

Publishing OLAP Cubes on the Semantic Web

Alejandro Vaisman^(✉)

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

Abstract. The availability of large repositories of semantically annotated data on the web is opening new opportunities for enhancing Decision-Support Systems. In addition, the advent of initiatives such as Open Data and Open Government, together with the Linked Open Data paradigm, are promoting publication and sharing of multidimensional data (MD) on the web. In this paper we address the problem of representing MD data using Semantic Web (SW) standards. We discuss how MD data can be represented and queried directly over the SW, without the need to download data sets into local data warehouses. We first comment on the RDF Data Cube Vocabulary (QB), the current W3C recommendation, and show that it is not enough to appropriately represent and query MD data on the web. In order to be able to support useful Online Analytical Process (OLAP) analysis, extension to QB, denoted QB4OLAP, has been proposed. We provide an in-depth comparison between these two proposals, and show that extending QB with QB4OLAP can be done without re-writing the observations, (the largest part of a QB data set). We provide extensive examples of the QB4OLAP representation, using a portion of the Eurostat data set and the well-known Northwind database. Finally, we present a high-level query language, called QL, that allows OLAP users not familiar with SW concepts or languages, to write and execute OLAP operators without any knowledge of RDF or SPARQL, the standard data model and query language, respectively, for the SW. QL queries are automatically translated into SPARQL (using the QB4OLAP metadata) and executed over an endpoint.

Keywords: Data warehousing · OLAP · Semantic web · RDF · SPARQL · Linked data

1 Introduction

Data Warehouses (DW) integrate data from multiple sources for analysis and decision support. They represent data according to *dimensions* and *facts*. The former reflect the perspectives from which data are viewed. The latter correspond to (usually) quantitative data (also known as *measures*) associated with different dimensions. Facts can be aggregated and disaggregated through operations called Roll-up and Drill-down, respectively, filtered, by means of Slice and Dice operations, and so on. This process is called Online Analytical Processing

(OLAP). 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, at certain periods of time. Dimensions are modeled as hierarchies of elements (also called members), such that each element belongs to a category (or level) in a hierarchy. DWs and OLAP systems are based on the multidimensional (MD) model, which views data in an n -dimensional space, usually called a data cube, whose axes are the dimensions, and whose cells contain the values for the measures. In the former example, a point in this space could be (*January 2015, Buenos Aires*), where the measure in the cell indicates the amount of the sales in January 2015, at the Buenos Aires branch.

Historically, DW and OLAP had been used as techniques for data analysis, typically using 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. In the last decade, several open source platforms for Business Intelligence (BI) have emerged, but, at the time this tutorial paper is being written, an open format to publish and share cubes among organizations is still needed. Further, Linked Data [1], a data publication paradigm, promotes sharing and reusing data on the web using Semantic Web (SW) standards and domain ontologies expressed in RDF (the basic data representation layer for the SW) [2], or in languages built on top of RDF (like RDF-Schema [3]). All of the above has widened the spectrum of users, and nowadays, in addition to the typical OLAP analysts, non-technical people are willing to analyze MD data.

1.1 Problem Statement

Two main approaches are found concerning OLAP analysis of MD data on the SW. The first one aims at extracting MD data from the Web, and loading them into traditional data management systems for OLAP analysis. The second one proposes to carry out OLAP-like analysis directly over SW data, typically, over MD data represented in RDF. In this tutorial we focus on the latter approach although, for completeness, in Sect. 2 we discuss and compare both lines of work.

Publishing and analyzing OLAP data directly over the SW, supports the concepts of self-service BI, or on-demand BI, aimed at incorporating web data into the decision-making process with little or no intervention of programmers or designers [4]. Statistical data sets are usually published using the RDF Data Cube Vocabulary (also denoted QB) [5], a W3C recommendation since January, 2014. However, as we explain later, among other limitations, the QB vocabulary does not support the representation of dimension hierarchies and aggregation functions needed for OLAP analysis. To address this challenge, a new vocabulary, called QB4OLAP has been proposed [6]. A key feature of QB4OLAP is that it allows reusing data already published in QB, by means of the addition of the hierarchical structure of the dimensions (and the corresponding instances

¹ <http://okfn.org/opendata/>.

² <http://opengovdata.org/>.

that populate the dimension levels). Once a data cube becomes published using QB4OLAP, users can perform OLAP operations over it. Moreover, high-level languages can be used to seamlessly query these data cubes, as we will show later.

1.2 Running Example

As our running example, we will use statistical data about asylum applications to the European Union (EU), provided by Eurostat, the EU’s statistical office³. This data set contains information about the number of asylum applicants by month, age, sex, citizenship, application type, and country that receives the application, and it is published using QB in the Eurostat - Linked Data dataspace⁴. For this tutorial, we extended the original QB data cube with dimension hierarchies, as shown in Fig. 1, using the MultiDim conceptual model [7]. The `Asylum_applications` fact contains only one measure (`#applications`) that represents the number of applications. This measure can be analyzed according to six analysis dimensions: the `sex` of the applicant, `age` which organizes applicants according to their age group, the time of the application (which includes a two-level hierarchy (with levels `month` and `year`), the `application_type`, which tells if the person is a first time applicant or a repeated applicant, and a geographical dimension that organizes countries into continents (the `Geography` hierarchy), or according to its government type (the `Government` hierarchy). This geographical dimension participates in the cube with two different roles: The `citizenship` of the asylum seeker, and the `destination` country of its application. Usually, these kinds of dimensions are denoted *role-playing* dimensions.

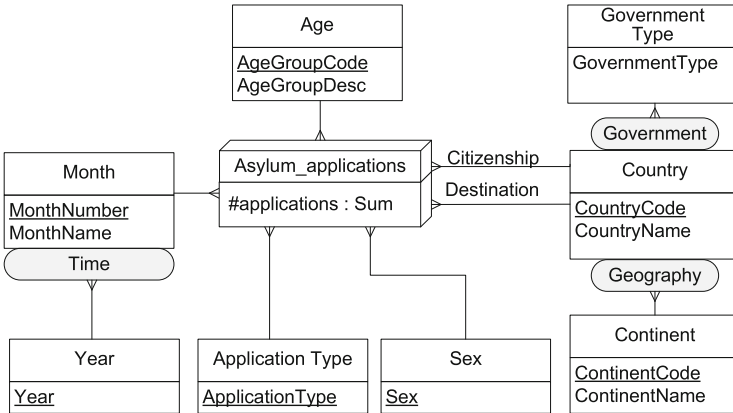


Fig. 1. Conceptual schema of the asylum applications cube

³ http://epp.eurostat.ec.europa.eu/cache/ITY_SDDS/EN/migr_asyapp_esms.htm.

⁴ <http://eurostat.linked-statistics.org/>.

1.3 Contributions

Although QB provides basic multidimensional information, this information is not enough to satisfy OLAP users' needs. In this way, a great part of the benefit of having MD data on the web gets lost. Further, since the QB model does not provide enough information for OLAP analysis, once downloaded, the data must be extended with the typical MD constructs. QB4OLAP has been proposed to address these drawbacks, allowing data owners to publish MD data on the SW, and to enrich existing data sets with structural metadata, and dimensional data. This enrichment can be done semi-automatically (a problem which is beyond the scope of the present paper, and is explained in detail in [8,9]). Also, QB4OLAP data cubes can be created from scratch, for example, integrating on-the-fly, data on the web. Last, but not least, a cube representation like the one allowed by QB4OLAP can not only be used to perform OLAP analysis through queries written in SPARQL [10] (the standard query language for RDF), but to express these queries using a high-level declarative query language, which can be then automatically translated into SPARQL (with the help of the QB4OLAP metadata), allowing non-technical users to perform OLAP data analysis without the need to understand how data are represented. In other words, typical OLAP users could be able to query MD data represented in RDF without the need of having any knowledge of SPARQL.

Concretely, in this tutorial paper we present:

- A comparison between the QB and QB4OLAP vocabularies;
- A description of how QB cubes can be enriched with OLAP metadata and data, and how existing DW can be published using the QB4OLAP vocabulary;
- A user-centric high-level query language, called QL, that expresses the most common OLAP operators independently of the underlying data representation, and a mechanism to automatically translate a QL expression into SPARQL, to query QB4OLAP cubes.

The remainder of the paper is organized as follows. Section 2 discusses related work. In Sect. 3 we introduce the basic concepts used throughout this paper. Section 4 studies the QB vocabulary, and discusses its limitations for representation and querying of MD data. Section 5 presents the QB4OLAP vocabulary, and an in-depth comparison against QB. Section 6 studies the Cube Algebra language, a high-level language to query cubes, and Sect. 7, the translation of Cube Algebra into SPARQL, to query cubes whose underlying representation is based on RDF and the QB4OLAP vocabulary. We conclude in Sect. 8.

This tutorial paper follows the presentation given by the author in the EBISS 2015 Summer School. It is not aimed at presenting original research material, but to put together, in a tutorial style, the main contributions of the work performed by the author in collaboration with other colleagues [6,8,9,11,12].

