


Efficient Analytical Queries on Semantic Web Data Cubes

Lorena Etcheverry¹  · Alejandro A. Vaisman²

Received: 2 September 2017 / Accepted: 13 October 2017 / Published online: 27 October 2017
© Springer-Verlag GmbH Germany 2017

Abstract The amount of multidimensional data published on the Semantic Web (SW) is constantly increasing, due to initiatives such as Open Data and Open Government Data, among others. Models, languages, and tools, that allow obtaining valuable information efficiently, are thus required. Multidimensional data are typically represented as data cubes and exploited using online analytical processing (OLAP) techniques. The RDF Data Cube Vocabulary, also denoted QB, is the current W3C standard to represent statistical data on the SW. Given that QB does not include key features needed for OLAP analysis, in previous work we have proposed an extension, denoted QB4OLAP, to overcome this problem without the need of modifying already published data. Once data cubes are appropriately represented on the SW, we need mechanisms to analyze them. However, in the current state-of-the-art, writing efficient analytical queries over SW data cubes demands a deep knowledge of standards like RDF and SPARQL. These skills are unlikely to be found in typical analytical users. Further, OLAP languages like MDX are far from being easily understood by the final user. The lack of friendly tools to exploit multidimensional data on the SW is a barrier that needs to be broken to promote the publication of such data. This is the problem we address in this paper. 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 techni-

calities. For this, we devised CQL (standing for Cube Query Language), a simple, high-level query language that operates over data cubes. Taking advantage of structural metadata provided by QB4OLAP, we translate CQL queries into SPARQL ones. Then, we propose query improvement strategies to produce efficient SPARQL queries, adapting general-purpose SPARQL query optimization techniques. We evaluate our implementation using the Star Schema benchmark, showing that our proposal outperforms others. The *QB4OLAP toolkit*, a web application that allows exploring and querying (using CQL) SW data cubes, completes our contributions.

Keywords Multidimensional Data Modeling · OLAP · Linked Open Data · Semantic Web

1 Introduction

Data warehouses (DW) integrate multiple data sources for analysis and decision support, representing data using the multidimensional (MD) model. This model organizes data in MD data cubes, where hierarchical *dimensions* represent the perspectives that characterize *facts*. The latter are usually associated with quantitative data, also known as *measures*. Data cube measures can be aggregated, disaggregated, and filtered using dimensions, and this process is called online analytical processing (OLAP).

DW and OLAP had been typically used as techniques for data analysis *within* organizations, based on high-quality internal data, and mostly using commercial tools with proprietary formats. However, initiatives such as Open Data¹ and Open Government Data² are encouraging organizations

✉ Lorena Etcheverry
lorenae@fing.edu.uy

Alejandro A. Vaisman
avaisman@itba.edu.ar

¹ Instituto de Computación, Facultad de Ingeniería, UdelAR, Ave Julio Herrera y Reissig 565, Montevideo, Uruguay

² Instituto Tecnológico de Buenos Aires, 25 de Mayo 457, Buenos Aires, Argentina

¹ <http://okfn.org/opendata/>.

² <http://opengovdata.org/>.

to publish and share MD data on the web. In addition, the *Linked Data* (LD) paradigm promotes a set of best practices for publishing and interlinking structured data on the web, using standards, like RDF³, and SPARQL.⁴ At the time of writing this paper, the amount of open data available as LD is approximately 90 billion triples in over 3,300 data sets, most of them freely accessible via SPARQL query endpoints.⁵ However, LD recommendations focus on the representation of relational data, but they are insufficient to represent other data models, in particular MD data.

In this new context, the business intelligence (BI) community faces several challenges. 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 Semantic Web (SW) standards. Second, it is necessary to provide mechanisms to analyze SW data *à la* OLAP. Regarding the first challenge, the *RDF Data Cube Vocabulary* [1] (QB) is the current W3C standard to represent statistical data following LD principles. There are already a considerable number of data sets published using QB. However, this vocabulary does not include key features needed for OLAP analysis, like dimensional hierarchies and aggregate functions. To address this problem, in previous work, we have proposed a new vocabulary called QB4OLAP [2,3], which extends QB in order to overcome these limitations. QB4OLAP also allows reusing data already published in QB, just by adding the needed MD schema semantics, and the corresponding data instances.

The work we present in this paper is aimed at tackling the second challenge above. To this end, we propose a high-level query language for OLAP, denoted CQL, where the *main data type is the data cube*. Our approach is based on a clear separation between the conceptual and the logical levels, a feature that is not common in traditional OLAP systems, where popular OLAP query and analysis languages, such as MDX,⁶ operate at the logical level and require, in order to be able to write queries, the user's deep understanding of how data are actually stored [4]. To achieve this separation, we start defining a data model for MD data cubes, and an algebra (which is a subset of the so-called Cube Algebra proposed in [4]), composed of a collection of operators, with a clearly defined semantics. This algebra will be the basis of our high-level OLAP query language, denoted *CQL* (standing for *Cube Query Language*), and is composed of a collection of operations that manipulate a data cube, which is the only kind of object that the user will be aware of. The user will thus write her queries at the conceptual level using CQL, and we provide mechanisms to translate these 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 *directly* over QB4OLAP cubes on the SW, without dealing with RDF or SPARQL technicalities. Note that, in general, OLAP users know how to manipulate a data cube through the typical roll-up, drill-down, and slice-dice operations, but it is unlikely that they would be familiar with SPARQL or the SW. Also, SPARQL optimization tips and best practices could be incorporated into the CQL to SPARQL translation process, to produce efficient queries, not an easy task for an average user. On the other hand, SW users know SPARQL and RDF very well, but the cube metaphor may help them to perform analytical queries easier and more intuitively than operating directly over the RDF representation.

More concretely, as our *first contribution*, we present a data model for OLAP and propose an algebra and a high-level query language based on it, namely CQL, where the main data type is the data cube. The semantics of the algebra operators is clearly defined using the notion of a *lattice of cuboids*, which is used for query processing and rewriting.

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

The remainder of this paper is organized as follows. Section 2 presents the running example we will use in this work. Section 3 briefly sketches the QB4OLAP vocabulary. Section 4 presents our approach to querying QB4OLAP data cubes. In Sect. 5 we concisely present our implementation, while Sect. 6 reports our experimental results. Section 7 discusses related work. We conclude in Sect. 8.

Remark 1 Our proposal for querying QB4OLAP data cubes has been previously briefly sketched in [5], while in this paper we develop those ideas in-depth, and provide a detailed experimental study, not included in previous work.

³ <https://w3.org/RDF/>.

⁴ <http://w3.org/TR/sparql11-query/>.

⁵ <http://stats.lod2.eu/>.

⁶ <http://microsoft.com/msj/0899/mdx/mdx.aspx>.

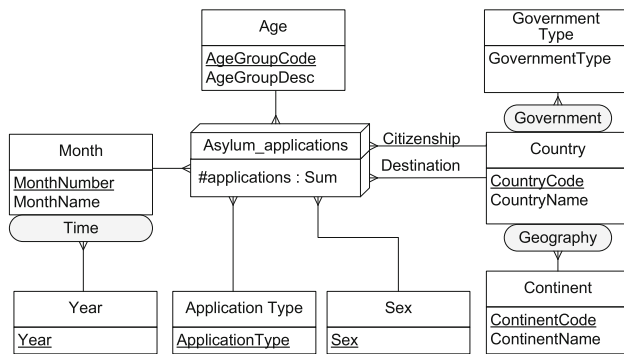


Fig. 1 Conceptual schema of the asylum applications cube

2 Running Example

Throughout this paper we use an example based on statistical data about asylum applications to the European Union, provided by Eurostat.⁷ This data set contains information about the number of asylum applicants per month, age, sex, citizenship, application type, and country that receives the application. It is published in the Eurostat LD dataspace,⁸ using the QB vocabulary. 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. We enriched the original data set in order to enhance the analysis possibilities. Making use of the features of QB4OLAP, we were able to reuse the published observations, so we only created new dimensions, and represented them using QB4OLAP structural metadata.

Figure 1 shows the resulting conceptual schema of the data cube, using the MultiDim notation [6]. The *asylum_applications* fact contains a measure (*#applications*) that represents the number of applications. This measure can be analyzed according to six analysis dimensions: sex of the applicant, age which organizes applicants according to their age group, time 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 one, 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. To create these hierarchies, we enriched the existent data set with DBpedia⁹ data, retrieving, for each country, its government type, and the continent it belongs to.

⁷ http://ec.europa.eu/eurostat/web/products-datasets/-/migr_asyappctzm.

⁸ <http://eurostat.linked-statistics.org/>.

⁹ <http://dbpedia.org>.

As an example, Table 1 shows some observations in tabular format. The first row lists the dimensions in the cube, and the second row lists the dimension level that corresponds to the observations.

Over the new cube, depicted in Fig. 1, we can pose queries like “Total asylum applications by year;” or “Total asylum applications by 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.

3 The QB4OLAP Vocabulary

In QB, the schema of a data set is specified by means of the *data structure definition* (DSD), an instance of the class `qb:DataStructureDefinition`. This specification is formed by *components*, which represent *dimensions*, *measures*, and *attributes*. *Observations* (in OLAP terminology, *fact instances*) represent points in a MD data space indexed by *dimensions*. These points are modeled using instances of the class `qb:Observation` and are organized in *data sets*, defined as instances of the class `qb:DataSet`, where each data set is associated with a DSD that describes the structure of a cube. Finally, each observation is linked to a member in each dimension of the corresponding DSD via instances of the class `qb:DimensionProperty`; analogously, each observation is associated with measure values via instances of the class `qb:MeasureProperty`.

