Dimensional Enrichment of Statistical Linked Open Data

Jovan Varga[a], Alejandro A. Vaisman[b], Oscar Romero[a],
Lorena Etcheverry[c], Torben Bach Pedersen[d], Christian Thomsen[d]

[a] *Universitat Politècnica de Catalunya, BarcelonaTech, Jordi Girona 1-3, Barcelona, Spain*
[b] *Instituto Tecnológico de Buenos Aires, 25 de Mayo 457, Buenos Aires, Argentina*
[c] *Instituto de Computación, Facultad de Ingeniería, UdelaR, Ave Julio Herrera y Reissig 565, Montevideo, Uruguay*
[d] *Aalborg Universitet, Selma Lagerlöfs Vej 300, Aalborg, Denmark*

Abstract

On-Line Analytical Processing (OLAP) is a data analysis technique typically used for local and well-prepared data. However, initiatives like Open Data and Open Government bring new and publicly available data on the web that are to be analyzed in the same way. The use of semantic web technologies for this context is especially encouraged by the Linked Data initiative. There is already a considerable amount of statistical linked open data sets published using the RDF Data Cube Vocabulary (QB) which is designed for these purposes. However, QB lacks some essential schema constructs (e.g., dimension levels) to support OLAP. Thus, the QB4OLAP vocabulary has been proposed to extend QB with the necessary constructs and be fully compliant with OLAP. In this paper, we focus on the enrichment of an existing QB data set with QB4OLAP semantics. We first thoroughly compare the two vocabularies and outline the benefits of QB4OLAP. Then, we propose a series of steps to automate the enrichment of QB data sets with specific QB4OLAP semantics; being the most important, the definition of aggregate functions and the detection of new concepts in the dimension hierarchy construction. The proposed steps are defined to form a semi-automatic enrichment method, which is implemented in a tool that enables the enrichment in an interactive and iterative fashion. The user can enrich the QB data set with QB4OLAP concepts (e.g., full-fledged dimension hierarchies) by choosing among the candidate concepts automatically discovered with the steps proposed. Finally, we conduct experiments with 25 users and use three real-world QB data sets to evaluate our approach. The evaluation demonstrates the feasibility of our approach and shows that, in practice, our tool facilitates, speeds up, and guarantees the correct results of the enrichment process.

Keywords: Linked Open Data, Multidimensional Data Modeling, OLAP, Semantic Web

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 places data in an *n*-dimensional space, usually called a *data cube*. In this way, a user can analyze data along several dimensions of interest. For instance, a user can analyze sales data according to time and location (dimensions). The simplicity of the MD model specially fits the business users who navigate and analyze the MD data by means of OLAP operations (typically via a graphical user interface).

A large number of MD models in the literature are based on the data cube metaphor [1, 2, 3]. 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 Data[1] and Open Government[2] 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* [4] initiative promotes sharing and reusing data on the web using *semantic web* (SW) standards and domain ontologies expressed in the Resource Description

*Corresponding author
Email address: jvarga@essi.upc.edu (Jovan Varga)

[1] http://okfn.org/opendata/
[2] http://opengovdata.org/

Framework (RDF) as the basic data representation layer for the SW [5], or in languages built on top of RDF (e.g., RDF-Schema [6] and OWL[3]).

Two main approaches can be found in the literature concerning OLAP analysis of SW data. The first one consists in extracting MD data from the web and loading them into traditional data management systems for OLAP analysis. The second one explores data models and tools that allow publishing and performing OLAP analysis directly over MD data on the SW. We discuss both approaches in detail in Section 7 and in this paper we follow the second one.

Statistical data sets on the SW are usually published using the *RDF Data Cube Vocabulary*[4] (also denoted QB), the current W3C standard. There is already a considerable amount of data sets published using QB. However, as we explain later, QB lacks (among other shortcomings) the structural metadata needed to automate the translation of OLAP operations into the underlying technology storing the MD data. For example, DWs have been typically implemented using relational technology and the definition of a *well-formed* MD schema allows the automatic translation of OLAP operations into SQL queries. To address this challenge, a new vocabulary, denoted QB4OLAP, has been proposed [7]. QB4OLAP 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. Thus, the main task that we address in this paper is *the enrichment of an existing QB data set with additional QB4OLAP semantics*. Once a data cube is published using QB4OLAP, users will be able to operate over it, not only through queries written in SPARQL [8] (the standard query language for RDF), but also by using a high-level OLAP query language [9]. Such a language allows OLAP users to query data cubes directly on the SW, without any knowledge of SPARQL or RDF, since OLAP queries and operations can be automatically translated into SPARQL, taking advantage of the structural metadata provided by QB4OLAP[5]. In addition, a language like this makes it easier to develop graphic tools, typically used to exploit data cubes.

Enriching an existing QB data set with QB4OLAP semantics implies a labor-intensive, repetitive, and error-prone task. Thus, it must be performed as automatically as possible. For instance, hierarchical structures of the dimensions can be discovered from the source data and metadata, and from external data. Once discovered, the structure must be populated with the members of hierarchy levels. In this paper, we present a method and a tool to facilitate the enrichment process. The method minimizes the user effort, by automatically detecting new potential semantics and performing otherwise time-consuming tasks, leaving to the user the task of providing the MD semantics that cannot be inferred.

Contributions. Our main contributions are:

- An in-depth comparison between the QB and QB4OLAP vocabularies, outlining the novel benefits of the latter.
- Techniques to automate (a) the association between measures and aggregate functions, by means of metadata; and (b) the discovery of dimension hierarchy schema and instances, based on an algorithm that detects implicit MD semantics.
- A method defining the steps described as SPARQL queries to semi-automatically enrich data already published in QB with dimensional (meta)data compliant with the QB4OLAP vocabulary.
- QB2OLAPem, a tool that in an iterative fashion implements the method and the algorithm for the detection of implicit MD semantics. The tool enables the user to semi-automatically produce a QB4OLAP description of a QB data cube with minimal manual effort.
- An evaluation of our approach based on the experiments conducted with 25 user and the use of three real-world QB data sets. The evaluation shows that our approach is feasible and that, in practice, QB2OLAPem reduces the enrichment time and user efforts and guarantees that the QB4OLAP schema created is correct.

The remainder of the paper is organized as follows. Section 2 explains the basic concepts used throughout the paper. Section 3 discusses the limitations of the QB vocabulary, with respect to its capability to represent MD data. This section also presents the QB4OLAP vocabulary, which addresses these limitations. Section 4 studies the automation challenges and provides possible solutions for the two most important ones. Section 5 describes the proposed enrichment method while Section 6 presents the approach evaluation. Finally, Section 7 discusses related work and we conclude in Section 8.

2. Preliminaries

In this section we present the basic concepts on OLAP and SW data models followed by a detailed elaboration on QB based on the running example used throughout the paper.

