Querying Semantic Web Data Cubes

Lorena Etcheverry[1] and Alejandro Vaisman[2]

[1] Instituto de Computación, Universidad de la República, Montevideo, Uruguay
lorenae@fing.edu.uy
[2] Instituto Tecnológico de Buenos Aires,Buenos Aires, Argentina
avaisman@itba.edu.ar

Abstract. We address the problem of querying data cubes for Online Analytical Processing (OLAP) analysis, directly on the Semantic Web (SW). We first introduce CQL, a simple algebra for querying data cubes at a *conceptual* level. Taking advantage of QB4OLAP metadata, we automatically translate CQL queries into SPARQL ones, and propose query optimization strategies that adapt, to the particular OLAP setting, general-purpose techniques. A web application allows exploring and querying OLAP cubes on the SW, using the machinery presented here.

1 Introduction

Data Warehouses (DW) integrate multiple data sources for analysis and decision support, representing data according to the Multidimensional Model (MD). Typically, DWs and Online Analytical Processing (OLAP), had been used as techniques for data analysis *within* organizations. However, initiatives such as Open Data[3], and the Linked Data[4] paradigm, are encouraging organizations to publish and share MD data, using Semantic Web (SW) standards like RDF[5], and the SPARQL[6] query language. In this context, the BI community faces two challenges: First, provide mechanisms to share MD metadata, essential to interpret and reuse MD data; Second, enable OLAP analysis of SW data. On the first direction, the RDF Data Cube Vocabulary (QB) [3], is the current W3C standard for statistical data. However, QB does not include key features, e.g., dimension hierarchies, needed for OLAP analysis. To address this challenge, the QB4OLAP vocabulary has been proposed [4], which allows reusing data already published in QB, just by adding the needed MD schema semantics, and the corresponding data instances. QB4OLAP also addresses the second challenge, providing means to represent MD metadata needed, e.g., to automatically produce a SPARQL representation of the most usual OLAP operations.
Contribution. We address the second challenge above, proposing a simple algebra for OLAP (denoted *CQL*, and standing for *cube query language*) whose

[3] http://okfn.org/opendata/
[4] http://linkeddata.org/
[5] http://www.w3.org/TR/rdf-concepts/
[6] http://www.w3.org/TR/sparql11-query/

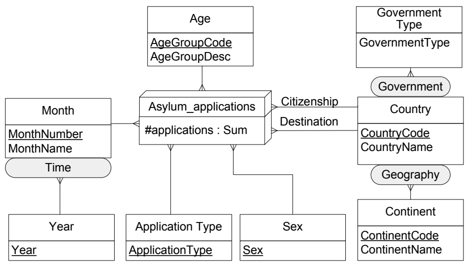

Fig. 1: Conceptual schema of the Asylum Applications cube

main data type is the data cube. We sketch a query simplification strategy for CQL, and propose algorithms to automatically translate CQL queries into equivalent SPARQL ones over QB4OLAP data cubes, and a strategy to improve the performance of those SPARQL queries. Preliminary results (discussed briefly) have shown that our strategies substantially speed up the query evaluation process, outperforming other proposals. A web application allowing to explore and query QB4OLAP cubes has been developed. We remark that our goal is to enable OLAP practitioners, *without any knowledge of SPARQL or SW concepts*, to write efficient queries using just operators well-known for any OLAP user, over a conceptual MD model, regardless the underlying data model and data types.
Running Example. Throughout this paper we use an example based on statistical data about asylum applications to the European Union, provided by Eurostat.[7] The original data set consists of observations reporting the number of applications by month, sex, age, application type, country of origin, and destination country. To show the potential of QB4OLAP, we enriched the existent data set by building aggregation hierarchies. Figure 1 shows the conceptual schema of the extended data cube, using the MultiDim notation [14]. The Asylum_applications fact contains a measure (#applications) representing the number of applications. There are six analysis dimensions: Sex and Age of the applicant, Time of the application, the Application_type (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 dimension participates with two roles: the Citizenship of the asylum applicant, and Destination country of the application.

