 schema:yearNum> ?plm21 . ?plm2 qb4o:memberOf schema:year .
 ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
 ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

 ?plm4 schema:nationName "UNITED_STATES" . ?plm4 qb4o:memberOf schema:nation .
 ?plm3 schema:inNation ?plm4 . ?plm3 qb4o:memberOf schema:city .
 ?lm2 schema:inCity ?plm3 . ?lm2 qb4o:memberOf rdfh:lo_custkey .

 ?plm6 schema:nationName "UNITED_STATES" . ?plm6 qb4o:memberOf schema:nation .
 ?plm5 schema:inNation ?plm6 . ?plm5 qb4o:memberOf schema:city .
 ?lm4 schema:inCity ?plm5 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .
 FILTER ((?plm21 >= 1992)&&(?plm21 <= 1997))
}}}
GROUP BY ?plm2 ?plm3 ?plm5
```

Figure 70: Query 8 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm3 ?plm4 (SUM(float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?o rdfh:lo_custkey ?lm2 .
  ?o <rdfh:lo_suppkey ?lm4
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm2 schema:yearNum> ?plm21 . ?plm2 qb4o:memberOf schema:year .
  ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

  ?plm3 schema:cityName ?plm31 . ?plm3 qb4o:memberOf schema:city .
  ?lm2 schema:inCity ?plm3 . ?lm2 qb4o:memberOf rdfh:lo_custkey .
  ?plm3 schema:cityName ?plm31.
  VALUES ?plm31 {"UNITED_KI1" "UNITED_KI5" }.

  ?plm4 schema:cityName ?plm41 . ?plm4 qb4o:memberOf schema:city .
  ?lm4 schema:inCity ?plm4 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .
  ?plm4 schema:cityName ?plm41.
  VALUES ?plm41 {"UNITED_KI1" "UNITED_KI5" }.
  FILTER ((?plm21 >= 1992)&&(?plm21 <= 1997))
}}}
GROUP BY ?plm2 ?plm3 ?plm4
```

Figure 71: Query 9(from improvement scenario ES11)

```
SELECT ?plm1 ?plm2 ?plm3 (SUM(xsd:float(?m4)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation .
  ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 .
  ?o rdfh:lo_orderdate ?lm1 .
  ?o rdfh:lo_custkey ?lm2 .
  ?o rdfh:lo_suppkey ?lm4
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm1 schema:yearmonthnum 199712 . ?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

  ?plm2 schema:cityName ?plm21 . ?plm2 qb4o:memberOf schema:city .
  ?lm2 schema:inCity ?plm2 . ?lm2 qb4o:memberOf rdfh:lo_custkey .
  ?plm2 schema:cityName ?plm21.
  VALUES ?plm21 {"UNITED_KI1" "UNITED_KI5" }.

  ?plm3 schema:cityName ?plm31 . ?plm3 qb4o:memberOf schema:city .
  ?lm4 schema:inCity ?plm3 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .
  ?plm3 schema:cityName ?plm31.
  VALUES ?plm31 {"UNITED_KI1" "UNITED_KI5" }.
 }}}
GROUP BY ?plm1 ?plm2 ?plm3
```

Figure 72: Query 10 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm4 (SUM(xsd:float(?m4)-xsd:float(?m5)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
 {GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation . ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 . ?o rdfh:lo_supplycost ?m5 .
  ?o rdfh:lo_orderdate ?lm1 . ?o rdfh:lo_custkey ?lm2 .
  ?o rdfh:lo_partkey ?lm3 . ?o rdfh:lo_suppkey ?lm4 .
 }}.
 {GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm2 schema:yearNum> ?plm21 . ?plm2 qb4o:memberOf schema:year .
  ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

  ?plm5 schema:regionName "AMERICA" . ?plm5 qb4o:memberOf schema:region .
  ?plm4 schema:inRegion ?plm5 . ?plm4 qb4o:memberOf schema:nation .
  ?plm3 schema:inNation ?plm4 . ?plm3 qb4o:memberOf schema:city .
  ?lm2 schema:inCity ?plm3 . ?lm2 qb4o:memberOf rdfh:lo_custkey .

  ?plm11 schema:regionName "AMERICA" .?plm11 qb4o:memberOf schema:region .
  ?plm10 schema:inRegion ?plm11 . ?plm10 qb4o:memberOf schema:nation .
  ?plm9 schema:inNation ?plm10 . ?plm9 qb4o:memberOf schema:city .
  ?lm4 schema:inCity ?plm9 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .