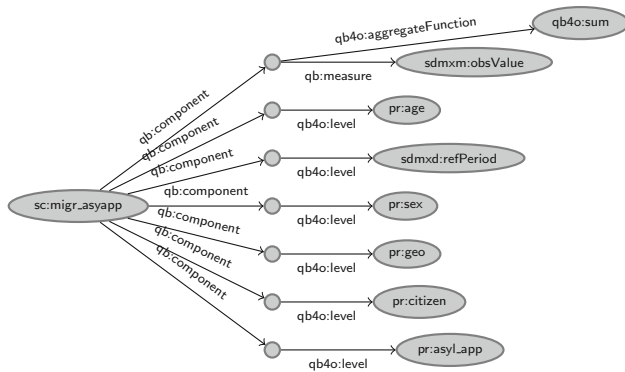
The QB4OLAP¹⁰ vocabulary extends QB to represent the most common features of the MD model. In this way, we can represent a dimension schema as composed of hierarchies of aggregation levels. We can also represent the allowed aggregate functions, rollup relationships (i.e., the parent-child relationships between dimension level members), and descriptive attributes of dimension levels. QB4OLAP allows operating over observations already published using QB, without the need of rewriting them. This is relevant since in a typical MD model, observations are the largest part of the data, while dimensions are usually orders of magnitude smaller. In this section we sketch the key aspects of the vocabulary and refer the reader to [7, 8] for details and a thorough comparison between QB and QB4OLAP.

In QB4OLAP, facts represent relationships *between dimension levels*, and observations (fact instances) map *level members* to measure values. Thus, QB4OLAP represents the structure of a data set in terms of *dimension levels* and measures, instead of *dimensions* and measures (which is the case of QB), allowing us to specify data cubes at different granularity levels in the cube dimensions. Accordingly, the schema of a cube in QB4OLAP is defined, like in QB, via a DSD, but in terms of dimen-

¹⁰ <http://purl.org/qb4olap/cubes>.

Table 1 Tabular representation of sample observations in the asylum applications datacube

Sex	Age	Time	Application type	Citizenship	Destination	Measures
<i>Sex</i>	<i>Age</i>	<i>Month</i>	<i>Application type</i>	<i>Country</i>	<i>Country</i>	<i>#applications</i>
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

**Fig. 2** QB4OLAP representation of Asylum applications data cube schema

sion levels. The class `qb4o:LevelProperty` is introduced to represent this. QB4OLAP also introduces the class `qb4o:AggregateFunction` to represent the *aggregate functions* that should be applied to summarize measure values. The property `qb4o:aggregateFunction` associates measures with aggregate functions in the DSD. Figure 2 shows an excerpt of the QB4OLAP representation of the Asylum applications data cube schema. In the figure, empty circles represent blank nodes. The node labeled `sc:migr_asyapp` represents the DSD of the cube.

Dimension hierarchies and levels are first-class citizens in a MD model for OLAP. Therefore, QB4OLAP focuses on their representation, and several classes and properties are introduced for that. *Dimension level attributes* are represented using class `qb4o:LevelAttribute`, and linked to `qb4o:LevelProperty` via `qb4o:hasAttribute` property. The class `qb4o:Hierarchy` represents *dimension hierarchies*, and the relationship between dimensions and hierarchies is represented via `qb4o:hasHierarchy` property and its inverse `qb4o:inDimension`. To support the fact that a level may belong to different hierarchies, and each level may have a different set of parent levels, the concept of `qb4o:HierarchyStep` is introduced. This represents the reification of the parent–child relationship between two levels. Hierarchy steps are implemented as blank nodes, and each hierarchy step is linked to its component levels using the properties `qb4o:childLevel` and `qb4o:parentLevel`, respectively. It is also associated with the hierarchy it belongs to, through the property `qb4o:inHierarchy`. Also,

the property `qb4o:pcCardinality` represents the cardinality of the relationships between level members in this step.

In earlier versions of QB4OLAP, the *rollup relationships* (in what follows, RUPs) between levels were represented, at the instance level, using the property `skos:broader`. Although this solution is enough for most kinds of MD hierarchies, it does not suffice to represent, at the instance level, dimensions with more than one RUP relationships (or functions) between the same pair of levels, usually denoted as *parallel dependent hierarchies* [6]. As an example, consider a geographical dimension with two levels: *Employee* and *City*. These levels participate in two hierarchies: one that represents the city where the employee lives (say, *LivesIn*), and another that represents the city where the employee works (*WorksIn*). It is easy to see that an employee may live and work in different cities; in order to represent this at the instance level, we need to define two different RDF properties, one for each RUP. Therefore, in QB4OLAP version 1.3 we introduced a mechanism to associate each hierarchy step with a user-defined property that implements the RUP at the instance level. These properties are instances of the class `qb4o:RollupProperty` and are linked to each hierarchy step via the property `qb4o:rollup`.

To conclude this section, Fig. 3 shows an excerpt of the Citizenship dimension schema represented using QB4OLAP. Again, empty circles represent blank nodes. We also include a sample dimension instance on the right hand side of the figure. We can see that the property `qb4o:memberOf` is used to tell that Asia (`citDim:AS`) is a member of the dimension level *Continent*. 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. This example also shows how RUPs are defined in the schema and used in the instances. For example `sc:inContinent` is stated as the implementation of the RUP between the levels *Country* and *Continent*, and it is used at the instance level to link members of these levels. B presents a complete QB4OLAP representation of the Asylum applications data cube.

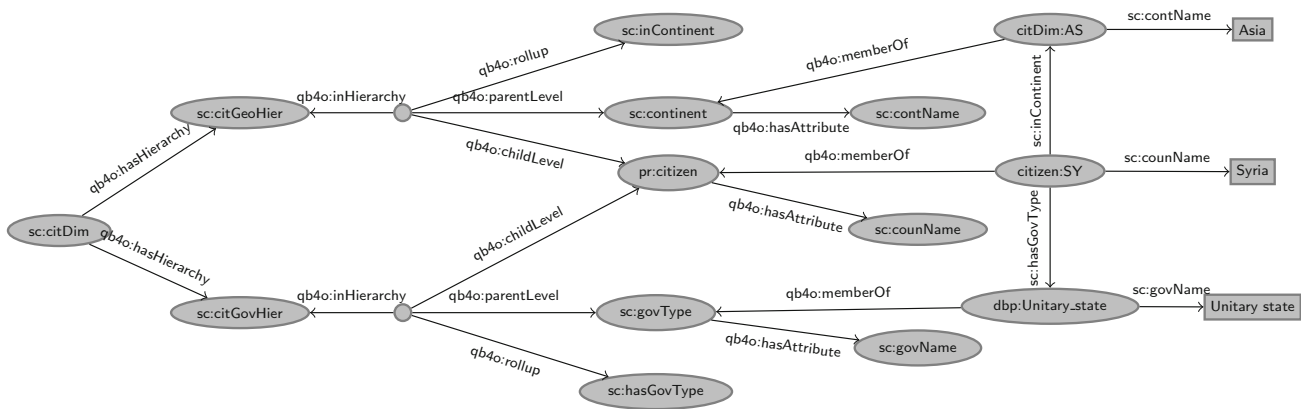


Fig. 3 Citizenship dimension: schema and sample instance

4 Querying QB4OLAP Cubes

We are now ready to get into the details of our approach for exploiting data cubes on the SW, basically, enabling analytical queries. The rationale of our approach is based on the definition of a clear separation between the conceptual and the logical levels, which, strangely, is not common in traditional OLAP. On the contrary, popular OLAP query and analysis languages, such as MDX, operate at the logical level and require, as we commented in Sect. 1, the user’s deep understanding of how data are actually stored in order to be able to write queries. Further, even though MDX is a popular language among OLAP experts, is far from being intuitive, and it would be a barrier for less technical users, who would like to manipulate a data cube to dive into the data. Thus, we follow an approach *aimed at promoting the data analysis directly on the SW*, and, for that, *we want to allow analytical users to focus on querying QB4OLAP cubes* using the operations they know well, for example, roll-up or drill-down, to aggregate or disaggregate data, respectively, minimizing the need of dealing with technical aspects. Our hypothesis is that most users are hardly aware of SW models and languages, but will easily capture the idea of languages dealing with cube operations. In addition, we consider, as explained, that MDX is too technical for our ultimate goal explained above. Thus, we propose a high-level language, denoted CQL, based on an algebra for OLAP, whose only data type is the data cube.

Figure 4 shows the query processing pipeline. The process starts with a CQL query that is first simplified (as explained in Sect. 4.2). This stage aims at rewriting the query to eliminate unnecessary operations, or operations written in a sequence that is probably not the best one.¹¹ The second step translates the simplified CQL query into a single SPARQL expression, following a *naïve* approach (Sect. 4.3). Finally, we apply

¹¹ We remark that in a self-service BI environment [9] users may not be experts, even to write queries in simple languages like CQL.

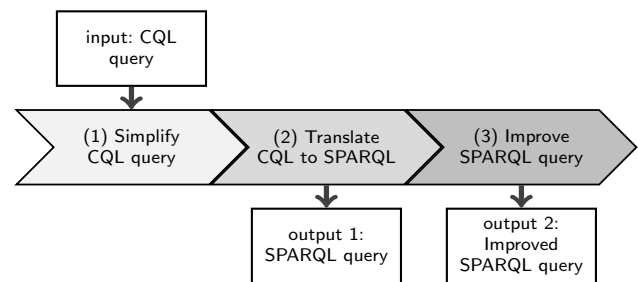


Fig. 4 Query processing pipeline

SPARQL optimization heuristics to improve the performance of the *naïve* queries (Sect. 4.4).

4.1 The CQL language

CQL follows the ideas introduced by Ciferri et al. [4], 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 because it includes the most common OLAP operations.

We next define a formal data model for cubes and define OLAP operations in CQL over this model. The model is based on the one proposed by Hurtado et al. [10], although we choose a different way to present it, which allows to define the semantics of the operations in a clean and elegant way. Due to space limitations, in the following we only present the main ideas to make this paper self-contained. We refer the reader to [7] for details.