2 Related Work

As mentioned above, two main approaches concerning OLAP analysis of MD data on the SW can be found in the literature. The first one consists in extracting

MD data from the SW and loading them into traditional MD data management systems for OLAP analysis, while the second one promotes performing OLAP-like analysis directly over SW data.

Along the first line of research, we find the works by Nebot and Llavori [13] and Kämpgen and Harth [14]. The former proposes a semi-automatic method for on-demand extraction of semantic data into an MD database, so data could be analyzed using traditional OLAP techniques. The authors present a methodology for discovering facts in SW data (represented as an OWL⁵ ontology), and populating an MD model with such facts. In this methodology, an MD schema is initially designed, indicating the subject of analysis that corresponds to a concept of the ontology, the potential dimensions, and the facts. Then, the dimension hierarchies are created, based on the knowledge available in the domain ontologies (i.e., the inferred taxonomic relationships). Finally, the user specifies the MD queries over the DW. Once queries are executed, a cube is built, and 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 an MD database. The authors propose a mapping between the concepts in QB, and an MD data model, and implement these mappings via SPARQL queries. In this methodology, the user first defines relevant data sets, which are retrieved from the web, and stored in a local triple store. A relational representation of the MD data model is then created and populated. Over this model, OLAP operations can be performed.

These two efforts are based on traditional MD data management systems, and require the existence of a local DW to store SW data. Also, they do not consider the possibility of directly querying *à la* OLAP MD data over the SW. Thus, a second line of research tries to overcome these drawbacks, exploring data models and tools that allow publishing and performing OLAP-like analysis directly over SW MD. The work we discuss in the remainder, follows this approach.

Terms like self-service BI [4], and Situational BI [15], refer to the capability of incorporating situational data into the decision process with little or no intervention of programmers or designers. In [4], the authors present a framework to support self-service BI, based on the notion of fusion cubes, i.e., multidimensional 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. These frameworks motivate the need for models and tools that allow to query MD data directly over the SW.

The RDF Data Cube vocabulary [5] is aimed at representing, using RDF, statistical data according to the SDMX⁶ information model discussed in Sect. 3.2. Although similar to traditional MD data models, the SDMX semantics imposes restrictions on what can be represented using QB. In particular, dimension hierarchies, a key concept in OLAP operations, are not appropriately supported in QB. To overcome this limitation, Etcheverry and Vaisman [6] proposed QB4OLAP, an extension to QB that allows representing analytical data

⁵ <http://www.w3.org/TR/owl2-overview/>.

⁶ <http://SDMX.org>.

according to traditional MD models, also proposing a preliminary implementation of some OLAP operators, using SPARQL queries over data cubes specified using QB4OLAP.

In [16] the authors present a framework for performing exploratory OLAP over Linked Open Data sources, where the multidimensional schema of the data cube is expressed in QB4OLAP and VoID. Based on this multidimensional schema the system is able to query data sources, extract and aggregate data, and build an OLAP cube. The multidimensional information retrieved from external sources is also stored using QB4OLAP.

The QB and QB4OLAP approaches will be compared in depth in Sect. 4, and, after this, the paper will be devoted to the study of QB4OLAP and its applications.

3 Preliminary Concepts

In this section we introduce the concepts that we will use in the rest of the paper. To set up a common analysis framework, we first need to briefly define the MD model for OLAP that will be used in our study. We do this in the first part of the section. In the second part we discuss statistical databases (SDB), and introduce the SDMX model, on which QB is based. We conclude with a definition of the basic SW concepts that we will need in the sequel.

3.1 OLAP

A broad number of MD models can be found in the literature [17–19]. We now describe the MD model for OLAP that we will use in our study.

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 . Although in practice, most MD models assume a function between the instances of parent and child dimension levels, we support relations between them, meaning that each member in the child level many have more than one associated member in the parent level, and vice versa (hierarchies including rollup relations are called non-strict). Cardinality constraints on these relations are then used to restrict the number of level members related to each other [7]. 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 Instance*, corresponding to a cube schema, is a partial function mapping coordinates from dimension instances into measure values.

A well-known set of operations is defined over cubes. For instance, based on the algebra sketched in [20], the *Roll-Up* operation summarizes data in a cube, along a dimension hierarchy. Analogously, *Drill-Down* disaggregates previously summarized data, and can be considered the inverse of Roll-Up. The *Slice* operation drops a dimension from a cube. The *Dice* operation receives a cube \mathcal{C} , and a first order formula ϕ over levels and measures in \mathcal{C} , and returns a new cube with the same schema, and whose instances are the ones that satisfy ϕ . There are more complex operators, but for the sake of simplicity, we will limit ourselves to the ones mentioned above.

3.2 Statistical Databases and the SDMX Model

Statistical Data Bases (SDB) also organize data as hypercubes whose axes are dimensions. Each point in this multidimensional space is mapped through observations into one or more spaces of measures. Dimensions are structured in classification hierarchies that allow analysis at different levels of aggregation. The Statistical Data and Metadata eXchange initiative (SDMX) proposes several standards for the publication, exchange and processing of statistical data, and defines an information model [21] from which we summarize some concepts next, since QB is based on SDMX.

In the SDMX model, a *Dimension* denotes a metadata concept used to classify a statistical series, e.g., a statistical concept indicating a certain economic activity. Two particular dimensions are identified: *TimeDimension*, specifying a concept used to convey the time period of the observation in a data set; and *MeasureDimension*, whose purpose is to specify formally the meaning of the measures and to enable multiple measures to be defined and reported in a data set. A *Primary Measure* denotes a metadata concept that represents the phenomenon to be measured in a data set. Dimensions, measures, and attributes are called *Components*.

Codelists enumerate a set of values to be used in the representation of dimensions, attributes, and other structural parts of SDMX. Additional structural metadata can indicate how codes are organized into hierarchies. Through the inheritance abstraction mechanism, the codelist comprises one or more codes, and the code itself can have one or more children codes in the (inherited) hierarchy association. Note that a child code can have only one parent code in this association.

A *Data Set* denotes a set of observations that share the same dimensionality, which is specified by a set of unique components (e.g., dimensions, measures). Each data set is associated with structural metadata, called *Data Structure Definition* (DSD), that includes information about how concepts are associated with the measures and dimensions of a data cube along descriptive (structural) metadata.

The value of the variable being measured for the concept associated to the *PrimaryMeasure* in the DSD is called an *Observation*. Each observation associates an *observation value* with a *key value*.

Several operators are defined over SDBs, although the SDMX standard does not define operators over data sets. Instead, it provides a mechanism to restrict the values within a data set via *constraints*. For example, the *CubeRegions* constraint, allows specifying a set of component values, defining a subset of the total range of the content of a data structure. The application of this constraint results in a *slice* of the original data set, fixing values for some components (e.g.: selecting some years in a TimeDimension). Therefore, the name *slice* may be misleading for OLAP practitioners, since in OLAP, a slicing operation reduces the cube's dimensionality, as explained in Sect. 3.1.

3.3 RDF and the Semantic Web

The 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 *subject* and *object* are nodes, and *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 must always be resources or blank nodes, predicates are always resources, and objects could be resources, blank nodes or literals. A set of reserved words defined in RDF Schema (called the *rdfs-vocabulary*) [3] 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* but it also implicitly states that object *c* is an instance of *rdf:Class* since there exists at least one resource that is an instance of *c*. Many formats for RDF serialization exist. In this paper we use Turtle [22].

SPARQL 1.1 [10] is the W3C standard query language for RDF, at the time this paper is being written. The query evaluation mechanism of SPARQL is based on subgraph matching: RDF triples are interpreted as nodes and edges of directed graphs, and the query graph is matched to the data graph, instantiating the variables in the query graph definition. The selection criteria is expressed as a graph pattern in the WHERE clause of a SPARQL query. Relevant to OLAP queries, SPARQL supports aggregate functions and the GROUP BY clause, as in classic SQL.

Due to space limitations, in the remainder we assume the reader is familiar with the basic notions of RDF and SPARQL.

4 QB: The RDF Data Cube Vocabulary

We now study in detail the QB vocabulary, and discuss its possibilities and limitations for representing and analyzing MD data.

4.1 Vocabulary Description

As mentioned above, QB is the W3C recommendation to publish statistical data and metadata in RDF, following the Linked Data principles. QB is based on the SDMX Information Model described in Sect. 3.2, and is the evolution of two previous attempts to represent statistical data in RDF: the Statistical Core Vocabulary (SCOVO) [23], and SDMX-RDF [24]. Figure 2 (taken from the W3C recommendation document [5]) depicts the QB vocabulary. Capitalized terms represent RDF classes and non-capitalized terms represent RDF properties. 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-classes or sub-properties. We describe the QB vocabulary next.