  ?plm8 schema:manufacturerName ?plm81 .
  ?plm8 qb4o:memberOf schema:manufacturer .
  ?plm7 schema:hasManufacturer ?plm8 . ?plm7 qb4o:memberOf schema:category .
  ?plm6 schema:hasCategory ?plm7 . ?plm6 qb4o:memberOf schema:brand .
  ?lm3 schema:hasBrand ?plm6 . ?lm3 qb4o:memberOf rdfh:lo_partkey .
 VALUES ?plm81 {"MFGR#1" "MFGR#2" } }}}
GROUP BY ?plm2 ?plm4
```

Figure 73: Query 11 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm7 ?plm10 (SUM(xsd:float(?m4)-xsd:float(?m5)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
  ?o a qb:Observation . ?o qb:dataSet rdfh-inst:ds .
  ?o rdfh:lo_revenue ?m4 . ?o rdfh:lo_supplycost ?m5 .
  ?o rdfh:lo_orderdate ?lm1 . ?o rdfh:lo_custkey ?lm2 .
  ?o rdfh:lo_partkey ?lm3 . ?o rdfh:lo_suppkey ?lm4 .
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
  ?plm2 schema:yearNum> ?plm21 . ?plm2 qb4o:memberOf schema:year .
  ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
  ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .

  ?plm5 schema:regionName "AMERICA" . ?plm5 qb4o:memberOf schema:region .
  ?plm4 schema:inRegion ?plm5 . ?plm4 qb4o:memberOf schema:nation .
  ?plm3 schema:inNation ?plm4 . ?plm3 qb4o:memberOf schema:city .
  ?lm2 schema:inCity ?plm3 . ?lm2 qb4o:memberOf rdfh:lo_custkey .

  ?plm11 schema:regionName "AMERICA" . ?plm11 qb4o:memberOf schema:region .
  ?plm10 schema:inRegion ?plm11 . ?plm10 qb4o:memberOf schema:nation .
  ?plm9 schema:inNation ?plm10 . ?plm9 qb4o:memberOf schema:city .
  ?lm4 schema:inCity ?plm9 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .

  ?plm8 schema:manufacturerName ?plm81 . ?plm8 qb4o:memberOf schema:manufacturer .
  ?plm7 schema:hasManufacturer ?plm8 . ?plm7 qb4o:memberOf schema:category .
  ?plm6 schema:hasCategory ?plm7 . ?plm6 qb4o:memberOf schema:brand .
  ?lm3 schema:hasBrand ?plm6 . ?lm3 qb4o:memberOf rdfh:lo_partkey .
  ?plm8 schema:manufacturerName ?plm81.
  VALUES ?plm81 {"MFGR#1" "MFGR#2" }. FILTER (?plm21 >= 1997)&&(?plm21 <= 1998) }}}
GROUP BY ?plm2 ?plm7 ?plm10
```

Figure 74: Query 12 (from improvement scenario ES11)

```
SELECT ?plm2 ?plm6 ?plm8
(SUM(xsd:float(?m4)-xsd:float(?m5)) as ?ag1)
FROM NAMED <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
FROM NAMED <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
WHERE {
{GRAPH <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap> {
 ?o a qb:Observation . ?o qb:dataSet rdfh-inst:ds .
 ?o rdfh:lo_revenue ?m4 . ?o rdfh:lo_supplycost ?m5 .
 ?o rdfh:lo_orderdate ?lm1 . ?o rdfh:lo_custkey ?lm2 .
 ?o rdfh:lo_partkey ?lm3 . ?o rdfh:lo_suppkey ?lm4 .
}}.
{GRAPH <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap> {
 ?plm2 schema:yearNum> ?plm21 . ?plm2 qb4o:memberOf schema:year .
 ?plm1 schema:monthInYear ?plm2 . ?plm1 qb4o:memberOf schema:month .
 ?lm1 schema:dateInMonth ?plm1 . ?lm1 qb4o:memberOf rdfh:lo_orderdate .
 ?plm2 schema:yearNum> ?plm21 .

 ?plm5 schema:regionName "AMERICA" . ?plm5 qb4o:memberOf schema:region .
 ?plm4 schema:inRegion ?plm5 . ?plm4 qb4o:memberOf schema:nation .
 ?plm3 schema:inNation ?plm4 . ?plm3 qb4o:memberOf schema:city .
 ?lm2 schema:inCity ?plm3 . ?lm2 qb4o:memberOf rdfh:lo_custkey .

 ?plm9 schema:nationName "UNITED_STATES" . ?plm9 qb4o:memberOf schema:nation .
 ?plm8 schema:inNation ?plm9 . ?plm8 qb4o:memberOf schema:city .
 ?lm4 schema:inCity ?plm8 . ?lm4 qb4o:memberOf rdfh:lo_suppkey .

 ?plm7 schema:categoryName "MFGR#14" . ?plm7 qb4o:memberOf schema:category .
 ?plm6 schema:hasCategory ?plm7 . ?plm6 qb4o:memberOf schema:brand .
 ?lm3 schema:hasBrand ?plm6 . ?lm3 qb4o:memberOf rdfh:lo_partkey .
 FILTER ((?plm21 >= 1997)&&(?plm21 <= 1998))}
}}
GROUP BY ?plm2 ?plm8 ?plm6
```

