Modeling and Querying Data Cubes on the Semantic Web

Lorena Etcheverry
Instituto de Computación,
Universidad de la República
Montevideo, Uruguay
lorenae@fing.edu.uy

Silvia Gomez
Instituto Tecnológico de
Buenos Aires
Buenos Aires, Argentina
sgomez@itba.edu.ar

Alejandro Vaisman
Instituto Tecnológico de
Buenos Aires
Buenos Aires, Argentina
avaisman@itba.edu.ar

ABSTRACT
The web is changing the way in which data warehouses are designed, used, and queried. With the advent of initiatives such as Open Data and Open Government, organizations want to share their multidimensional data cubes and make them available to be queried online. The RDF data cube vocabulary (QB), the W3C standard to publish statistical data in RDF, presents several limitations to fully support the multidimensional model. The QB4OLAP vocabulary extends QB to overcome these limitations, allowing to implement the typical OLAP operations, such as rollup, slice, dice, and drill-across using standard SPARQL queries. In this paper we introduce a formal data model where the main object is the data cube, and define OLAP operations using this model, independent of the underlying representation of the cube. We show then that a cube expressed using our model can be represented using the QB4OLAP vocabulary, and finally we provide a SPARQL implementation of OLAP operations over data cubes in QB4OLAP.

1. INTRODUCTION
On-Line Analytical Processing (OLAP) is a well-established approach for data analysis to support decision making, that typically relates to Data Warehouse (DW) systems. It is based on the multidimensional (MD) model which views data in an n-dimensional space, usually called a data cube. There is a large number of MD models in the literature [9, 12, 20], based on the data cube metaphor. Historically, DW and OLAP have been used as techniques for data analysis within an organization, using mostly commercial tools with proprietary formats. However, initiatives like Open Data1 and Open Government2 are pushing organizations to publish MD data using standards and non-proprietary formats. Although several open source platforms for business intelligence (BI) have emerged in the last decade, an open format to publish and share cubes among organizations is still missing. The Linked Data [11] initiative promotes sharing and reusing data on the web using semantic web (SW) standards and domain ontologies expressed in the Resource Description Framework (RDF) (the basic data representation layer for the SW) [15], or in languages built on top of RDF (e.g., RDF-Schema [4]).

The need for tools and techniques allowing to publish and sharing data cubes did not take long to arise. Statistical data sets are usually published using the RDF Data Cube Vocabulary [6] (also denoted QB), the current W3C standard. However, as we discussed in [7, 8], the QB vocabulary does not support dimension hierarchies and aggregate functions needed for OLAP analysis. To address this challenge, we proposed a new vocabulary called QB4OLAP [8], that allows reusing data already published in QB just by adding the needed MD schema semantics (e.g., the hierarchical structure of the dimensions) and the corresponding instances that populate the dimension levels. Once a data cube is published using QB4OLAP, users are able to operate over it, not only through queries written in SPARQL [18] (the standard query language for RDF), but also using a high-level declarative OLAP language built taking advantage of the QB4OLAP metadata.

In this paper we extend our previous work, presenting a formal data model for data cubes, and we use it to provide the semantics of a set of high-level OLAP operators over data cubes. We then show that a data cube represented using this model can be represented on the semantic web using the QB4OLAP vocabulary. Finally, we provide a SPARQL implementation of the OLAP operators over QB4OLAP.

To put the reader in context, we next present the basic concepts on OLAP and SW data models.

1.1 RDF and the Semantic Web
Resource Description Framework (RDF) is a data model for expressing assertions over resources identified by an internationalized resource identifier (IRI). Assertions are expressed as triples of the form (subject, predicate, object).

A set of RDF triples or RDF data set, can be seen as a directed graph where the subject and object are nodes, and the predicates are arcs. Data values in RDF are called literals. Blank nodes are used to represent anonymous resources or resources without an IRI, typically with a structural function, e.g., to group a set of statements. Subjects are always resources or blank nodes, predicates are always resources,
and object could be resources, blank nodes or literals. A set of reserved words defined in RDF Schema (called the rdfs-vocabulary) [4] is used to define classes, properties, and to represent hierarchical relationships between them. For example, the triple (s, rdf:type, c) explicitly states that s is an instance of c instance. Many formats for RDF serialization exist. In this paper we use Turtle [3].

SPARQL 1.1 [18] is the current W3C standard query language for RDF. Its query evaluation mechanism is based on subgraph matching: RDF triples are interpreted as nodes and edges of directed graphs, and the query graph is matched to the data graph, instantiating the variables in the query. The selection criteria is expressed using a graph pattern in the WHERE clause. Relevant to OLAP queries, SPARQL supports aggregate functions and the GROUP BY clause.

1.2 OLAP

Data Warehouses (DW) integrate data from multiple sources, also keeping their history for analysis and decision support. DWs represent data according to dimensions and facts. The former reflect the perspectives from which data are viewed, and we may have several of them. The latter corresponds to (usually) quantitative data (also known as measures) associated with different dimensions. Dimensions organize elements, or members, in hierarchies, where each element belongs to a category (or level) in a hierarchy. These are the main components of the multidimensional model, which represents facts in an n-dimensional space, usually called a data cube, whose axes are the dimensions, and whose cells contain the values for the measures.

Online Analytical Processing (OLAP) is the process of querying a data cube, where facts can be aggregated and disaggregated via operations called roll-up and drill-down, respectively, and filtered through slice and dice, among other operations.

As an illustration, the facts related to the sales of a company may be associated with the dimensions Time and Location, representing the sales at certain locations in certain periods of time. Assuming that facts (sales) are recorded at granularities month and city in dimensions Time and Location, respectively, a point in this space could be (January 2014, Buenos Aires), and the measure in this cell indicates the amount of the sales in January 2014, at the Buenos Aires branch. A roll-up operation over dimension Time up to level year would produce the yearly amount of sales for each city.

1.3 Problem Statement and Contributions

Ciferri et al. [5] have shown that, opposite to the usual belief, most of the multidimensional data models in the literature are at the logical level rather than at a conceptual level, and that the data cube is far from being the focus of these models. Therefore, the authors sketched (quite informally) a model and algebra where the data cube is a first-class citizen. Along the same lines, Gómez et al. [10] showed that such a model can be used to seamlessly query many kinds of multidimensional data (e.g., discrete and continuous geographic data). We follow these lines of thought, and, as our first contribution, formalize a data model where the main object is the data cube over which a conceptual query language where the operators manipulate the data cube, can be defined. This way, the user just queries data cubes, independently of the underlying data representation. Moreover, in our data model, the semantics of these operators is clearly defined using the notion of a lattice of cuboids, which is later used for query processing and rewriting. As a second contribution we show that a data cube represented using our data model can be published on the semantic web using the QB4OLAP vocabulary. Finally, as a third contribution, taking advantage of the structural metadata provided by the QB4OLAP representation, we present algorithms that produce a SPARQL implementation of the main OLAP operators over QB4OLAP data cubes. That is, an OLAP user would not need to have any knowledge of SPARQL at all, and still be able to query cubes over the semantic web.

The remainder of this paper is organized as follows. Section 2 introduces our running example. Section 3 presents a formal data model for data cubes. Then, Section 4 describes the representation of data cubes in QB4OLAP. Section 5 introduces a high-level OLAP query language purely based on operations over a data cube, and presents a set of algorithms to produce a SPARQL implementation of these operations over QB4OLAP data cubes. Finally, Section 6 discusses related work, and Section 7 concludes the paper.