Definition 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 $A_l = \langle a_1, \dots, a_n \rangle$ is a tuple of level *attributes*. Each attribute a_i has a domain $Dom(a_i)$; (c) “ \rightarrow ” is a partial order over the levels in \mathcal{L} , with a unique bot-

tom 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 is a set of levels such that $L_h \subseteq \mathcal{L}$, and there is at least one path between the bottom level in d , and the top level All composed of all the levels in L_h . \square

Definition 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 $\langle v_1, v_2, \dots, v_n \rangle$, such that $\forall l, L = \langle l, \langle a_1, \dots, a_n \rangle \rangle \in \mathcal{L}$, and $\forall i, i = 1, \dots, n, v_i \in \text{Dom}(a_i)$; (b) \mathcal{R} is a finite set of relations, called *rollup*, denoted $RUP_{L_i}^{L_j}, L_i, L_j \in \mathcal{L}$, where $L_i \rightarrow L_j \in \text{'}\rightarrow\text{'}$, \square

Definition 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, 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 (cf. Def. Definition 1); (c) \mathcal{M} is a finite set of attributes, where each $m \in \mathcal{M}$, called *measure*, has domain $\text{Dom}(m)$; (d) $\mathcal{F} : \mathcal{M} \rightarrow \mathcal{A}$ is a function that maps measures in \mathcal{M} to an aggregate function in \mathcal{A} . \square

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

Definition 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 $d_i \in \mathcal{D}, i = 1, \dots, r$; and (c) A set of levels $\mathcal{V}_{Cb} = \{L_1, L_2, \dots, L_D\}$ where $L_j \in \mathcal{L}_{d_i}$ in $d_i, i = 1, \dots, r$, such that not two levels belong to the same dimension, a *cuboid instance* is a partial function $Cb : \mathcal{T}_{L_1} \times \dots \times \mathcal{T}_{L_D} \rightarrow \text{Dom}(m_1) \times \dots \times \text{Dom}(m_M)$, where $m_k \in \mathcal{M}, \forall k, k = 1, \dots, p$. The elements in the domain of Cb are called *cells* (whose content are elements in the range of Cb), and \mathcal{V}_{Cb} is called the *level set* of the cuboid. \square

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 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., $|\mathcal{V}_{Cb_1} - \mathcal{V}_{Cb_2}| = |\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1}| = 1$. \square

Definition 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 $\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1} = \{L_r\}$, and L_r and L_c are levels in a dimension d_k such that $L_c \rightarrow L_r$; then, we define the order $Cb_1 \preceq Cb_2$ between both cuboids. Moreover, for each pair of adjacent cuboids $Cb_1 \preceq Cb_2$, each cell $c = (c_1, \dots, c_{k-1}, c_k, c_{k+1}, \dots, c_n, m_1, m_2, \dots, m_p) \in Cb_2$ can be obtained from the cells in Cb_1 as follows.

Let $(c_1, \dots, c_{k-1}, b_{k1}, c_{k+1}, \dots, c_n, m_{1,1}, m_{2,1}, \dots, m_{p,1}), (c_1, \dots, c_{k-1}, b_{k2}, c_{k+1}, \dots, c_n, m_{1,2}, m_{2,2}, \dots, m_{p,2}), \dots, (c_1, \dots, c_{k-1}, b_{kq}, c_{k+1}, \dots, c_n, m_{1,q}, m_{2,q}, \dots, m_{p,q})$ be all the cells in Cb_1 where $(b_{ki}, c_k) \in RUP_{L_c}^{L_r}, i = 1 \dots q$. Measures in $c \in Cb_2$ are computed as $m_i = AGG_i(m_{i,1}, \dots, m_{i,j}), j = 1..q$, where AGG_i is the aggregate function related to m_i . \square

A *Cube Instance* is the lattice of all cuboids that share the same cube schema, defined over the \preceq order relation above. 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 $Cb_i \preceq^* Cb_j$.

Now, we are ready to give a precise semantics for the operations in the OLAP algebra that will be the basis for CQL (see [7] for details).

The ROLL-UP operation summarizes data to a higher level along a dimension hierarchy; that is, it receives a cuboid Cb_1 in a cube instance, and a level L in dimension D , and returns another cuboid Cb_2 in the same instance, such that $Cb_1 \preceq^* Cb_2, L \in \mathcal{V}_{Cb_2}$, and $\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1} = \{L\}$. The DRILL-DOWN operation does the inverse, i.e., it receives a cuboid Cb_1 , and a level L in a dimension D , and returns a cuboid Cb_2 such that $Cb_2 \preceq^* Cb_1$, and $\mathcal{V}_{Cb_2} - \mathcal{V}_{Cb_1} = \{L\}$. Note that the cuboids resulting from a ROLL-UP or DRILL-DOWN on a dimension D are always reachable from the bottom of the cube instance. Thus, a DRILL-DOWN over a dimension D to a level L can be obtained performing a ROLL-UP over d from the bottom cuboid up to L . Since ROLL-UP and DRILL-DOWN only imply a navigation across a lattice (and do not modify it), we call them Instance Preserving Operations (IPO).

The DICE operation selects the cells in a cube that satisfy a boolean condition ϕ . It is analogous to the selection operation in the relational algebra. The condition ϕ is expressed over level member attributes, and/or measure values.

The SLICE operation removes one of the dimensions or measures in the cube. It is analogous to the projection operation in relational algebra. In the case of eliminating a dimension, it is required that, before slicing, the dimension contains a single element at the instance level [11]. If this condition is not satisfied, a ROLL-UP to the All level must be applied over this dimension before removing it.

We denote operations 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. Again, see [7] for details.

In the remainder, we will make use of the following properties. For the sake of space, we omit the proofs.

Property 1 (ROLL- UP/DRILL- DOWN commutativity) A sequence of two consecutive ROLL- UP (DRILL- DOWN) operations over different dimensions is commutative. □

Property 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 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 4 (Slicing ROLL- UP and DRILL- DOWN) Performing 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. □

A CQL query is a sequence of OLAP operations defined above, where the input cuboid of an operation is the output cuboid produced by the previous one. We assume that the input cuboid for the first operation in the sequence is the bottom cuboid of a certain cube instance.

4.1.1 CQL by Example

We now present the syntax of a CQL expression by means of an example. Consider Query 1 below.

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

Example 1 (CQL query) The following CQL query produces a cuboid that answers Query 1. For clarity, intermediate results are stored in variables C_i , although this is not mandatory.

```

$C1:=ROLLUP(migr_asyapp, timeDim, year);
$C2:=ROLLUP($C1, citizenshipDim, continent);
$C3:=DICE($C2, (citizenshipDim|continent|contName = "Africa"));
$C4:=DICE($C3, (destinationDim|geo|counName = "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 remark that the user only deals with the elements of the MD model (e.g., cubes, dimensions), and not the unfriendly (for non-experts) technical issues concerning MDX, SPARQL, RDF, etc. Also note the use of the notation dimension|level|attribute in the DICE expressions. □

4.1.2 Well-Formed CQL Queries

We define *well-formed* CQL queries as follows.

Definition 7 (Well-formed CQL query). A *well-formed* CQL query 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 over the same dimension; (iii) There is no DICE operation mentioning conditions over measure values, in-between a ROLL- UP and/or 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 with the following example.

Example 2 (Condition (iii) in Definition 7) 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 CQL program below produces the answer to Query 2, although it is not well-formed. We next explain why.

```

$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. Thus, the measure now contains the *aggregated* values, not the original ones. A DICE operation is then applied to keep cells that satisfy the restriction over the aggregated measure value. However, since we want the results 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 7 prevents this. □

To summarize, the following patterns define 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:** (SLICE*|DICE*|ROLL- UP*)⁺
- P2:** (SLICE*|ROLL- UP⁺|DRILL- DOWN⁺| DICE_l⁺)⁺
- P3:** (SLICE*|ROLL- UP⁺|DRILL- DOWN⁺|DICE_l^{*})⁺DICE_m⁺

4.2 CQL Simplification Process

As we have already mentioned, CQL is aimed at being used by non-experts. Thus, although 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 Sect. 4.1, we define the following set of rewriting rules. Between brackets we indicate the properties in which the rules are founded.

Rule 1 Remove all the ROLL- UP or DRILL- DOWN operations with the same start and target levels (Property 3).

Rule 2 Find sequences of ROLL- UP and/or DRILL- DOWN operations over the same dimension D , with no DICE _{l} operation in-between, where l is a level in D . Find the last level l_D in the sequence. If l_D is not the bottom level of D (call this level l_{bD}), replace the sequence with a single ROLL- UP from l_{bD} to l_D . Otherwise, remove all the operations in the group (Properties 1 and 2).

Rule 3 If there is a SLICE operation over a dimension D , and no DICE operation that mentions 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 DRILL- DOWN operations over D , and no DICE operation that mentions levels of D , remove all the ROLL- UP and DRILL- DOWN operations, and keep only the SLICE operation (Property 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 omitted).

Property 5 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 SLICE operations are either at the beginning or at the end of q_{out} , but not in the middle.

We now present an example of the simplification process, where we apply the rules above.