Figure 75: Query 13 (from improvement scenario ES11)

BIBLIOGRAPHY

[1] Alberto Abelló, Jérôme Darmont, Lorena Etcheverry, Matteo Golfarelli, Jose-Norberto Mazón, Felix Naumann, Torben Bach Pedersen, Stefano Rizzi, Juan Trujillo, Panos Vassiliadis, and Gottfried Vossen. "Fusion Cubes: Towards Self-Service Business Intelligence." In: *IJDWM* 9.2 (2013), pp. 66–88 (cit. on pp. 17, 18).

[2] Alberto Abelló, Oscar Romero, Torben B. Pedersen, Rafael Berlanga, Victoria Nebot, María J. Aramburu, and Alkis Simitsis. "Using Semantic Web Technologies for Exploratory OLAP: A Survey." In: *IEEE Trans. Knowl. Data Eng.* 27.2 (2015), pp. 571–588 (cit. on pp. 17, 18).

[3] Elham Akbari Azirani, François Goasdoué, Ioana Manolescu, and Alexandra Roatis. "Efficient OLAP operations for RDF analytics." In: *31st IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2015, Seoul, South Korea, April 13-17, 2015*. 2015, pp. 71–76 (cit. on pp. 102–105).

[4] Dave Beckett and Tim Berners-Lee. *RDF 1.1 Turtle*. 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/ (cit. on p. 14).

[5] Dan Brickley and Ramanathan Guha. *RDF Schema 1.1*. 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/ (cit. on pp. 1, 14).

[6] Cristina Ciferri, Ricardo Ciferri, Leticia Gómez, Markus Schneider, Alejandro A. Vaisman, and Esteban Zimányi. "Cube Algebra: A Generic User-Centric Model and Query Language for OLAP Cubes." In: *IJDWM* 9.2 (2013), pp. 39–65 (cit. on pp. 62, 67).

[7] Dario Colazzo, François Goasdoué, Ioana Manolescu, and Alexandra Roatiş. "RDF Analytics: Lenses over Semantic Graphs." In: *Proceedings of the 23rd International Conference on World Wide Web*. 2014, pp. 467–478 (cit. on pp. 100, 102, 104, 105).

[8] Richard Cyganiak and Dave Reynolds. *The RDF Data Cube Vocabulary (W3C Recommendation)*. 2014. URL: http://www.w3.org/TR/vocab-data-cube/ (cit. on pp. 3, 19).

[9] Souripriya Das, Seema Sundara, and Richard Cyganiak. *R2RML: RDB to RDF Mapping Language*. 2012. URL: http://www.w3.org/TR/r2rml/ (cit. on p. 15).

[10] Johan Montagnat Franck Michel and Catherine Faron-Zucker. *A survey of RDB to RDF translation approaches and tools*. 2013. URL: https://hal.archives-ouvertes.fr/hal-00903568 (cit. on p. 55).

[11] Leticia Gómez, Silvia Gómez, and Alejandro A. Vaisman. "A generic data model and query language for spatiotemporal OLAP cube analysis." In: *Proceedings of EDBT*. ACM. 2012, pp. 300–311 (cit. on pp. 1, 11).

[12] Silvia Gomez. "Un Modelo y Lenguaje de Consulta Genérico para el Procesamiento Analítico Online y su Aplicación a Campos de Datos Continuos." PhD thesis. Instituto Tecnológico de Buenos Aires, 2014 (cit. on p. 69).