2. RUNNING EXAMPLE

Throughout this paper we will be using an example based on statistical data about asylum applications to countries in the European Union, provided by Eurostat³. This dataset contains information about the number of asylum applicants by month, age, sex, citizenship, and country that receives the application. It is published using QB in the Eurostat - Linked Data dataspace⁴. Basically, a QB dataset is composed of a set of observations that represent data instances that adhere to a schema, represented by a data structure definition (DSD). However, QB lacks of the capability to represent dimension aggregation hierarchies. To allow OLAP operations, QB4OLAP allows building cube schemas on top of the observations already published using QB. In this way, the cost of adding OLAP capabilities to existing datasets is the cost of building the new dimension schema (the analysis dimensions), and populating its instances. In this example we built simple dimension hierarchies to organize countries into continents, and months into years.

Figure 1 shows the conceptual schema of the data cube (extended with hierarchies), using the MultiDim notation [19]. The asylum_applications cube has a measure (#applications) that represents the number of applications. This measure can be analyzed according to six dimensions: the 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 represents if the applicant is a first-time applicant or a returning applicant, and a geographical dimension that organizes countries into continents (Geography hierarchy) or according to its government type (Government hierarchy). This geographical dimension participates in the cube with

³http://epp.eurostat.ec.europa.eu/cache/ITY_SDDS/EN/migr_asyapp_essm.htm
⁴http://eurostat.linked-statistics.org/
two different roles: as the citizenship of the asylum applicant, and as the destination country of the application.

3. DATA CUBES

In multidimensional models, data are organized as cubes whose axes are dimensions. Each point in this multidimensional space is mapped into one or more spaces of measures. Dimensions are organized in hierarchies that allow analysis at different aggregation levels. The values in a dimension level are called members, and may have properties or attributes. Members in a level must have a corresponding member in the upper level in the hierarchy, and this correspondence is defined through so-called rollup functions. In this section we present a formal model for data cubes upon which we build our query language.

3.1 Data Cubes Formalization

Definition 3.1. (Dimension schema). A dimension schema is a tuple \((nD, L, \rightarrow, H)\) where: (a) \(nD\) is the name of the dimension; (b) \(L\) is a set of tuples \((\text{name}_i, A_i)\), called levels, where \(\text{name}_i\) identifies a level in \(L\), and \(A_i = (a_1, \ldots, a_n)\) is a tuple of level attributes. Each attribute \(a_i\) has a domain \(\text{Dom}(a_i)\) with \(1 \leq i \leq n\); (c) \((L, \rightarrow)\) represents a lattice with a unique bottom level and a unique top level (All), where \(\rightarrow\) is a partial order that defines a parent-child relation between pairs of levels in \(L\); (d) \(H\) is a set of tuples \((\text{name}_i, L_i)\), called hierarchies, where \(\text{name}_i\) identifies the hierarchy, and \(L_i \subseteq L\) is the set of levels that participate in the hierarchy.

Remark 1. All levels in a dimension schema must belong to at least one hierarchy. For each dimension \(nD\) with schema \((nD, L, \rightarrow, H)\), where \(H = \{(h_1, L_{h_1}), \ldots, (h_i, L_{h_i})\}\), then \(\bigcup_{j=0}^{m} L_{h_j} = L\).

In addition, all the dimensions contain at least one default hierarchy \((\text{name}_0, L_0)\), with \(L_0 = L\).

Example 3.1. (Citizenship dimension schema) The schema of the Citizenship dimension in Figure 1 is defined as:
\[
\text{Citizenship} = \{(\text{Country}, \{\text{countryCode, countryName}\}),
\text{Continent}, \{\text{continentCode, continentName}\},
\text{GovernmentType}, \{\text{governmentType}\}\).
\]

\(\rightarrow\) = \{\text{Country} \rightarrow \text{Continent}, \text{Country} \rightarrow \text{GovernmentType},
\text{Continent} \rightarrow \text{All}, \text{GovernmentType} \rightarrow \text{All}\);
\(H = \{\text{Geography}, \{\text{Country, Continent, All}\},
\text{Government}, \{\text{Country, GovernmentType, All}\}\}\)

Definition 3.2. (Dimension instance). A dimension instance for a dimension schema \((nD, L, \rightarrow, H)\) is a tuple \((\text{nD}, L, \rightarrow, H)\), \(\mathcal{R}\) where: (a) \(\mathcal{R}\) is a finite set of tuples of the form \((v_1, v_2, \ldots, v_n)\) \(\forall L = (L_1, L_2, \ldots, L_k) \in L\) such that \(v_i = 1, \ldots, n, v_i \in \text{Dom}(a_i)\); (b) \(\mathcal{R}\) is a finite set of relations \(\text{RUP}^{L_1}_{L_0}, \text{L}_1, \text{L}_0 \subseteq L\), such that \(\text{L}_1 \rightarrow \text{L}_0 \subseteq L\).
Each relation \(\text{RUP}^{L_1}_{L_0}\) relates members of \(L_0\) (child level) with members of \(L_1\) (parent level).

Example 3.2. (Citizenship dimension instance) A possible instance of the Citizenship dimension in Figure 1 is:
\[
\mathcal{R} = \{
\text{RUP}_{\text{Country}}^{\text{Continent}}, \text{RUP}^{\text{All}}_{\text{Country}}, \text{RUP}^{\text{GovernmentType}}_{\text{Country}},
\text{RUP}^{\text{Country}}_{\text{GovernmentType}}, \text{RUP}^{\text{All}}_{\text{GovernmentType}}\},
\text{with RUP}_{\text{Country}}^{\text{Continent}} = \{\{\text{AD, Andorra},\{\text{EU, Europe}\}\},
\{\text{ZW, Zimbabwe},\{\text{AF, Africa}\}\}\},
\text{RUP}^{\text{GovernmentType}}_{\text{Country}} = \{\{\text{AD, Andorra},\{\text{Unitary state}\}\},
\text{RUP}^{\text{All}}_{\text{Country}} = \{\{\text{x}, \{\text{All}\}\} | x \in \mathcal{T}_{\text{All}}\},
\text{RUP}^{\text{All}}_{\text{GovernmentType}} = \{\{\text{x}, \{\text{All}\}\} | x \in \mathcal{T}_{\text{All}}\}\}.
\]

Definition 3.3. (Cube schema). A cube schema is a tuple \((nC, D, M, F)\) where: (a) \(nC\) is the name of the cube; (b) \(D\) is a finite set of dimension schemas (see Def. 3.1); (c) \(M\) is a finite set of attributes, where each \(m \in M\), called measure, has domain \(\text{Dom}(m)\); (d) \(F : M \rightarrow A\) is a function that maps each measure in \(M\) to an aggregate function in \(A\).

Example 3.3. (Asylum_application cube schema) We define the cube schema, presented in Figure 1 as:
\[
\text{Asylum_application} = \{\text{Sex,Age,Time,Application_type, Citizenship, Destination}, \{\#applications\},
\{\#applications,\sum\}\},
\text{where dimension Citizenship is defined as in Example 3.1. We omit the definition of the other dimensions, for the sake of brevity and to avoid redundancy.}
\]

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

Definition 3.4. (Cuboid instance). Given: (a) a cube schema \((nC, D, M, F)\), where \(|D| = D\) and \(|M| = M\), (b) a dimension instance \(I_i\) for each \(D_i \in D, i = 1, \ldots, D\), and (c) a set of levels \(\mathcal{V}_C = \{l_1, l_2, \ldots, l_D\}\) where \(l_i \in L_i\) of \(D_i \in D, i = 1, \ldots, D\), such that there are not two levels belonging to the same dimension, a cuboid instance \(C\) is a partial function \(C : \prod_{i=1}^{D} \mathcal{V}_C \rightarrow \text{Dom}(m)\times \cdots \times \text{Dom}(m)\), where \(m_k \in M, \forall k, 1, \ldots, M\). The elements in the domain of \(C\) are called cells, and \(\mathcal{V}_C\) it the set of levels of the cuboid.