In the remainder, Section 2 sketches the QB4OLAP vocabulary. Section 3 presents the CQLalgebra and the data model, for querying QB4OLAP data cubes. In Section 4 we describe the CQL-to-SPARQL translation process. Section 5 briefly discusses related work, and we conclude in Section 6.

[7] `http://eurostat.linked-statistics.org/`

2 Representing cubes in QB4OLAP

We now use our running example to concisely show how data cubes, dimension schemas and dimension instances can be represented using QB4OLAP.[8] Like in QB, the schema of a data set is specified by means of the DSD. QB4OLAP represents the structure of a data set (i.e., a cube) in terms of dimension *levels* (introducing the class `qb4o:LevelProperty`), and measures. The prefixes used in this work are presented in Figure 2 in Appendix A.

Example 1. (QB4OLAP cube structure) Below we show the representation of the structure of a data cube for the Eurostat example, using QB4OLAP.

```
schema:migr_asyappctzmQB4O rdf:type qb:DataStructureDefinition ;
  qb:component [ qb4o:level property:age ; qb4o:cardinality qb4o:ManyToOne ] ;
  qb:component [ qb4o:level sdmxd:refPeriod ; qb4o:cardinality qb4o:ManyToOne ] ;
...
  qb:component [ qb4o:level property:citizen ; qb4o:cardinality qb4o:ManyToOne ] ;
  qb:component [ qb:measure sdmx−measure:obsValue ; qb4o:aggregateFunction qb4o:sum ] .
```

Note that dimension levels in this cube are the lowest levels in the dimension hierarchies. *Observations* (in OLAP terminology, *facts*), represent points in an MD space, complying with the schema given above. □

The class `qb4o:Hierarchy` represents *dimension hierarchies*, attached to a dimension via the property `qb4o:hasHierarchy`. Class `qb4o:HierarchyStep` represents the parent-child relationship between two levels. Each step is associated via `qb4o:rollup` with a custom property that implements the rollup relationship at the instance level.

Example 2. (QB4OLAP dimensions) We now define the Citizenship dimension of Figure 1 (`schema:citizenshipDim`), and its Geography hierarchy.

```
# Dimension definition
schema:citizenshipDim a qb:DimensionProperty; rdfs:label "Citizenship dimension"@en;
  qb4o:hasHierarchy schema:citizenshipGeoHier, schema:citizenshipGovHier .
# Hierarchy definition
schema:citizenshipGeoHier a qb4o:Hierarchy; rdfs:label "Citizenship Geo Hierarchy"@en;
   qb4o:inDimension schema:citizenshipDim; qb4o:hasLevel property:citizen, schema:continent.
# Base level
property:citizen a qb4o:LevelProperty; rdfs:label "Country of citizenship"@en;
  qb4o:hasAttribute schema:countryName.
schema:countryName a qb4o:LevelAttribute;
  rdfs:label "Country name"@en; rdfs:range xsd:string.
#Upper hierarchy levels
schema:continent a qb4o:LevelProperty; rdfs:label "Continent"@en;
  qb4o:hasAttribute schema:continentName.
schema:continentName a qb4o:LevelAttribute;
  rdfs:label "Continent name"@en; rdfs:range xsd:string.
#rollup relationships
schema:inContinent a qb4o:RollupProperty.
schema:hasGovernment a qb4o:RollupProperty.
#Hierarchy step
_:ih1 a qb4o:HierarchyStep;
  qb4o:inHierarchy schema:citizenshipGeoHier;
  qb4o:childLevel property:citizen; qb4o:parentLevel schema:continent;
  qb4o:pcCardinality qb4o:OneToMany; qb4o:rollup schema:inContinent.
```

[8] http://purl.org/qb4olap/cubes

Note that levels `property:citizen` and `schema:continent` are defined, as well as their attributes, e.g., `schema:continentName`. □

Example 3. (Dimension Instances) Level members are attached to dimension levels via the property `qb4o:memberOf`. Below we show dimension members corresponding to France, for dimension `schema:citizenshipDim`.

```
citizen:FR
  qb4o:memberOf property:citizen ; schema:countryName "France"@en;
  schema:inContinent citDim:EU ; schema:hasGovernment dbpedia:Unitary_state .

citDim:EU
  qb4o:memberOf schema:continent ; schema:continentName "Europe" .

dbpedia:Unitary_state
  qb4o:memberOf schema:governmentType ; schema:governmentName "Unitary state"@en .
```

