QB4OLAP: A New Vocabulary for OLAP Cubes on the Semantic Web *

Lorena Etcheverry and Alejandro A. Vaisman

[1] Instituto de Computación, Facultad de Ingeniería, Universidad de la República
lorenae@fing.edu.uy
[2] Université Libre de Bruxelles
avaisman@ulb.ac.be

Abstract. On-Line Analytical Processing (OLAP) tools allow querying large multidimensional (MD) databases called data warehouses (DW). OLAP-style data analysis over the semantic web (SW) is gaining momentum, and thus SW technologies will be needed to model, manipulate, and share MD data. To achieve this, the definition of a vocabulary that adequately represents OLAP data is required. Unfortunately, so far, the proposals in this direction have followed different roads. On the one hand, the QB vocabulary (a proposal by the W3C Government Linked Data Working Group) follows a model initially devised for analyzing statistical data, but does not adequately support OLAP multidimensional data. Another recent proposal, the Open Cube vocabulary (OC) follows closely the classic MD models for OLAP and allows implementing OLAP operators as SPARQL queries, but does not provide a mechanism for reusing data already published using QB. In this work, we propose a new vocabulary, denoted QB4OLAP, which extends QB to fully support OLAP models and operators. We show how data already published in QB can be analyzed *à la* OLAP using the QB4OLAP vocabulary, and vice versa. To this end we provide algorithms that build the structures that allow performing both kinds of analysis, and show that compatibility between QB and QB4OLAP can be achieved at low cost, only adding dimensional information.

1 Introduction

Business Intelligence (BI) comprises a collection of techniques used for extracting and analyzing business data to support decision-making. As part of the BI machinery, On-Line Analytical Processing (OLAP) [9] tools and algorithms allow querying large multidimensional (MD) databases called data warehouses (DW). Data in a DW come from heterogeneous and distributed operational sources and go through a process denoted ETL (standing for Extraction, Transformation, and Loading). In OLAP, data are usually seen as a *cube*, where each cell contains measures representing facts and contextual information (the latter called *dimensions*). Traditional OLAP tools have proven to be successful in analyzing large sets of enterprise data, but sometimes these highly curated data is not enough in today's business dynamics. External data (particularly web data) can enhance local analysis by means of, e.g., *fusion cubes* [1]. Also, OLAP-style analysis of semantic web (SW) data is likely to become crucial in the near future, as large

* This research has been partially funded by LACCIR project R1210LAC004.

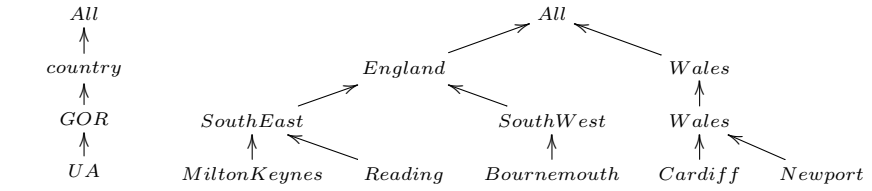
repositories of semantically annotated data are becoming available [11]. So far, this kind of analysis of SW data is performed extracting MD information from the Semantic Web into traditional OLAP databases. This approach requires the existence of a local data warehouse (DW) to store semantic web data, and this restriction clashes with the autonomous and high volatile nature of web data sources. Changes in the sources may lead not only to updates on data instances, but also to changes in the structure of the DW that will also impact on the ETL processes. Since the DW construction process needs human supervision, this approach not only does not automatically reflect changes on the sources, but also can be very hard to update and maintain. In [5] we have discussed these drawbacks, and claimed that performing OLAP operations *directly over RDF data* can be useful and plausible in certain scenarios.

BI over the SW requires, as a starting point, the definition of a precise vocabulary allowing to represent OLAP data. Unfortunately, so far, the proposals in this direction have followed different roads. The RDF Data Cube vocabulary (QB) [4] follows models originally devised to analyze statistical data. On the other hand, the recently proposed Open Cube (OC) vocabulary [5] (also expressed in RDF) closely follows the classic MD models for OLAP existing in the literature. In this paper we show that the former presents problems when dealing with hierarchical data, and that OC does not directly support data cubes defined over QB (Section 3). In light of this, we introduce the QB4OLAP vocabulary (Section 4), which extends QB to fully support OLAP models, and allows implementing OLAP operators directly over the RDF representation. We also provide algorithms that build the QB4OLAP structures needed to analyze observations already published using QB, and vice versa (Section 5), showing that for this we only need to modify dimensional information, which is usually small compared with the size of OLAP fact data (called *observations* in statistical databases).