[3] http://www.w3.org/TR/owl2-overview/

[4] http://www.w3.org/TR/vocab-data-cube/

[5] A prototype is available at http://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/

2.1. OLAP

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. We denote '$\rightarrow$' a partial order on these levels, with a unique bottom, and a unique top, the latter being a distinguished level denoted *All*, whose only member is called *all*. We denote '$\rightarrow^*$' the reflexive and transitive closure of '$\rightarrow$'. Levels can have *attributes* describing them. A *Dimension Instance* assigns a set of dimension members to each dimension level in the dimension schema. 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. In a rollup relationship, l_j is called the *child* level and l_k the *parent* level. In practice, to guarantee the correct aggregation of the measure values, rollup relations actually become functions. *Cardinality constraints* on these relations are then used to restrict the number of level members related to each other [10]. A *Cube Schema* is defined by a set of dimensions and a set of measures, and for each measure a *default aggregate function* is specified. Each dimension is represented by a level, defining the *granularity* of the cube. The cube composed by the bottom levels of each dimension is called a *base cube*. All other cubes are called *cuboids*. A *Cube Instance*, corresponding to a cube schema, is a partial function mapping coordinates from dimension instances (at the cube's granularity level) into measure values.

A well-known set of operations can be defined over cubes [11]. For example, given a cube C, a dimension $D \in C$, dimension levels $l_l, l_u \in D$ such that $l_l \rightarrow^* l_u$, and an aggregate function F_{agg}, $RollUp(C, D, l_u, F_{agg})$ returns a new cube where measure values are aggregated along D, from the current level l_l up to a level l_u, using F_{agg}. Analogously, $DrillDown(C, D, l_l, F_{agg})$ disaggregates previously summarized data, from the current level l_u down to a level l_l and can be considered the inverse of *RollUp*. Note that we do not need to use the starting levels as parameters of these operations, because we assume that they are applied over the 'current' aggregation level of the cube, thus they would be redundant, since the cube 'knows' the current aggregation level for dimension D. $Slice(C, D, F_{agg})$ receives a cube C, a dimension $D \in C$, and an aggregate function F_{agg}, and returns a new cube, with the dimension D removed from the original schema, such that measure values are aggregated along D up to level *All* before removing the dimension, using F_{agg}. Note that in all cases, F_{agg} could be omitted if the default aggregate function is used. Finally, given a cube C, and a first order formula σ over levels and measures in C, $Dice(C, \sigma)$ returns a new cube with the same schema, and whose instances are the instances in C that satisfy σ. For example, given a *Sales* data cube, with dimensions Time and Location, the query "Total sales by region, in December 2015" can be expressed with the following sequence of operations: $C1 := Dice(Sales, Time.month = "12 - 2015")$; $C2 := Rollup(C1, Location, Location.region)$. The first operation takes as input the *Sales* data cube, and produces another data cube $C1$, whose cells contain only sales data corresponding to December, 2015; $C1$ is then summarized by geographical region.

2.2. RDF and the Semantic Web

The basic construct of RDF is a triple, of the (s, p, o) form, where s stands for *subject*, p for *predicate*, and o for *object*. In general, s, p, and o are *resources*, identified with internationalized resource identifiers (IRIs). An object can also be a data value, denoted a *literal* in RDF, or a *blank node*, typically used to represent anonymous resources. Subjects can also be represented by blank nodes. A set of RDF triples is called an *RDF graph*. An *RDF data set* is a collection of RDF graphs, comprising one default RDF graph with no name, and zero or more *named graphs* (a named graph is a graph with a name, typically an IRI or a blank node). Graph names must be unique within an RDF data set.

In addition, the *RDF Schema* (RDF-S) [6] is composed by a set of reserved keywords which define classes, properties, and hierarchical relationships between them. For example, the triple (r, `rdf:type`, c) explicitly states that r is an instance of c, and it also implicitly states that object c is an instance of `rdfs:Class`. Many formats for RDF serialization exist. In this paper we use Turtle [12].

SPARQL 1.1 [8] is the W3C standard query language for RDF. 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. Relevant to OLAP queries, SPARQL 1.1 supports aggregate functions and the `GROUP BY` clause, a feature not present in previous versions.

From here on we assume that the reader is familiar with RDF and SPARQL concepts.

2.3. QB: The RDF Data Cube Vocabulary

As mentioned before, QB is the W3C recommendation to publish statistical data and metadata in RDF. QB is based on the main components of the SDMX information model [13], proposed by the Statistical Data and Metadata eXchange initiative (SDMX)[6] for the publication, exchange, and processing of statistical data. The elements with white background in Figure 1 depict the QB vocabulary. Capitalized terms represent RDF classes and non-capitalized terms represent RDF properties. Capitalized terms in italics represent classes with no instances. An arrow with black triangle head 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. The range of a property can also be denoted using ":". For better comprehension, we next introduce the running example that is used to explain the QB elements and then followed throughout the paper.

We use data published by the World Bank[7], a financial institution supporting developing countries, basically through loans for strategic projects. Free and open access to data about these countries is provided through the World Bank Open Data (WBOD), that includes a collection of indicators[8] measured for different countries and regions across time. Data are available in tabular, RDF, and many other formats depending on the particular portion of the data set. World Bank Linked Data (WBLD) is a Linked Data data set created from WBOD data via its rdf-ization (where needed) and it is annotated with the QB vocabulary. The WBLD is organized in four subsets, stored in different files, including demographic and financial indicators, projects and operations, and climate data. Additionally, there is a VoiD[9] file which contains metadata that describe the data sets. Moreover, a SPARQL endpoint[10] is also available to query the WBLD. Our running example is based on the "Market capitalization of listed companies (current US$)" indicator (CM.MKT.LCAP.CD)[11], where market capitalization refers to the share price times the number of shares outstanding. Each indicator is provided as a QB data set, i.e., as an instance of the class `qb:DataSet`.

The schema of a QB data set is specified by means of the *data structure definition* (DSD), an instance of the class `qb:DataStructureDefinition`. This specification is composed of a set of *component* properties, instances of subclasses of the `qb:ComponentProperty` class, representing *dimensions*, *measures*, and *attributes*. Component properties are not directly related to the DSD: the `qb:ComponentSpecification` class is an intermediate class typically instantiated as RDF blank nodes, that allows specifying additional attributes for a component in a DSD (e.g., a component may be tagged as *required* (i.e., mandatory), using the `qb:componentRequired` property). The different components that belong to a component specification are linked using specific properties that depend on the type of the component: `qb:dimension` for dimensions, `qb:measure` for measures, and `qb:attribute` for attributes. Component specifications are linked to DSDs via the `qb:component` property. Note that a DSD can be shared by different QB data sets (and each QB data set is linked to its DSD) by means of the `qb:structure` property. Example 1 below presents the triples that represent the DSD of our running example.