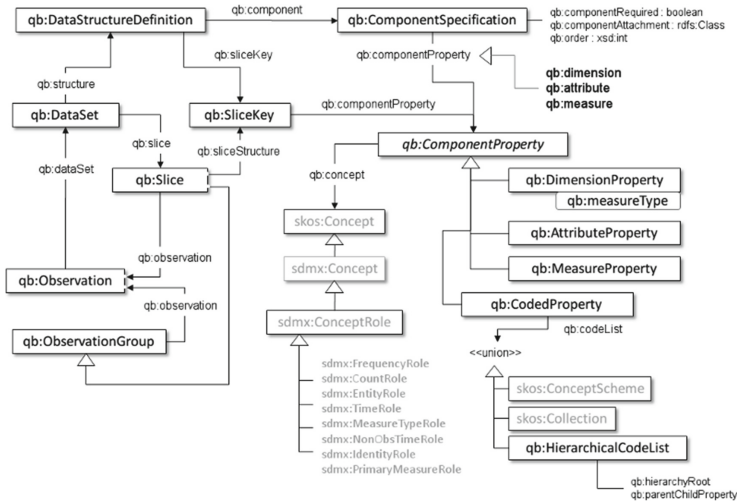


Fig. 2. The QB vocabulary (cf. [5])

The schema of a data set is specified by means of the DSD (like in SDMX), an instance of the class `qb:DataSetDefinition`. This specification comprises a set of *Component* properties, instances of the class `qb:ComponentProperty` (in italics in Fig. 2), representing *Dimensions*, *Measures*, and *Attributes*. This is shown in Example 1. Note that a DSD can be shared by many data sets by means of the `qb:structure` property. *Observations* (in OLAP terminology, *facts*), are instances of the class `qb:Observation`, and represent points in an MD data space indexed by dimensions. They are associated with data sets (instances of the class `qb:DataSet`), through the `qb:dataSet` property (see Example 2). Each observation can be linked to a value in each dimension of the DSD via instances of `qb:DimensionProperty`; analogously, values for each observation are associated with measures via instances of the class `qb:MeasureProperty`. Instances of the class `qb:AttributeProperty` are used to associate attributes with observations.

Finally, note that QB allows observations in a data set to be expressed at different levels of granularity in each dimension. For example, one observation may refer to a country, and another one may refer to a region.

Component properties are not directly related to the DSD: the class `qb:ComponentSpecification` is an intermediate class which allows to specify additional attributes for a component in a DSD. For example, a component may be tagged as *required* (i.e., mandatory), using the `qb:componentRequired` property. Components that belong to a specification are linked using specific properties that depend on the type of the component, that is, `qb:dimension` for dimensions, `qb:measure` for measures, and `qb:attribute` for attributes. Component specifications are linked to DSDs via the `qb:component` property. For instance, in Example 1 we can see how dimensions are defined in the DSD, through the `qb:dimension` and `qb:component` properties.

In order to allow reusing the concepts defined in the SDMX Content Oriented Guidelines [25], QB provides the `qb:concept` property which links components to the general concepts they represent. The latter are modeled using the `skos:Concept` class defined in the SKOS vocabulary.⁷

Although QB can define the structure of a fact (via the DSD), it does not provide a mechanism to represent an OLAP dimension structure (i.e., the dimension levels and the relationships between levels). However, QB allows representing hierarchical relationships between level members in the dimension instances. The QB specification describes three possible scenarios with respect to the organization of dimensions, as we explain next.

- If there is no need to define hierarchical relationships within dimension members, QB recommends representing the members using instances of the class `skos:Concept`, and the set of admissible values using `skos:ConceptScheme`. A SKOS concept scheme allows organizing one or more SKOS concepts, linked to the concept schemes they belong to, via the `skos:inScheme` property.
- To represent hierarchical relationships, the recommendation is to use the semantic relationship `skos:narrower`, with the following meaning: if two concepts A and B are related using `skos:narrower`, B represents a finer concept than A (e.g., `animals skos:narrower mammals`). In addition, SKOS defines a `skos:hasTopConcept` property, which allows linking a concept scheme to the (possibly many) most general concept it contains. To reuse existing data, QB provides the class `qb:HierarchicalCodeList`. An instance of this class defines a set of root concepts in the hierarchy using `qb:hierarchyRoot` and a parent-child relationship via `qb:parentChildProperty` which links a term in the hierarchy to its immediate sub-terms.

Finally, *Slices* represent subsets of observations. They are not defined as operators over an existing cube, but as new structures and new instances (observations), where one or more values of dimension members are fixed. The structure of a slice is defined using a DSD, and an instance of the `qb:SliceKey` class.

⁷ <http://www.w3.org/TR/2009/REC-skos-reference-20090818/>.

Example 1 below presents the triples that represent a portion of the structure of the QB data set in our running example. Note that components are defined as RDF blank nodes.

Example 1 (Data Set Structure Definition).

```

1 @prefix qb: <http://purl.org/linked-data/cube#> .
2 @prefix sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#> .
3 @prefix sdmx-measure: <http://purl.org/linked-data/sdmx/2009/measure#> .
4 @prefix dsd: <http://eurostat.linked-statistics.org/dsd/> .
5 @prefix property: <http://eurostat.linked-statistics.org/property#> .
6
7 dsd:migr_asyappctzm rdf:type qb:DataStructureDefinition ;
8   qb:component [qb:dimension property:age] ;
9   qb:component [qb:dimension property:geo] ;
10  qb:component [qb:dimension property:sex] ;
11  qb:component [qb:dimension property:citizen] ;
12  qb:component [qb:dimension property:asyl_app] ;
13  qb:component [qb:dimension sdmx-dimension:refPeriod] ;
14  qb:component [qb:measure sdmx-measure:obsValue] .
15
16 <http://eurostat.linked-statistics.org/data/migr_asyappctzm> qb:structure dsd:migr_asyappctzm

```

Line 7 defines the IRI of the DSD. The lines that follow, indicate the components of such structure, and Line 16 tells that the DSD is the structure of the data set in the subject of the triple. □

Continuing with the Eurostat running example, Example 2 below shows the triples that represent an observation (in OLAP jargon, a *fact*), corresponding to the schema above.

Example 2 (Observations). The following triples represent an observation corresponding to the number of citizens of Andorra submitting applications to migrate to Austria in 2014.

```

1 @prefix data: <http://eurostat.linked-statistics.org>;
2 <http://eurostat.linked-statistics.org/data/migr_asyappctzm#M,AD,F,TOTAL,ASY_APP,AT,2014M10>
3   a qb:Observation ;
4   qb:dataSet <http://data/migr_asyappctzm> ;
5   property:age data:dic/age#TOTAL;
6   property:geo data:dic/geo#AT;
7   property:sex data:dic/sex#F;
8   property:citizen data:dic/citizen#AD;
9   property:asyl_app data:dic/asyl_app#ASY_APP;
10  sdmx-dimension:refPeriod 2014-10-0;
11  sdmx-measure: obsValue 0 .

```

Line 2 tells that the IRI in the subject is an instance of the class `qb:Observation`, and Line 4 indicates the data set to which the observation belongs. The other triples correspond to the dimension instances and the observed value (the measure, in Line 11). □

4.2 Is QB Suitable for OLAP?

Although QB can be used to publish MD observations, it cannot represent the most typical features of the MD model that are used to navigate data in an OLAP fashion. We discuss this next.

1. *QB does not provide native support for dimension structures.* Typical OLAP operations, like Roll-up and Drill-down, rely on the organization of dimension members into hierarchies that define aggregation levels. However, as explained above, QB cannot represent the structural metadata needed to appropriately support such operations. The mechanisms described in Sect. 4.1 allows only to organize dimension members hierarchically, that means, they can only represent relationships between instances, for example, to say that Argentina is a finer concept than South America, but not to say that Argentina is a country, South America is a continent, and that countries aggregate over continents.
2. *QB does not provide native support to represent aggregate functions.* Most OLAP operations aggregate or disaggregate cube data along a dimension (e.g., a Roll-up operation over the Time dimension can aggregate measure values from the Month level up to Year level), using an aggregate function defined for each measure. Normally, it is not possible to assume a single aggregate function for all measures. The ability to link each measure with an aggregation function is not present in QB.
3. *QB does not provide native support for descriptive attributes.* In the MD model, each dimension level is associated with a set of *attributes* that describe the characteristics of the dimension members (e.g. the level Country may have the attributes *countryName*, *area*, etc.), and one or more *identifiers* [7]. However, in QB, dimension members are represented as coded values, which in most cases are represented as IRIs (although this is not mandatory). We will see later, that this limitation can have impact over some operations, typically, when dicing over a dimension.

5 The QB4OLAP Vocabulary

From the discussion in Sect. 4.2, the need of a more powerful vocabulary was evident. Thus, the QB4OLAP⁸ vocabulary has been proposed, extending QB with a set of RDF terms that allow representing the most common features of the MD model. The main features of QB4OLAP are:

- QB4OLAP can represent the most common features of the MD model. Given that there is no standard (or widely accepted) conceptual model for OLAP, the features considered were based on the MultiDim model [7].
- QB4OLAP includes the metadata needed to automatically implement OLAP operations as SPARQL queries. Using these metadata (e.g., the aggregation paths in a dimension), the operations could be written in a high-level language (or submitted using a graphic navigation tool), and translated into SPARQL. In this way, OLAP users, with no knowledge of SPARQL at all, would be able to exploit data on the SW.
- QB4OLAP allows operating 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

⁸ <http://purl.org/qb4olap/cubes>.

schema of the cube can be specified in terms of dimension levels, using the (QB) class `qb:DataStructureDefinition` (allowing reusing existing QB observations, if needed). To represent aggregate functions the class `qb4o:AggregateFunction` is defined. The property `qb4o:aggregateFunction` associates measures with aggregate functions, and, together with the concept of component sets, allows a given measure to be associated with different aggregate functions in different cubes, addressing one of the drawbacks of QB. Finally, when a fact (observation) is related to more than one dimension level member (this is called a many-to-many dimension [7]), the property `qb4o:cardinality` allows representing the cardinality of this relationship.