2 OLAP Preliminary Concepts

We first define the MD model for OLAP that will be used in our study. Detailed formal models can be found in [6,7,12], among other ones. We assume the reader is familiar with basic notions of the SW, RDF, and SPARQL.

In OLAP, data are organized as hypercubes whose axes are called *dimensions*. Each point in this MD space is mapped into one or more spaces of *measures*, representing *facts* that are analyzed along the cube's dimensions. Dimensions are structured in *hierarchies* that allow analysis at different aggregation *levels*. The actual values in a dimension level are called *members*. A **Dimension Schema** is composed of a non-empty finite set of levels, with a distinguished level denoted *All*. We denote '$\rightarrow$' a partial order on these levels; the reflexive and transitive closure of '$\rightarrow$' ('$\rightarrow^*$') has a unique bottom level and a unique top level (the latter denoted *All*). Levels can have attributes describing them. A **Dimension Instance** assigns to each dimension level in the dimension schema a set of dimension *members*. For each pair of levels (l_j, l_k) in the dimension schema, such that $l_j \rightarrow l_k$, a relation (denoted *rollup*) is defined, associating members from level l_j with members of level l_k. A **Cube Schema** contains a set of dimension schemas and a set of measures, where for each measure an aggregate function is specified. A **Cube**



(a) Schema (b) An instance

Fig. 1: Dimension schema and instance: geoDim

Instance, corresponding to a cube schema, is a partial function mapping coordinates from dimension instances into measure values.

Example 1. (Cube schema and instance) We want to analyze the household projection in the UK (a measure) along the dimensions geographic location and date. Let us call these dimensions geoDim and dateDim, respectively. Dimension geoDim is organized in a hierarchy of levels: unitary authority (UA), Government Office Region (GOR) and country; dimension dateDim has only one level: year. The household measure is associated with the SUM aggregate function. Based on these dimensions and measures, we define a cube, denoted householdCS. Figure 1 shows dimension geoDim (its schema and a sample instance), while Figure 2a presents an instance of householdCS cube retrieved from StatsWales[3] and Open Data Communities[4].The symbol $\perp$ means that the value of a measure, corresponding to a set of coordinates, is unknown or undefined. □

A well-known set of operations is defined over cubes. We present some of these operations next. They are based on the recently proposed *Cube Algebra* [2].

Roll-Up summarizes data in a cube, along a dimension hierarchy. Given a cube $\mathcal{C}$, a dimension $D \in \mathcal{C}$, and a dimension level $l_u \in D$ such that $l_l \rightarrow^* l_u$ with l_l the level in dimension D used to represent cube instances in $\mathcal{C}$, Roll-Up($\mathcal{C}, D, l_u$) returns a new cube $\mathcal{C}'$ where measure values are aggregated along D up to the level l_u, using the aggregate function defined for each measure. Analogously, **Drill-Down** disaggregates previously summarized data, and can be considered the inverse of Roll-Up. Note that this requires to store the aggregation path. **Slice** receives a cube $\mathcal{C}$ and a dimension $D \in \mathcal{C}$, and returns a new cube $\mathcal{C}'$, with the same schema as $\mathcal{C}$ except for dimension D. Measure values in the cube are aggregated along dimension D up to level All before removing the dimension. **Dice** receives a a cube $\mathcal{C}$, and a first order formula σ over levels and measures in $\mathcal{C}$, and returns a new cube $\mathcal{C}'$ which has the same schema as cube $\mathcal{C}$. Instances in $\mathcal{C}'$ are the instances in $\mathcal{C}$ that satisfy σ.

Example 2 (OLAP Operators). Consider the cube schema of Example 1 and the cube instance in Figure 2a. The result of Slice($householdCS$, geoDim) is shown in Figure 2b. Figure 2c shows the result of Dice($householdCS$, household $>= 60 \wedge$ GOR $<>$ South West). Figure 2d shows Roll-Up($householdCS$, geoDim, GOR). □

[3] Report number 028727 in `http://statswales.wales.gov.uk/index.htm`

[4] Household projections by district, England, 1991-2033 `http://opendatacommunities.org/id/dataset/housing/household-projections`

			year		
country	**GOR**	**UA**	*2006*	*2007*	*2008*
England	*South East*	*Milton Keynes*	92	94	96
		Reading	58	58	60
	South West	*Bournemouth*	71	72	73
Wales	*Wales*	*Cardiff*	132.1	134.2	136.7
		Newport	58.7	⊥	59.6

(a) An instance of householdCS

year		
2006	*2007*	*2008*
411.8	358.2	425.3

(b) $Slice$

			year		
country	**GOR**	**UA**	*2006*	*2007*	*2008*
England	*South East*	*Milton Keynes*	92	94	96
		Reading	⊥	⊥	60
Wales	*Wales*	*Cardiff*	132.1	134.2	136.7