Example 1. *The DSD of the running example is defined in the file* `meta.rdf`[12] *and looks as follows.*

```
1  @prefix qb: <http://purl.org/linked-data/cube#> .
2  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3  @prefix sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#> .
4  @prefix sdmx-measure: <http://purl.org/linked-data/sdmx/2009/measure#>.
5
6  <http://worldbank.270a.info/dataset/world-bank-indicators/structure>
7    a qb:DataStructureDefinition ;
8    qb:component [
9      a qb:ComponentSpecification ;
10     qb:dimension <http://worldbank.270a.info/property/indicator> ;
11     qb:order "1"^^xsd:int],
12     [
13     a qb:ComponentSpecification ;
14     qb:dimension sdmx-dimension:refArea ;
15     qb:order "2"^^xsd:int],
16     [
17     a qb:ComponentSpecification ;
18     qb:dimension sdmx-dimension:refPeriod ;
19     qb:order "3"^^xsd:int],
20     [
21     a qb:ComponentSpecification ;
22     qb:measure sdmx-measure:obsValue ;
23     qb:order "4"^^xsd:int] .
```

This DSD is composed of three dimensions: `<http://worldbank.270a.info/property/indicator>` *(lines 9-11), representing an indicator,* `sdmx-dimension:refArea` *(lines 13-15) which represents the geographical reference*

[6] http://SDMX.org
[7] http://www.worldbank.org
[8] http://data.worldbank.org/indicator
[9] http://semanticweb.org/wiki/VoID
[10] http://worldbank.270a.info/sparql
[11] http://data.worldbank.org/indicator/CM.MKT.LCAP.CD
[12] http://worldbank.270a.info/data/meta/meta.rdf

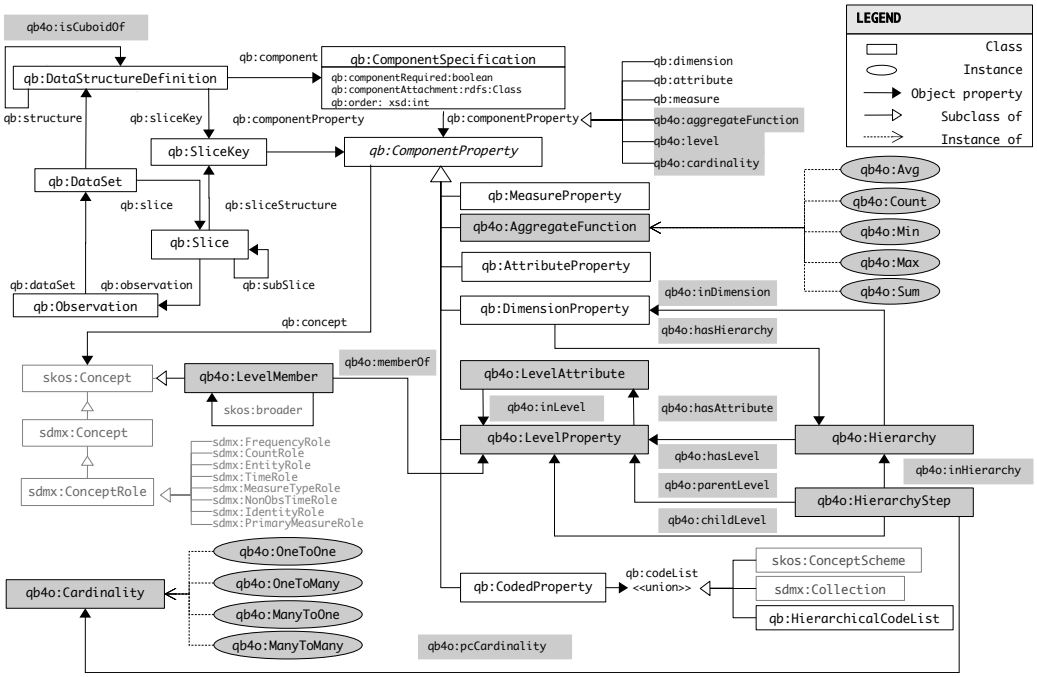

Figure 1: QB (cf. [14]) and QB4OLAP vocabularies

area, and `sdmx-dimension:refPeriod` *(lines 17-19) which represents the time period. The measure of the data set is the generic* `sdmx-measure:obsValue` *predicate (lines 21-23).*

Instances of the running example data set are described in an RDF graph contained in the file `CM.MKT.LCAP.CD.rdf`.[13] Such instances are called *observations* in the QB vocabulary. *Observations* (in OLAP terminology, *facts*) are instances of the `qb:Observation` class and represent points in an MD data space indexed by *dimensions*. They are associated with *data sets* (instances of the `qb:DataSet` class), through the `qb:dataSet` property. Each observation can be linked to a value in each dimension of the DSD via instances of the `qb:DimensionProperty` class; analogously, values for each observation are associated to measures via instances of the `qb:MeasureProperty` class; and instances of the `qb:AttributeProperty` class are used to associate attributes to observations. Example 2 below presents the triples of an observation from our running example.

Example 2. *The triples representing an observation corresponding to the market capitalization for Serbia in 2012 (we do not repeat prefixes previously defined).*

```
1  @prefix property: <http://worldbank.270a.info/property/> .
2  @prefix indicator: <http://worldbank.270a.info/classification/indicator/>.
3  @prefix country: <http://worldbank.270a.info/classification/country/> .
4  @prefix year: <http://reference.data.gov.uk/id/year/> .
5
6  <http://worldbank.270a.info/dataset/world-bank-indicators/
7    CM.MKT.LCAP.CD/RS/2012> a qb:Observation ;
8    qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> ;
9    property:indicator indicator:CM.MKT.LCAP.CD ;
10   sdmx-dimension:refArea country:RS ;
11   sdmx-dimension:refPeriod year:2012 ;
12   sdmx-measure:obsValue 7450560827.04874 .
```

Note that each of the RDF properties defined as components of the data set DSD (Example 1) is used here to link the observation with either dimension members or measure values. In particular, the recorded value for `CM.MKT.LCAP.CD` *indicator is linked to the observation via the* `sdmx-measure:obsValue` *predicate (line 12), and the semantics of this measure is given by the indicator linked to the observation via the* `property:indicator` *predicate (line 9).*

To give further semantics to the components of a DSD, they may be associated with concepts in an ontology. For this, we can make use of the property `qb:concept`, to link components in a DSD, with instances of the class `skos:Concept` defined in the SKOS vocabulary.[14] More specifically, this property can be used to link component properties (i.e., dimensions or

[13] http://worldbank.270a.info/data/world-development-indicators/CM.MKT.TRAD.CD.rdf

[14] http://www.w3.org/TR/skos-reference/

measures), with standard concepts defined in the SDMX guidelines (e.g., reference area, frequency, etc.) [15]. We illustrate this in Example 3 below, to define the dimension `sdmx-dimension:refPeriod`.

Example 3. *An excerpt of the triples that define the dimension `sdmx-dimension:refPeriod`, and associate it with the SDMX concept `sdmx-concept:refPeriod` (a SKOS concept) are shown below.*

```
1  @prefix sdmx−concept: <http://purl.org/linked−data/sdmx/2009/concept#> .
2  @prefix sdmx−dimension: <http://purl.org/linked−data/sdmx/2009/dimension#> .
3  @prefix qb: <http://purl.org/linked−data/cube#> .
4
5  sdmx−dimension:refPeriod a qb:DimensionProperty, rdf:Property ;
6          rdfs:range rdfs:Resource;
7          qb:concept sdmx−concept:refPeriod ;
8          rdfs:label "Reference Period"@en ;
9          rdfs:comment """The period of time or point in time to which the
10         measured observation is intended to refer."""@en .
11
12 sdmx−concept:refPeriod a sdmx:Concept, skos:Concept ;
13         rdfs:label "Reference Period"@en ;
14         rdfs:comment """The period of time or point in time to which the
15         measured observation is intended to refer."""@en;
16         skos:inScheme sdmx−concept:cog .
```

Linking the dimension `sdmx-dimension:refPeriod` with the concept `sdmx-concept:refPeriod` (line 7) allows to give semantics to this dimension.