Example 3 (CQL simplification)

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

The following CQL expression answers Query 3.

```
$C1:=ROLLUP(migr_asyapp, 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 to Rule 1. By Rule 3, operation \$C5 can be moved to the beginning of the query. Finally, by Rule 5, we can remove \$C3, as operation \$C5 performs a SLICE over the same dimension. The result of the process is:

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

□

4.3 CQL to SPARQL Translation

The next step in the process is the translation of queries in CQL (which are expressed at the conceptual level), into SPARQL expressions over QB4OLAP cubes (expressed at the logical level). Our translation algorithms produce an SPARQL implementation of the CQL operators. For this, we use the QB4OLAP representation of the formal model defined in Sect. 4.1, and the semantics of the operators defined in terms of this formal model. Recall that a cube instance CB is the lattice of all possible cuboids that adhere to a cube schema, and \leq is the partial order between adjacent cuboids in CB . Definitions 5 and 6 provide a mechanism to compute the cells of adjacent cuboids. Therefore, starting from the bottom cuboid in the lattice (the one composed of the bottom levels in each dimension), all the cuboids that form the cube instance can be computed incrementally. Thus, to compute the ROLL- UP operation over an input cuboid CB_{in} , it suffices to start at CB_{in} , and navigate the cube lattice visiting adjacent cubes that differ only in the level associated to dimension D , until we reach a cuboid CB_{out} , that contains the desired level in dimension D (note that this path is unique, by definition).

We do not materialize intermediate results. Instead, we directly compute the target cuboid via a SPARQL query that navigates the dimension hierarchies up to the desired level, aggregating measure values using the aggregate functions declared in the QB4OLAP schema. Note that this is a direct implementation of Definition 5 using SPARQL over a data cube represented using QB4OLAP. Due to space limitations we do not present the translation algorithms (which can be found in [7]), but we present the ideas behind the implementation of each CQL operator using SPARQL 1.1, by means of an example.

Let us consider Query 4 below, and the CQL query that expresses it.

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_asyapp, citizenshipDim, continent);
$C2:=ROLLUP($C1, timeDim, year);
$C3:=DICE($C2, (citizenshipDim|continent|contName="Asia"));
```



```

$C4:=DICE($C3,( obsValue > 5000 AND
  (destinationDim|country|counName = "France")
  OR
  (destinationDim|country|counName
  ="United Kingdom")));

```

Example 4 (CQL to SPARQL translation) The 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.

```

1 SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 ?ag1
2 WHERE {
3   { SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
4     (SUM(xsd:integer(?m1)) as ?ag1)
5     FROM loc-ins:migr_asyapp_clean
6     FROM loc-sch:migr_asyappQB4013
7     WHERE { ?o a qb:Observation .
8       ?o qb:dataSet data:migr_asyapp .
9       ?o sdmxm:obsValue ?m1 .
10      ?o pr:citizen ?lm1 .
11      ?lm1 qb4o:memberOf pr:citizen .
12      ?lm1 sc:inContinent ?plm1 .
13      ?plm1 qb4o:memberOf sc:continent .
14      ?o sdmxd:refPeriod ?lm2 .
15      ?lm2 qb4o:memberOf sdmxd:refPeriod .
16      ?lm2 sc:inYear ?plm2 .
17      ?plm2 qb4o:memberOf sc:year .
18      ?o pr:geo ?lm3 .
19      ?o pr:sex ?lm4 .
20      ?o pr:age ?lm5 .
21      ?o pr:asy1_app ?lm6 .
22      ?plm1 sc:contName ?plm11 .
23      ?lm3 sc:counName ?lm31 .
24      FILTER ( ?plm11 = "Asia" &&
25        (?lm31 = "France" ||
26        ?lm31 = "United Kingdom" ))}
27     GROUP BY ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
28 } FILTER ( ?ag1 > 5000 )

```

Lines 10 through 13 implement the first `ROLLUP` ($C1$). 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` (lines 12 and 13). 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 3), in the `GROUP BY` clause of the inner query (line 27), and in the result of the outer query (line 1). Analogously, the navigation that corresponds to the `ROLLUP` in $C2$ is performed in lines 14 through 17. Lines 18 to 21 will 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 `?xg1` in line 4. The aggregated value is added to the 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 22) and then use a `FILTER` clause to keep only the cells that correspond to “Asia” (line 24). The DICE operation in

statement $C4$ is split as follows: the restriction on country names is implemented adding lines 25 and 26 to the `FILTER` clause (country names are retrieved in line 23), 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 28). \square

4.4 SPARQL Queries Improvement

We have shown a *naïve* procedure to automatically produce SPARQL queries that implement CQL queries over QB4OLAP. To improve the performance of such queries, we adapted three existing techniques to the characteristics of MD data in general, taking into account particularities of QB4OLAP representation.

First, we adapted to our setting the heuristics proposed by Loizou et al. [12] 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 [13]. Since the SPARQL queries we produce do not include the `OPTIONAL` operator, we do not use this rule.

H2-Use named graphs to localize SPARQL graph patterns This heuristic is based on the correlation between the performance of a query and the number of triples it is evaluated against. We apply this heuristic as follows. We organize QB4OLAP data into two named graphs, namely: (a) A *schema* graph, which stores the schema and dimension members; (b) An *instance* graph, which stores only observations. Normally, the size of the instance 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 $2+|D|+|M|$, where D is the set of dimensions, and M the set of measures.

H3-Reduce intermediate results This heuristic 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. This is due to the fact that QB4OLAP proposes to use a different predicate to represent each RUP relationship between level members, instead of using, as in QB, a single predicate like `skos:narrower`. We give an example of this in B.

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

H5-Rewriting FILTER clauses. Proposes to transform `FILTER` clauses with disjunction (`||`) of equality constraints,

using either the UNION of patterns, or a VALUES expression. In Example 5 we show these transformations. Since the reported results are not conclusive on which of these strategies leads to better performant queries, we decided to evaluate both of them (see Sect. 6).

Example 5 (Rewriting FILTER clauses) The queries below show how FILTER clauses with disjunction of equality constraints can be replaced using H5.

```

1 SELECT ?x
2 WHERE {
3   ?x <predicate> ?y .
4   FILTER (?y = value1 || ?y = value2)}
5 #rewriting FILTER using UNION
6 SELECT ?x
7 WHERE {
8   { ?x <predicate> value1 }
9   UNION
10  { ?x <predicate> value2 } }
11 #rewriting FILTER using VALUES
12 SELECT ?x
13 WHERE {
14   ?x <predicate> ?y .
15   VALUES ?y (value1 value2)}

```

□

As our second strategy, we considered the recommendations in [14], 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 one allows the query processor to use indexes to select the patterns that match the criteria.

Example 6 (Improving FILTERs) Below, we give an example of the second strategy.

```

1 SELECT ?x
2 WHERE { ?x ?y ?z .
3   FILTER (?y = <predicate> && ?z > value1)}
4 #splitting FILTER conjunction
5 SELECT ?x
6 WHERE { ?x ?y ?z .
7   FILTER (?y = <predicate>)
8   FILTER (?z > value1)}
9 #replace FILTER equality constraints with a BGP
10 SELECT ?x
11 WHERE { ?x <predicate> ?z.
12   FILTER (?z > value1)}

```

The query in Lines 1 to 3 asks for the values of ?x that are associated via <predicate>, with values greater than “value1.” We then rewrite the query applying the strategies mentioned above, i.e., splitting and rewriting. □

The next example shows the result of applying the above two strategies to the query in Example 4.

Example 7 (SPARQL queries improvement) The application of H2 organizes graph patterns in the inner query in two GRAPH clauses: one that corresponds to patterns in the instance graph (lines 8 to 15), and another in the schema graph

(lines 16 to 26). Applying H5, the FILTER clause on country names is replaced by a VALUES clause (line 25). Finally, using the second strategy, FILTER clauses are split, and the one on continent name is replaced by a graph pattern (line 20).

```

1 SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 ?xg1
2 WHERE {
3   {SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
4     (SUM(xsd:integer(?m1)) as ?xg1)
5   FROM NAMED loc-ins:migr_asyapp_clean
6   FROM NAMED loc-sch:migr_asyappQB4013
7   WHERE {
8     {GRAPH loc-ins:migr_asyapp_clean
9       {?o a qb:Observation .
10        ?o qb:dataSet data:migr_asyapp .
11        ?o sdmxm:obsValue ?m1 .
12        ?o pr:citizen ?lm1 .
13        ?o sdmxd:refPeriod ?lm2 .
14        ?o pr:geo ?lm3 . ?o pr:sex ?lm4 .
15        ?o pr:age ?lm5 . ?o pr:asyl_app ?lm6 .}}.
16    {GRAPH loc-sch:migr_asyappQB4013
17      {?lm1 qb4o:memberOf pr:citizen .
18       ?lm1 sc:inContinent ?plm1 .
19       ?plm1 qb4o:memberOf sc:continent .
20       ?plm1 sc:contName "Asia" .
21       ?lm2 qb4o:memberOf sdmxd:refPeriod .
22       ?lm2 sc:inYear ?plm2 .
23       ?plm2 qb4o:memberOf sc:year .
24       ?lm3 sc:counName ?lm31 .
25       VALUES ?lm31 {"France"@en "United Kingdom"@en}
26     }}}
27   GROUP BY ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
28   } FILTER (?xg1 > 5000) }

```

□

Our third, and final, strategy is based on the work of Stocker et. al [15]. 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 , $|L_i| \geq |L_j|$. Moreover, in most cases $|L_i| > |L_j|$ holds. Based on the above, we define alternative *ordering criteria* (OC) for the graph patterns.

- Ordering Criterion 1 (OC1)-For each dimension appearing in the query, apply first the patterns that correspond to higher levels.
- Ordering Criterion 2 (OC2)-For each dimension, apply OC1. Then, reorder dimensions as follows: 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 Criterion 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 $|l_A| \geq |l_B|$, then dimension A goes before dimension B.

Example 8 (Reordering triple patterns) We show the result of applying OC2 to reorder the triple patterns on the schema graph from Example 7.

```

1 GRAPH loc-sch:migr_asyappQB4013 {
2 ?plm1 sc:contName "Asia" .
3 ?plm1 qb4o:memberOf sc:continent .
4 ?plm1 sc:inContinent ?plm1 .
5 ?plm1 qb4o:memberOf pr:citizen .
6 ?lm3 sc:counName ?lm31 .
7 VALUES ?lm31 {"France"@en "United Kingdom"@en}
8 ?plm2 qb4o:memberOf sc:year .
9 ?plm2 sc:inYear ?plm2 .
10 ?plm2 qb4o:memberOf sdmxd:refPeriod .}

```

Triples in lines 2 through 5 correspond to the Citizenship dimension, lines 6 and 7 correspond to Destination dimension, and lines 8 through 10 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”). □

We end this section with some remarks on the complexity of the generated SPARQL queries. 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 [13]. Moreover, the evaluation of queries that only contain AND and UNION operators is already NP-complete, as proved in [16]. Perez et. al [13] also proved that the main source of complexity in SPARQL 1.0 queries is the introduction of the OPTIONAL, that 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 functionalities incorporated in SPARQL 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.

5 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 5 presents a screenshot of this module.