(c) $Dice$

		year		
country	**GOR**	*2006*	*2007*	*2008*
England	*South East*	150	152	156
	South West	71	72	73
Wales	*Wales*	190.8	134.2	196.3

(d) $Roll - Up$

Fig. 2: Applying OLAP operations to a cube

3 Representing Multidimensional Data in RDF

3.1 The RDF Data Cube Vocabulary (QB)

The RDF Data Cube vocabulary [4] (also denoted QB) is a proposal by the W3C Government Linked Data (GLD) Working Group, focused on the publication of statistical data and metadata using RDF and adhering to Linked Data principles. It represents a simplified version of the SDMX Information Model [10]. Although it has not yet become an standard, several datasets are currently being published using this vocabulary.

Observations (in OLAP terminology, facts) are described in QB as instances of `qb:Observation`, representing points in a MD data space indexed by *dimensions*. They are grouped in *datasets* (`qb:DataSet`) by means of the `qb:dataSet` property. The schema of a dataset is specified by the *data structure definition* (DSD) (instance of `qb:DataStructureDefinition`). Although QB can define the structure of a fact (via the DSD), it does not provide a mechanism to represent an OLAP dimension structure (i.e., the dimension levels and the relationships between levels). However, QB allows representing hierarchical relationships between level *members* in the *dimension instances*, using the SKOS vocabulary. The `qb:Slice` construct does not represent an operator over an existent dataset, but the result of the application of a constraint over the dimension values on an existent data cube. Observations that populate the slice are not automatically computed from the observations in the original data cube and must be explicitly added to the RDF graph.We further explain the vocabulary through an example, and refer the reader to the reference for details.

Figure 3 (a) shows a representation using QB of the dimension geoDim. Dimension members are represented using URIs defined in Data.gov.uk[5] according to the Ordnance Survey Administrative Geography Ontology[6]. For the sake of space we only include the data corresponding to the Reading UA and omit prefix declarations. A dimension property is used to define the dimension (line 1). Then, a concept scheme defines the

[5] `http://statistics.data.gov.uk/def/administrative-geography`

[6] `http://www.ordnancesurvey.co.uk/oswebsite/ontology/v1/AdministrativeGeography.rdf`

```
1  eg:geoDim a qb:DimensionProperty,
2      qb:CodedProperty; qb:codeList eg:geo.
3  eg:geo a skos:ConceptScheme;
4      skos:hasTopConcept ns2:921;
5
6  ns2:921 a adgeo:Country;
7      rdfs:label "England@en";
8      skos:inScheme eg:geo;
9      skos:narrower ns1:J.
10 ns1:J a adgeo:GovernmentOfficeRegion;
11     rdfs:label "South East@en" ;
12     skos:inScheme eg:geo;
13     skos:narrower ns0:00mc.
14 ns0:00mc a adgeo:UnitaryAuthority;
15         rdfs:label "The Borough of Reading@en";
16         skos:inScheme eg:geo.
```

(a) The geoDim dimension structure.

```
1  eg:dateDim a qb:DimensionProperty,
2                              qb:CodedProperty.
3  eg:household a qb:MeasureProperty.
4  eg:householdCS a qb:DataStructureDefinition;
5      qb:component [qb:dimension eg:geoDim];
6      qb:component [qb:dimension eg:dateDim];
7      qb:component [qb:measure eg:household];
8      qb:component [qb:attribute
9                              sdmx−attribute:unitMeasure] .
10 eg:dataset−hh a qb:DataSet;
11     rdfs:label "Household in UK"@en;
12     qb:structure eg:householdCS.
13 eg:o1 a qb:Observation;
14     qb:dataSet eg:dataset−hh ;
15     eg:geoDim ns0:00mc ;
16     eg:dateDim db:2007;
17     eg:household 58 ;
18     sdmx−attribute:unitMeasure db:Thousand.
```

(b) Cube structure and instances.

Fig. 3: Expressing schemas and instances in QB

hierarchical relationship between dimension members, which is linked to the dimension property using the `qb:codelist` property (line 2). The most general concepts within the hierarchy are defined using the `skos:topConcept` property. In this example there is only one top concept (line 4) and it corresponds to an instance of the Country class (England). Hierarchical relationships among members (from the more general concepts down to more specific ones) are stated using the `skos:narrower` property (lines 9 and 13). Figure 3 (b) shows a representation, using QB, of the data cube of Figure 2a. The observation depicted represents the household corresponding to Reading in 2007.