Finally, *slices*, as defined in QB, represent subsets of observations, not as operators over an existing cube, but as new structures and new instances (observations) in which 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. The class `qb:Slice` allows grouping the observations that correspond to a particular slice (using the `qb:observation` property) and the structure of each slice is attached using the `qb:sliceStructure` property.

3. Representing Multidimensional Data in RDF

Although appropriate to represent and publish statistical data, QB has a set of limitations when it comes to represent an MD model for OLAP. Thus, in this section we elaborate on these limitations of QB, introduce the QB4OLAP vocabulary that extends QB with the necessary concepts, discuss some QB4OLAP design decisions, and provide hints about the use of QB4OLAP.[15]

[15] Parts of the material in this section have previously appeared in [7, 16]. However, the content of [7] have been updated and now refer to newer versions of QB4OLAP. Further, the examples based on WBLD are new. Finally, we remark that [16] is a tutorial on QB4OLAP, produced for the 2015 edition of the Business Intelligence Summer School.

3.1. Limitations of QB

Lack of support for an OLAP dimension structure. Although QB allows representing hierarchical relationships between *level members* in the *dimension instances*, it does not provide a mechanism to represent an OLAP dimension structure (i.e., the dimension levels and the relationships between levels). That means, QB allows stating that Serbia is a narrower concept than Europe, but not that Serbia is a Country, Europe is a Continent, and that countries aggregate to continents. To represent hierarchical relationships between dimension members, the semantic relationship `skos:narrower` should be used, with the following meaning: If two concepts `A` and `B` are such that `A skos:narrower B`, `B` represents a narrower concept than `A` (e.g., `continent skos:narrower country`).

Additional information that can be used to build *dimension instances* is scattered across many graphs. For example, we can obtain information about `country:RS` (Serbia) from the graph in the file `countries.rdf`.[16] Example 4 shows the triples that can be obtained about Serbia.

Example 4. *The triples about the dimension member Serbia, obtained from `countries.rdf`.*

```
1  @prefix skos: <http://www.w3.org/2004/02/skos/core#> .
2  @prefix dbpedia: <http://dbpedia.org/resource/> .
3  @prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
4  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
5  @prefix dcterms: <http://purl.org/dc/elements/1.1/> .
6  @prefix region: <http://worldbank.270a.info/classification/region/> .
7  @prefix income: <http://worldbank.270a.info/classification/income−level/> .
8  @prefix lending: <http://worldbank.270a.info/classification/lending−type/> .
9
10 <http://worldbank.270a.info/classification/country> skos:hasTopConcept
11   country:RS .
12
13 country:RS
14   a skos:Concept, <http://dbpedia.org/ontology/Country> ;
15   skos:inScheme <http://worldbank.270a.info/classification/country> ;
16   skos:topConceptOf <http://worldbank.270a.info/classification/country> ;
17   skos:notation "RS" ;
18   skos:exactMatch country:SRB ;
19   skos:prefLabel "Serbia"@en ;
20   property:region region:ECS ;
21   property:admin−region region:ECA ;
22   property:income−level income:UMC ;
23   property:lending−type lending:IBD ;
24   dbpedia:capital "Belgrade"@en ;
25   geo:lat "20.4656"^^xsd:float ;
26   geo:long "44.8024"^^xsd:float ;
27   foaf:page <http://data.worldbank.org/country/RS> ;
28   ...
29   dcterms:created "2012−02−29T00:00:00Z"^^xsd:dateTime ;
30   dcterms:issued "2013−11−04T13:37:18Z"^^xsd:dateTime .
31
32 country:SRB skos:exactMatch country:RS ; skos:notation "SRB" .
```

[16] http://worldbank.270a.info/data/meta/countries.rdf

Some of the triples provide information that can be used to define dimension hierarchies, when producing the QB4OLAP representation. Line 14 states that `country:RS` *is a country as it says that this IRI is of type* `<http://dbpedia.org/ontology/Country>`*. Lines 20 and 21 state that Serbia belongs to two different regions*[17]*:* `region:ECS` *(Europe & Central Asia (all income levels)) and* `region:ECA` *(Europe & Central Asia (developing only)), respectively. Lines 22 and 23 provide information about the income level and the type of lending Serbia is eligible for, respectively.*[18]

We have said that a typical OLAP user explores data, e.g., performing aggregations along dimension hierarchies. For example, she would like to compute the total capitalization by region, income level, or lending type, given that these data are available in the data set, as Examples 1 through 4 show. However, we can also see that these data are given at the instance level, that means, no hierarchical structure is defined for the refArea dimension. This is because QB does not allow to define aggregation paths. Nevertheless, it is clear that from the information available, we could infer and build a dimension hierarchy. A possible structure for a geographical dimension (like refArea) is shown in Figure 2. We can see that there are five levels, namely country (i.e., intial refArea), region, lending-type, income, and, following traditional MD design, a distinguished level All which has only one level member, denoted all. These levels are organized into three hierarchies: a *geographical* hierarchy country → region → All, a *lending type* hierarchy country → lending-type → All, and an *income* hierarchy country → income → All.

Lack of support for aggregate functions. QB does not provide native support to represent aggregate functions. Many OLAP operations change the granularity level of the data represented in a data cube (e.g., a rollup over the Time dimension from the Month level up to the Year level). This involves aggregating measure values along dimensions, using the aggregate function defined for each measure. These aggregate functions depend on the nature of the measure (i.e., additive, semi additive, non additive [10]). The ability to link each measure with an aggregate function is therefore crucial and, although present in OLAP tools, it is not considered in QB.

[17] http://worldbank.270a.info/classification/region

[18] These concepts are defined in http://worldbank.270a.info/classification/income-level and http://worldbank.270a.info/classification/lending-type.

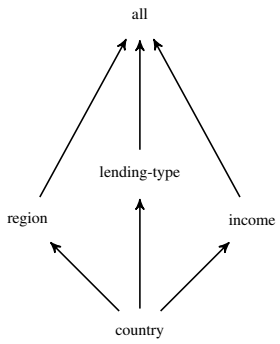

Figure 2: Dimension levels and hierarchies with bottom level country.