Example 3.4. (Cuboid instance) Consider the cube schema Asylum_application defined in Example 3.3. A possible instance of the cuboid \(C_1\), where \(\mathcal{V}_C = \{\text{Sex, Age, Month, Country}\}\).
Application_type, Country, Country) is presented in Figure 2, using a tabular representation, where the first row lists the dimensions in the cuboid, and the second row lists the dimension level corresponding to the cuboid.

The sets of the cuboid instances that refer to the same cube schema, can be organized using the concepts of adjacent cuboids and order between cuboids, defined as follows.

Definition 3.5. (Adjacent Cuboids). Two cuboids \( Cb_1 \) and \( Cb_2 \), that refer to the same cube schema, are adjacent if their corresponding level sets \( V_{Cb_1} \) and \( V_{Cb_2} \) differ in exactly one level, i.e., \( |V_{Cb_1} - V_{Cb_2}| = |V_{Cb_2} - V_{Cb_1}| = 1 \).

Example 3.5. (Adjacent cuboids) Consider the cube schema defined in Example 3.3 and the cuboids \( Cb_1, Cb_2, \) and \( Cb_3 \) given by \( V_{Cb_1} = \{\text{Sex, Age, Month, Application-type, Country, Country}\} \), \( V_{Cb_2} = \{\text{Sex, Age, Year, Application-type, Country, Continent}\} \) and \( V_{Cb_3} = \{\text{Sex, Age, Year, Application-type, Country, Continent}\} \). According to Definition 3.5, \( Cb_1 \) is adjacent to \( Cb_2 \), and \( Cb_2 \) is adjacent to \( Cb_3 \), but \( Cb_1 \) is not adjacent to \( Cb_3 \).

Definition 3.6. (Order between Adjacent Cuboids). Given two adjacent cuboids \( Cb_1 \) and \( Cb_2 \), such that \( V_{Cb_1} - V_{Cb_2} = \{l_i\} \) and \( V_{Cb_2} - V_{Cb_1} = \{l_p\} \) and \( l_i \) and \( l_p \) are levels of the lattice \( \langle L, \rightarrow \rangle \) of the dimension \( D_k \) such that \( l_i \rightarrow l_p \), then \( Cb_1 \preceq Cb_2 \).

Moreover, for each pair of adjacent cuboids \( Cb_1 \preceq Cb_2 \) each cell \( c = \langle c_1, \ldots, c_k, \ldots, c_n, m_1, m_2, \ldots, m_s \rangle \in Cb_2 \) can be obtained from the cells in \( Cb_1 \) as follows. Let \( \langle c_1, \ldots, c_k-1, b_{1k}, c_{k+1}, \ldots, c_n, m_1, m_2, \ldots, m_s \rangle \), \( \langle c_1, \ldots, c_k-1, b_{2k}, c_{k+1}, \ldots, c_n, m_1, m_2, \ldots, m_s \rangle \), \( \langle c_1, \ldots, c_k-1, b_{3k}, c_{k+1}, \ldots, c_n, m_1, p, m_2, \ldots, m_s \rangle \) be cells in \( Cb_1 \) where \( b_{ik}, c_{ik} \in RUP_{lk_i}^{n_k}, i = 1 \ldots q \) that means that all members \( b_{ik} \) in level \( lk_i \) in dimension \( D_k \) are in a parent-child relation with the element \( c_{ik} \) in level \( lk_p \) in that dimension, and the measures in cell \( e \in Cb_2 \) are obtained as \( m_1 = AGG_{i_{e}}(m_1, \ldots, m_s) \), where \( AGG_{i_{e}} \) is the aggregation function related to measure \( m_i \).

Example 3.6. (Order between cuboids) Consider the cuboids \( Cb_1, Cb_2, \) and \( Cb_3 \) in Example 3.5. Then \( Cb_1 \preceq Cb_2 \), because \( \text{Month} \rightarrow \text{Year} \) holds, and \( Cb_2 \preceq Cb_3, \) because \( \text{Country} \rightarrow \text{Continent} \) holds.

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

Definition 3.7. (Cube Instance). Given a cube schema \( \langle nC, D, M, F, \rangle \), where \( |D| = D \) and \( |M| = M \), and a dimension instance \( I_i \) for each \( D_i \in D, i = 1, \ldots, D \), a cube instance \( CI \) is the lattice \( \{CB, \preceq\} \) where \( CB \) is the set of all possible cuboids, and \( \preceq \) is the order between adjacent cuboids in \( CB \).

Example 3.7. (Cuboids of Asylum_application) Consider the cube schema defined in Example 3.3. All possible combinations of the levels in the six dimensions of the cube, lead to 216 cuboids, which are organized in a lattice. Assuming the instance of cuboid \( Cb_1 \) in Figure 2, Figures 4a and 4b present tabular representations of instances of cuboids \( Cb_2 \) and \( Cb_3 \) given by \( V_{Cb_1} = \{\text{Sex, Age, Year, Application-type, Continent, Country}\} \), and \( V_{Cb_3} = \{\text{All, Age, Year, Application-type, Continent, Country}\} \).

4. QB4OLAP AND DATA CUBES

In this section we first present QB4OLAP distinctive features, and then we describe in detail how the formal model defined in Section 3 can be represented in QB4OLAP.

4.1 QB4OLAP

QB4OLAP\(^5\) extends QB with a set of RDF terms that allow representing the most common features of the MD model. Figure 4 depicts the QB4OLAP vocabulary. Original QB terms are prefixed with "qb:", while QB4OLAP terms are prefixed with "qb4o:" and displayed in gray background. Capitalized terms represent RDF classes, non-capitalized terms represent RDF properties, and capitalized terms in italics represent class instances. An arrow from class \( A \) to class \( B \), labeled \( rel \) means that \( rel \) is an RDF property with domain \( A \) and range \( B \). White triangles represent sub-class or sub-property relationships. Black diamonds represent \( rdf:\text{type} \) relationships (instances). The range of a property can also be denoted using \( ^* \).

The rationale behind QB4OLAP includes:

- QB4OLAP must be able to represent the most common features of the MD model. The features considered are based on the MultiDim model [19].
- QB4OLAP must include all the metadata needed to implement OLAP operations as SPARQL queries. In this way, OLAP users do not need to know SPARQL (which is the case of typical OLAP users), and even wrappers for OLAP tools could be developed to query RDF data sets directly. We comment on this issue at the end of this section.
- QB4OLAP must allow to operate over already published observations which conform to DSDs defined in QB, without the need of rewriting the existing observations, and with the minimum possible effort. Note that in a typical MD model, dimensions are usually orders of magnitude smaller than observations, which are the largest part of the data.

4.2 Implementing data cubes in QB4OLAP

We next sketch how each of the concepts introduced in Section 3.1 can be represented in QB4OLAP. The formal proof is outside the scope of this paper. We assume the reader has basic knowledge of RDF syntax.