The implementation of *OLAP operations* is not considered in the specification of the QB vocabulary. In spite of this, some OLAP operations can be defined over a structure based on QB, although in a limited way. For example, since dimension levels and aggregate functions for each measure are not modeled, is not possible to implement *Roll-Up* over QB. The same issues apply to *Drill-down* and *Slice*. Finally, a particular case of *Dice* could be implemented over QB, given that the FO formula σ can only involve cube measures (again, because of the lack of support of dimension levels).

Kämpgen et al. [8] attempt to override the lack of structure in QB defining an OLAP data model on top of QB, but still do not provide a mechanism to represent aggregate functions. In addition, the work proposes a mechanism for implementing some OLAP operators over these extended cubes, using SPARQL queries, but only provide an implementation of *Roll-Up* in the case of one-level dimensions.

3.2 The Open Cube Vocabulary

Open Cube (OC) [5] is an RDF vocabulary focused on publishing MD data and performing analysis *à la* OLAP directly over such data. It is based on the MD data model presented in Section 2. In OC each level in a dimension is *explicitly* declared (opposite to QB). For each level, its parent in the dimension hierarchy is declared using the `oc:parentLevel` property. Dimension instances are represented separately, and each level member is related to the level it belongs to using the `oc:inLevel`

	MD concept	Data Cube (QB) [4]	Kämpgen et al. [8]	Open Cube [5]
1	Dimensions	Yes	Yes	Yes
2	Levels	No	Yes	Yes
3	Level members	Yes	Yes	Yes
4	Rel. between Level Members and Levels	No	Yes	Yes
5	Roll-up relations between Levels	No	Yes	Yes
6	Roll-up relations between Level Members	Yes	Yes	Yes
7	Multiple hierarchies in a dimension	Yes	No	Yes
8	Measures	Yes	Yes	Yes
9	Multiple measures in a cube	Yes	Yes	Yes
10	Aggregation functions	No	No	Yes

Table 1: Multidimensional modeling support.

property. Rollup relations between level members are indicated using the property `oc:parentLevelMember`.

The Open Cube vocabulary overcomes the problems of QB to support *OLAP operations*. In [5] we showed that, based on OC, OLAP operators could be implemented through SPARQL queries *over the RDF representation of data cubes.* Since in Open Cube, data cubes are represented as RDF graphs, and the result of any of the OLAP operators must be an RDF graph, OLAP operators have to be implemented as SPARQL `CONSTRUCT` queries.

Comparing approaches. Table 1 summarizes the modeling features supported by each proposal. Note that OC allows representing all the MD concepts in Table 1, however at the expense of the need of adopting a new vocabulary, which prevents reusing data cubes already published using QB. Conversely, [8] allows reusing QB cubes, at the expense of modeling limitations.

4 QB4OLAP: a Proposal for a Unified Data Cube Vocabulary

4.1 Vocabulary Specification

In Section 2 we have shown that QB presents some shortcomings to support BI analysis on the semantic web. On the other hand, OC does not support existing cubes based on QB, that means, data already published using QB (and applications built over this model) cannot be reused. This represents a problem since, as we already mentioned, although QB is not yet a W3C candidate standard, many datasets have already been published using QB. Therefore, to allow BI analysis and, at the same time, supporting existing applications over data published using QB, we propose the QB4OLAP vocabulary[7]. QB4OLAP allows either publishing and analyzing MD from scratch (similarly to what can be done using OC), or performing OLAP analysis on observations already published using QB. To achieve this, QB4OLAP adds to QB the capability of representing dimension levels, level members, rollup relations between levels and level members, and associating aggregate functions to measures.

Figure 4 presents the QB4OLAP vocabulary, where classes and properties that are added to QB (with prefix `qb4o`) are depicted in light gray background. The class