Lack of support for descriptive attributes. In the MD model, the instances (members) of each dimension level usually contain a set of real-world concepts with similar characteristics. Further, the schema of each level is composed of a set of *attributes* that describe the characteristics of their members (e.g., the level country may have the attributes countryName, surface, etc.) and one or more *identifiers* [10]. QB does not provide a mechanism to associate a set of attributes with a dimension level. This affects the expressiveness and efficiency of some OLAP operations, in particular, *Dice*, which filters a cube according to a Boolean condition. For example, to obtain a cube containing just data about Serbia, without descriptive attributes we would need to filter such data using the *IRI* representing Serbia, instead of the proper string. This would not only be unnatural for a user, but also highly inefficient. Note that the `qb:AttributeProperty` class, used in QB to associate attributes to observations as mentioned before, differs from descriptive level attributes as defined in QB4OLAP, and cannot be used in the way explained in the example above. Section 3.4 illustrate the use of descriptive attributes in a *Dice* operation.

3.2. The QB4OLAP Vocabulary

QB4OLAP[19] extends QB with a set of RDF terms and 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 [10].
- QB4OLAP must allow to operate over already published observations which conform to DSDs defined in QB, without the need of rewriting the existing observations. Note that in a typical MD model, observations

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

are the largest part of the data while dimensions are usually orders of magnitude smaller.

- QB4OLAP must include all the metadata needed to automatically generate SPARQL queries implementing OLAP operations. 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 can be developed to query RDF data sets directly (we give an example of this in Section 3.4).

The elements with gray background in Figure 1 depict the QB4OLAP vocabulary. Moreover, original QB terms are prefixed with "qb:"; QB4OLAP terms are prefixed with "qb4o:". In addition to the QB graphical notation, QB4OLAP introduces ellipses representing class instances and dashed arrows representing `rdf:type` relationships.

As already mentioned, dimension hierarchies and levels are first-class citizens in an MD model for OLAP. Therefore, QB4OLAP focuses on their representation and several classes and properties are introduced to this end. QB4OLAP represents the structure of a data set in terms of *levels* and measures, instead of *dimensions* and measures (which is the case of QB), thus allowing us to specify the granularity level considered for each dimension. *Dimension levels* are represented in QB4OLAP in the same way that QB represents dimensions: as classes of properties. The class `qb4o:LevelProperty` represents dimension levels. Declaring it as a subclass of `qb:ComponentProperty` allows specifying the schema of the cube in terms of dimension levels, using `qb:DataStructureDefinition`. To represent *aggregate functions* the class `qb4o:AggregateFunction` can be used. 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. Given the structure described above, in QB4OLAP, fact instances (observations) map *level members* to measure values. It is also worth noting that, in general, each fact is related with at most one level member, for each level that participates in the fact. However, there are cases where this restriction does not hold, yielding so-called many-to-many dimensions [10]; thus, to support these dimensions, the property `qb4o:cardinality` can be used to represent the cardinality of the relationship between a fact and a level.

Example 5 shows how the cube in our running example would look like in QB4OLAP (we explain how we came up with this schema in Section 5). Figure 3 presents the definition of the prefixes (not included in the examples so far) that we use in the sequel.

```
@prefix classification: <http://worldbank.270a.info/classification/> .
@prefix dataset: <http://worldbank.270a.info/dataset/> .
@prefix qb4o: <http://purl.org/qb4olap/cubes#> .
@prefix year: <http://reference.data.gov.uk/id/year/> .

#QB4OLAP schema and instances
@prefix schema:
<http://www.fing.edu.uy/inco/cubes/schemas/world−bank−indicators#> .
@prefix instances:
<http://www.fing.edu.uy/inco/cubes/instances/world−bank−indicators#> .
```

Figure 3: RDF prefixes to be used in the examples

Example 5. *The new DSD for the running example data cube defined using QB4OLAP.*

```
schema:QB4O_CM_MKT_LCAP_CD
        a qb:DataStructureDefinition ;
        qb:component [ qb:measure sdmx−measure:obsValue;
                qb4o:aggregateFunction qb4o:sum ] ;
        qb:component [ indicator:CM.MKT.LCAP.CD ] ;
        qb:component [ qb4o:level sdmx−dimension:refArea ] ;
        qb:component [ qb4o:level sdmx−dimension:refPeriod ] .

indicator:CM.MKT.LCAP.CD a qb4o:LevelProperty.
sdmx−dimension:refArea a qb4o:LevelProperty.
sdmx−dimension:refPeriod a qb4o:LevelProperty.
sdmx−measure:obsValue a qb:MeasureProperty.

dataset:CM.MKT.LCAP.CD qb:structure
                schema:QB4O_CM_MKT_LCAP_CD.
```

The DSD is defined in terms of dimension levels such that the dimension properties of the original QB cube are declared as instances of `qb4o:LevelProperty` and considered the lowest levels in the dimension hierarchy. Thus, we avoid rewriting the observations. Readers familiar with OLAP technology may note that `indicator:CM.MKT.LCAP.CD` refers to the MDX's Measures *dimension. MDX is a de facto standard language for OLAP (see [10] for details).*

To represent *dimension hierarchies* the `qb4o:Hierarchy` class is introduced. The relationship between dimensions and hierarchies is represented via the property `qb4o:hasHierarchy` and its inverse `qb4o:inDimension`. To support the most common conceptual models, we need to allow declaring that a level may belong to different hierarchies, and that each level may have a different set of parent levels. Also, the relationship between level members may have different cardinality constraints (e.g., one-to-many, many-to-many, etc.). The class `qb4o:HierarchyStep` allows this by means of the reification of the parent-child relationship between two levels in a hierarchy. Each hierarchy step is linked to its two component levels using the `qb4o:childLevel` and the `qb4o:parentLevel` properties, respectively,

and is attached to the hierarchy it belongs to, using the `qb4o:inHierarchy`. The `qb4o:pcCardinality` property allows representing the cardinality constraints of the relationships between level members in this step, using members of the `qb4o:Cardinality` class, whose instances are depicted in Figure 1. Example 6 illustrates the above.

Example 6. *The definition of the geographical dimension `schema:geoDim` according to Figure 2.*

```
schema:geoDim a qb:DimensionProperty ;
  rdfs:label "Geographical dimension"@en;
  qb4o:hasHierarchy schema:geoHier, schema:lendingHier,
                    schema:incomeHier.
```

We now define each hierarchy, declare to which dimension it belongs, and which levels it traverses. We have three hierarchies in our schema, namely `schema:geoHier`, `schema:lendingHier`, and `schema:incomeHier`. We just show the first of them, the other ones are analogous.

```
schema:geoHier a qb4o:Hierarchy ;
  rdfs:label "Geographical Hierarchy"@en ;
  qb4o:inDimension schema:geoDim;
  qb4o:hasLevel sdmx-dimension:refArea, schema:region, schema:geoAll.
```

Next, we define the base (i.e., finest granularity) level for the geographical dimension, that means, the one whose instances compose the observations, and the upper levels in each hierarchy. Note that the former are defined to be compatible with QB, but as levels instead of dimensions. The example shows only the geographical dimension while construction of other dimensions is analogous where only the All level is added to each of them.

```
# Base levels
sdmx-dimension:refArea a qb4o:LevelProperty;
  rdfs:label "country level"@en.

#Upper hierarchy levels
schema:region a qb4o:LevelProperty;
  rdfs:label "Geographical regions"@en.
schema:lendingtype a qb4o:LevelProperty;
  rdfs:label "Lending type level"@en.
schema:income a qb4o:LevelProperty;
  rdfs:label "Income level"@en.
schema:geoAll a qb4o:LevelProperty;
  rdfs:label "All reference areas"@en.
```