Definition 4.1. (Dimension schema in QB4OLAP) A dimension schema in QB4OLAP is an RDF graph that uses terms defined in the QB and QB4OLAP vocabularies as follows:

- Dimensions are defined as instances of the class \( \text{qb4o:DimensionProperty} \).
- Levels are defined as properties, which are instances of the class \( \text{qb4o:LevelProperty} \).
- Level attributes are defined as properties, which are instances of the class \( \text{qb4o:LevelAttribute} \) and related to levels with the property \( \text{qb4o:inLevel} \) or its inverse \( \text{qb4o:hasAttribute} \).

\[^{5}\text{http://purl.org/qb4olap/cubes}\]
Hierarchies are defined as instances of the class qb4o:Hierarchy, and related to dimensions with properties qb4o:inDimension and qb4o:hasHierarchy. Levels are connected with the hierarchies they belong using the property qb4o:hasLevel.

Hierarchies are composed of pairs of levels, which are represented using the class qb4o:HierarchyStep. Pairs are related to hierarchies via the property qb4o:inHierarchy. For each pair we distinguish the role of each level using the properties qb4o:childLevel and qb4o:parentLevel. The property qb4o:rollup is used to relate each pair with the property that implements the RUP relation for that step, which is an instance of qb4o:RollupProperty. The cardinality of this relationship (1 to 1, 1 to N, etc.) is stated using the qb4o:pcCardinality property and instances of the qb4o:Cardinality class.

**Example 4.1. (Citizenship dimension schema in QB4OLAP)**

We show next the representation in QB4OLAP of the schema of the Citizenship dimension in Example 3.1.

| Sex | Age | Time   | Application_type | Citizenship | Destination | Measures |
|-----|-----|--------|------------------|-------------|-------------|----------|
| M   | 14 to 17 | 201301 | new applicant | CM, Cameroon | BE, Belgium | 5        |
| P   | less than 14 | 201303 | new applicant | CM, Cameroon | FR, France  | 5        |
| M   | 18 to 34 | 201301 | new applicant | CM, Cameroon | BE, Belgium | 10       |
| P   | 18 to 34 | 201303 | new applicant | CM, Cameroon | BE, Belgium | 55       |

We show next the representation in QB4OLAP of the schema of the Citizenship dimension in Example 3.1.

| Sex | Age | Time   | Application_type | Citizenship | Destination | Measures |
|-----|-----|--------|------------------|-------------|-------------|----------|
| M   | 14 to 17 | 201301 | new applicant | AF, Africa  | BE, Belgium | 5        |
| P   | less than 14 | 201303 | new applicant | AF, Africa  | FR, France  | 5        |
| M   | 18 to 34 | 201301 | new applicant | AF, Africa  | FR, France  | 10       |
| P   | 18 to 34 | 201303 | new applicant | AF, Africa  | BE, Belgium | 55       |

**Figure 2:** Tabular representation of a cuboid instance of the Asylum_application cube schema

**Figure 3:** Two cuboid instances of the Asylum_application cube schema

Definition 4.2. (Dimension instance in QB4OLAP)

A dimension instance in QB4OLAP is an RDF graph that uses the terms defined in QB and QB4OLAP vocabularies, and also the IRIs defined to represent the dimension schema, as follows: (a) Each level member is represented by an IRI and is related to each level it belongs to, via the qb4o:memberOf property. (b) For each step, the RUP relation between level members is represented using the property linked to the step in the schema via qb4o:rollup property.

**Example 4.2. (Citizenship dimension instance in QB4OLAP)**

Below we show part of the triples in the QB4OLAP representation of the instance of the Citizenship dimension in Example 3.2. Note that some triples are enriched with links to external data, like DBpedia.
Definition 4.3. (Cube schema in QB4OLAP) A cube schema in QB4OLAP is represented as an instance of the `qb:DataStructureDefinition` class. The dimensions and measures in the cube are represented using the properties `qb:measure` and `qb:dimension`. The cardinality between observations and dimensions is represented using the property `qb4o:cardinality`. Aggregate functions are represented via the `qb4o:AggregateFunction`, and a property is defined to associate measures with aggregate functions (`qb4o:aggregateFunction`). This property allows a given measure to be associated with different aggregate functions in different cubes.

Example 4.3. (Cube schema in QB4OLAP) We show below the QB4OLAP representation of the Asylum application cube schema.

```
schema:migrasyappctzmCUBE rdf:type qb:DataStructureDefinition ;
qb:component [ qb:measure sdmx-measure:obsValue; qb4o:aggregateFunction qb4o:sum ] ;
qb:component [ qb:dimension schema:sexDim ; qb4o:cardinality qb4o:ManyToMany ;
qb:component [ qb:dimension schema:ageDim ; qb4o:cardinality qb4o:ManyToMany ;
qb:component [ qb:dimension schema:timeDim ; qb4o:cardinality qb4o:ManyToMany ;
qb:component [ qb:dimension schema:asylappDim ; qb4o:cardinality qb4o:ManyToMany ] ;
qb:component [ qb:dimension schema:citizenshipDim ; qb4o:cardinality qb4o:ManyToMany ] ;
skos:notation "migrasyappctzmCUBE".
```

Definition 4.4. (Cuboid instance in QB4OLAP) A cuboid instance in QB4OLAP is a set of `qb:Observations` organized in a `qb:DataSet`. The set of levels of the cuboid is an instance of the `qb:DataStructureDefinition` class, and the property `qb:structure` is used to relate them. To state the fact that a cuboid instance adheres to a specific cube schema we use the property `qb4o:isCuboidOf`.

Example 4.4. (Cuboid instance in QB4OLAP) Let us consider the cuboid that corresponds to the schema of Example 4.3, which considers the lowest level for each dimension in the cube (i.e., the bottom of the lattice). Below we show the QB4OLAP representation of the set of levels of the cuboid, the definition of a dataset that represents the cuboid instance, and also a cell in this cuboid (`qb:Observation`), which corresponds to the last row in Table 2.

```
@prefix eurostat: <http://eurostat.linked-statistics.org/data/> .
@prefix cell: <http://eurostat.linked-statistics.org/data/migrasyappctzm#> .
schema:migrasyappctzmBOTTOM rdf:type qb:DataStructureDefinition ;
qb4o:isCuboidOf schema:migrasyappctzmCUBE ;
qb:component [ qb:measure sdmx-measure:obsValue; qb4o:aggregateFunction qb4o:sum ] ;
qb:component [ qb4o:level property:age ] ;
qb:component [ qb4o:level sdmx-dimension:refPeriod ] ;
qb:component [ qb4o:level property:sex ] ;
qb:component [ qb4o:level property:geo ] ;
qb:component [ qb4o:level property:citizen ] ;
qb:component [ qb4o:level property:asylapp ] ;
skos:notation "migrasyappctzmBOTTOM".
```

5. QUERYING DATA CUBES

We now present our proposal for querying data cubes on the semantic web. We follow the approach presented in [5], where a clear separation between the conceptual and the logical levels is made. In this way, we formally define a collection of operators, whose semantics is clearly defined using...
the data model of Section 3.1. This set of operators conforms our query language (QL) at the conceptual level. The user then will write her queries at this level, and they will be translated into a SPARQL query over the QBEOLAP-based RDF representation (at the logical level) using the algorithms presented in Section 5.2.

5.1 Data Cube Algebra

The operators proposed in the Cube Algebra sketched in [5] can be classified into two groups: (1) Operations that navigate the cube instance to which the input cuboid belongs, which are called instance preserving operations (IPO); and (2) Operations that generate a new cube instance, which are called instance generating operations (IGO). The Roll-up and Drill-down operations belong to the first group, while Slice, Dice and Drill-across belong to the second. We now define each of the operations and provide their semantics.