[7] `http://publishing-multidimensional-data.googlecode.com/git/index.html`

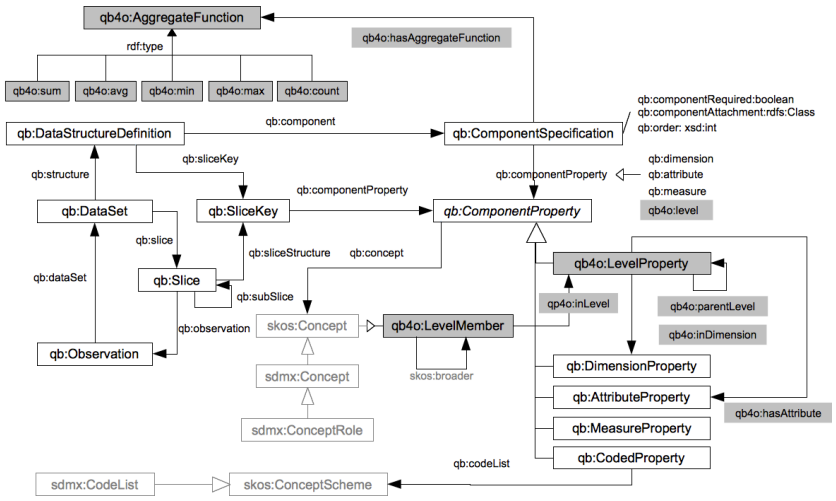


Fig. 4: The QB4OLAP Vocabulary

`qb4o:LevelProperty` models dimension levels; the `qb4o:parentLevel` property represents relations between dimension levels. Level members are represented as instances of the class `qb4o:LevelMember`, and rollup relations between them can be expressed using the property `skos:broader`. Level attributes are defined via the `qb4o:hasAttribute` property. In the DSD, level properties are stated. These levels, like in OC, correspond to the lowest levels in the hierarchy of each dimension in the cube. Also, like in OC, although dimensions are not directly linked to the data structure, they can be reached navigating from any level using the `qb4o:inDimension` property. The class `qb4o:AggregateFunction` represents aggregate functions, and the association between measures and aggregate functions in a data cube is represented using the property `qb4o:hasAggregateFunction`. This property, together with the idea of component sets, allows a given measure to be related to different aggregate functions in different cubes, which is an *improvement with respect to OC where each measure could only be related to one aggregate function.*

We now show how QB4OLAP can be used to publish MD data from scratch. Later, in Section 5, we present how this vocabulary can also be used to analyze data already published using QB. Figure 5 (a) shows the definition of the geoDim dimension schema and instances using the QB4OLAP vocabulary. Figure 5 (b) shows how the data cube of Figure 2a can be represented using QB4OLAP. Although resembling the definition of the cube in QB presented in Figure 3 (b), there are some relevant differences. The data cube schema is declared in lines 8 through 12, using an instance of `qb:DataStructureDefinition`. Level properties (`qb4o:level`) are used to specify the cube schema, instead of dimension properties (`qb:dimension`). Also, the aggregate function corresponding to measure `eg:household` is stated using the `qb4o:hasAggregateFunction` property (line 12). Data cube instances are modeled using the `qb:Observation` class, and grouped using the `qb:DataSet` class, which is linked to the corresponding fact schema using the property `qb:structure`.

```
1  eg:geoDim a qb:DimensionProperty.
2
3  eg:unitaryAuthority a qb4o:LevelProperty;
4    qb4o:inDimension eg:geoDim;
5    qb4o:parentLevel eg:governmentOfficeRegion.
6  eg:governmentOfficeRegion a qb4o:LevelProperty;
7    qb4o:inDimension eg:geoDim;
8    qb4o:parentLevel eg:country.
9  eg:country a qb4o:LevelProperty;
10   qb4o:inDimension eg:geoDim;
11   skos:closeMatch adgeo:Country.
12
13 ns0:00mc qb4o:inLevel eg:unitaryAuthority;
14   rdfs:label "The Borough of Reading@en";
15   skos:broader ns1:J.
16 ns1:J qb4o:inLevel eg:governmentOfficeRegion;
17   rdfs:label "South East@en" ;
18   skos:broader ns2:921.
19 ns2:921 qb4o:inLevel eg:country;
20   rdfs:label "England@en".
```

(a) The geoDim dimension structure

```
1  eg:dateDim a qb:DimensionProperty.
2
3  eg:year a qb4o:LevelProperty;
4    skos:closeMatch db:Year;
5    qb4o:inDimension eg:dateDim.
6
7  eg:household a qb:MeasureProperty.
8  eg:householdCS a qb:DataStructureDefinition;
9        qb:component [qb4o:level eg:unitaryAuthority];
10       qb:component [qb4o:level eg:year];
11       qb:component [qb:measure eg:household;
12                qb4o:hasAggregateFunction qb4o:sum].
13 eg:dataset−hh a qb:DataSet;
14       rdfs:label "Household in UK"@en;
15       qb:structure eg:householdCS.
16 eg:o1 a qb:Observation;
17       qb:dataSet eg:dataset−hh ;
18       eg:unitaryAuthority ns0:00mc ;
19       eg:year db:2007;
20       eg:household 58.
```

(b) Data cube structure and instances

Fig. 5: Expressing schemas and instances in QB4OLAP

4.2 OLAP Operators over QB4OLAP

One of the key aspects of OC is that it supports performing OLAP operators presented in Section 2, directly over the RDF representation of cubes. In [5] we discussed why being able to operate directly over this representation is relevant in BI scenarios, and showed how the operators can be implemented as SPARQL queries. The implementation of OLAP operators as SPARQL queries, when QB4OLAP is used, is also possible. Moreover, it is straightforward to transform the algorithms presented in [5] to be used over this new vocabulary. As an example, Figure 6 shows the specification in QB4OLAP of the HouseholdByGOR schema and the SPARQL 1.1 query that implements Roll-Up($householdCS$, geoDim, GOR). The outer CONSTRUCT query (line 1) builds triples based on the results of an inner SELECT query (line 3), which implements the GROUP BY according to members of the levels in the target schema (in this case GOR and year). The dimension hierarchy is traversed via the `skos:broader` property to find corresponding members in level GOR (line 9). New IRIs must be generated to identify each of the new observations resulting from the application of the operator (lines 4 to 6).