Finally, the hierarchy steps (i.e., parent-child relationships) are defined. Again, we just show the ones corresponding to the `schema:geoHier` hierarchy.

```
_:hs1 a qb4o:HierarchyStep;
  qb4o:inHierarchy schema:geoHier;
  qb4o:childLevel sdmx-dimension:refArea;
  qb4o:parentLevel schema:region;
  qb4o:pcCardinality qb4o:ManyToOne.

_:hs2 a qb4o:HierarchyStep;
  qb4o:inHierarchy schema:geoHier;
  qb4o:childLevel schema:region;
  qb4o:parentLevel schema:geoAll;
  qb4o:pcCardinality qb4o:ManyToOne.
```

To represent *level attributes*, QB4OLAP provides the class of properties `qb4o:LevelAttribute`, linked to `qb4o:LevelProperty` via the `qb4o:hasAttribute` property. Instances of this class are used to link level instances with attribute values. Example 7 shows the definition of an attribute for the `sdmx-dimension:refArea` dimension level.

Example 7. *Definition of a level attribute.*

```
sdmx-dimension:refArea qb4o:hasAttribute
  schema:capital.
schema:capital a qb4o:LevelAttribute;
  rdfs:range xsd:string .
```

We assume that we add the attribute `schema:capital` to the `sdmx-dimension:refArea` dimension level.

At the instance level, *dimension level members* are represented as instances of the class `qb4o:LevelMember`, which is a sub-class of `skos:Concept`. Members are attached to the levels they belong to, using the property `qb4o:memberOf`, which resembles the semantics of `skos:member`. Rollup relationships between members are expressed using the property `skos:broader`, conveying the idea that hierarchies of level members should be navigated from finer granularity concepts up to coarser granularity concepts. Example 8 below shows some examples of dimension members for the dimension `schema:geoDim`.

Example 8. *The details for the dimension members corresponding to Serbia.*

```
country:RS  a  qb4o:LevelMember ;
    qb4o:memberOf sdmx-dimension:refArea ;
    skos:broader  lending:IBD ;
    skos:broader  income:UMC ;
    skos:broader  region:ECS ;
        skos:prefLabel  "Serbia"@en .

lending:IBD  a  qb4o:LevelMember ;
    qb4o:memberOf  schema:lending ;
    skos:broader  instance:geoAll ;
        skos:prefLabel  "IBRD"@en .

income:UMC  a  qb4o:LevelMember ;
        qb4o:memberOf  schema:income ;
        skos:broader  instance:geoAll ;
        skos:prefLabel  "Upper middle income"@en .

region:ECS  a  qb4o:LevelMember ;
    qb4o:memberOf  schema:region ;
    skos:broader  instance:geoAll ;
        skos:prefLabel  "Europe & Central Asia (all income levels)"@en .

instance:geoAll  a  qb4o:LevelMember ;
    qb4o:memberOf  schema:geoAll ;
    skos:prefLabel  "Geo ALL"@en .
```

Note that, for attribute instances, we need to link IRIs corresponding to level members, with attribute values. In our example for the geographical dimension:

```
country:RS schema:capital "Belgrade"^^xsd:string .
```

3.3. Discussion: From QB observations to QB4OLAP

Let us now comment on some decisions underlying the QB4OLAP design. As we have already mentioned, observations in QB are specified as dimension properties, while in QB4OLAP they are specified as level properties, to allow defining hierarchies, as usual in OLAP. Therefore, in order to be able to work with existing QB observations, a new DSD is defined in terms of QB4OLAP *dimension levels*. This saves the cost that would imply adding, for each observation, triples for linking the observation with *level* members using newly defined level properties. We thus propose to define, in the new DSD, a *level property* for each *dimension property* in the existing DSD, and consider the former as the bottom level of each corresponding dimension in the new DSD. Of course, as a consequence, the same elements that in QB are considered dimensions, in QB4OLAP play the role of dimension *levels*, as in the case of `sdmx-dimension:refPeriod` and `sdmx-dimension:refArea`.

Further, instead of defining a new concept in QB4OLAP to represent dimensions, QB4OLAP uses the QB class `qb:DimensionProperty`. This does not produce a semantic contradiction, given that the QB specification states that this class is "The class of component properties which represent the dimensions of the cube", a definition that still holds in the QB4OLAP interpretation. In addition, since level members in QB4OLAP are instances of the class `skos:Concept`, we can use existing QB dimension members to populate QB4OLAP dimension levels. To accomplish this, IRIs that represent dimensions members have to be linked with level members via the `qb4o:memberOf` property.

3.4. Using QB4OLAP

We have mentioned that one of the main advantages of using QB4OLAP instead of QB to represent MD data on the SW, is that QB4OLAP allows us to write high-level OLAP queries and automatically translate them into SPARQL. This way, OLAP users may exploit MD data directly over the web, and query them without any knowledge of SPARQL, or without the need of exporting these data to a relational repository. The obvious consequence is an enhancement of the usability of data published on the web. Giving complete details of how this can be achieved is beyond the scope of this paper, but we would like to at least convey the main idea through an example query.

Let us consider the query "Total market capitalization of listed companies, grouped by income level, in the period [2010,2012]." This is a typical OLAP query involving two main operations, as described in Section 2.1: First, a selection of the values corresponding to the years mentioned in the query (a *Dice* operation). Second, an aggregation, using the SUM aggregate function, along the geographical dimension, using the `schema:incomeHier` up to the `schema:income` level. This can be expressed, in a high-level, cube-based, conceptual algebra like the one proposed in [11] (using the operations defined in Section 2.1), as follows:

```
$C1 := DICE (<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD>,
        (timeDim.refPeriod.yearNumber >= 2010) AND
        (timeDim.refPeriod.yearNumber <= 2012) );
$C2 := ROLLUP ($C1, geoDim, geoDim.income);
```

In the syntax above, we use the notation *dimension.level.attribute*, to represent the dimension's structure. Also, to be concise, we omitted the F_{agg} parameter in the expression for `ROLLUP` (as indicated in Section 2.1), because we assume it is the only one defined for this cube in the DSD. Finally, the variables $C1$ and $C2$ store the results of the operations in the right hand sides of the expressions. With the help of QB4OLAP metadata (which, e.g., describes the hierarchical structure), this query can be automatically translated to the following SPARQL expression:

```
SELECT ?year ?income SUM(xsd:integer(?measureValue)) AS ?sumMeas
FROM <http://www.fing.edu.uy/inco/cubes/schemas/wbld>
FROM <http://www.fing.edu.uy/inco/cubes/instances/wbld>
WHERE {
SERVICE <http://worldbank.270a.info/sparql>
    {?o a qb:Observation ;
        qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD>;
        sdmx−dimension:refArea ?country;
        sdmx−dimension:refPeriod ?year;
        sdmx−measure:obsValue ?measureValue.
        }
    ?country skos:broader ?income.
    ?income qb4o:memberOf schema:income.
    ?year schema:yearNumber ?yearNum
    FILTER (?yearNum >= 2010 && ?yearNum <= 2012 )
}
GROUP BY ?year ?income
```