5.1.1 Instance Preserving Operations (IPO)

IPOs take as input a cuboid in a cube instance, and return another cuboid in the same instance. We consider the following sets: $C$ is the set of all the cuboids in a cube instance, $D$ is the set of dimensions, $M$ is the set of measures, $V$ is the set of dimension levels, and $B$ is the set of boolean expressions over level attributes and measures. For clarity, and to simplify the definitions, we assume that the aggregate function associated to the measures is SUM, so we drop $F$ from the cube schema definition.

The Roll-up operator is a function Roll-up: $D \times C \rightarrow C$ that summarizes data at a higher level in a dimension hierarchy. It is defined as follows:

Definition 5.1. (Roll-up Operator). Given a cube instance Cl with schema $(C, D_in, M_{in})$, a cuboid $C_{in} \in C$ with its correspondent set of levels $V_{C_{in}}$, a dimension $D \in D_in$ with schema $(D, L, \rightarrow)$, and two levels $l_{in}, l_{out}$ in $D_in$ such that: $l_{in} \in V_{C_{in}}$ and $l_{in} \rightarrow l_{out}$ in $(L, \rightarrow)$, then Roll-up($C_{in}, l_{in}, l_{out}$) returns a cuboid $C_{out}$ in $C$ such that $V_{C_{out}} = \{ V_{C_{in}} - \{ l_{in} \} \} \cup \{ l_{out} \}$. Notice that $C_{in} \subset C_{out}$ in the lattice Cl.

The Drill-down operator is a function Drill-down: $D \times C \rightarrow C$ that disaggregates data down to a specific level in a dimension hierarchy.

Definition 5.2. (Drill-down Operator). Given a cube instance Cl with schema $(C, D_{in}, M_{in})$, a cuboid $C_{in} \in C$ with its correspondent set of levels $V_{C_{in}}$, a dimension $D \in D_{in}$ with schema $(D, L, \rightarrow)$, and two levels $l_{in}, l_{out}$ in $D_{in}$ such that: $l_{in} \in V_{C_{in}}$ and $l_{in} \rightarrow l_{out}$ in $(L, \rightarrow)$, then Drill-down($C_{in}, D, l_{in}, l_{out}$) returns a cuboid $C_{out}$ in $C$ such that $V_{C_{out}} = \{ V_{C_{in}} - \{ l_{in} \} \} \cup \{ l_{out} \}$. Notice that $C_{in} \subset C_{out}$ in the lattice Cl.

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

5.1.2 Instance Generating Operations (IGO)

IGOs generate a new cube instance, which may have the same schema as the original one (e.g. in the case of Dice), or may have a different schema (e.g. Slice and Drill-across). In all the cases, these operations take as input a cuboid in a cube instance, and return a cuboid in another instance or cuboid lattice, induced by the application of the operation.

The Dice operator is a function Dice: $D \times B \rightarrow C$ that selects the values in dimension levels and measures that satisfy a boolean condition. It resembles the Selection ($\sigma$) operation in relational algebra.

Definition 5.3. (Dice Operator). Given a cube instance Cl with schema $(C, D_{in}, M_{in})$, a cuboid $C_{in} \in C$ with its correspondent set of levels $V_{C_{in}}$, and a boolean condition $\phi$ over the measures in $M_{in}$ and/or the attributes of the levels in $V_{C_{in}}$, Dice($C_{in}, \phi$) returns a cuboid $C_{out} \in C$ as follows:

(a) $C_{out} = C_{in}$ if $\exists l_{in} \in V_{C_{in}}$ and $l_{in} \rightarrow l_{out}$ in $(L, \rightarrow)$, then Dice($C_{in}, \phi$) returns a cuboid $C_{out} \in C$ such that $V_{C_{out}} = \{ V_{C_{in}} - \{ l_{in} \} \} \cup \{ l_{out} \}$. Notice that $C_{in} \subset C_{out}$ in the lattice Cl.

(b) $V_{C_{out}} = V_{C_{in}}$

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

Definition 5.4. (Slice Operator). Given a cube instance Cl with schema $(C, D_{in}, M_{in})$, where $| D_{in} | > 1$ or where $| M_{in} | > 1$ a cuboid $C_{in} \in C$ with its correspondent set of levels $V_{C_{in}}$, and a dimension $D \in D_{in}$ or a measure $M \in M_{in}$, according to the input parameters:

(1) Slice($C_{in}, D$) returns a cuboid $C_{out} \in C$ as follows:

(a) $C_{out} = C_{in}$ if $\exists l_{in} \in V_{C_{in}}$ and $l_{in} \rightarrow l_{out}$ in $(L, \rightarrow)$, then Slice($C_{in}, D$) returns a cuboid $C_{out} \in C$ such that $V_{C_{out}} = \{ V_{C_{in}} - \{ l_{in} \} \} \cup \{ l_{out} \}$. Notice that $C_{in} \subset C_{out}$ in the lattice Cl.

(b) $V_{C_{out}} = V_{C_{in}}$

The Drill-across operator is a function Drill-across: $C \rightarrow C \rightarrow C$ that performs the union of two cuboids that are defined over the same dimensions, and contain the same instance, but differ in the measures. It allows to compare measures from different cuboids and resembles the Join ($\Join$) operation in relational algebra. In this paper we will not address this operator, and limit ourselves to unary operations, that means, operations over single data cubes. Thus, we omit the formal definition of the operation.

5.2 Algebra Operations as SPARQL Queries
| Function signature | Description |
|--------------------|-------------|
| newVarName()       | Generates and returns a unique SPARQL variable name. |
| val(v)             | Returns the value stored in variable v. |
| levels(s)          | Returns all the levels in a schema s (i.e., all the values of ?l that satisfy s qb:component ?c. ?c qb4o:level ?l). |
| getLevel(s,d)      | Returns the only level l that corresponds to dimension d in the schema s (i.e., the only value of ?l that satisfies s qb:component ?c. ?c qb4o:level ?l. ?h qb4o:hasLevel ?l. ?h qb4o:inDimension ?d). |
| levelsPath(l0,l0)  | Returns a levels path from level l0 to l0. |
| getRollup(l0,l0)   | Returns the predicate that implements the RUP function from level l0 to level l0. |
| measures(s)        | Returns all the measures in a schema s (all the values of ?m that satisfy s qb:component ?c. ?c qb:measure ?m). |
| aggFunction(m,s)   | Returns the aggregation function of measure m (all the values of ?l that satisfy s qb:component ?c. ?c qb:measure ?m. ?l qb4o:aggregateFunction ?f). |

Table 1: Auxiliary functions

In this section we show how the operators above can be implemented as SPARQL queries over QB4OLAP-based RDF data cubes. Before that, we would like to discuss on the result form of these queries.

SPARQL queries may return results in different formats. In particular SELECT queries return a table of values, while CONSTRUCT queries return a graph (i.e., a set of triples). Since each operator returns a cuboid in a certain cube instance, and according to Definitions 4.3 and 4.4, cuboids in QB4OLAP are RDF graphs, it is evident that algebra operators should be implemented in SPARQL using CONSTRUCT queries. Despite this, SPARQL 1.1 does not allow to compute aggregations in a CONSTRUCT query, and therefore our approach is to use subqueries. We then produce two queries: (a) an inner SELECT query to compute aggregations, and (b) an outer CONSTRUCT query that generates the graph using the computed results. Notice that the inner query is responsible for the actual computation of values, while the outer query just generates the output as a graph.