[13] Steve Harris and Andy Seaborne. *SPARQL 1.1 Query Language*. 2013. URL: http://www.w3.org/TR/sparql11-query/ (cit. on pp. 1, 15).

[14] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool Publishers, 2011, pp. 1–136. URL: http://linkeddatabook.com/ (cit. on p. 1).

[15] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. *OWL 2 Web Ontology Language Primer*. 2012. URL: http://www.w3.org/TR/2012/REC-owl2-primer-20121211/ (cit. on p. 1).

[16] Carlos A. Hurtado, Alberto O. Mendelzon, and Alejandro A. Vaisman. "Maintaining Data Cubes under Dimension Updates." In: *Proceedings of the 15th International Conference on Data Engineering*. 1999, pp. 346–355 (cit. on pp. 1, 11).

[17] Dilshod Ibragimov, Katja Hose, Torben Bach Pedersen, and Esteban Zimányi. "Towards Exploratory OLAP Over Linked Open Data - A Case Study." In: *Enabling Real-Time Business Intelligence - International Workshops, BIRTE 2013, Riva del Garda, Italy, August 26, 2013, and BIRTE 2014, Hangzhou, China, September 1, 2014, Revised Selected Papers*. Vol. 206. 2014, pp. 114–132 (cit. on pp. 18, 104).

[18] Kim A. Jakobsen, Alex B. Andersen, Katja Hose, and Torben Bach Pedersen. "Optimizing RDF Data Cubes for Efficient Processing of Analytical Queries." In: *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, US, October 12th, 2015*. Vol. 1426. 2015 (cit. on pp. 104, 105).

[19] Evangelos Kalampokis, Bill Roberts, Areti Karamanou, Efthimios Tambouris, and Konstantinos A. Tarabanis. "Challenges on Developing Tools for Exploiting Linked Open Data Cubes." In: *Proceedings of the 3rd International Workshop on Semantic Statistics co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, Pennsylvania, USA, October 11th, 2015*. Vol. 1551. 2015. URL: http://ceur-ws.org/Vol-1551/article-07.pdf (cit. on pp. 104, 105, 109).

[20] Benedikt Kämpgen and Andreas Harth. "Transforming statistical linked data for use in OLAP systems." In: *Proceedings of ICSS*. 2011, pp. 33–40 (cit. on pp. 102, 104, 105, 108).

[21] Benedikt Kämpgen and Andreas Harth. "No Size Fits All - Running the Star Schema Benchmark with SPARQL and RDF Aggregate Views." In: *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*. Vol. 7882. 2013, pp. 290–304 (cit. on pp. 87, 92, 99, 102, 104, 105, 108).

[22] Benedikt Kämpgen and Andreas Harth. "OLAP4LD - A Framework for Building Analysis Applications Over Governmental Statistics." In: *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*. Vol. 8798. 2014, pp. 389–394 (cit. on pp. 102, 104, 105, 108).

[23] Benedikt Kämpgen, Seán O'Riain, and Andreas Harth. "Interacting with Statistical Linked Data via OLAP Operations." In: *The Semantic Web: ESWC 2012 Satellite Events - ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012. Revised Selected Papers*. Vol. 7540. 2012, pp. 87–101 (cit. on pp. 99, 102, 104, 105, 108).

[24] Benedikt Kämpgen, Steffen Stadtmüller, and Andreas Harth. "Querying the Global Cube: Integration of Multidimensional Datasets from the Web." In: *Knowledge Engineering and Knowledge Management - 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014. Proceedings*. Vol. 8876. 2014, pp. 250–265 (cit. on pp. 102, 104, 105, 108).

[25] Bart Kuijpers and Alejandro Vaisman. "An Algebra for OLAP (Submitted." In: (2016) (cit. on pp. 63, 67).

[26] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. "Static analysis and optimization of semantic web queries." In: *ACM TODS* 38.4 (2013), p. 25 (cit. on p. 80).

[27] Antonis Loizou, Renzo Angles, and Paul T. Groth. "On the formulation of performant SPARQL queries." In: *J. Web Sem.* 31 (2015), pp. 1–26 (cit. on pp. 80, 81, 90, 93).

[28] Alexander Löser, Fabian Hueske, and Volker Markl. "Situational Business Intelligence." In: *Business Intelligence for the Real-Time Enterprise*. Vol. 27. 2009, pp. 1–11 (cit. on p. 17).