Example 3 below, shows how the cube in our Eurostat running example would look like in QB4OLAP. Figure 4 presents the definition of the prefixes that we will use in the sequel.

```

1 PREFIX qb: <http://purl.org/linked-data/cube#>
2 PREFIX qb4o: <http://purl.org/qb4olap/cubes#>
3
4 #QB4OLAP schema and instances
5 PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
6 PREFIX instances: <http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp#>

```

Fig. 4. RDF prefixes to be used in the examples

Example 3 (Eurostat Cube Structure in QB4OLAP). Below, we show the structure of a data cube for the Eurostat example, represented using QB4OLAP. The reader is suggested to compare against the DSD in Example 1.

```

1 schema:migr_asyappctzmQB4O rdfs:type qb:DataStructureDefinition;
2
3 qb:component [ qb4o:level property:age ; qb4o:cardinality qb4o:ManyToOne ] ;
4 qb:component [ qb4o:level property:geo ; qb4o:cardinality qb4o:ManyToOne ] ;
5 qb:component [ qb4o:level property:sex ; qb4o:cardinality qb4o:ManyToOne ] ;
6 qb:component [ qb4o:level property:citizen qb4o:cardinality qb4o:ManyToOne ] ;
7 qb:component [ qb4o:level property:asyl_app ; qb4o:cardinality qb4o:ManyToOne ] ;
8 qb:component [ qb:measure sdmx—measure:obsValue; qb4o:aggregateFunction qb4o:sum ] ;
9
10 <http://eurostat.linked-statistics.org/data/migr_asyappctzm> qb:structure
11 schema:migr_asyappctzmQB4O.
12 sdmx—measure:obsValue a qb:MeasureProperty;
13 rdfs:label "Number of applications"@en; rdfs:range xsd:integer .

```

Note that, opposite to QB, the structure is defined in terms of dimension levels, which represent the granularity of the observations in the data set. Each level is associated to a cardinality, using the property `qb4o:cardinality`. In this case, all cardinalities are many-to-one, indicating that an observation is associated to exactly one member in every dimension level. To avoid rewriting the observations, a QB4OLAP DSD schema:`migr_asyappctzmQB4O` is created, and associated with the data set `<http://eurostat.linked-statistics.org/data/migr_asyappctzm>` (recall that in Example 1, the data set structure was `dsd:migr_asyappctzm`). This allows reusing, as QB4OLAP level properties, the dimension properties already defined in the QB structure, allowing to use

the existing observations, since the data set will “point” to this new DSD. Thus, we must declare those properties as instances of `qb4o:LevelProperty`. For example, for the Time dimension, we must define (we explain this dimension in detail later):

```

1 schema:timeDim a qb:DimensionProperty ;
2   rdfs:label "Time dimension"@en ;
3   qb4o:hasHierarchy schema:timeHier .
4
5 sdmx-dimension:refPeriod a qb4o:LevelProperty ;
6   rdfs:label "Month level"@en .

```

We can see that `sdmx-dimension:refPeriod` (the Time dimension) is redefined as a dimension level using the class `qb4o:LevelProperty`; a dimension `schema:timeDim` is defined using the QB class `qb:DimensionProperty`. In addition, a dimension hierarchy `schema:timeHier` is defined. Since the dimension levels defined in this way are the lowest ones in the dimension hierarchies, a QB4OLAP cube schema can then be defined using these properties. We explain this below. \square

Dimension hierarchies are represented using the class `qb4o:Hierarchy`; further, the properties `qb4o:hasHierarchy` and `qb4o:inDimension`, tell that a dimension contains a certain hierarchy, and that a certain hierarchy belongs to a dimension, respectively. Also, hierarchies are composed of levels, and the relationship between levels in a hierarchy may have different cardinality constraints (e.g. one-to-many, many-to-many, etc.). We call these relationships *hierarchy steps*, which are represented by the class `qb4o:HierarchyStep`. Each hierarchy step is linked to its two component levels using the `qb4o:childLevel` and `qb4o:parentLevel` properties, and can be attached to the hierarchy it belongs to, using the property `qb4o:inHierarchy`. The property `qb4o:pcCardinality` represents the cardinality constraints of the relationships between level members in this step, associating a hierarchy with a member of the `qb4o:Cardinality` class, whose instances are depicted in Fig. 3. Example 4 shows a part of the definition of the dimension hierarchies for our running example.

Example 4 (Dimension Structure and Hierarchies in QB4OLAP). In addition to the definition of the Time dimension structure (`schema:timeDim`) shown in Example 3, we can define one or more hierarchies, and declare which dimension they belong to, and the levels that they traverse. In this example, we create a hierarchy denoted `schema:timeHier`, with two levels, `sdmx-dimension:refPeriod`, and `schema:year`, representing the aggregation levels `month` (the bottom level) and `year`, respectively. Also, the distinguished level `All` is defined, as `schema:timeAll`. Below, we show these definitions.

```

1 schema:timeHier a qb4o:Hierarchy ;
2   rdfs:label "Time Hierarchy"@en ;
3   qb4o:inDimension schema:timeDim ;
4   qb4o:hasLevel sdmx-dimension:refPeriod, schema:year, schema:timeAll .
5
6 sdmx-dimension:refPeriod a qb4o:LevelProperty ;
7   rdfs:label "Month level"@en .
8

```

```

9 schema:year a qb4o:LevelProperty ;
10   rdfs:label "Year"@en .
11
12 schema:timeAll a qb4o:LevelProperty ;
13   rdfs:label "All dates"@en .

```

We remark that the lowest granularity level for the time dimension is defined as in QB (i.e., `sdmx-dimension:refPeriod`), but as a dimension level instead of a dimension.

The parent-child relationships between levels are defined as *hierarchy steps*, using the class `qb4o:HierarchyStep`, as we show below.

```

1 ..ih21 a qb4o:HierarchyStep ;
2   qb4o:inHierarchy schema:timeHier ;
3   qb4o:childLevel sdmx-dimension:refPeriod ;
4   qb4o:parentLevel schema:year; qb4o:pcCardinality qb4o:ManyToOne .
5
6 ..ih22 a qb4o:HierarchyStep;
7   qb4o:inHierarchy schema:timeHier ;
8   qb4o:childLevel schema:year ;
9   qb4o:parentLevel schema:timeAll ; qb4o:OneToManyToOne .

```

Note that we indicated, for each step (represented using a blank node), to which hierarchy it belongs, which level is the parent (i.e., the level with coarser granularity), and which level is the child (i.e., the level with finer granularity), and the cardinality of the relationship. □

Finally, in order to address the lack of support for *level attributes* in QB, QB4OLAP provides the class of properties `qb4o:LevelAttribute`. This class is linked to `qb4o:LevelProperty`, via the `qb4o:hasAttribute` property. For completeness, QB4OLAP includes the `qb4o:inLevel` property, with domain in the class `qb4o:LevelAttribute` and range in the class `qb4o:LevelProperty`. The `qb4o:inLevel` property is rarely used, but is included for completeness, as kind of an “inverse” of `qb4o:hasAttribute` (note that RDF does not allow to represent the inverse of a property). Level attributes are useful in OLAP to filter cubes according to some attribute property. Example 5 shows the definition of an attribute for the time dimension level `sdmx-dimension:refPeriod`.

Example 5 (Level Attributes). For this example, assume we add attribute `schema:monthNumber` to the level `sdmx-dimension:refPeriod` in the time dimension.

```

1 sdmx-dimension:refPeriod qb4o:hasAttribute schema:monthNumber .
2
3 schema:monthNumber a qb4o:LevelAttribute;
4   rdfs:label "Month number"@en.

```

Note that the attribute `schema:monthNumber` is declared indicating that it is an instance of the class `qb4o:LevelAttribute`. □

5.2 Dimension Instances in QB4OLAP

Typically, instances of OLAP dimensions levels are composed of so-called level members. In QB4OLAP, level members are represented as instances of the class

`qb4o:LevelMember`, which is a sub-class of `skos:Concept`. Members are associated with the level they belong to, using the property `qb4o:memberOf`, whose semantics is similar to `skos:member`. Rollup relationships between members are expressed using the property `skos:broader`. The choice of this property, instead of `skos:narrower`, like it is recommended in QB, aims at indicating that the hierarchies of level members are usually navigated from finer granularity concepts up to higher granularity concepts. Example 6 below illustrates this.