We now present the algorithms that generate the SPARQL implementation of each operator. To improve the clarity of the presentation, we use the auxiliary functions defined in Table 1. We also use an abstract representation of a SPARQL query, where for each query: (a) queryType can be SELECT or CONSTRUCT, (b) resultFormat represents the set of variables and expressions included in the SELECT clause or the set of BGP s included in the CONSTRUCT clause, depending on the type of the query, (c) qrPatterns represents the set of graph patterns in the WHERE clause, (d) subQueries represents the set of subqueries in the WHERE clause, (e) filter represents a FILTER clause, (f) and groupBy represents the set of variables included in the GROUP BY clause. We assume that each of these parts can be accessed and modified, and we use the dot notation (\("\). ) to access them. We also consider a function add\(f\), such that add\(s\) appends \(s\) to a particular part of the query. For example, given a query \(q\) such that \(q\).queryType == "SELECT" the instruction \(q\).resultFormat.add\(v\) adds the variable \(v\) to the SELECT clause of \(q\).

5.2.1 IPO Operations as SPARQL Queries

According to Definition 3.7, a cube instance is the lattice \(\{Cuboids, \leq\}\) where \(Cuboids\) is the set of all possible cuboids that adhere to a cube schema, and \(\leq\) is the order between adjacent cuboids in \(Cuboids\). As stated by Definition 3.5 for each pair of adjacent cuboids \(C_{1} \preceq C_{2}\), each cell in \(C_{2}\) can be computed from the cells in \(C_{1}\). Therefore, starting from the bottom cuboid in the lattice, which is the cube instance whose cells are members of the bottom levels in each dimension of the schema, all the possible cuboids that form the cube instance can be computed incrementally.

To compute the ROLL-UP operation over a cuboid \(C_{0}\) and a dimension \(D\) it suffices to start at \(C_{0}\), and navigate the cube lattice visiting adjacent cuboids that differ only in the level associated to dimension \(D\), until we reach a cuboid \(C_{out}\) that has the desired level in dimension \(D\) (this path is unique).

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

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

Algorithm 1 builds both queries simultaneously, using the add function. Lines 2 and 3 state the query type for each query. Line 4 states that generated observations belong to the dataset newDS. Lines 6 through 12 project the members of each level in the schema into the result of both queries, also adding triples to the WHERE clause of the inner query and adding the variables that represent the level members to the GROUP BY clause also in the inner query. Lines 13 through 19 do the same for measures. In Lines 14 and 18, \(f\) represents the SPARQL function corresponding to the aggregate function for each measure, and \(f(val(m))\) is the string that should be included to calculate the aggregated value (e.g. \(\sum?m\) if \(val(m) = ?m\)). Lines 20 to 33 add the triples needed to navigate the dimension hierarchy. Line 22 retrieves the RDF property that implements the RUP relation for each step in the path. Line 24 adds to the newDS the string that should be included to calculate the aggregation for each level member. Line 25 retrieves the RDF property that implements the RUP relation for each step in the path. Line 27 adds a triple that allows us to state to which level the level member belongs, and line 29 retrieves the parent level member of the current level applying the RUP function obtained in Line 22 (this is done for all the levels in the path except for the target level \(l_{out}\). When level \(l_{out}\) is reached Lines 31 and 32 add the target level to the GROUP BY and SELECT clauses of the inner query, respectively, while...
Algorithm 1: Generates a SPARQL query that implements a Roll-up in QB4OLAP

**Precondition:** \( C_n \) is a cuboid instance in QB4OLAP, where \( V_C \) is the set of levels of the cuboid and \( D \) is the dataset that represents the cuboid instance, \( l_{in} \) is a level such that \( l_{in} \in V_C \) and \( l_{out} \in D \) (i.e., \( l_{in} \) and \( l_{out} \) belong to the lattice \( L \)).

**Postcondition:** \( q_{out} \) is a SPARQL CONSTRUCT query that represents a cuboid instance \( C_{out} = \text{ROLL-UP}(C_n, D, l_{out}) \).

1. **function createRollupQuery(C_n, D, l_{out})**
2. \( q_{out}.queryType = \text{CONSTRUCT} \)
3. \( q_{out}.queryType = \text{SELECT} \)
4. \( q_{out}.patterns.add(newObs, qb:dataset, newDS) \)
5. \( l_{in} \leftarrow \text{getLevel}(C_n, D) \)
6. for all \( l \in L = \text{levels}(C_n) \) do
7. if \( l \neq l_{in} \) then
8. \( l_{in} \leftarrow \text{newVar}() \)
9. \( q_{in}.patterns.add(\text{newObs}, l, \text{val}(l)) \)
10. \( q_{in}.groupBy.add(\text{val}(l)) \)
11. \( q_{in}.resultFormat.add(\text{val}(l)) \)
12. \( q_{out}.resultFormat.add(\text{newObs}, l, \text{val}(l)) \)
13. for all \( m \in M = \text{measures}(C_n) \) do
14. if \( m \neq \text{newVar}() \) then
15. \( m_{in} \leftarrow \text{newVar}() \)
16. \( q_{in}.patterns.add(m_{in}, m, \text{val}(m)) \)
17. \( q_{in}.resultFormat.add(f(\text{val}(m)) \text{ AS } m_{in}) \)
18. \( q_{out}.resultFormat.add(\text{newObs}, m, \text{val}(m)) \)
19. for all \( (l_1, l_2) \in \text{path} = \text{levelsPath}(l_{in}, l_{out}) \) do
20. \( l_{in} \leftarrow \text{newVar}() \)
21. \( l_{out} \leftarrow \text{getRollup}(l_1, l_2) \)
22. if \( l \neq l_{out} \) then
23. \( q_{in}.patterns.add(\text{obs}, \text{val}(l), \text{val}(m)) \)
24. \( q_{in}.patterns.add(\text{办出}, \text{val}(l), \text{val}(m)) \)
25. else
26. \( \text{where} \leftarrow \text{newVar}() \)
27. \( \text{q_{in}.patterns.add(\text{val}(\text{w}), l, \text{val}(\text{w}))} \)
28. \( \text{q_{in}.patterns.add(\text{val}(\text{w}), \text{val}(\text{w}))} \)
29. \( q_{in}.patterns.add(\text{办出}, \text{val}(\text{w}), \text{val}(\text{w})) \)
30. else
31. \( q_{in}.groupBy.add(\text{plm}) \)
32. \( q_{in}.resultFormat.add(\text{plm}) \)
33. \( q_{out}.resultFormat.add(\text{newObs}, l_{op}, \text{val}(\text{plm})) \)
34. \( q_{out}.subqueries.add(q_{in}) \)
35. return \( q_{out} \)

Line 33 adds the target level to the outer query result. Finally, Line 34 sets the inner query as a subquery within the WHERE clause of the outer query, which is returned in Line 35. For clarity, we have omitted the clause that generates the expression that binds variable \( \text{newOBS} \) to a dynamically generated URI from the values in the observation.