5 Compatibility between QB and QB4OLAP

The QB4OLAP vocabulary is compatible with QB, in the sense that QB4OLAP cube schemas can be built on top of data cube instances (observations) already published using QB. Existing applications, or applications that do not require OLAP style-analysis, can still use the QB schema and instances. Therefore, the cost of adding OLAP capabilities to existing datasets *is the cost of building the new schema,* in other words, the cost of building the analysis dimensions. Conversely, cubes built over QB4OLAP from scratch can be transformed into QB cubes in order to be exploited by existing applications

```
eg:householdByGOR a qb:DataStructureDefinition;
        qb:component [qb4o:level eg:governmentOfficeRegion];
        qb:component [qb4o:level eg:year];
        qb:component [qb:measure eg:household;
                        qb4o:hasAggregateFunction qb4o:sum].
eg:dataset−hh1 a qb:DataSet;
        rdfs:label "Household in UK by GOR"@en;
        qb:structure eg:householdByGOR.
```

HouseholdByGOR schema

```
CONSTRUCT { ?id a qb:Observation . ?id qb:dataSet eg:dataset−hh1 . ?id eg:year ?year .
                    ?id eg:governmentOfficeRegion ?gor . ?id eg:household ?sumHhold }
WHERE{ { SELECT ?gor ?year (SUM(?hhold) AS ?sumHhold)
                        (iri(fn:concat("http://example.org/hhold#", "hholdGOR","_",
                          fn:substring−after(?gor,"http://example.org/hhold#"),"_",
                          fn:substring−after(?year,"http://example.org/hhold#"))) AS ?id)
            WHERE { ?o qb:dataSet eg:dataset−hh . ?o eg:year ?year .
                        ?o eg:household ?hhold . ?o eg:unitaryAuthority ?ua .
                        ?ua skos:broader ?gor . ?gor qb4o:inLevel eg:governmentOfficeRegion;
            }GROUP BY ?gor ?year}}
```

HouseholdByGOR instances

Fig. 6: RollUp implementation over QB4OLAP

supporting the latter. As in the case above, *the cube instances remain untouched*. Figure 7 depicts this scenario.

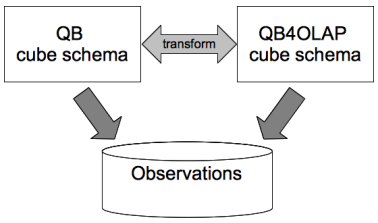


Fig. 7: Analyzing observations using QB and QB4OLAP simultaneously.

5.1 From QB to QB4OLAP

One of the main differences between QB4OLAP and QB is the possibility of the former of specifying a dimension hierarchy. On the contrary, QB allows dimension members to be hierarchically organized into a `skos:ConceptScheme` structure using `skos:narrower` and `skos:broader` properties. This `skos:ConceptScheme` represents a hierarchy of level *members* instead of a hierarchy of levels. Therefore, generating a QB4OLAP cube from a QB cube requires knowing the hierarchy of levels. Thus, we assume that this hierarchy can be obtained from the hierarchy of level members, either automatically or manually (e.g., by a curator). Implicitly, this also assumes that the association between members and dimension levels can also be obtained. If the dimension does not have hierarchical information, we assume a hierarchy of only one

level. Also, generating a QB4OLAP cube from a QB cube requires knowing the association between measures and aggregate functions, which we assume can be obtained automatically or through the intervention of an expert user.

Recall that observations in QB are expressed in terms of *dimensions*, while QB4OLAP requires observations to be expressed in terms of *dimension levels*. Therefore, to avoid rewriting observations, we propose to reuse the URIs that represent dimensions in QB, to represent the bottom level of each dimension in the generated QB4OLAP schema (we show this in lines 8 and 9 in Figure 8 (a)).

Algorithm 1 receives a QB cube schema, and produces a QB4OLAP cube schema. The new schema must be linked to the dataset containing the (existing) observations[8].The algorithm creates and populates the dimension structure, and creates a new DSD.