Example 6 (Dimension Instances in QB4OLAP). We show now some examples of members of levels in the dimension `schema:timeDim`.

```

1 @prefix time:<http://purl.org/qb4olap/dimensions/time#> .
2
3 time:TOTAL
4   qb4o:memberOf schema:timeAll .
5
6 time:200801
7   qb4o:memberOf sdmx-dimension:refPeriod;
8   skos:broader time:2008 .
9 ...
10 time:2008
11   qb4o:memberOf schema:year;
12   skos:broader time:TOTAL .
13 ...
14 time:2009
15   qb4o:memberOf schema:year;
16   skos:broader time:TOTAL .
17 ...
18 time:201401
19   qb4o:memberOf sdmx-dimension:refPeriod;
20   skos:broader time:2014 .
21 ...
22 time:2014
23   qb4o:memberOf schema:year;
24   skos:broader time:TOTAL .
25 ...

```

In Lines 6 through 8 we indicate that the month January of 2008 belongs to level `sdmx-dimension:refPeriod`, and rolls up to the element `time:2008`, an IRI representing the year 2008. In turn, `time:2008`, defined in Lines 10 through 12, rolls up to the level `time:TOTAL`, which represents the distinguished member all (although it is not mandatory to indicate this element).

Analogously to level members, we must define the instances of level attributes. For this, associate the IRIs corresponding to level members, with literals corresponding to attribute values (i.e., attribute instances). In our example, for the Time dimension we have:

```

1 time:201401 schema:monthNumber "201401"^^xsd:integer .

```

Note that, opposite to level members, which are IRIs, attribute instances are always literals (since QB4OLAP does not define, for attributes, a class analogous to `qb4o:MemberOf`). □

5.3 How Can We Use QB4OLAP?

There are three basic ways of using QB4OLAP: (a) To enrich an existing data set published in QB, with structural metadata and dimensional data; (b) To

publish an existing data cube/data warehouse; (c) To build a new cube, using QB4OLAP, from scratch. We already discussed option (a). We do not specifically address option (c) here, since it comprises the tasks of the first two ones. We briefly address option (b) in this section.

To illustrate how we can publish an existing DW on the SW using QB4OLAP, we use the well-known Northwind DW (see [7] for a detailed explanation of the Northwind DW design). Figure 5 shows the conceptual model of the Northwind DW using the MultiDim model.

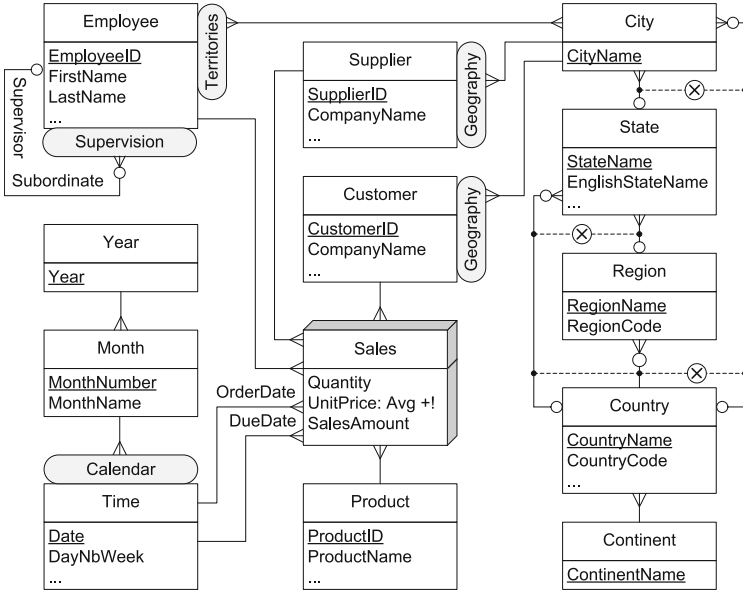


Fig. 5. Conceptual schema of the Northwind DW

It has been already shown that most of the widely used features of the MultiDim conceptual model, and, in general, of the MD model, can be represented using QB4OLAP [12]. Therefore, we do not extend here on this explanation, but below, we give some examples using the Northwind DW.

Example 7 (Northwind DW Structure Definition). This example shows a portion of the DSD that exposes the structure of the Northwind DW in QB4OLAP. The DSD is denoted `nw:Northwind`. It comprises nine dimensions and six measures.

```

1 @prefix nw: <http://dwbook.org/cubes/schemas/northwind#> .
2
3 # Cube definition
4
5 nw:Northwind a qb:DataStructureDefinition ;
6
7 # Lowest level for each dimension in the cube

```

```

8   qb:component [ qb4o:level nw:employee ; qb4o:cardinality qb4o:ManyToOne ] ;
9   qb:component [ qb4o:level nw:orderDate ; qb4o:cardinality qb4o:ManyToOne ] ;
10  qb:component [ qb4o:level nw:dueDate ; qb4o:cardinality qb4o:ManyToOne ] ;
11  qb:component [ qb4o:level nw:shippedDate ; qb4o:cardinality qb4o:ManyToOne ] ;
12  qb:component [ qb4o:level nw:product ; qb4o:cardinality qb4o:ManyToOne ] ;
13  qb:component [ qb4o:level nw:order ; qb4o:cardinality qb4o:OneToOne ] ;
14  qb:component [ qb4o:level nw:shipper ; qb4o:cardinality qb4o:ManyToOne ] ;
15  qb:component [ qb4o:level nw:supplier ; qb4o:cardinality qb4o:ManyToOne ] ;
16  qb:component [ qb4o:level nw:customer ; qb4o:cardinality qb4o:ManyToOne ] ;
17
18  # Measures in the cube
19  qb:component [ qb:measure nw:quantity ; qb4o:hasAggregateFunction qb4o:sum ] ;
20  qb:component [ qb:measure nw:unitPrice ; qb4o:hasAggregateFunction qb4o:avg ] ;
21  qb:component [ qb:measure nw:discount ; qb4o:hasAggregateFunction qb4o:avg ] ;
22  qb:component [ qb:measure nw:salesAmount ; qb4o:hasAggregateFunction qb4o:sum ] ;
23  qb:component [ qb:measure nw:freight ; qb4o:hasAggregateFunction qb4o:sum ] ;
24  qb:component [ qb:measure nw:netAmount ; qb4o:hasAggregateFunction qb4o:sum ] .
25
26  # Measures definition
27
28  nw:quantity a rdf:Property , qb:MeasureProperty ;
29      rdfs:label "Quantity"@en ;
30      rdfs:subPropertyOf sdmx—measure:obsValue ;
31      rdfs:range xsd:positiveInteger .
32
33  nw:unitPrice a rdf:Property , qb:MeasureProperty ;
34      rdfs:label "Unit Price"@en ;
35      rdfs:subPropertyOf sdmx—measure:obsValue ;
36      rdfs:range xsd:decimal .
37  ...

```

Next, we show the schema of part of the *Employee* dimension, illustrating the representation of the recursive *Supervision* hierarchy, and the definition of level attributes.

```

1  # -- Employee dimension definition
2
3  nw:employeeDim a rdf:Property , qb:DimensionProperty ;
4      rdfs:label "Employee Dimension"@en ;
5      qb4o:hasHierarchy nw:supervision , nw:territories .
6
7  # -- Supervision hierarchy
8
9  nw:supervision a qb4o:Hierarchy ;
10     rdfs:label "Supervision Hierarchy"@en ;
11     qb4o:inDimension nw:employeeDim ;
12     qb4o:hasLevel nw:employee .
13
14  _:supervision_hs1 a qb4o:HierarchyStep ;
15     qb4o:inHierarchy nw:supervision ;
16     qb4o:childLevel nw:employee ;
17     qb4o:parentLevel nw:employee ;
18     qb4o:pcCardinality qb4o:ManyToOne .
19  ...
20
21  # -- Employee level
22
23  nw:employee a qb4o:LevelProperty ;
24     rdfs:label "Employee Level"@en ;
25     qb4o:hasAttribute nw:employeeID ;
26     qb4o:hasAttribute nw:firstName ;
27     qb4o:hasAttribute nw:lastName ;
28     qb4o:hasAttribute nw:title ;
29     ... .
30
31  nw:employeeID a qb4o:LevelAttribute ;
32     rdfs:label "Employee ID"@en ;

```

```

33         rdfs:range xsd:positiveInteger .
34 nw:firstName a qb4o:LevelAttribute ;
35         rdfs:label "First Name"@en ; rdfs:range xsd:string .
36 nw:lastName a qb4o:LevelAttribute ;
37         rdfs:label "Last Name"@en ;
38         rdfs:range xsd:string .
39 nw:title a qb4o:LevelAttribute ;
40         rdfs:label "Title"@en ;
41         rdfs:range xsd:string .
42 ...

```

We can see that, in the recursive hierarchy `nw:supervision`, there is only one level, `nw:employee`, that is also the parent and child level of the hierarchy step `_:supervision_hs1` (the level `All` can be omitted). We can also see some of the dimension level attributes, and their definitions. \square

The translation from an existing data cube (for example, a cube represented in the relational model), can be done in an automatic way, using the R2RML standard.⁹ The study of this mechanism is out of the scope of this paper (see [26] for an implementation).

In the next section we use the Eurostat data cube to illustrate how we can query it using a high-level language, and its automatic translation into SPARQL.