4. Automating Metadata Definition

Considering that QB4OLAP brings benefits in terms of additional schema constructs that are necessary for state-of-the-art OLAP analysis, we next discuss the possibilities for introducing these enhancements into existing QB data sets. Currently, a considerable number of data sets are published in QB. Thus, in this section we elaborate on how to define and/or discover new concepts

(e.g., dimension levels) that can be used for enriching existing QB data sets. As this can be a very cumbersome, error-prone, and labor-intensive task, we especially focus on its maximal possible automation such that it involves the least possible user intervention. To achieve that, we take advantage of the semantics that is explicitly or implicitly present in the data set or in external data sources. Once discovered, these concepts are used for the enrichment method explained in the next section. In this section, we discuss the automation challenges and how far they can be solved in a semi-automatic way, followed by solutions that we propose for the two most relevant tasks, namely *the definition of aggregate functions* and *the discovery of the dimension hierarchy schema and instances*.

4.1. Automation Challenges

The enrichment tasks needed to turn a QB into a QB4OLAP data set must be done at the *metadata* (e.g., the cube schema concepts) and *data* (e.g., the cube instances) levels. Simply stated, we need to build the hierarchical structure of the dimensions involved (i.e., metadata enrichment) and populate this structure with actual data (i.e., data enrichment). Further, the enrichment includes associating aggregate functions with measures which is also a metadata-related task.

Exploiting the existing QB semantics (i.e., metadata) and the analysis of the data set instances (i.e., data) enable the automatic discovery of potentially new metadata concepts (e.g., new dimension levels). These metadata concepts can be suggested to the user that needs to select the concepts of her interests and, if needed, provide the minimum possible input about missing semantics and specific situations (e.g., data conflicts). The new metadata then support the rest of the enrichment process. For instance, in Example 4 we can see that a `country` is related to a `region` via the `property:region` property. When this property is identified as a parent-child relationship between two levels in a dimension, the rollup instances can be automatically created for all `country` and `region` instances (using the `skos:broader` property).

Performing OLAP analysis directly over SW data in the RDF and Linked Data settings is likely to bring certain challenges. This is due to the fact that, unlike in the traditional DWs settings where data are prepared by a well-defined and complex ETL process, the Linked Data settings do not guarantee clean and formatted data. On the contrary, working in a Linked Data environment typically involves external data sources where it is not rare to find incomplete and imperfect data. These problems directly influence the automation possibilities and user involvement is typically required to fully enrich a data set. The user needs to choose and/or to add semantic information to the data set, or to manage problems that can be present in the data sets. The less these situations occur, the higher level of automation can be achieved and vice-versa. Next, we describe these challenges starting with the semantic-related challenges – partial and imperfect semantics – and then we address data-related challenges – partial and imperfect data.

Partial semantics. This challenge arises when the data set does not contain enough schema information for the enrichment tasks. For instance, a data set may lack information about the aggregate functions that can be applied over measures or the semantics for building the dimension hierarchies.

This challenge is the most relevant for our approach and needs to be tackled. The problem can be addressed either by enrichment from external sources, or by manual user intervention. In the next subsections we discuss two possible approaches for defining the additional semantics needed for enrichment tasks.

Imperfect semantics. This challenge comprises different cases that occur when semantics obstructs automation. For instance, when the same semantics is represented with different concepts, the same concepts have different semantics, and other conflicts that may arise in schema comparison [17]. This is called *semantic heterogeneity* [18]. For example, we may have different currencies, different measure units, etc. As other cases of imperfect semantics, we can mention incorrectly defined semantics (e.g., when a continent is located in a city), and outliers, i.e., unexpected semantic concepts that cannot be aligned with the rest (e.g., cantons as a geographical concept that is not used in most of the cases).

This problem should be addressed by detecting the potential deviations in semantics (e.g., the same concept playing different roles) and enabling the user to address these cases. These situations are typically solved in the data set cleaning and transformation stages [17] while our approach focuses on semantic enrichment.

Partial data. Partial data may affect aggregation, a key task in OLAP analysis. Missing data may raise many challenges. For example, aggregating data to the continent level depends on the availability of data about all the belonging countries.

To tackle this problem, missing data can possibly be imported from external sources. However, since it is often the case that the data set in hand contains the only

available data (e.g., only some countries in the EU are covered), we focus on constructing the cube on top of the *available pieces of data*. In the case that it is possible to detect the problems caused by missing data, the user should be notified (e.g., marking the aggregated values as incomplete/partial in case of missing values).

Imperfect data. Many different cases can illustrate this problem, where data instances cause difficulties to achieve automation. One of these cases is *data heterogeneity* where not all data are well formatted or do not satisfy explicit or implicit constraints. For example, in RDF it is impossible to impose data instances to satisfy MD integrity constraints (see Section 4.3). Imperfect data may also include data errors and data instances that do not satisfy constraints but still represent correct information.

This challenge generates a wide spectrum of cases and we focus on *cardinality detection* based on data analysis. Other cases, like the detection of the data instances that do not match their type(s) (e.g., instead of an expected integer we find a string), out-of-range values, and similar cases should be detected and reported to the user. Then, the user can discard or, if possible, fix these cases. In this context, an extensive overview of methodologies for data quality assessment and improvement can be found in [19].

4.2. Associating measures with aggregate functions

To enable automatic navigation along dimension hierarchies, each measure in the data cube needs to have an associated aggregate function. Since QB does not allow to provide this information, the QB data set must be enriched with a mapping of measures to aggregate functions. We call this mapping *MAggMap*. Not every aggregate function can be applied to a measure and give a valid result. Defining the appropriate aggregate function depending on the measure type is a well-known problem in the literature related to the summarizability problem in OLAP and statistical databases [20]. In that context, measure types are flow (e.g., monthly sales value), stock (e.g., inventory of a product), and value-per-unit (e.g., product item price). For instance, while it makes sense to compute the sum of the monthly sales by year, it does not make sense to sum a product's unit price over time. This summarizability condition is called *type compatibility*, i.e., the compatibility between the measure type (i.e., its semantics), the measure category (i.e., temporal or non-temporal), and the aggregate function's type. This condition, together with *disjointness* and *completeness* (see next section), are necessary to guarantee correct data summarization [20].

The large variety of measure and aggregate function types makes the compatibility check a tedious task that can hardly be fully automated. Even in such a case, the user would still need to choose among different options. Therefore, the user must be involved in this process. However, this involvement can be guided and semi-automated based on the compatibility definition presented in [20]. We address the interested reader to [21] for further details.

In this paper, we assume that the user explicitly provides the *MAggMap* mapping, which is a needed input for our enrichment tasks (see Section 5). We define it as [measure IRI, aggregate function IRI] pairs. For example, [`sdmx-measure:obsValue`, `qb4o:Sum`]. In our current implementation (see Section 6), we suggest a default aggregate function (e.g., sum) but it can be changed by the user.