Fig. 5 QB4OLAP toolkit: Explorer module

Query cubes

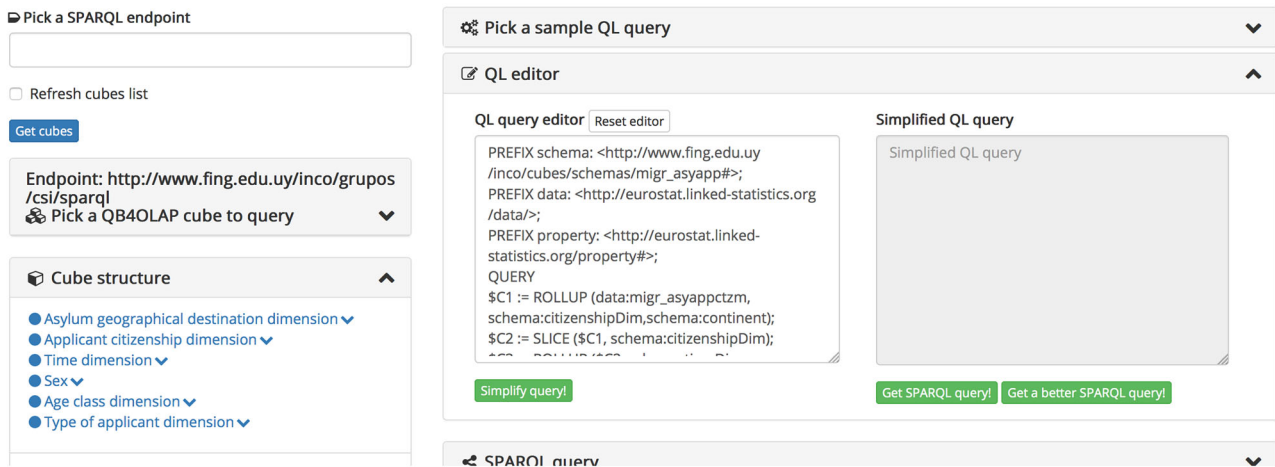


Fig. 6 QB4OLAP toolkit: Querying module

The *Querying module* implements the querying processing pipeline presented in Fig. 4. The user first writes a CQL query. Then, the application simplifies this CQL query and displays the result to the user, who can choose to generate either a naïve SPARQL query or an improved one. The query produced is presented to the user and executed. Results are presented in tabular format. Figure 6 presents a screenshot of this module.

The QB4OLAP toolkit has been entirely developed in Java Script over the Node.js platform using Express. Handlebars, jQuery, and D3.js are used to implement the front-end. Virtuoso Open Source version 7 is used for RDF storage and SPARQL back-end. The communication with Virtuoso is implemented via HTTP and using JSON format to exchange data. Figure 7 presents the technology stack of QB4OLAP toolkit.

The QB4OLAP toolkit is available online.¹² We also provide example queries that the user will edit and run. Source code is available at GitHub.¹³

6 Evaluation

We now report and discuss experimental results. Our primary goal is to show that, with our proposal, OLAP users can write complex analytical queries in an algebra that is familiar to them, manipulating just what they know well: data cubes, regardless of how they are physically stored. For what we are

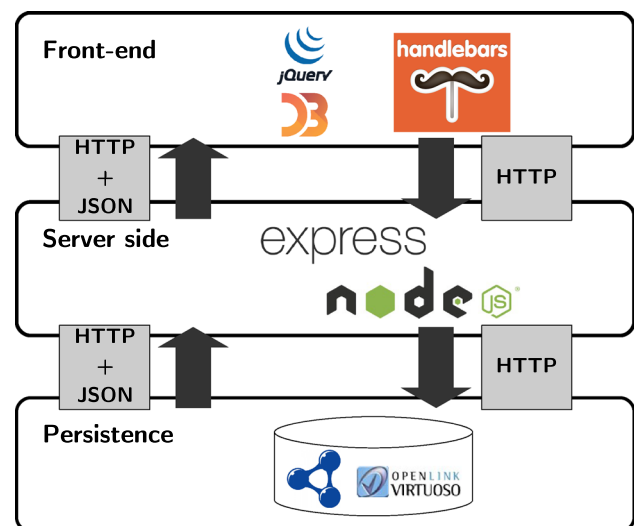


Fig. 7 QB4OLAP toolkit: technology stack

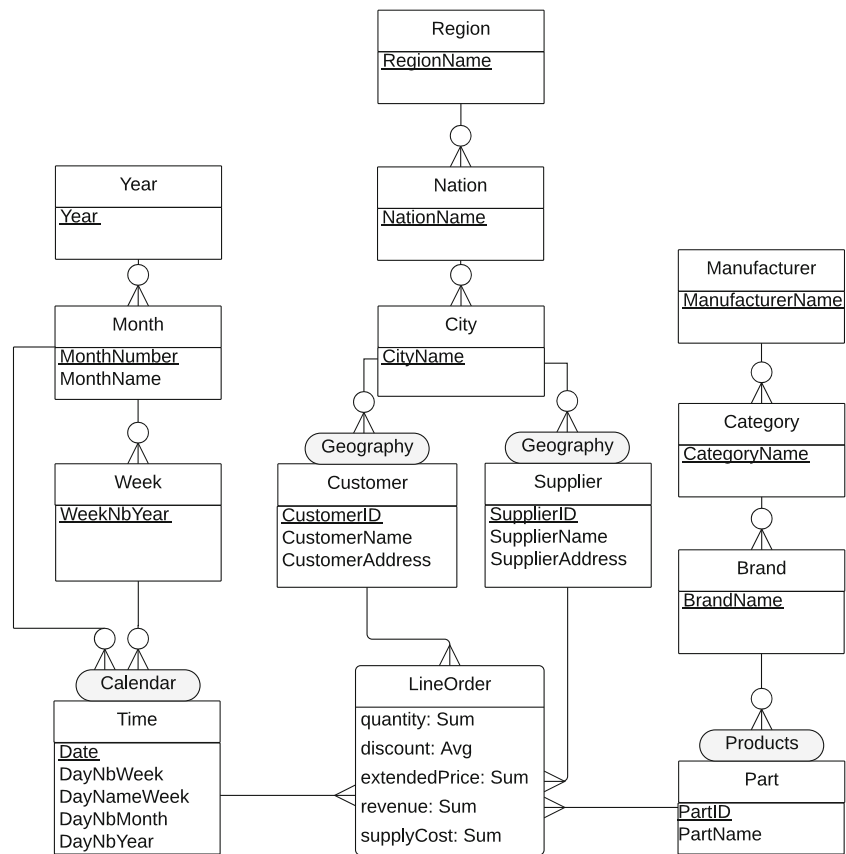
interested in this paper, OLAP users should be able to query cubes on the SW, without having to deal with technical issues such as QB4OLAP, RDF, or SPARQL, and still obtain good query performance.

Our evaluation goal is thus 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 look for the best possible combinations of query optimization strategies. For the first goal, we compare our approach against the one by Kämpgen et al. [17, 18], who propose a mechanism for implementing some OLAP operations over extended QB cubes using SPARQL queries (see

¹² <https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/>.

¹³ <https://github.com/lorenae/qb4olap-tools>.

Fig. 8 Conceptual schema of the SSB-QB4OLAP cube



Sect. 7 for details). To evaluate their approach, they adapted the Star Schema benchmark (SSB) [19], and produced the SSB-QB benchmark, which 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; (iii) a set of thirteen SPARQL queries over these data. These queries are equivalent to SSB queries and aim at representing the most common types of star schema queries in an OLAP setting. Based on this work, 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 as in SSB-QB; (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 [17]. It also allows us to measure the impact of our improvement strategies, in order to address our second goal. For this, we translated the CQL queries into SPARQL using the naïve approach and explore which combination of strategies yields the best query results, based on several metrics.

Next, we introduce the SSB-QB4OLAP benchmark (Sect. 6.1), describe the experimental setup and experiments (Sect. 6.2), and discuss the results (Sect. 6.3). The complete

experimental environment is available for download as a virtual machine at the benchmark site.¹⁴

6.1 The SSB-QB4OLAP Benchmark

SSB-QB4OLAP data represents SSB data cube at Scale 1 and is organized in three sets of triples that represent: (1) facts (observations); (2) the cube schema; and (3) the dimension instances (i.e., level members, attribute values, and RUP relationships). The set of observations, as in SSB-QB, consists of about 132,000,000 triples, representing 6,000,000 line orders. The cube schema is represented in QB4OLAP, consists of about 250 triples, and corresponds to the conceptual schema presented in Fig. 8. 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. Finally, a set of about 2,800,000 triples represents level members, attribute values, and rollup relationships. Table 2 shows the number of members in each level. Data are available for querying at our endpoint.¹⁵

¹⁴ <https://github.com/lorenae/ssb-qb4olap>.

¹⁵ <https://www.fing.edu.uy/inco/grupos/csi/sparql>.

Table 2 SSB-QB4OLAP dataset statistics

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

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 varying fractions of the set of line orders (selectivity coverage). The **first query flight (QF1)** is composed of three queries (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.

6.2 Experimental Setup and Results

We ran our evaluation on an Ubuntu Server 14.04.1 LTS, a single Intel(R) Xeon(R) E5620 @2.40GHz with 4 cores and 8 hardware threads, 32GB RAM, and 500GB for local data storage. We use Virtuoso Open source (V 07.20.3214) as RDF store. BIBM tool¹⁶ was used to perform *TPC-H power tests*, and in each test suit, a mix of 13 queries was used with scale 1 and 2 client streams. We also ran a test suit using the query mix from SSB-QB. We measured the average response time for each query and the following TPC-H metrics for each query mix: *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

¹⁶ <http://sourceforge.net/projects/bibm/>.

Table 3 Strategies used to improve queries performance

-
- S1: Use named graphs to reduce the search space [12]
 - S2: Replace FILTER equality constraints that compare a variable and a constant with BGP's [14]
 - S3: Split FILTER clauses with CONJUNCTION of constraints into a cascade of FILTER clauses with atomic constraints [14]
 - S4: Replace FILTER clauses with DISJUNCTION of equality constraints using UNION or VALUES [12]
 - S5: Reorder triple patterns applying most restrictive patterns for each dimension first (using criteria OC1, OC2, or OC3)
-

of the measurement interval; and *TPC-H Composite*, the geometric mean of the previous metrics, that reflects the query processing power when queries are submitted in a single stream, and the query throughput for queries submitted by multiple concurrent users [20].