□

3 Querying QB4OLAP cubes

To allow users to query QB4OLAP cubes without dealing with SPARQL, we propose an algebra, denoted *CQL*, based on [2]. To evaluate a query in CQL, first, CQL queries are simplified (i.e. to eliminate redundancy and reorder operations), and then translated into a single SPARQL expression, following a *naïve* approach. Finally, we apply SPARQL optimization heuristics to improve the performance of the *naïve* queries.

The CQL Language We next define a formal data model for cubes, and OLAP operators over this model. A cube expressed in this model can be represented using the QB4OLAP vocabulary. Due to space limitations, we only present informally the main ideas, and refer the reader to [5] for details.

A *dimension schema* is a tuple $\langle d, \mathcal{L}, \rightarrow, \mathcal{H} \rangle$ where d is the name of the dimension; $\mathcal{L}$ is a set of pairs $\langle l, \mathsf{A}_l \rangle$, where l is a level in $\mathcal{L}$, and A_l is a set of attributes associated with l; '$\rightarrow$' is a partial order between pairs of levels in $\mathcal{L}$ with a unique bottom level and a unique level, denoted All; $\mathcal{H}$ is a set of pairs $\langle h_n, L_h \rangle$, called *hierarchies*, where h identifies the hierarchy, and $L_h \subseteq \mathcal{L}$ is the set of levels in the hierarchy. Given a schema for dimension d, a *dimension instance* I_d for d is defined as follows: for each level l in d there is a set of tuples $\mathcal{T}_l$, and such that for each attribute $a_i \in \mathsf{A}_l$ there is a value from $Dom(a_i)$ in $\mathcal{T}_l(a_i)$. In addition, there is $\mathcal{R}$, a finite set of *rollup relations*, one for each pair of levels in '$\rightarrow$', denoted as follows: $\mathsf{RUP}_{L_i}^{L_j}, L_i, L_j \in \mathcal{L}$, where $L_i \rightarrow L_j$, associating elements in $\mathcal{T}_{l_i}$ with elements in $\mathcal{T}_{l_j}$.

A *cube schema* is a tuple $\langle C_n, \mathcal{D}, \mathcal{M}, \mathcal{F} \rangle$ where C_n is the name of the cube; $\mathcal{D}$ is a finite set of dimension schemas; $\mathcal{M}$ is a finite set of attributes called *measures*; and $\mathcal{F}$ is a function mapping a measure $m \in \mathcal{M}$ to an aggregate function. Given a cube schema with D dimensions and M measures, and a set of levels $\mathcal{V}_{Cb} = \{l_1, l_2, \ldots, l_D\}$, such that there are not two levels belonging to the same dimension, a *cuboid instance* Cb is a partial function $Cb : \mathcal{T}_{l_1} \times \cdots \times \mathcal{T}_{l_D} \rightarrow Dom(m_1) \times \cdots \times Dom(m_M)$, where $m_k \in \mathcal{M}, \forall k, k = 1, \ldots, M$, where $\mathcal{T}_{l_1}$ are the instances of level $l_i \in I_{d_i}$. The elements in the domain of Cb are called *cells*, and $\mathcal{V}_{Cb}$ it the *set of levels* of the cuboid.

The above allows us to define a *lattice of cuboids*, provided that we define an order between cuboids, as follows. 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 belonging to the same dimension d. 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 $l_c \;\rightarrow\; l_p \in d$, then $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$. Moreover, for each pair of adjacent cuboids $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$, if we assume that $\mathsf{RUP}_{l_p}^{lc}$ is a function, each cell in each cell Cb_2 can be obtained from the cells in Cb_1, aggregating the measures along the dimension d using such function, and the aggregation functions associated to each measure. Finally, a *Cube Instance* is the lattice of all cuboids that share the same cube schema. The bottom of this lattice is the original cube, and the top of the lattice is the cuboid with just the All level for all the dimensions in the cube.