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

Input: dsd_1 is the data structure definition of a data cube c_1 in QB, D_1 is the set of dimensions in c_1, M is the set of pairs (m_i, ag_i) where m_i is a measure and ag_i is its corresponding aggregate function.
Output: dsd_2 is the data structure definition of a data cube c_2 in QB4OLAP that allows to analyze the observations that populated dsd_1

1: **for all** $d_i \in D_1$ (d_i a `qb:DimensionProperty`) **do**
2: Create a new dimension $d_j \in D_2$ (d_j a `qb:DimensionProperty`)
3: Obtain a hierarchy of levels hl_i and a hierarchy of level members hm_i for d_i
4: **for all** $l_i \in hl_i$ **do**
5: Add triples (l_i a `qb4o:LevelProperty`) and (l_i `qb4o:inDimension` d_j)
6: **for all** $lm_i \in hm_i$ such that lm_i belongs to level l_i **do**
7: Add a triple (lm_i `qb4o:inLevel` l_i).
8: **end for**
9: **end for**
10: **for all** $(l_j, l_k) \in hl_i$ such that $l_j \rightarrow l_k$ **do**
11: Add a triple (l_j `qb:parentLevel` l_k)
12: **end for**
13: **if** l_i is the bottom level in hl_i **then**
14: Add a triple (dsd_2 `qb:component` [`qb4o:level`l_i])
15: **end if**
16: **end for**
17: **for all** m_i such that (dsd_1 `qb:component` [`qb:measure` m_i]) **do**
18: Add a triple (dsd_2 `qb:component` [`qb:measure` m_i;`qb:hasAggregateFunction` ag_i])
19: **end for**

Example 3. Figure 8 (a) presents the result of applying Algorithm 1 to the data cube of Figure 3. A new prefix `eg1` is used to define new dimensions and the new cube schema. In lines 1 to 10 we show the triples that represent geoDim dimension, lines 12 to 15 show the representation of the dateDim dimension, while lines 19 to 23 present the DSD of the new cube. This DSD has to be added as another structure of the dataset that contains the observations to be analyzed, which is done in line 25. □

5.2 From QB4OLAP to QB

Algorithm 2 receives a QB4OLAP cube schema and creates a QB cube schema. To avoid rewriting observations we propose a similar strategy than in Algorithm 1. In this case we reuse the URIs that represent the lowest levels in each dimension in QB4OLAP,

[8] We are not interested here in discussing the most efficient way of generating cubes. We are just proving that this generation is feasible.

```
1  eg1:newGeoDim a ab:DimensionProperty.
2  eg1:country a qb4o:LevelProperty;
3          qb4o:inDimension eg1:newGeoDim.
4  ns2:921 qb4o:inLevel eg1:country.
5  eg1:governmentOfficeRegion a qb4o:LevelProperty;
6          qb4o:inDimension eg1:newGeoDim.
7  ns1:J qb4o:inLevel eg1:governmentOfficeRegion.
8  eg:geoDim a qb4o:LevelProperty;
9          qb4o:inDimension eg1:newGeoDim.
10 ns0:00mc qb4o:inLevel eg:geoDim.
11
12 eg1:newDateDim a ab:DimensionProperty.
13 eg:dateDim a qb4o:LevelProperty;
14         qb4o:inDimension eg1:newDateDim.
15 db:2007 qb4o:inLevel eg:dateDim.
16
17 eg1:household a qb:MeasureProperty.
18
19 eg1:householdCS a qb:DataStructureDefinition;
20       qb:component [qb4o:level eg:geoDim];
21       qb:component [qb4o:level eg:dateDim];
22       qb:component [qb:measure eg1:household;
23           qb:hasAggregateFunction qb4o:sum] .
24
25 eg:dataset−hh qb:structure eg1:householdCS.
```

(a) Creating a cube in QB4OLAP from a cube in QB

```
1  eg:unitaryAuthority a qb:DimensionProperty;
2           qb:codeList eg2:cl1.
3  eg2:cl1 a skos:ConceptScheme;
4          skos:hasTopConcept ns0:00mc.
5
6  ns0:00mc skos:inScheme eg2:cl1.
7  ns1:J skos:inScheme eg2:cl1.
8  ns2:921 skos:inScheme eg2:cl1.
9
10
11
12 eg:year a qb:DimensionProperty;
13          qb:codeList eg2:cl2.
14 eg2:cl2 a skos:ConceptScheme.
15 db:2007 skos:inScheme eg2:cl2.
16
17 eg2:household a qb:MeasureProperty.
18
19 eg2:householdCS a qb:DataStructureDefinition;
20       qb:component [qb:dimension eg:unitaryAuthority];
21       qb:component [qb:dimensionl eg:year];
22       qb:component [qb:measure eg2:household].
23
24
25 eg:dataset−hh qb:structure eg2:householdCS.
```

(b) Creating a cube in QB from a cube in QB4OLAP

Fig. 8: Creating cube schemas

to represent dimensions in the QB schema. Algorithm 2 creates and populates the dimension structure, and produces the DSD. For each dimension d_i a new dimension is created, identified by the URI corresponding to the lowest level in its hierarchy (l_j). The `skos:ConceptScheme` cs_i is created to represent the hierarchy of dimension members, and it is associated with the dimension d_i through a `qb:CodeList` property (line 5). Then, for each level l_i in the dimension, all its level members lm_i are added to cs_i (line 8). We also need to state which level members lm_i are top concepts in the concept scheme, to allow traversing the hierarchy via `skos:broader` relationships (lines 9 to 11). Measures are added to the new DSD in lines 15 to 17.