[29] Elzbieta Malinowski and Esteban Zimányi. *Advanced data warehouse design: From conventional to spatial and temporal applications*. Springer, 2008 (cit. on p. 4).

[30] Pat O Neil, Betty O Neil, and Xuedong Chen. *Star Schema Benchmark*. 2009. URL: http://www.cs.umb.edu/{~}poneil/StarSchemaB.PDF (cit. on p. 87).

[31] Andriy Nikolov. *Report on the OpenCube Toolkit and Platforms Prototype*. 2014. URL: http://opencube-project.eu/sites/default/files/OpenCube_D32_Report%20on%20the%20OpenCube%20Tookit%20and%20Platforms%20Prototype_v2.pdf (cit. on pp. 104, 105).

[32] J. Pérez, M. Arenas, and C. Gutierrez. "Semantics and Complexity of SPARQL." In: *ACM Transactions on Database Systems (TODS)* 34.3 (2009), pp. 1–45 (cit. on pp. 81, 84).

[33] Markus Lanthaler Richard Cyganiak David Wood. *RDF 1.1 Concepts and Abstract Syntax*. 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (cit. on pp. 1, 13).

[34] SDMX. *Content Oriented Guidelines*. 2009. URL: http://sdmx.org/?page_id=11 (cit. on p. 21).

[35] SDMX. *SDMX Standards: Information Model*. 2011. URL: http://sdmx.org/wp-content/uploads/2011/08/SDMX_2-1-1_SECTION_2_InformationModel_201108.pdf (cit. on p. 12).

[36] Michael Schmidt, Michael Meier, and Georg Lausen. "Foundations of SPARQL query optimization." In: *Proceedings of ICDT*. 2010, pp. 4–33 (cit. on p. 84).

[37] Arie Shoshani. "OLAP and Statistical Databases: Similarities and Differences." In: *PODS*. 1997, pp. 185–196 (cit. on pp. 1, 13).

[38] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. "SPARQL basic graph pattern optimization using selectivity estimation." In: *Proceedings of WWW*. ACM. 2008, pp. 595–604 (cit. on p. 83).

[39] TPC.org. *TPC-H Benchmark*. 2014. URL: http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpch2.17.1.pdf (cit. on p. 90).

[40] Petros Tsialiamanis, Lefteris Sidirourgos, Irini Fundulaki, Vassilis Christophides, and Peter Boncz. "Heuristics-based query optimisation for SPARQL." In: *Proceedings of EDBT*. ACM. 2012, pp. 324–335 (cit. on p. 80).

[41] Alejandro Vaisman and Esteban Zimányi. "Data Warehouses: Next Challenges." In: *Business Intelligence*. Vol. 96. 2012, pp. 1–26 (cit. on p. 17).

[42] Alejandro Vaisman and Esteban Zimányi. *Data Warehouse Systems: Design and Implementation*. Springer, 2014 (cit. on pp. 4–6, 11, 23, 27, 29, 31, 48).

[43] Jovan Varga, Alejandro A. Vaisman, Oscar Romero, Lorena Etcheverry, Torben Bach Pedersen, and Christian Thomsen. "Dimensional Enrichment of Statistical Linked Open Data." In: (). Submitted to the Journal of Web Semantics (December 2015) (cit. on pp. 18, 56).

[44] Jovan Varga, Lorena Etcheverry, Alejandro Vaisman, Oscar Romero, Torben Pedersen, and Christian Thomsen. "QB2OLAP: Enabling OLAP on Statistical Linked Open Data." In: *32nd IEEE International Conference on Data Engineering ICDE 2016, Helsinki, Finland, May 16-20, 2016. Demo paper* (cit. on p. 57).

[45] Panos Vassiliadis. "Modeling Multidimensional Databases, Cubes and Cube Operations." In: *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*. 1998, pp. 53–62 (cit. on pp. 1, 11).

[46] Rob Vesse. *SPARQL Optimization 101, Tutorial at ApacheCon North America 2014*. 2014. URL: http://events.linuxfoundation.org/sites/events/files/slides/SPARQL%20Optimisation%20101%20Tutorial.pdf (cit. on pp. 80, 82, 90).

DECLARATION

Put your declaration here.

Montevideo, Uruguay, July 2016

Lorena Etcheverry Venturini

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/