Now that we have the intuition of what a cube is, we are ready to give a precise semantics for the operators composing the CQL algebra (see [5] for details).

The Roll-up operator summarizes data to a higher level along a dimension hierarchy; that is, it receives a cuboid Cb_1 in a cube instance, and returns another cuboid Cb_2 in the same instance, such that $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$. The Drill-down operator does the inverse, i.e., it returns a cuboid Cb_2 such that $\mathsf{Cb}_2 \preceq \mathsf{Cb}_1$. It is clear that a Drill-down over a dimension d can be obtained performing a Roll-up over d from the bottom cuboid. We will use this result in the sequel. Since Roll-up and Drill-down only imply a navigation across a lattice (and do not modify it), we call them Instance Preserving Operators (IPO).

The Dice operator selects the cells in a cuboid that satisfy a boolean condition ϕ, expressed over level member attributes, and/or measure values. The Slice operator removes one of the dimensions or measures in the cube. We denote these operators Instance Generating Operators (IGO), since they induce a new lattice (because they reduce the dimensionality of the cube, or because they reduce the number of cells in the cuboid), whose bottom cuboid is the result of the corresponding operation. Again, see [5] for details.

Example 4. (CQL syntax) Given the query: *Total asylum applications submitted by African citizens to France in 2013, by sex, time, age, and citizenship country*; the CQL query below produces the answer.

```
$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);
```

The notation of the DICE operation is (dimension|level|attribute). Note that the ROLLUP to the continent level, is performed to allow selecting African citizens. Note that the DRILLDOWN takes the cube down to the Citizenship level (the applicant's country), and the two final SLICE operations remove dimensions Application Type and Destination since they are not wanted in the result. □

To restrict the problem to queries that can be evaluated without storing the computation trace, we limit ourselves to consider the subset of CQL queries that satisfy the following patterns, where DICE$_l$ and DICE$_m$ denote dicing operations on level attribute or measure values, respectively:(a) $(\text{SLICE}^*|\text{DICE}^*|\text{ROLL-UP}^*)^+$;
(b)$(\text{SLICE}^*|\text{ROLL-UP}^+|\text{DRILL-DOWN}^+|\text{DICE}_l^+)^+$;
(c)$(\text{SLICE}^*|\text{ROLL-UP}^+|\text{DRILL-DOWN}^+|\text{DICE}_l^*)^+\text{DICE}_m^+$.

CQL simplification Since CQL is aimed at being used by non-experts, input CQL queries may include unnecessary operations. Further, operations can be reordered to reduce the size of the cuboid as early as possible. Thus, we devised the following set of rewriting rules: (1) Remove all the ROLL-UP or DRILL-DOWN operations with the same origin and target level; (2) Given a sequence of ROLL-UP and/or DRILL-DOWN operations over a dimension d, without DICE$_l$ operations in-between, and where $l \in d$, find the last level l_D in the sequence, and if l_D is not the bottom level of d, replace the sequence with a single ROLL-UP from the bottom to l_d; Otherwise, remove all the operations in the group; (3) If there is a SLICE over a dimension d (respectively, a measure m), and no DICE operation that considers level members of d (respectively, mentions m), move the SLICE to the beginning of the query; otherwise move it to the end; (4) If there is a SLICE over a dimension d, a sequence of ROLL-UP and DRILL-DOWN operations over d, and no DICE operation mentions levels in d, remove all the ROLL-UP and DRILL-DOWN operations, and keep only the SLICE one.

It can be proved that, applying these rules produces a query such that: (a) If there are no DICE operations in a CQL query, there is at most one ROLL-UP and no DRILL-DOWN operation for each Dimension d; (b) SLICE operations are either at the beginning or at the end of the query, but not in the middle.