6.2.1 Evaluation of the Improvement Strategies

We measured the impact on performance of the improvement strategies presented in Sect. 4.4, in order to find out which combination of strategies results more beneficial. The strategies are summarized in Table 3.

For each of the 13 queries in the benchmark, Table 4 indicates which strategies in Table 3 can be applied to them. 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 9 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 (as we can see in Table 4). Then we consider the cases of applying S3 (ES3) or not. For S4 we consider both flavors: 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 Sect. 4.4. As an example, ES11 is the result of applying improvement strategies S1, S2, S4 (VALUES) and S5 (OC1), to naïve SPARQL queries.

Table 5 reports the results for the naïve approach and all the evaluation scenarios. ES7 and ES11 are the scenarios with better performance. Figure 10 reports the average execution time for each query at the best improvement scenarios.

6.2.2 Comparison with SSB-QB

We also wanted to compare the queries produced by our naïve approach, and the best and worst cases of the improved

Table 4 Applicability of each improvement strategy to SSB-QB4OLAP queries

	Q ₁	Q ₂	Q ₃	Q ₄	Q ₅	Q ₆	Q ₇	Q ₈	Q ₉	Q ₁₀	Q ₁₁	Q ₁₂	Q ₁₃
S1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S2	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓
S3	✓	✓	✓		✓		✓	✓	✓	✓		✓	✓
S4									✓	✓	✓	✓	
S5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

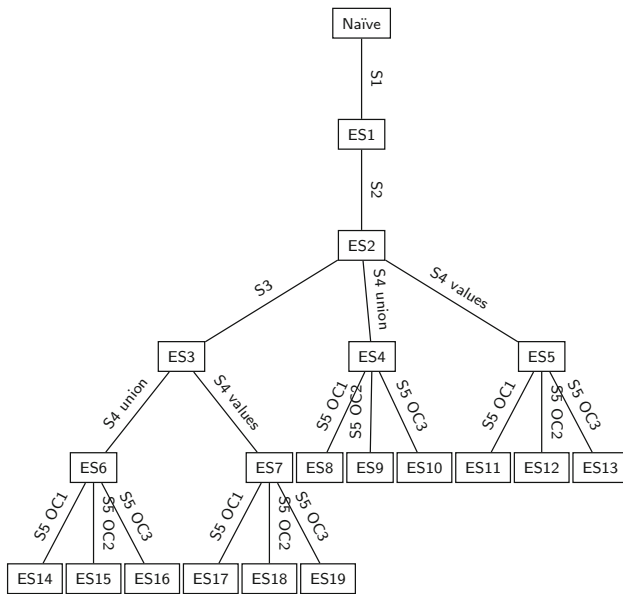


Fig. 9 Improvement Strategies Evaluation Scenarios

queries, against the SSB-QB queries. Thus, we implemented SSB-QB in our experimental setting and ran the queries. Table 6 shows the results obtained for each TPC-H metric, and Fig. 11 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).

6.3 Discussion

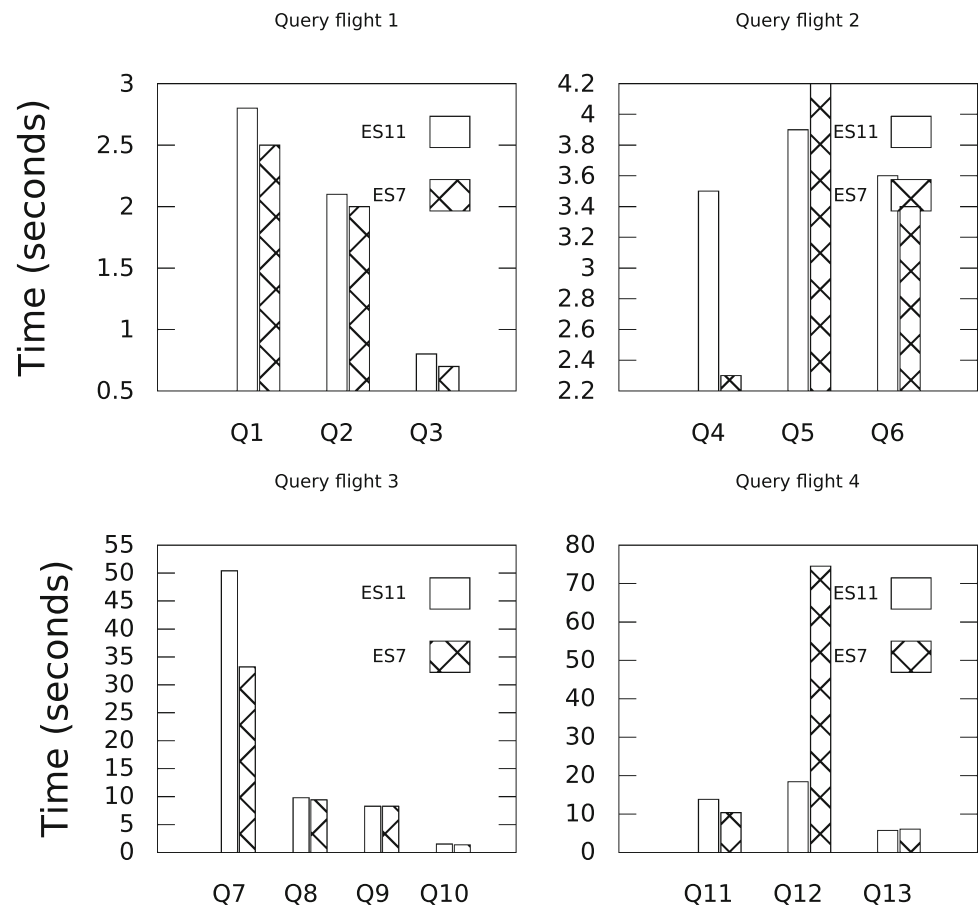
Regarding the improvement scenarios, results show that, for the TPC-H Composite metric, scenario ES11 outperforms the other ones, with a 10X improvement with respect to the naïve scenario (see Table 5), and a 10X speedup in the execution time for the query mix. The second best scenario is ES7, with a 9X improvement on TPC-H Composite with respect to the naïve scenario and a 9X speedup. 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),

Table 5 TPC-H metrics: improvement evaluation

	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

but ES7 applies S3, while ES11 applies S5 with OC1 reordering (Fig. 9).

Regarding the impact of each improvement strategy (Table 5), 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). Finally, we cannot be conclusive on the impact of reordering graph patterns.

Fig. 10 Naïve vs. improved queries execution time**Table 6** TPC-H metrics comparison

	Power (QpH)	Throughput (QpH)	Composite (QpH)	Interval (sec)
SSB-QB [17]	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

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 6). 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 Fig. 11) shows that our approach outperforms SSB-QB for Q1, Q4, Q7, Q11, and Q12.

We next further discuss the reasons why our naïve approach performs better than the SSB-QB queries.

- SSB-QB queries include an ORDER BY clause to order results, while our queries do not.
- As a consequence of the absence of level attributes, SSB-QB queries use string comparison on IRIs to fix level

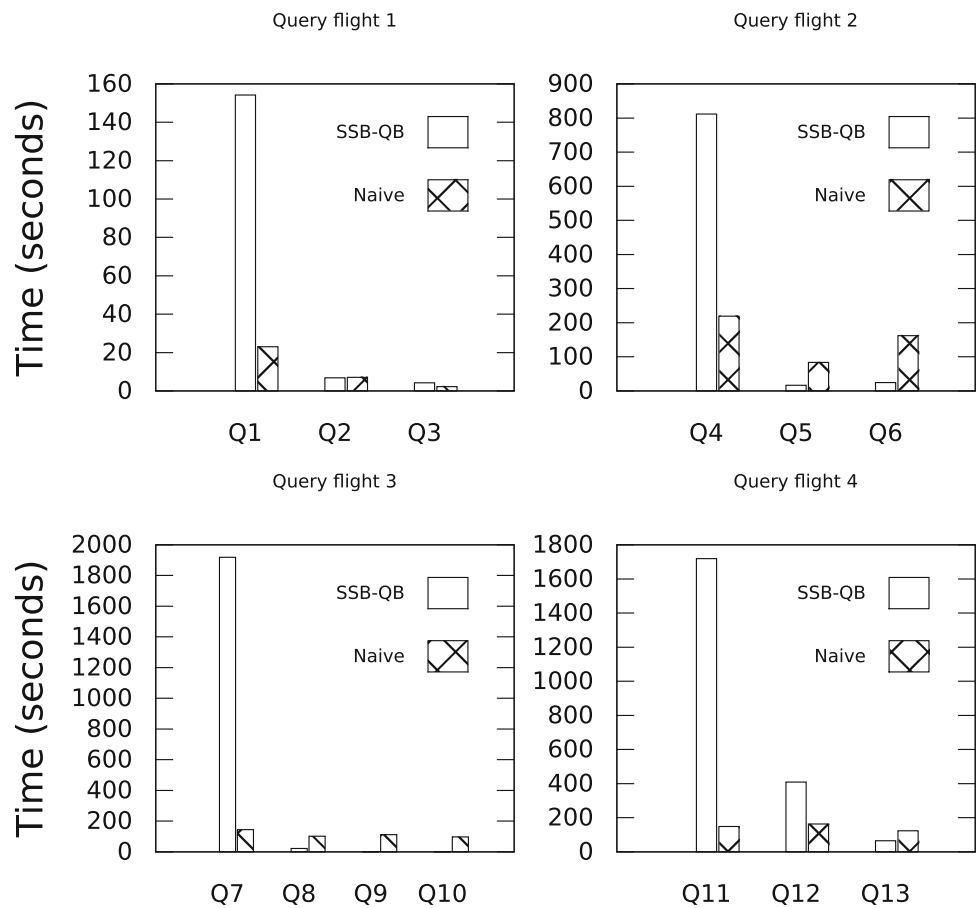
members, while we can use comparison over other data types, like, for example, numeric values. It is well known that string comparison is usually slower than integer comparison.