6 Querying QB4OLAP Cubes

The machinery described above can be applied to query data cubes on the SW, following the approach presented in [20], where a clear separation between the conceptual and the logical levels is made, and a high-level language, called Cube Algebra, is defined. Cube Algebra is a user-centric language operating at the conceptual level. This is the reason why the design of QB4OLAP puts emphasis on representing most of the features of OLAP conceptual models. To take advantage of the vocabulary, a subset of Cube Algebra, called QL, was defined, in a way such that the user can write her queries at the conceptual level, and these queries will be automatically translated into a SPARQL query over the QB4OLAP-based RDF representation (at the logical level). There are also a set of rules to ameliorate and simplify QL queries before obtaining an equivalent SPARQL query, which we explain succinctly below.

Remark 1. The content of this section, is based on the work in [27, 28], adapted and simplified for the EBISS 2015 tutorial.

6.1 The QL Language

Ciferri et al. [20] have shown that, opposite to the usual belief, most of the MD data models in the literature operate 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 proposed a model and an algebra where the data

⁹ <http://www.w3.org/TR/r2rml/>.

cube is a first-class citizen, and OLAP operators are used to manipulate the only type of this model: again, the data cube. Following these ideas, Gómez et al. [17] showed that such a model can be used to seamlessly perform OLAP analysis over discrete and continuous geographic data. That means, the user will write the queries in Cube Algebra, without caring about which kind of data lies underneath. The framework takes care of the spatial data management, and of translating the expressions into the language supported by the underlying database (PostGIS in the case of [17]). Along these lines, the use of a high-level query language (as mentioned, called QL), based on the Cube Algebra, for querying cubes represented in RDF following the QB4OLAP model, has been proposed. In this way, the user will only see a collection of dimensions, dimension levels, and measures, and will write the queries in QL, which will then be translated to SPARQL, and executed on the QB4OLAP underlying data cube.

In this section we briefly outline the portion of QL that we will use in the sequel. We remark that we have simplified the language to make the paper easier to read, keeping the most important features, relevant to our main goal, which is, to show how a QB4OLAP cube can be queried without the need of knowing SPARQL programming.

We start the presentation describing the operators, using the Eurostat data cube as our running example.

Operators. The *ROLLUP* operation aggregates measures along a dimension hierarchy to obtain measures at a coarser granularity. The syntax for this operation is:

```
ROLLUP(CubeName, Dimension, Level)
```

where *Level* is the level in *Dimension* to which the aggregation is performed.

Example 8 (ROLLUP). To compute *the total number of applications by country*, we should write

```
ROLLUP(Asylum_applications, Citizenship, Country)
```

The names of the dimensions and levels, are based on the conceptual model in Fig. 1. □

The *DRILLDOWN* operation performs the inverse of *ROLLUP*; that is, it goes from a more general level to a more detailed level down in a hierarchy. The syntax of this operation is as follows:

```
DRILLDOWN(CubeName, Dimension, Level)
```

where *Level* is the level in *Dimension* to which the operation is performed.

Example 9 (DRILLDOWN). After rolling-up to the year level, we may want to drill-down to the month level. For that, we write:

```
DRILLDOWN(YearCube, Time, Month)
```

Note that we assume that we created the cube *YearCube* after rolling-up to year. □

The *SLICE* operation removes a dimension from a cube (i.e., a cube of $n - 1$ dimensions is obtained from a cube with n dimensions) by selecting one instance in a dimension level. The syntax of this operation is:

```
SLICE(CubeName, Dimension)
```

where the *Dimension* will be dropped by fixing a single value in the *Level* instance. The other dimensions remain unchanged.

The *DICE* operation returns a cube with the same dimensionality of the original one, but only containing the cells that satisfy a Boolean condition. The syntax for this operation is

```
DICE(CubeName, Condition)
```

where *Condition* is a Boolean condition over dimension levels, attributes, and measures. The *DICE* operation is analogous to a selection in the relational algebra.

Usually, slicing and dicing operations are applied together.

Example 10 (SLICE and DICE). If in our running example we want to remove the *Time* dimension, we would write:

```
SLICE(Asylum_applications, Time)
```

If we want to keep only applications made by Egyptian citizens, we write:

```
DICE(Asylum_applications, Citizenship.CountryName = 'Egypt')
```

Note that the dicing condition is applied on the value of a level attribute. This is easier than applying a condition over an IRI, illustrating one of the advantages of supporting level attributes in QB4OLAP. \square

We remark that in this paper we limit ourselves to show only the four operations above, since they are enough to illustrate the main idea behind this proposal. A more detailed explanation, and further operations, can be found in [7].

A QL query (or program), is a sequence of OLAP operators, which can store intermediate results in variables bound to cubes, which can be used as arguments in subsequent operations. For example, the following query performs a slicing operation over the *Destination* dimension, an aggregation to the year level in the *Time* dimension, and finally filters the result to obtain only the number of asylum applications submitted by citizens from African countries.

```

1 PREFIX data: <http://eurostat.linked-statistics.org/data/>;
2 PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
3 QUERY
4 $C1 := SLICE(data:migr_asyappctzm, schema:destinationDim);
5 $C2 := ROLLUP($C1, schema:timeDim, schema:year);
6 $C3 := DICE ($C2, (schema:citizenshipDim|schema:continent|schema:continentName = "Africa"));

```

Note that we have included in the language the Turtle prefixes, which, of course do not belong to the conceptual level, but we think this helps, from a pedagogical point of view, to better convey the idea. In a user-oriented implementation these names can be easily hidden, that is, it would be trivial to write *year* instead of *schema:year*.

Finally, to make the presentation simpler, in what follows we assume that QL queries have the following pattern: $(\text{ROLLUP} \mid \text{SLICE} \mid \text{DRILLDOWN})^* (\text{DICE})^*$. That means, DICE operators are the last ones in a query, i.e., no ROLLUP, DRILLDOWN or SLICE operations can follow a DICE one.

6.2 Query Simplification

Automatic query simplification and amelioration is important for two reasons: (a) Users will not always write “good” QL queries: although syntactically correct, redundant and/or unnecessary operations could be included; (b) The order in which the operations are written in a query is not always the best one. Thus, a set of rules simplify and ameliorate the queries proposed by users. The **simplification** process deals with the elimination of redundancy in the queries. The **amelioration** process typically aims at producing a query, equivalent to the original one, but which performs better than it. We briefly explain the simplification process next. To organize the discussion we consider two cases:

- Queries that do not contain DICE operators;
- Queries that contain DICE operators.

Queries Not Including a DICE Operation. In this case, we apply the following rules:

- **Rule 1:** Group all the ROLLUP and DRILLDOWN operations over the same dimension, and replace each group of such operations with a single ROLLUP *from the bottom level* of the dimension to the lowest lever indicated in the drill-down operation(s).
- **Rule 2:** If the query contains a SLICE and a sequence of ROLLUP and DRILLDOWN operations over the same dimension, *remove the sequence* of ROLLUPS and DRILLDOWNS and keep only the SLICE.
- **Rule 3:** Reduce intermediate results by performing SLICE operations as soon as possible.

The rationale of the rules is clear. Rule 1 eliminates the ROLLUPS that will be traversed later down in the hierarchy, when performing the DRILLDOWN. Rule 2 addresses the case in which a SLICE removes a dimension that is traversed using ROLLUPS and DRILLDOWNS. In this case, none of the two latter operations will contribute to the result. Rule 3 reduces the size of the intermediate results as early as possible.

Queries Including a DICE Operation. Taking advantage of the assumption that DICE operators are the last ones in a query, we can split the query in two subsets of statements: one that does not contain DICE operators, and another one that is composed only of DICE operators. We can then apply the rules presented above, to the first portion of the query, keeping the statements that involve DICE operators as in the original query.

6.3 QL by Example

In this section we present some examples of QL queries, and their simplification process.

We start the presentation with a query not containing a DICE operation: *Asylum applications by year and continent where the applicant lives*. This is a typical OLAP query, involving two ROLLUP operations, to the Year and Continent levels in dimensions Time and Citizenship.

Query 1: Asylum applications by year and continent.

```

1 PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
2 PREFIX data: <http://eurostat.linked-statistics.org/data/>;
3 QUERY
4 $C1 := ROLLUP (data:migr_asyappctzm, schema:citizenshipDim,schema:continent);
5 $C2 := ROLLUP ($C1, schema:citizenshipDim,schema:citAll);
6 $C3 := ROLLUP ($C2, schema:timeDim, schema:year);
7 $C4 := SLICE ($C3, schema:destinationDim);
8 $C5 := SLICE ($C4, schema:asyappDim);
9 $C6 := SLICE ($C5, schema:sexDim);
10 $C7 := SLICE ($C6, schema:ageDim);
11 $C8 := DRILLDOWN ($C7, schema:citizenshipDim,schema:continent);

```

Note that this is not the best way of writing this query, since the ROLLUP to All is clearly not needed (recall that we want to promote the analysis within non-expert OLAP users). Thus, applying Rule 1, the sequence of ROLLUPs and DRILLDOWNS over schema:citizenshipDim dimension is replaced by a single ROLLUP from the bottom level up to the level reached by the last operation in the sequence (in this case schema:continent). The simplified query looks as follows.

```

1 PREFIX data: <http://eurostat.linked-statistics.org/data/>;
2 PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
3 QUERY
4 $C1 := SLICE ($data:migr_asyappctzm, schema:destinationDim);
5 $C2 := SLICE ($C1, schema:asyappDim);
6 $C3 := SLICE ($C2, schema:sexDim);
7 $C4 := SLICE ($C3, schema:ageDim);
8 $C5 := ROLLUP ($C4, schema:citizenshipDim,schema:continent);
9 $C6 := ROLLUP ($C5, schema:timeDim, schema:year);

```

Let us now show a query including dicing operations. We want to obtain *Asylum applications by year submitted by Asian citizens, where applications count >5000 whose destination is France or Germany*.