4 CQL to SPARQL translation

The next step in the process is the translation of CQL queries (expressed at the conceptual level), into SPARQL expressions over QB4OLAP cubes (logical level), avoiding the materialization of intermediate results. We present the ideas by means of an example. Let us consider the query: *Total asylum applications per year submitted by Asian citizens to France or United Kingdom, where the number of applications is > 5000*, expressed in CQL as:

```
$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")));
```

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.

```
1  SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 ?ag1
2  WHERE {
3    { SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 (SUM(xsd:integer(?m1)) as ?ag1)
4      FROM loc−ins:migr_asyapp_clean
5      FROM loc−sch:migr_asyappctzmQB4O13
6      WHERE { ?o a qb:Observation . ?o qb:dataSet dt:migr_asyappctzm .
7              ?o sdmxm:obsValue ?m1 .
8              ?o property:citizen ?lm1 . ?lm1 qb4o:memberOf property:citizen .
9              ?lm1 schema:inContinent ?plm1 . ?plm1 qb4o:memberOf schema:continent .
10             ?o sdmxd:refPeriod ?lm2 . ?lm2 qb4o:memberOf sdmxd:refPeriod .
11             ?lm2 schema:inYear ?plm2 . ?plm2 qb4o:memberOf schema:year .
12             ?o property:geo ?lm3 . ?o property:sex ?lm4 .
13             ?o property:age ?lm5 . ?o property:asyl_app ?lm6 .
14             ?plm1 schema:continentName ?plm11 .
15             ?lm3 schema:countryName ?lm31 .
16             FILTER ( ?plm11="Asia" && ((?lm31="France"@en) || (?lm31="United␣Kingdom"@en)))
17   } GROUP BY ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
18 } FILTER ( ?ag1 > 5000) }
```

Lines 8 and 9 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. Then, we navigate the hierarchy up to the level Continent, using the rollup property `schema:inContinent`. 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 18), and in the result of the outer query (line 1). The navigation corresponding to the Rollup in C2 is performed analogously. Lines 12 and 13 instantiate the level members of the remaining dimensions, and variables are added to the `GROUP BY` and `SELECT` clauses of the inner and outer queries, respectively. Line 7 retrieves the value of the measure in each observation, and the SUM aggregate function computes ?ag1 in line 3. The aggregated value is added to result of the outer query (line 1) (measure values are converted to integer before applying the SUM, 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 14) and then use a `FILTER` clause to keep only the cells that correspond to "Asia" (line 16). Analogously, the restrictions on country names are implemented in line 16 (country names are retrieved in line 15), 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 19).

SPARQL queries improvement To improve the performance of the queries produced by the *naïve* algorithm, we adapted existing SPARQL optimization techniques to the characteristics of MD data and QB4OLAP.

As a first strategy, we adapted to our setting the heuristics proposed by Loizou et al. [9] From the five heuristics proposed, we choose two that are applicable (e.g., heuristics related to the `OPTIONAL` clauses do not apply, since the SPARQL queries we produce do not use that clause), and have shown to improve performance across different triplestores. Concretely, we use two heuristics:

H1 - Use named graphs to localize SPARQL graph patterns. To take advantage of this, we organize QB4OLAP data into two named graphs. The *schema* graph stores the schema and dimension members, while the *instance* 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.
H2 - Specifying alternative URIs. Proposes to transform `FILTER` clauses with disjunction (||) of equality constraints, using either the `UNION` of patterns, or a `VALUES` expression. Since the reported results are not conclusive on which of these strategies leads to better performant queries, we decided to try them both in our experimental evaluation (not discussed in this paper).

As a second strategy, we considered the recommendations in [15], namely: (i) Split conjunctive `FILTER` equality constraints into a cascade of `FILTER` equality constraints; (ii) Replace `FILTER` equality constraints that compare a variable and a constant with graph patterns.