- The BGPs used to traverse hierarchies in SSB-QB may not take advantage of Virtuoso indexes.

To illustrate the last point we first give some insight on Virtuoso, and then present an example. The Virtuoso triple store uses a relational database 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). Two full and three partial indices are implemented:¹⁷

¹⁷ <http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtRDFPerformanceTuning>.

Fig. 11 SSB-QB and Naïve queries execution time



- 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 Q8 from SSB-QB4OLAP.

Q8: Revenue volume for lineorder transactions by customer city, supplier city and year, for suppliers and clients within the USA, and transactions issued between 1992 and 1997.

Figures 12 and 13 present the SPARQL representation of Query 6.3 according to SSB-QB and to our naïve approach, respectively. In particular, notice the BGPs that implement the ROLL-UP operation over the Time dimension (lines 8–12 in Fig. 12 and lines 8–13 in Fig. 13): 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 also unknown.

```

1 SELECT ?c_city ?s_city ?d_year
2       sum(?rdfh_lo_revenue) as ?lo_revenue
3 FROM <http://lod2.eu/schemas/rdfh-inst#ssb1_ttl_qb>
4 FROM <http://lod2.eu/schemas/rdfh#ssb1_ttl_dsd>
5 FROM <http://lod2.eu/schemas/rdfh#ssb1_ttl_levels>
6 WHERE {
7   ?obs qb:dataSet rdfh-inst:ds.
8   ?obs rdfh:lo_orderdate ?d_date.
9   ?d_yearmonthnum skos:narrower ?d_date.
10  ?d_yearmonth skos:narrower ?d_yearmonthnum.
11  ?d_year skos:narrower ?d_yearmonth.
12  rdfh:lo_orderdateYearLevel skos:member ?d_year.
13  ?obs rdfh:lo_custkey ?c_customer.
14  ?c_city skos:narrower ?c_customer.
15  ?c_nation skos:narrower ?c_city.
16  ?c_region skos:narrower ?c_nation.
17  rdfh:lo_custkeyRegionLevel skos:member ?c_region.
18  ?obs rdfh:lo_suppkey ?s_supplier.
19  ?s_city skos:narrower ?s_supplier.
20  ?s_nation skos:narrower ?s_city.
21  ?s_region skos:narrower ?s_nation.
22  rdfh:lo_suppkeyRegionLevel skos:member ?s_region.
23  ?obs rdfh:lo_revenue ?rdfh_lo_revenue.
24  FILTER(?c_nation = rdfh:lo_custkeyNationUNITED-STATES ).
25  FILTER(?s_nation = rdfh:lo_suppkeyNationUNITED-STATES ).
26  FILTER(
27    str(?d_year) >= "http://lod2.eu/schemas/rdfh#
28    http://lo_orderdateYear1992" and
29    str(?d_year) <= "http://lod2.eu/schemas/rdfh#
30    http://lo_orderdateYear1997").
31 }
32 GROUP BY ?d_year ?c_city ?s_city
33 ORDER BY ASC(?d_year) DESC(?lo_revenue)

```

Fig. 12 Query 8 (SSB-QB)

```

1 SELECT ?plm2 ?plm3 ?plm5 (SUM(xsd:float(?m4)) as ?ag1)
2 FROM <http://www.fing.edu.uy/inco/cubes/instances/ssb_qb4olap>
3 FROM <http://www.fing.edu.uy/inco/cubes/schemas/ssb_qb4olap>
4 WHERE {
5   ?o a qb:Observation .
6   ?o qb:dataSet rdfs:inst:ds .
7   ?o rdfs:lo_revenue ?m4 .
8   ?o rdfs:lo_orderdate ?l1 .
9   ?l1 qb4o:memberOf rdfs:lo_orderdate .
10  ?l1 schema:dateInMonth ?plm1 .
11  ?plm1 qb4o:memberOf schema:month .
12  ?plm1 schema:monthInYear ?plm2 .
13  ?plm2 qb4o:memberOf schema:year .
14  ?o rdfs:lo_custkey ?l2 .
15  ?l2 qb4o:memberOf rdfs:lo_custkey .
16  ?l2 schema:inCity ?plm3 .
17  ?plm3 qb4o:memberOf schema:city .
18  ?plm3 schema:inNation ?plm4 .
19  ?plm4 qb4o:memberOf schema:nation .
20  ?o rdfs:lo_partkey ?l3 .
21  ?o rdfs:lo_supkey ?l4 .
22  ?l4 qb4o:memberOf rdfs:lo_supkey .
23  ?l4 schema:inCity ?plm5 .
24  ?plm5 qb4o:memberOf schema:city .
25  ?plm5 schema:inNation ?plm6 .
26  ?plm6 qb4o:memberOf schema:nation .
27  ?plm4 schema:nationName > ?plm41 .
28  ?plm6 schema:nationName > ?plm61 .
29  ?plm2 schema:yearNum ?plm21 .
30  FILTER ( ?plm41 = "UNITED STATES" &&
31           ?plm61 = "UNITED STATES" &&
32           ?plm21 >= 1992 && ?plm21 <= 1997)
33 }
34 GROUP BY ?plm2 ?plm3 ?plm5

```

Fig. 13 Query 8 (SSB-QB4OLAP naïve)

7 Related Work

We identify two major approaches in OLAP analysis of SW data. The first one consists in extracting MD data from the web, and loading them into traditional data management systems for OLAP analysis. This approach requires a local DW to store the extracted data, a restriction that clashes with the highly volatile and autonomous nature of web data sources. Relevant to this line of research, are the works by Nebot and Llavori [21] and Kämpgen and Harth [22]. We will discuss here a different line of work, which explores data models and tools that allow publishing and performing OLAP-like analysis directly over the SW, *representing MD data in RDF*. This is closely related with the concepts of *self-service BI*, which aims at incorporating web data into the decision-making process [9], and *exploratory OLAP* [23].

Ibrahimov et al. [24] 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 proposal uses the QB4OLAP vocabulary 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 it. Although the envisioned framework should be able to answer MDX queries, few details are provided on the translation process from MDX queries to SPARQL queries over QB4OLAP. Although expert OLAP users are likely to know MDX, in a self-service BI environment most users are not so proficient, in our opinion, we need a *more intuitive language*, that

can deal only with cubes, an intuitive data structure for most analytical users.

Literature on MD data representation in RDF can be further organized in two categories: (i) those that use specialized RDF vocabularies to explicitly define the data cubes; and (ii) those that implicitly define a data cube over existing RDF data graphs. Our work follows the explicit approach and extends the QB vocabulary to include the MD structure. Kämpgen et al. [17, 18] also attempt to override the lack of structure in QB defining an OLAP data model on top of QB and other vocabularies. They use extensions to the SKOS vocabulary¹⁸ to represent the hierarchical structure of the dimensions. In this representation, levels can belong to only one hierarchy, and level attributes are not supported. In [17] the authors implement some OLAP operators over those extended cubes, using SPARQL queries, restricted to data cubes with only one hierarchy per dimension. They also explore the use of RDF aggregate views to improve performance. This approach requires specialized OLAP engines for analytical queries over RDF data, instead of traditional triple stores.

The WaRG project¹⁹ proposes an analytical model to implicitly define data cubes over RDF graphs. The core concepts are the analytical schema (AnS), a graph that represents an MD view over existing RDF data, following the classical global-as-view data integration approach, and analytical queries (AnQ) over AnS, which can be implemented as SPARQL BGPs [25, 26]. Although they show how some OLAP operations can be implemented as AnQs, key operations like ROLL-UP are just briefly sketched. Moreover, AnS does not support the definition of complex dimension hierarchies.

Regarding SPARQL query processing, many works study the complexity of query evaluation [13, 16]. In [27] the authors focus on the static analysis of SPARQL queries, in particular those that contain the OPTIONAL operator. Tsalimanis et al. [28] propose a heuristic approach to the optimization for SPARQL joins, based on the selectivity of graph patterns. All of these are general-purpose studies. On the contrary, *we take advantage of the characteristics of our data model* (e.g., the OLAP operators, and the information provided by QB4OLAP metadata) to define optimization rules that may not apply to a more generic scenario.

Jakobsen et al. [29] study how to improve SPARQL queries over QB4OLAP data cubes. To reduce the number of joins (BGPs) needed to traverse hierarchies, they propose to generate denormalized representations of 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)

¹⁸ http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS.

¹⁹ <http://team.inria.fr/oak/projects/warg/>.

with attribute values of related level members. Although preliminary results show an improvement in queries performance, this approach prevents level members from being reused and referenced, breaking the Linked Data nature of QB4OLAP data instances.

8 Conclusion

In this paper we proposed the use of a high-level language (CQL) over data cubes, to express OLAP queries at a conceptual level. We showed that these queries can be automatically translated into efficient SPARQL ones. For this, we first used the metadata provided by the QB4OLAP vocabulary to obtain a naïve translation of CQL programs to SPARQL queries, and then, we adapted general-purpose SPARQL optimization techniques to the OLAP setting, to obtain better performance. Our experiments over synthetic data (an adaptation of the Star Schema TPC-H benchmark) showed that even the naïve approach outperforms other proposals, and suggest the best combinations of optimization strategies. An application to explore SW cubes, write, and execute CQL queries, completes our contributions. We believe that these results can encourage and promote the publication and sharing of MD data on the SW. We plan to continue working in this direction, extending CQL (and the corresponding translations) with other OLAP operations.

Acknowledgements Alejandro Vaisman was partially supported by PICT-2014 Project 0787, from the Argentinian Scientific Agency.

A Prefixes used in this paper

Figure 14 shows the prefixes used in this paper.