Query 2: Asylum applications by year submitted by Asian citizens, where the number of applications is larger than 5000, and whose destination is France or Germany.

```

1 PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
2 PREFIX data: <http://eurostat.linked-statistics.org/data/>;
3 PREFIX property: <http://eurostat.linked-statistics.org/property#>;
4 PREFIX sdmx-measure: <http://purl.org/linked-data/sdmx/2009/measure#>;
5 QUERY
6 $C1 := ROLLUP (data:migr_asyappctzm, schema:citizenshipDim,schema:citAll);
7 $C2 := ROLLUP ($C1, schema:timeDim, schema:year);
8 $C3 := DRILLDOWN ($C2, schema:citizenshipDim,schema:continent);
9 $C4 := SLICE ($C3, schema:asyappDim);
10 $C5 := SLICE ($C4, schema:sexDim);
11 $C6 := SLICE ($C5, schema:ageDim);

```

```

12 $C7 := DICE ($C6, (schema:citizenshipDim|schema:continent|schema:continentName = "Asia"));
13 $C8 := DICE ($C7, ( sdmx-measure:obsValue > 5000 AND
14 (schema:destinationDim|property:geo|schema:countryName = "France") OR
15 (schema:destinationDim|property:geo|schema:countryName = "Germany")));

```

Below, we show the “simplified” query. Again, the sequence of roll-ups and drill-downs is replaced by a roll-up from the bottom level of the hierarchy.

```

1 PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
2 PREFIX data: <http://eurostat.linked-statistics.org/data/>;
3 PREFIX property: <http://eurostat.linked-statistics.org/property#>;
4 PREFIX sdmx-measure: <http://purl.org/linked-data/sdmx/2009/measure#>;
5 QUERY
6 $C1 := SLICE (data:migr_asyappctzm, schema:asyappDim);
7 $C2 := SLICE ($C1, schema:sexDim);
8 $C3 := SLICE ($C2, schema:ageDim);
9 $C4 := ROLLUP($C3,schema:timeDim,schema:year);
10 $C5 := ROLLUP($C4,schema:citizenshipDim,schema:continent);
11 $C6 := DICE ($C5(schema:citizenshipDim|schema:continent|schema:continentName = "Asia"));
12 $C7 := DICE ($C6, ( sdmx-measure:obsValue > 5000 AND
13 (schema:destinationDim|property:geo|schema:countryName = "France") OR
14 (schema:destinationDim|property:geo|schema:countryName = "Germany")));

```

7 Translating QL Queries into SPARQL

We expressed above that QB4OLAP provides the metadata needed to automatically translate a high-level language into SPARQL. This is a key feature to promote the use of the semantic web: users would not need to learn a new and complex language like SPARQL. In our case, OLAP users will only need to write relatively simple QL programs, and they will have the flexibility to analyze data cubes on-the-fly.

We now describe a mechanism for translating a QL program into a single SPARQL query. Again, we consider two cases: (1) Queries that do not contain DICE operations, and (2) Queries that contain DICE operations. In Sect. 7.1 we describe the generation of SPARQL queries in the first group, while in Sect. 7.2 we present the rules for the second group of queries.

7.1 Queries Not Including a DICE Operation

After applying the rules presented in the previous section to the original query, we reduce all the possible queries to some kind of “normal form”, where, for each dimension D in the data cube only one of the following conditions is satisfied:

- No operation is performed over D
- A ROLLUP operation is performed over D
- A SLICE operation is performed over D

ROLLUPS are implemented navigating the rollup relationships between members, guided by the dimension hierarchy, and aggregations are performed using GROUP BY clauses. The former are performed through SPARQL joins, as we show in the example below. The reader can now better understand why we cannot do this for QB-annotated data sets: they lack the necessary metadata.

SLICES over dimensions correspond to “slicing out” dimensions. This operation requires measure values to be aggregated up to the ALL level of the dimension being sliced out. The mechanism for this is the same one that is used to compute a ROLLUP.

Therefore, after simplifying and ameliorating the query, we can automatically translate it into a single SPARQL expression.

Example 11. We next show the SPARQL query produced for Query 1.

```

1 PREFIX qb: <http://purl.org/linked-data/cube#>
2 PREFIX qb4o: <http://purl.org/qb4olap/cubes#>
3 PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
4 SELECT ?pim1 ?plm2 (SUM(<http://www.w3.org/2001/XMLSchema#integer>(?m1)) as ?ag1)
5 FROM <http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp_clean>
6 FROM <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyappctzmQB40>
7 WHERE {
8   ?o a qb:Observation .
9   ?o qb:dataSet <http://eurostat.linked-statistics.org/data/migr_asyappctzm> .
10  ?o <http://purl.org/linked-data/sdmx/2009/measure#obsValue> ?m1 .
11  ?o <http://purl.org/linked-data/sdmx/2009/dimension#refPeriod> ?lm1 .
12  ?lm1 qb4o:memberOf <http://purl.org/linked-data/sdmx/2009/dimension#refPeriod> .
13  ?lm1 <http://www.w3.org/2004/02/skos/core#broader> ?plm1 .
14  ?plm1 qb4o:memberOf <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#year> .
15  ?o <http://eurostat.linked-statistics.org/property#citizen> ?lm2 .
16  ?lm2 qb4o:memberOf <http://eurostat.linked-statistics.org/property#citizen> .
17  ?lm2 <http://www.w3.org/2004/02/skos/core#broader> ?plm2 .
18  ?plm2 qb4o:memberOf <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#continent> .
19 }
20 GROUP BY ?plm1 ?plm2

```

Note that the SLICE operations are implemented omitting, in the SELECT clause, the variables corresponding to the dropped dimensions. Navigation is performed through joins. Lines 8 through 10 (within the WHERE clause), identify the observations, and Line 11 takes the bottom level of the time dimension, which is used to navigate, through the skos:broader predicate, up to the year level. We proceed analogously with the Citizenship dimension: variable ?lm2 is used to navigate the hierarchy up to the continent level, bound to variable ?plm2. Finally, the GROUP BY clause is applied, and an aggregation using function SUM is performed. □

7.2 Queries Including DICE Operations

In this case, we know that the rules have been applied to the first part of the query, which reduces this part of the query to the cases already described above. The second part of the query contains only DICE operations. Each DICE operation is associated with a condition over measures and/or attribute values, and its result filters out of cells that do not satisfy the condition. We implement the DICE conditions using SPARQL FILTER clauses, also making use of the expressions presented in Sect. 7.1 as subqueries.

The SPARQL query is produced applying the following steps:

1. Obtain a SPARQL query that implements the part of the QL query that does not contain DICE operators, applying the method presented in Sect. 7.1. We will refer to this query as the *inner query*.

2. Produce an *outer SPARQL query* such that:

- (a) Its SELECT clause has the same variables as the SELECT clause of the inner query
- (b) Its WHERE clause contains:
 - i. The inner query
 - ii. A set of graph patterns to obtain the values of the attributes involved in DICE conditions
 - iii. A FILTER clause with the conjunction of the conditions of all the DICE operations

DICE conditions are thus translated into SPARQL expressions. For example, conditions over attributes with range `xsd:string` are implemented using the REGEX function.

Example 12. This example shows the translation of Query 2, which contains a DICE clause. Here, we use the REGEX clause (which handles regular expressions) within the FILTER condition, to obtain the citizens from Asia, and the destination countries.

```

1 PREFIX qb: <http://purl.org/linked-data/cube#>
2 PREFIX qb4o: <http://purl.org/linked-data/cube#>
3 PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
4 SELECT ?ag1 ?plm1 ?lm4 ?plm2
5 WHERE {
6   { ?plm2 <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#continentName> ?plm21 .
7     ?lm4 <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#countryName> ?lm41 .
8   }
9   { SELECT ?plm1 ?lm4 ?plm2
10    (SUM(<http://www.w3.org/2001/XMLSchema#integer>( ?m1)) as ?ag1)
11    FROM <http://www.fing.edu.uy/inco/cubes/instances/migr_asyapp_clean>
12    FROM <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyappctzmQB4O>
13    WHERE {
14      ?o a qb:Observation .
15      ?o qb:dataSet <http://eurostat.linked-statistics.org/data/migr_asyappctzm> .
16      ?o <http://purl.org/linked-data/sdmx/2009/measure#obsValue> ?m1 .
17      ?o <http://eurostat.linked-statistics.org/property#age> ?lm1 .
18      ?o <http://purl.org/linked-data/sdmx/2009/dimension#refPeriod> ?lm2 .
19      ?lm2 qb4o:memberOf <http://purl.org/linked-data/sdmx/2009/dimension#refPeriod> .
20      ?lm2 <http://www.w3.org/2004/02/skos/core#broader> ?plm1 .
21      ?plm1 qb4o:memberOf <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#year> .
22      ?o <http://eurostat.linked-statistics.org/property#sex> ?lm3 .
23      ?o <http://eurostat.linked-statistics.org/property#geo> ?lm4 .
24      ?o <http://eurostat.linked-statistics.org/property#citizen> ?lm5 .
25      ?lm5 qb4o:memberOf <http://eurostat.linked-statistics.org/property#citizen> .
26      ?lm5 <http://www.w3.org/2004/02/skos/core#broader> ?plm2 .
27      ?plm2 qb4o:memberOf <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#continent> .
28      ?o <http://eurostat.linked-statistics.org/property#asyl_app> ?lm6 .
29    }
30    GROUP BY ?plm1 ?lm4 ?plm2
31  }
32  FILTER (((REGEX (?plm21,"Asia" , "i")))&&(((?ag1 > 5000) && ((REGEX (?lm41,"France" , "i")) ||
33    (REGEX (?lm41,"Germany" , "i"))))))