We next show the result of applying these strategies to the SPARQL query above. The application of **H1** organizes graph patterns in the inner query in two `GRAPH` clauses: one for the instance graph (lines 7 to 11), and another for the schema graph (lines 12 to 19). Applying **H2**, the `FILTER` clause on country names is replaced by a `VALUES` clause (line 19). Filter clauses are split, and the `FILTER` clause on continent name is replaced by a graph pattern (line 15).

```
1  SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 ?ag1
2  WHERE {
3   { SELECT ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6 (SUM(xsd:integer(?m1)) as ?ag1)
4     FROM NAMED loc−ins:migr_asyapp_clean
5     FROM NAMED loc−sch:migr_asyappctzmQB4O13
6     WHERE {
7      {GRAPH loc−ins:migr_asyapp_clean
8        {?o a qb:Observation . ?o qb:dataSet dt:migr_asyappctzm .
9         ?o sdmxm:obsValue ?m1 . ?o property:citizen ?lm1 .
10        ?o sdmxd:refPeriod ?lm2 . ?o property:geo ?lm3 .
11        ?o property:sex ?lm4 . ?o property:age ?lm5 . ?o property:asyl_app ?lm6 . }}.
12     {GRAPH loc−sch:migr_asyappctzmQB4O13
13       {?lm1 qb4o:memberOf property:citizen .
14        ?lm1 schema:inContinent ?plm1 . ?plm1 qb4o:memberOf schema:continent .
15        ?plm1 schema:continentName "Asia" .
16        ?lm2 qb4o:memberOf sdmxd:refPeriod . ?lm2 schema:inYear ?plm2 .
17        ?plm2 qb4o:memberOf schema:year .
18        ?lm3 schema:countryName ?lm31 .
19        VALUES ?lm31 {"France"@en "United␣Kingdom"@en} }}
20  } GROUP BY ?plm1 ?plm2 ?lm3 ?lm4 ?lm5 ?lm6
21 } FILTER (?ag1 > 5000) }
```

Finally, based on Stocker et. al [12], we propose to reorder triple patterns on the schema graph to further improve the performance of SPARQL queries. This optimization is based on graph pattern selectivity. The idea is to apply first the most selective patterns. This requires to keep estimates on the selectivity of each pattern. 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 exact one parent on the level immediately above. Thus, for each pair of levels l_i and l_j such that $l_i \rightarrow l_j$, $|l_i| \geq |l_j|$. Based on the above, we define the following *ordering criteria* (OC) for the graph patterns: (1) For each dimension appearing in the query, apply first the patterns that correspond to higher levels (**OC1**); (2) For each dimension, apply OC1. Then, reorder dimensions considering first, dimensions with conditions that fix a certain member, then dimensions with conditions that restrain to a range of members, and then the other dimensions (**OC2**); (3) For each dimension, apply OC1, then OC2. If more than one dimension satisfy any of the criteria in OC2, proceed as follows: If dimensions A and 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.

Below, we show the result of applying OC2 to reorder the triple patterns on the second graph of the query. For each dimension, the graph patterns are ordered from higher levels in the hierarchy to lower ones. 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_asyappctzmQB4O13 {
  ?plm1 schema:continentName "Asia" .
  ?plm1 qb4o:memberOf schema:continent .
  ?lm1 schema:inContinent ?plm1 . ?lm1 qb4o:memberOf property:citizen .
  ?lm3 schema:countryName ?lm31 .
  VALUES ?lm31 {"France"@en "United␣Kingdom"@en}
  ?plm2 qb4o:memberOf schema:year . ?lm2 schema:inYear ?plm2 .
  ?lm2 qb4o:memberOf sdmxd:refPeriod .}
```

Experimental results and implementation For the sake of space we just report that comprehensive experiments have been run adapting the Star Schema Benchmark (SSB) [10], with 132,000,000 triples, running the 13 queries in the benchmark and applying different combinations of our strategies. The conclusions were that our proposal largely outperforms [6,7], and that some combinations of optimization strategies improve the SPARQL query produced by the naïve translation up to 10 times. In addition, a toolkit allowing exploring and querying QB4OLAP cubes is publicly available.[9]

5 Related Work

Kämpgen et al. [6,7] attempt to override the lack of structure in QB by defining an OLAP data model on top of QB using other vocabularies, to represent the hierarchical structure of the dimensions. Further, in [6] the authors implement some OLAP operators over those extended cubes, using SPARQL queries, restricted to data cubes with only one hierarchy per dimension, and 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 use of SW technologies in OLAP is surveyed in [1].

Regarding SPARQL query processing, many works study the complexity of query evaluation [11]. In [8] the authors focus on the static analysis of SPARQL

[9] `https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/`

queries, in particular those that contain the `OPTIONAL` operator. Tsialimanis et. al [13] 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.

6 Conclusion

We have described a simple algebra (CQL) over data cubes, that can be used to express OLAP queries, and automatically translated into efficient SPARQL queries. First, we use QB4OLAP metadata to obtain a naïve translation of CQL to SPARQL; then, we adapted general-purpose SPARQL optimization techniques to the OLAP setting. Our experiments over a modified SSB showed that our techniques outperform other proposals, and suggested the best combinations of optimization strategies. An application to explore and query SW cubes completes our contibutions. We think that these results can encourage and promote the publication and sharing of MD data on the SW, and we plan to continue working in this direction, extending CQL with other OLAP operations.

A Prefixes definition