**Example 5.1.** The SPARQL query generated by Algorithm 1 for \( \text{ROLL-UP}((\text{Asylum-application}, \text{Citizenship, Continent})) \) is:

```sparql
PREFIX property: <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/property/>
PREFIX qb: <http://purl.org/linked-data/cube/
PREFIX sdmx: <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/">
PREFIX sdmx: <http://www.fing.edu.uy/inco/cubes/schemas/migr_appapp/>
PREFIX qb: <http://www.fing.edu.uy/inco/cubes/schemas/migr_appapp/>

CONSTRUCT { ?newOBS a qb:Observation; qb:dataset=qb:dataset; qb:app=qb:appapp; property:asyl-area-property:app=appapp; property:geo = geo; property:age = age; property:sex = sex; property:refPeriod = refPeriod; property:sdmx-2009/measure#geo; property:sdmx-2009/measure#age; property:sdmx-2009/measure#sex; qb:memberOf qb:appapp; sdmx:measure:obsValue ?sumApp } WHERE { ?lm1 a qb:Observation; qb:dataset <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; qb:app <http://www.fing.edu.uy/inco/cubes/schemas/migr_appapp/>; property:asyl-area-property:app <http://www.fing.edu.uy/inco/cubes/schemas/migr_appapp/>; property:geo <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; property:age <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; property:sex <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; property:refPeriod <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; qb:memberOf <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; sdmx:measure:obsValue ?sumApp } }
```

In Section 5.1 we have discussed that it is possible to transform a **DRILL-DOWN** operation into a **ROLL-UP**, therefore there is no need to provide a specific implementation for **DRILL-DOWN** in QB4OLAP.

5.2.2 IGO Operations as SPARQL Queries

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

The **DICE** operation takes as input a cuboid in a cube instance, and a boolean expression \( \phi \) over measure values and/or attributes values, and returns a cuboid in a new cube instance keeping only the cells from the input cuboid that satisfy \( \phi \). The implementation of this operator in SPARQL selects the **qb:Observations** that satisfy \( \phi \). Since measures and attributes are literals, conditions over them can be implemented as **FILTER** clauses. Also, conditions that only involve equality can be efficiently implemented via graph patterns, restricting the result of the query to observations that are related to a particular level member. Inequalities over level members are represented as **FILTER** clauses. Algorithm 2 generates the SPARQL implementation of the **DICE** operator, which is also based in an inner query which performs the filter, and an outer query that produces the results.

**Example 5.2.** (SPARQL query that implements the **DICE** operation) The operation:

\[ \text{DICE}((\text{Asylum-application}, ((201303<=month<=201307) \lor (#applications > 80) \land \text{Destination.country.countryName} = \text{Belgium})) \) is implemented in SPARQL as follows.

```sparql
CONSTRUCT { ?a ?b ?c } WHERE { ?lm1 a qb:Observation; qb:dataset <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; qb:app <http://www.fing.edu.uy/inco/cubes/schemas/migr_appapp/>; property:asyl-area-property:app <http://www.fing.edu.uy/inco/cubes/schemas/migr_appapp/>; property:geo <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; property:age <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; property:sex <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; property:refPeriod <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; qb:memberOf <http://www.fing.edu.uy/inco/cubes/instances/migr_appapp/>; sdmx:measure:obsValue ?sumApp } FILTER (?timeMonthNum > 201305 && ?timeMonthNum <= 201307 && ?val(integer(?m)) < 40) }
```

The **SLICE** operation comes in two flavors. In one case, it takes as input a cuboid instance and a dimension. In the other, it takes a cuboid instance and a measure. In both cases the implementation of this operator in QB4OLAP requires the creation of a new schema, where the input dimension or the measure are removed. In the case where the operation receives a dimension (**SLICE**(\(C_n, D\))), the new cube instance \( C_{out} \) is computed as (\( \text{ROLL-UP}(C_n, D) \)). The SPARQL query generation algorithm is straightforward, and we omit it here.
Algorithm 2: Generates a SPARQL query that implements a Dice in QB4OLAP

Precondition: $C_{in}$ is a cuboid instance in QB4OLAP, where with its correspondent set of levels $V_{C_{in}}$ and a boolean condition $\phi$ over the measures in $M_{in}$ and/or the attributes of the levels in $V_{C_{in}}$. $V_{C_{in}}$ is the set of levels of the cuboid, $\phi$ is a boolean condition over the measures in $M_{in}$ and/or the attributes of the levels in $V_{C_{in}}$, and $d_{i}$ is the dataset that represents the cuboid instance.

Postcondition: $q_{out}$ is a SPARQL CONSTRUCT query that represents a cuboid instance $C_{out} = Dice(C_{in}, \phi)$

```java
1: function CREATE_DICE_QUERY($C_{in}$, $\phi$)  
2: $q_{in}.queryType = 'CONSTRUCT'  
3: $l_{vars} = []  
4: $m_{vars} = []  
5: bgps.add(\{
6: \})  
7: $q_{out}.grPatterns.add(?newObs , qb: dataSet , newDS)  
8: for all $i \in L = levels(C_{in})$ do  
9: $l_{vars}[i] = l_{i}$  
10: $q_{a}.grPatterns.add(\{obs , l_{i} , val(l_{i})\})  
11: $q_{a}.resultFormat.add(\{\})  
12: $q_{a}.resultFormat.add(\{\})  
13: $q_{out}.resultFormat.add(?newObs , l_{i} , val(l_{i}))  
14: for all $m \in M = measures(C_{in})$ do  
15: $m_{i} = newVar()  
16: $m_{a}.add(\{\})  
17: $q_{a}.grPatterns.add(\{obs , m , val(m_{i})\})  
18: $q_{a}.resultFormat.add(\{\})  
19: $q_{out}.resultFormat.add(?newObs , m , val(m_{i}))  
20: $treeCond = parseCondition(\phi)  
21: $treeCond = parseCondition(\phi)  
22: $treeCond = parseCondition(\phi)  
23: $treeCond = parseCondition(\phi)  
24: $treeCond = parseCondition(\phi)  
25: $treeCond = parseCondition(\phi)  
26: $treeCond = parseCondition(\phi)  
27: function PRO conditional($tree$, $l_{vars}$, $m_{vars}$, $bgps$, $filter$)  
28: if $tree$ = leaf then  
29: $v = findVariable($tree$.element, $l_{vars}$)  
30: $la = newVar()  
31: bgps.add($v , $tree$.level , $la$)  
32: $filter = la , tree.oper , tree.value)  
33: else  
34: $v = findVariable($tree$.element, $m_{vars}$)  
35: $filter = v , tree.oper , tree.value)  
36: else  
37: $procCondition($tree.left$ , $l_{vars}$, $m_{vars}$, $bgps_{left}$,$filter_{left}$)  
38: $procCondition($tree.right$ , $l_{vars}$, $m_{vars}$, $bgps_{right}$,$filter_{right}$)  
39: $bgps.add(\{\})  
40: $bgps.add(\{\})  
41: $bgps.add(\{\})  
42: $filter.add(\{filter_{left} , tree.oper , filter_{right}\})
```

6. RELATED WORK

There are two main lines of research addressing OLAP analysis of SW data, namely (1) extracting MD data from the SW and loading them into traditional MD data management systems for OLAP analysis; and (2) performing OLAP-like analysis directly over SW data, e.g., over MD data represented in RDF. We next discuss them in some more detail.