```

Note that the inner and outer queries have the same variables. Also, the outer query contains the FILTER clause, that makes use of the REGEX function. The inner query is solved in the same way as in Example 11. \square

8 Conclusion

In this tutorial paper we have explained how MD data can be represented and queried directly over the SW, without the need of downloading data sets into local DWs. We have shown that, to this end, the RDF Data Cube Vocabulary (QB), the current W3C recommendation must be extended with structural metadata, and dimensional data, in order to be able to support useful OLAP-like analysis. We provided an in-depth comparison between these proposals, and we showed that extending QB with QB4OLAP can be done without re-writing the observations (the largest part of the data). We also presented a high-level query language that allows OLAP users that are not familiar with SW concepts or languages, to write and execute OLAP operators without any knowledge of SPARQL. Queries are automatically translated into SPARQL and executed over an endpoint.

The asylum applications data cube that we have used as running example in this tutorial, as well as an RDF representation of the Northwind DW, and other example cubes, are available at a public SPARQL endpoint.¹⁰ As an exercise, the interested reader can execute the queries presented in this paper, and compare them against the actual Eurostat data, where data are provided in many different ways (reports, graphics, etc.). The analysis allowed by publishing data directly over the SW, using QB4OLAP to represent and enrich data, provides a flexibility that cannot be achieved by traditional publication methods. Moreover, based on the existing observations, expressed in QB, the cost of enriching the original data set is relatively low.

Current work is being carried out along two main lines: (a) Developing further optimization techniques, and providing a benchmark to run queries and study query performance [27, 28]; (b) Enhancing usability, by developing semi-automatic techniques to enrich and build existing QB data sets with QB4OLAP metadata [8, 9].

Acknowledgments. The author is partially funded by the project PICT 2014 - 0787 CC 0800082612, awarded by the Argentinian Scientific Agency.

References

1. Heath, T., Bizer, C.: *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, San Rafael (2011)
2. Klyne, G., Carroll, J.J., McBride, B.: *Resource description framework (RDF): Concepts and abstract syntax* (2004). <http://www.w3.org/TR/rdf-concepts/>
3. Brickley, D., Guha, R., McBride, B.: *RDF vocabulary description language 1.0: RDF schema* (2004). <http://www.w3.org/TR/rdf-schema/>

¹⁰ <https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/>.

4. Abelló, A., Darmont, J., Etcheverry, L., Golfarelli, M., Mazón, J., Naumann, F., Pedersen, T.B., Rizzi, S., Trujillo, J., Vassiliadis, P., Vossen, G.: Fusion cubes: towards self-service business intelligence. *Int. J. Data Warehous. Min.* **9**(2), 66–88 (2013)
5. Cyganiak, R., Reynolds, D.: The RDF Data Cube Vocabulary (W3C Recommendation) (2014). <http://www.w3.org/TR/vocab-data-cube/>
6. Etcheverry, L., Vaisman, A.: QB4OLAP: a vocabulary for OLAP cubes on the semantic web. In: *Proceedings of the 3rd International Workshop on Consuming Linked Data, COLD 2012, Boston, USA. CEUR-WS.org* (2012)
7. Vaisman, A., Zimányi, E.: *Data Warehouse Systems: Design and Implementation*. Springer, Heidelberg (2014)
8. Varga, J., Etcheverry, L., Vaisman, A.A., Romero, O., Pedersen, T.B., Thomsen, C.: Enabling OLAP on statistical linked open data. In: *Proceedings of the 32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland* (2016, to appear)
9. Varga, J., Romero, O., Vaisman, A.A., Etcheverry, L., Pedersen, T.B., Thomsen, C.: Dimensional enrichment of statistical linked open data (2016) (Submitted for publication)
10. Prud'hommeaux, E., Seaborne, A.: SPARQL 1.1 Query Language for RDF (2011). <http://www.w3.org/TR/sparql11-query/>
11. Etcheverry, L., Vaisman, A.A.: Enhancing OLAP analysis with web cubes. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) *ESWC 2012. LNCS, vol. 7295*, pp. 469–483. Springer, Heidelberg (2012)
12. Etcheverry, L., Vaisman, A., Zimányi, E.: Modeling and querying data warehouses on the semantic web using QB4OLAP. In: Bellatreche, L., Mohania, M.K. (eds.) *DaWaK 2014. LNCS, vol. 8646*, pp. 45–56. Springer, Heidelberg (2014)
13. Nebot, V., Llavori, R.B.: Building data warehouses with semantic web data. *Decis. Support Syst.* **52**(4), 853–868 (2012)
14. Kämpgen, B., Harth, A.: Transforming statistical linked data for use in OLAP systems. In: *Proceedings of the 7th International Conference on Semantic Systems, I-Semantics 2011, Graz, Austria*, pp. 33–40 (2011)
15. Löser, A., Hueske, F., Markl, V.: Situational business intelligence. In: Castellanos, M., Dayal, U., Sellis, T. (eds.) *BIRTE 2008. LNBIP, vol. 27*, pp. 1–11. Springer, Heidelberg (2009)
16. Ibragimov, D., Hose, K., Pedersen, T.B., Zimányi, E.: Towards exploratory OLAP over linked open data – a case study. In: Castellanos, M., Dayal, U., Pedersen, T.B., Tatbul, N. (eds.) *BIRTE 2013 and 2014. LNBIP, vol. 206*, pp. 114–132. Springer, Heidelberg (2015)
17. Gómez, L.I., Gómez, S.A., Vaisman, A.A.: A generic data model and query language for spatiotemporal OLAP cube analysis. In: *Proceedings of the 15th International Conference on Extending Database Technology, EDBT 2012*, pp. 300–311. ACM (2012)
18. Hurtado, C.A., Mendelzon, A.O., Vaisman, A.A.: Maintaining data cubes under dimension updates. In: *Proceedings of the 15th International Conference on Data Engineering, ICDE 1999, Sydney, Australia*, pp. 346–355. IEEE Computer Society (1999)
19. Vassiliadis, P.: Modeling multidimensional databases, cubes and cube operations. In: *Proceedings of the 10th International Conference on Scientific and Statistical Database Management, SSDBM 1998, Capri, Italy*, pp. 53–62. IEEE Computer Society (1998)

20. Ciferri, C., Ciferri, R., Gómez, L., Schneider, M., Vaisman, A., Zimányi, E.: Cube algebra: a generic user-centric model and query language for OLAP cubes. *Int. J. Data Warehous. Min.* **9**(2), 39–65 (2013)
21. SDMX: SDMX standards: Information model (2011). http://sdmx.org/wp-content/uploads/2011/08/SDMX_2-1-1_SECTION_2_InformationModel_201108.pdf
22. Beckett, D., Berners-Lee, T.: Turtle - Terse RDF Triple Language (2011). <http://www.w3.org/TeamSubmission/turtle/>
23. Hausenblas, M., Ayers, D., Feigenbaum, L., Heath, T., Halb, W., Raimond, Y.: The Statistical Core Vocabulary (SCOVO) (2011). <http://vocab.deri.ie/scovo>
24. Cyganiak, R., Field, S., Gregory, A., Halb, W., Tennison, J.: Semantic statistics : bringing together SDMX and SCOVO. In: *Proceedings of the WWW2010 Workshop on Linked Data on the Web*, pp. 2–6. CEUR-WS.org (2010)
25. SDMX: Content Oriented Guidelines (2009). <https://sdmx.org/?p=163>
26. Bouza, M., Elliot, B., Etcheverry, L., Vaisman, A.A.: Publishing and querying government multidimensional data using QB4OLAP. In: *Proceedings of the 9th Latin American Web Congress, LA-WEB 2014, Ouro Preto, Minas Gerais, Brazil*, pp. 82–90 (2014)
27. Etcheverry, L., Gómez, S., Vaisman, A.A.: Modeling and querying data cubes on the semantic web (2015). CoRR abs/1512.06080
28. Etcheverry, L., Vaisman, A.A.: Querying semantic web data cubes efficiently (2016) (Submitted for publication)