```
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 property: <http://eurostat.linked-statistics.org/property#>
PREFIX citizen: <http://eurostat.linked-statistics.org/dic/citizen#>
PREFIX dt: <http://eurostat.linked-statistics.org/data/>
PREFIX loc-ins: <http://www.fing.edu.uy/cubes/instances/>
PREFIX loc-sch: <http://www.fing.edu.uy/cubes/schemas/>
PREFIX schema: <http://www.fing.edu.uy/cubes/schemas/migr_asyapp#>
PREFIX citDim: <http://www.fing.edu.uy/cubes/dims/migr_asyapp/citizen#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
```

Fig. 2: RDF prefixes used in this work

References

1. Abelló, A., Romero, O., Pedersen, T.B., Berlanga, R., Nebot, V., Aramburu, M.J., Simitsis, A.: Using semantic web technologies for exploratory OLAP: A survey. IEEE TKDE 27(2), 571–588 (2015)

2. Ciferri, C., Ciferri, R., Gómez, L., Schneider, M., Vaisman, A., Zimányi, E.: Cube algebra: A generic user-centric model and query language for OLAP cubes. IJDWM 9(2), 39–65 (2013)
3. Cyganiak, R., Reynolds, D.: The RDF Data Cube Vocabulary (W3C Recommendation) (January 2014), `http://www.w3.org/TR/vocab-data-cube/`
4. Etcheverry, L., Vaisman, A.: QB4OLAP: A vocabulary for OLAP cubes on the semantic web. In: Proc. of COLD. CEUR-WS.org, Boston, USA (2012)
5. Etcheverry, L., Gomez, S., Vaisman, A.: Modeling and querying data cubes on the semantic web. arXiv preprint arXiv:1512.06080 (2015)
6. Kämpgen, B., Harth, A.: No size fits all - running the star schema benchmark with SPARQL and RDF aggregate views. In: The Semantic Web: Semantics and Big Data, LNCS, vol. 7882, pp. 290–304. Springer (2013)
7. Kämpgen, B., O'Riain, S., Harth, A.: Interacting with Statistical Linked Data via OLAP Operations. In: Proceedings of ESWC. CEUR-WS.org (2012)
8. Letelier, A., Pérez, J., Pichler, R., Skritek, S.: Static analysis and optimization of semantic web queries. ACM TODS 38(4), 25 (2013)
9. Loizou, A., Angles, R., Groth, P.: On the formulation of performant SPARQL queries. Web Semantics: Science, Services and Agents on the WWW 31, 1–26 (2014), `http://linkinghub.elsevier.com/retrieve/pii/S1570826814001061`
10. Neil, P.O., Neil, B.O., Chen, X.: Star Schema Benchmark (2009), `http://www.cs.umb.edu/{~}poneil/StarSchemaB.PDF`
11. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. ACM Transactions on Database Systems (TODS) 34(3), 1–45 (2009)
12. Stocker, M., Seaborne, A.: SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. Proceedings of WWW'08 pp. 595–604 (2008)
13. Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., Boncz, P.: Heuristics-based query optimisation for SPARQL. In: Proceedings of EDBT. pp. 324–335. ACM (2012)
14. Vaisman, A., Zimányi, E.: Data Warehouse Systems: Design and Implementation. Springer (2014)
15. Vesse, R.: SPARQL Optimization 101, Tutorial at ApacheCon North America 2014 (2014), `http://events.linuxfoundation.org/sites/events/files/slides/SPARQL%20Optimisation%20101%20Tutorial.pdf`