Fig. 14 RDF prefixes used in this work

```

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX qb4o: <http://purl.org/qb4olap/cubes#>
PREFIX sdmxm: <http://purl.org/linked-data/sdmx/2009/measure#>
PREFIX sdmxd: <http://purl.org/linked-data/sdmx/2009/dimension#>
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 dt: <http://eurostat.linked-statistics.org/data/>
PREFIX ds: <http://eurostat.linked-statistics.org/data/migr_asyappctzm#>
PREFIX loc-ins: <http://www.fing.edu.uy/cubes/instances/>
PREFIX loc-sch: <http://www.fing.edu.uy/cubes/schemas/>
PREFIX sc: <http://www.fing.edu.uy/cubes/schemas/migr_asyapp#>
PREFIX instances: <http://www.fing.edu.uy/cubes/instances/migr_asyapp#>
PREFIX citDim: <http://www.fing.edu.uy/cubes/dims/migr_asyapp/citizen#>
PREFIX time: <http://purl.org/qb4olap/dimensions/time#201409>

```

B QB4OLAP Representation of the Asylum Applications Data Cube

In this appendix we show how the Eurostat data cube in our running example, looks like in QB4OLAP. Figure 15 shows the cube schema. Note that the structure is defined in terms of dimension levels, which represent the granularity of the observations in the data set (i.e., these levels are the lowest levels in the dimension hierarchies).

An observation (represented in QB4OLAP) corresponding to the schema in Fig. 15 is shown below. It corresponds to the first row of Table 1.

```

ds:M,SY,F,Y18-34,NASY_APP,DE,2014M09 a qb:Observation ;
pr:age age:Y18-34 ;
sdmxd:refPeriod time:2001409 ;
pr:sex sex:F ;
pr:geo geo:DE ;
pr:citizen citizen:SY ;
pr:asyl_app app:NASY_APP ;
sdmxm:obsValue 425 .

```

Dimensions are represented in QB4OLAP as follows. We define the citizenship dimension `sc:citDim` of Fig. 1, and the hierarchy `sc:citGeoHier`, also declaring levels `pr:citizen` and `sc:continent`. Also, we associate attributes with levels, e.g., `sc:contName` with `sc:continent`.

```

sc:migr_asyapp rdf:type 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 ] .

dt:migr_asyappctzm qb:structure sc:migr_asyappctzmQB40;.

```

Fig. 15 Asylum Applications Data Cube

Finally, the rollups and hierarchy steps (i.e. parent–child relationships) are defined.

```
# Dimension definition
sc:citDim a qb:DimensionProperty ;
rdfs:label "Applicant citizenship dimension"@en ;
qb4o:hasHierarchy sc:citGeoHier, sc:citGovHier .

# Hierarchy definition
sc:citGeoHier a qb4o:Hierarchy ;
rdfs:label "Applicant citizenship Geo Hierarchy"@en ;
qb4o:inDimension sc:citDim ;
qb4o:hasLevel pr:citizen, sc:continent .

# Base level
pr:citizen a qb4o:LevelProperty ;
rdfs:label "Country of citizenship"@en ;
qb4o:hasAttribute sc:counName ;
sc:counName a qb4o:LevelAttribute ;
rdfs:label "Country name"@en ; rdfs:range xsd:string .

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

#rollup relationships
sc:inContinent a qb4o:RollupProperty .
sc:hasGovType a qb4o:RollupProperty .
#hierarchy step
_:ih1 a qb4o:HierarchyStep ;
qb4o:inHierarchy sc:citGeoHier ;
qb4o:childLevel pr:citizen ;
qb4o:parentLevel sc:continent ;
qb4o:pcCardinality qb4o:OneToMany ;
qb4o:rollup sc:inContinent .
```

Level members are represented as instances of the class `qb4o:LevelMember` and attached to the levels they belong to via the property `qb4o:memberOf`, as shown next, using the dimension members for dimension `sc:citDim`, corresponding to Syria. Note that, for attribute instances, we need to link IRIs representing level members, with literals, corresponding to attribute values.

```
citizen:SY
qb4o:memberOf pr:citizen ;
sc:counName "Syria"@en ;
sc:inContinent citDim:AS ;
sc:hasGovType dbpedia:Unitary_state .

citDim:AS
qb4o:memberOf sc:continent ;
sc:contName "Asia" .

dbpedia:Unitary_state
qb4o:memberOf sc:governmentType ;
sc:govName "Unitary state"@en .
```

References

- Cyganiak R, Reynolds D (2014) The RDF Data Cube Vocabulary (W3C Recommendation). <http://www.w3.org/TR/vocab-data-cube/>
- Etcheverry L, Vaisman AA (2012) Enhancing OLAP analysis with web cubes. In: Simperl E, Cimiano P, Polleres, A, Corcho Ó, Prestutti V (eds), The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings, Vol. 7295 of Lecture Notes in Computer Science, Springer, pp. 469–483
- Etcheverry L, Vaisman AA (2012) QB4OLAP: A vocabulary for OLAP cubes on the semantic web. In: Sequeda J, Harth A, Hartig O (eds), Proceedings of the Third International Workshop on Consuming Linked Data, COLID 2012, Boston, MA, USA, November 12, 2012, Vol. 905 of CEUR Workshop Proceedings, CEUR-WS.org
- Ciferri C, Ciferri R, Gómez L, Schneider M, Vaisman AA, Zimányi E (2013) Cube algebra: A generic user-centric model and query language for OLAP cubes. IJDM 9(2):39–65
- Etcheverry L, Vaisman AA (2016) 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, CEUR-WS.org
- Vaisman A, Zimányi E (2014) Data Warehouse Systems: Design and Implementation. Springer, Berlin
- Etcheverry L, and Gómez SA, Vaisman AA, Modeling and Querying Data Cubes on the Semantic Web, CoRR [arXiv:1512.06080](https://arxiv.org/abs/1512.06080)
- Vaisman A A (2015) Publishing OLAP cubes on the semantic web. In: Business Intelligence - 5th European Summer School, eBISS 2015, Barcelona, Spain, July 5-10, 2015, Tutorial Lectures, pp. 32–61
- Abelló A, Darmont J, Etcheverry L, Golfarelli M, Mazón J, Naumann F, Pedersen TB, Rizzi S, Trujillo J, Vassiliadis P, Vossen G (2013) Fusion cubes: towards self-service business intelligence. IJDM 9(2):66–88
- Hurtado C, Gutiérrez C, Mendelzon A (2005) Capturing summarizability with integrity constraints in OLAP. ACM Trans Database Syst 30(3):854–886
- Agrawal R, Gupta A, Sarawagi S (1997) Modeling multidimensional databases. In: Proceedings of the 15th International Conference on Data Engineering (ICDE'97), IEEE Computer Society, Birmingham, UK, pp. 232–243
- Loizou A, Angles R, Groth PT (2015) On the formulation of performant SPARQL queries. J Web Sem 31:1–26
- Pérez J, Arenas M, Gutierrez C (2009) Semantics and complexity of SPARQL. ACM Trans Database Syst (TODS) 34(3):1–45
- Vesse R (2014) SPARQL Optimization 101, Tutorial at ApacheCon North America 2014. <http://events.linuxfoundation.org/sites/events/files/slides/SPARQL%20Optimisation%20101%20Tutorial.pdf>
- Stocker M, Seaborne A, Bernstein A, Kiefer C, Reynolds D (2008) SPARQL basic graph pattern optimization using selectivity estimation. In: Proceedings of WWW, ACM, pp. 595–604
- Schmidt M, Meier M, Lausen G (2010) Foundations of SPARQL query optimization. In: Proceedings of ICDT, ACM, New York, NY, pp. 4–33
- Kämpgen B, Harth A (2013) No size fits all - running the Star Schema Benchmark with SPARQL and RDF aggregate views. In: Cimiano P, Corcho Ó, Presutti V, Hollink L, Rudolph S (eds), The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. In: Proceedings, Vol. 7882 of Lecture Notes in Computer Science, Springer, pp. 290–304
- Kämpgen B, O'Riain S, Harth A (2012) Interacting with statistical linked data via OLAP operations. In: Simperl E, Norton B, Mladenic D, Valle E D, Fundulaki I, Passant A, Troncy R (eds), The Semantic Web: ESWC 2012 Satellite Events-ESWC 2012 Satellite Events, Heraklion, Crete, Greece, May 27-31, 2012. Revised Selected Papers, Vol. 7540 of Lecture Notes in Computer Science, Springer, pp. 87–101
- Neil P O, Neil B O, Chen X (2009) Star Schema Benchmark. <http://www.cs.umb.edu/~poneil/StarSchemaB.PDF>
- TPC.org, TPC-H Benchmark (2014). http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpch2.17.1.pdf
- Nebot V, Llavori RB (2012) Building data warehouses with semantic web data. Decis Support Syst 52(4):853–868

22. Kämpgen B, Harth A (2011) Transforming statistical linked data for use in OLAP systems. In: Proceedings of ICSS, Graz, Austria, pp. 33–40
23. Abelló A, Romero O, Pedersen TB, Berlanga R, Nebot V, Aramburu MJ, Simitsis A (2015) Using semantic web technologies for exploratory OLAP: a survey. *IEEE Trans Knowl Data Eng* 27(2):571–588
24. Ibragimov D, Hose K, Pedersen T B, Zimányi E (2014) Towards exploratory OLAP over linked open data - A case study. In: Castellanos M, Dayal U, Pedersen TB, Tatbul N (eds), *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 of *Lecture Notes in Business Information Processing*, Springer, pp. 114–132
25. Colazzo D, Goasdoué F, Manolescu I, Roatis A (2014) Rdf analytics: Lenses over semantic graphs. In: Proceedings of the 23rd International Conference on World Wide Web, WWW '14, ACM, pp. 467–478
26. Azirani E A, Goasdoué F, Manolescu I, Roatis A (2015) 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, IEEE, pp. 71–76
27. Letelier A, Pérez J, Pichler R, Skritek S (2013) Static analysis and optimization of semantic web queries. *ACM TODS* 38(4):25
28. Tsialiamanis P, Sidiropoulos L, Fundulaki I, Christophides V, Boncz P (2012) Heuristics-based query optimisation for SPARQL. In: Proceedings of EDBT, ACM, pp. 324–335
29. Jakobsen KA, Andersen AB, Hose K, Pedersen TB (2015) Optimizing RDF data cubes for efficient processing of analytical queries. In: Hartig O, Sequeda J, Hogan A (eds), 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 of *CEUR Workshop Proceedings*, CEUR-WS.org