Relevant to the first line are the works by Nebot and Llavori [17] and Kämpgen and Harth [14]. The former proposes a semi-automatic method for on-demand extraction of semantic data into a MD database. In this way, data could be analyzed using traditional OLAP techniques. The authors present a methodology for discovering facts in SW data, and populating a MD model with such facts. They assume that data are represented as an OWL\textsuperscript{6} ontology. The proposed methodology has four main phases: (1) Design of the MD schema, where the user selects the subject of analysis that corresponds to a concept of the ontology, and then selects potential dimensions. Then, she defines the measures, which are functions over data type properties; (2) Identification and extraction of facts from the instance store according to the MD schema previously designed, producing the base fact table of a DW; (3) Construction of the dimension hierarchies based on the instance values of the fact table and the knowledge available in the domain ontologies (i.e., the inferred taxonomic relationships) and also considering desirable OLAP properties for the hierarchies; (4) User specification of MD queries over the DW. Once queries are executed, a cube is built. Then, typical OLAP operations can be applied over this cube.

Kämpgen and Harth [14] study the extraction of statistical data published using the QB vocabulary into a MD database. The authors propose a mapping between the concepts in QB and a MD data model, and implement these mappings via SPARQL queries. There are four main phases in the proposed methodology: (1) Extraction, where the user defines relevant data sets which are retrieved from the web and stored in a local triple store. Then, SPARQL queries are performed over this triple store to retrieve meta-data on the schema, as well as data instances; (2) Creation of a relational representation of the MD data model, using the metadata retrieved in the previous step, and the population of this model with the retrieved data; (3) Creation of a MD model to allow OLAP operations over the underlying relational representation. Such model is expressed using XML for Analysis (XMLA)\textsuperscript{7}, which allows the serialization of MD models and is implemented by several OLAP clients and servers; (4) Specification of queries over the DW, using OLAP client applications.

The proposals described above are based on traditional MD data management systems, thus they capitalize the existent knowledge in this area and can reuse the vast amount of available tools. However, they require the existence of a local DW to store SW data. This restriction clashes with the autonomy and highly volatile nature of web data sources as changes in the sources may lead not only to updates on data instances but also in the structure of the DW, which

\textsuperscript{6}http://www.w3.org/TR/owl2-overview/
\textsuperscript{7}http://xmlforanalysis.com
would become hard to update and maintain. In addition, these approaches solve only one part of the problem, since they do not consider the possibility of directly querying à la OLAP MD data over the SW.

The second line of research tries to overcome the drawbacks of the first one, exploring data models and tools that allow publishing and performing OLAP-like analysis directly over SW MD data. Terms like self-service BI [1], Situational BI [16], on-demand BI, or even Collaborative BI, refer to the capability of incorporating situational data into the decision process with little or no intervention of programmers or designers. The web, and in particular the SW, is considered as a large source of data that could enrich decision processes. Abelló et al. [1] present a framework to support self-service BI, based on the notion of fusion cubes, i.e., MD cubes that can be dynamically extended both in their schema and their instances, and in which data and metadata can be associated with quality and provenance annotations.

To support the second approach mentioned above, the RDF Data Cube vocabulary [6] proposes an RDF representation for statistical data according to the SDMX information model. Although similar to traditional MD data models, the SDMX semantics imposes restrictions on what can be represented using QB. Etcheverry and Vaisman [8] proposed QB4OLAP, an extension to QB that allows to represent analytical data according to traditional MD models, also presenting a preliminary implementation of some OLAP operators (Roll-Up, Dice, and Slice), using SPARQL queries over data cubes specified using QB4OLAP.

In [13] the authors present a framework for Exploratory OLAP over Linked Open Data sources, where the MD schema of the data cube is expressed in QB4OLAP and VoID. Based on this MD schema the system is able to query data sources, extract and aggregate data, and build an OLAP cube. The MD information retrieved from external sources is also stored using QB4OLAP.

For an exhaustive study of the possibilities of using SW technologies in OLAP, we refer the reader to the survey by Abelló et al. [2].

7. CONCLUSION AND FUTURE WORK

In this work, we presented a formal data model for data cubes, that allows the definition of a conceptual query language to manipulate data cubes, and showed that a data cube represented using this model can be published on the semantic web using the QB4OLAP vocabulary. With a focus on querying and publishing data cubes on the semantic web, we defined a conceptual query language for the data model previously described. Finally, we showed how SPARQL queries over QB4OLAP cubes can be automatically produced.

In future work we will concentrate on the composition of operators and the experimental evaluation of this proposal, which, to the best of our knowledge, is the first of its kind.

8. REFERENCES

[1] A. Abelló, J. Darmont, L. Etcheverry, M. Gelfarelli, J. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, and G. Vossen. Fusion cubes: Towards self-service business intelligence. IJDWM, 9(2):66–88, 2013.
[2] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis. Using semantic web technologies for exploratory olap: A survey. IEEE Transactions on Knowledge and Data Engineering, 27(2):571–588, 2015.
[3] D. Beckett and T. Berners-Lee. Turtle - Terse RDF Triple Language, 2011.
[4] D. Brickley, R. Guha, and B. McBride. RDF Vocabulary Description Language 1.0: RDF Schema, 2004.
[5] C. Ciferri, R. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi. Cube algebra: A generic user-centric model and query language for OLAP cubes. International Journal of Data Warehousing and Mining, 9(2):39–65, 2013.
[6] R. Cyganiak and D. Reynolds. The RDF Data Cube Vocabulary (W3C Recommendation), January 2014. http://www.w3.org/TR/vocab-data-cube/.
[7] L. Etcheverry and A. Vaisman. Enhancing OLAP analysis with web cubes. In Proceedings of ESWC, pages 469–483, Heraklion, Crete, Greece, 2012. Springer.
[8] L. Etcheverry and A. Vaisman. QB4OLAP: A vocabulary for OLAP cubes on the semantic web. In Proc. of COLD, Boston, USA, 2012. CEUR-WS.org.
[9] L. I. Gómez, S. A. Gómez, and A. A. Vaisman. A generic data model and query language for spatiotemporal OLAP cube analysis. In Proceedings of EDBT, pages 300–311. ACM, 2012.
[10] L. I. Gómez, S. A. Gómez, and A. A. Vaisman. A generic data model and query language for spatiotemporal OLAP cube analysis. In Proceedings of EDBT, pages 300–311. ACM, 2012.
[11] T. Heath and C. Bizer. Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.
[12] C. A. Hurtado, A. O. Mendelzon, and A. A. Vaisman. Maintaining Data Cubes under Dimension Updates. In Proceedings of the 15th International Conference on Data Engineering, ICDE ’99, pages 346–355, Washington, DC, USA, 1999. IEEE Computer Society.
[13] D. Ibragimov, K. Hose, T. Pedersen, and Z. E. Towards exploratory OLAP over linked open data: A case study. In Proceedings of the 7th International Workshop on Business Intelligence for the Real-Time Enterprises, BIRTE 2014. Springer, 2014.
[14] B. Kämpgen and A. Harth. Transforming statistical linked data for use in OLAP systems. In Proceedings of the 7th International Conference on Semantic Systems, I-Semantics ’11, pages 33–40, New York, NY, USA, 2011. ACM.
[15] G. Klyne, J. J. Carroll, and B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax, 2004.
[16] A. Löser, F. Hueske, and V. Markl. Situational Business Intelligence. In M. Castellanos, U. Dayal, and V. Markl, editors, Business Intelligence for the
[17] V. Nebot and R. B. Llavori. Building data warehouses with semantic web data. *Decision Support Systems*, 52(4):853–868, 2012.

[18] E. Prud’hommeaux and A. Seaborne. SPARQL 1.1 Query Language for RDF, 2011.

[19] A. Vaisman and E. Zimányi. *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[20] P. Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, SSDBM ’98, pages 53–62, Washington, DC, USA, 1998. IEEE Computer Society.