6 Conclusion

We have shown that the RDF Data Cube Vocabulary (QB) does not suffice for modeling and querying OLAP data cubes. On the other hand, the OC vocabulary, a former proposal by the authors of the present paper, overrides those problems, at the expense of not being compatible with already existing applications based on QB. Thus, we proposed a new vocabulary, which we denoted QB4OLAP, that fills the gap between QB and OC. We also provide algorithms to transform cubes based on QB into equivalent cubes supporting QB4OLAP cubes (and vice versa), showing that compatibility is obtained only at the cost of building the analysis dimensions.

We are currently implementing the OLAP operators using QB4OLAP as http RESTful services, and working in a query language based (at the conceptual level) on the Cube Algebra [2], within a general framework of enabling OLAP analysis over the semantic web.

Algorithm 2 Creating a cube in QB from a cube in QB4OLAP

Input: dsd_1 is the data structure definition of a data cube c_1 in QB4OLAP, D_1 is the set of dimensions in c_1.
Output: dsd_2 is the data structure definition of a data cube c_2 in QB that allows to analyze the observations that populated dsd_1

1: **for all** $d_i \in D_1$ such that (d_i a `qb:DimensionProperty`) **do**
2: Let l_j be a level that satisfies (dsd_1 `qb:component[qb4o:level` l_j]) .(l_j `qb4o:inDimension` d_i)
3: Add a triple (l_j a `qb:DimensionProperty`)
4: Add a triple (dsd_2 `qb:component [qb:dimension` l_j])
5: Add triples (l_j `qb:CodeList` cs_i). (cs_i a `skos:ConceptScheme`)
6: **for all** l_i such that (l_i `qb4o:inDimension` d_i) **do**
7: **for all** lm_i such that (lm_i `qb4o:inLevel` l_i) **do**
8: Add a triple (lm_i skos:inScheme cs_i)
9: **if** lm_i is the top level in the hierarchy **then**
10: Add a triple (cs_i skos:hasTopConcept lm_i)
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: **for all** m_i such that (dsd_1 `qb:component [qb:measure` m_i]) **do**
16: Add a triple (dsd_2 `qb:component [qb:measure` m_i])
17: **end for**

References

1. A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J.-N. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, and G. Vossen. Fusion cubes: Towards self-service business intelligence. *IJDWM Vol 9, Number 2 (to appear)*, 2013.
2. C. Ciferri, R. Ciferri, L. I. Gómez, M. Schneider, A. A. Vaisman, and E. Zimányi. Cube Algebra: A Generic User-Centric Model and Query Language for OLAP Cubes. *IJDWM Vol 9, Number 2 (to appear)*, 2013.
3. J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. Mad skills: New analysis practices for big data. *PVLDB*, 2(2):1481–1492, 2009.
4. R. Cyganiak and D. Reynolds. The RDF Data Cube Vocabulary (W3C Working Draft), April 2012. http://www.w3.org/TR/vocab-data-cube/.
5. L. Etcheverry and A. A. Vaisman. Enhancing OLAP Analysis with Web Cubes. In *ESWC 2012*, pages 53–62, 2012.
6. S. Gómez, L. Gómez, and A. A. Vaisman. A Generic Data Model and Query Language for Spatiotemporal Olap Cube Analysis. In *EDBT 2012*, 2012.
7. C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Maintaining Data Cubes under Dimension Updates. In *ICDE '99*, pages 346-355, 1999.
8. B. Kämpgen, S. O'Riain, and A. Harth. Interacting with Statistical Linked Data via OLAP Operations. In *ESWC workshops*, 2012.
9. R. Kimball. *The Data Warehouse Toolkit*. J. Wiley and Sons, 1996.
10. SDMX. SDMX Standards: Information Model, 2011.
11. A. Vaisman and E. Zimányi. Data Warehouses: Next Challenges. In *Business Intelligence*, Vol 96 of *Lecture Notes in Business Information Processing*, pages 1–26. Springer, 2012.
12. P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In *SSDBM*, pages 53–62, 1998.