4.3. Discovering dimensional data

Dimensional concepts consist of dimensions, hierarchies, levels, and level attributes. Briefly, dimensions contain different levels of aggregation (i.e., dimension levels), which are organized in hierarchies (in short, each hierarchy correspond to a path of rollup relationships) and may contain attributes (see Section 2.1). Enriching a QB data set with dimensional data implies properly identifying all these constructs and annotating them according to QB4OLAP. Current OLAP state-of-the-art identifies dimensional concepts from functional dependencies (FDs) [22]. Arranging the dimensional concepts according to FDs guarantee the summarizability disjointness and completeness and these are necessary conditions to guarantee the summarizability correctness. Accordingly, FDs must be guaranteed between facts and dimensions (e.g., between market capitalization and geographical dimension) and between the levels forming dimension hierarchies (e.g., between the `country` and `region` levels). [23] discusses the role of FDs for automatic MD modeling and how to discover them for Description Logics (DL). To discover FDs, the most widespread technique consists of sampling data to identify functional properties that fulfill the underlying many-to-one[20] cardinality of the relationship. Briefly, many-to-one (i.e., m:1) cardinalities require that every child level instance is related to one parent level instance (e.g., `Serbia` is related to the `Europe & Central Asia (ECS)` region), while each parent level instance can be related to one or more child

[20] Note that "many" stands for "one or more instances".

level instances and these sets (e.g., countries and regions) do not mutually overlap [24]. These guarantee completeness and disjointness. A comprehensive and detailed overview of the summarizability challenges in MD modeling is presented in [24]. Dimension hierarchies whose properties satisfy many-to-one cardinalities guarantee a correct summarizability as the aggregate values at parent levels (e.g., region) include all related child level instances (e.g., countries) and no parent level instance is without child level instance(s).[21] Furthermore, there is no double-counting of child level instances at the parent levels. Many-to-one cardinalities enable the automation of the MD design [25] where the potential new levels can be discovered by detecting these cases in data instances.

In some expressive languages, such as the OWL 2 RL profile[22] based on DL-Lite [26], it is possible to state that a property is functional. However, most available RDF data sets omit such definitions. Therefore, in the spirit of [25], we analyze the instances to identify FDs from data. To avoid the inherent computational complexity discussed in [27], we benefit from the QB semantics by considering the QB dimensions as the initial set of dimension levels from which start building richer dimensional structures. Thus, the QB dimensions serve as the starting point from where to discover possible new dimension levels, hierarchies, and level attributes based on FDs. Given the relevance of this step for MD modeling, we provide an algorithm for the detection of implicit FDs by discovering functional properties (i.e., satisfying a many-to-one cardinality) for dimension levels. Moreover, one-to-one (i.e., 1:1) cardinalities are identified to detect potential dimension level attributes. We only consider linear hierarchies since, in practice, complex hierarchies (see [28] for more details) with many-to-many cardinalities are typically transformed to linear hierarchies with many-to-one cardinalities. The pseudo code is presented in Algorithm 1. The algorithm runs over an implicit RDF graph.

The algorithm starts from the set of original QB data set *dimensions* (*L*), which from now on we call *initial levels*. Moreover, it also takes the *minCompl* and *minCard* parameters, which are later discussed in Algorithm 2. The output of the algorithm are the sets of all levels (*allL*), rollup properties (*allP*), all hierarchy steps (*allHS*), and all [level, level attribute] pairs (*allLLA*)

Algorithm 1: Detect implicit MD semantics

```
Input: L, minCompl, minCard;     // initial levels set (i.e.,
       former QB dimensions), minimum completeness, and
       minimum cardinality parameters, respectively
Output: allL, allP, allHS, allLLA;          // all levels, all
        properties, all hierarchy steps, and all
        level-level attribute pairs, respectively
1  begin
2      allL = L;
3      allP = ∅;
4      allHS = ∅;
5      allLLA = ∅;
6      foreach level ∈ allL following a bottom-up order do
7          foreach property ∈ getProperties(level) do
8              if getCardinality(level, property, minCompl, minCard)
               = m : 1 then
9                  parentLevel = getObjectElement(level, property);
10                 if noCycles(level, parentLevel) then
11                     allL ∪= parentLevel;
12                     allP ∪= property;
13                     allHS ∪= (level, property, parentLevel);
14             else if
               getCardinality(level, property, minCompl, minCard) =
               1 : 1 then
15                 levelAttribute =
                   getObjectElement(level, property);
16                 allLLA ∪= (level, levelAttribute);
```

available in the input QB data set. The set of all levels is initially populated with the initial levels set (line 2), while the other sets are initially empty (see lines 3 to 5). For each level (line 6), e.g., the `country` level, we check all of its properties (line 7), e.g., the region property, and infer their cardinalities. We iterate over the levels by following a bottom–up approach; i.e., we start from the finer (e.g., the `country` level) and later visit coarser granularity levels (e.g., the `region` level). Details on how to retrieve the property cardinality are shown in Algorithm 2. If a property yields a *many-to-one* cardinality (line 8) its object (i.e., the RDF property range) is considered as a potential coarser granularity level to rollup to. Therefore, a potential new *parent* level is retrieved in line 9. Importantly, to guarantee the MD integrity constraints, before adding this new parent level to the set of all levels, we check that this addition does not produce cycles (line 10), i.e., that the current level cannot be reached from the newly identified parent level (e.g., that there is no direct or indirect rollup relationship from `region` to `country`). If there are no cycles, we add the new parent level to the set of all levels (line 11). Then, in lines 12 and 13, the property and the hierarchy step triples are added to the corresponding sets. Otherwise, if the property cardinality is one-to-one (line 14), the new concept is considered as a level attribute (e.g., `label`), and it is added to the set of [level, level attribute] pairs (lines 15 and 16). The output sets

[21] Note that we do not consider the special case of non-covering dimensions where there could exist parent level instances with no child level instances.

[22] https://www.w3.org/TR/2008/WD-owl2-profiles-20081008/#OWL_2_RL

of Algorithm 1 are later consumed as inputs in our enrichment tasks (see Section 5).

Algorithm 2: Get cardinality for a property

```
   Input: l, p, minCompl, minCard;    // level, property, minimum
          completeness, and minimum cardinality parameters
   Output: cardinality;    // cardinality of the property p for
           the level l
1  begin
2      to − oneFromChild = 0;      // number of to-one property
       instances from the child side
3      to − oneFromParent = 0;     // number of to-one property
       instances from the parent side
4      to − manyFromParentSet = ∅;      // set of parent level
       instances for to-many property instances from the
       parent side
5      foreach li ∈ getInstances(l);        // li - level instance
6      do
7          if countSubjectPropertyInstances(li, p) = 1 then
8              to − oneFromChild ++;
9              parentLI = getObjectElement(li, p);
10             if countObjectPropertyInstances(parentLI, p) = 1 then
11                 to − oneFromParent ++;
12             else if countObjectPropertyInstances(parentLI, p) > 1
               then
13                 to − manyFromParentSet ∪= parentLI ;
14     if to − oneFromChild ≥ getInstanceNumber(l) * minCompl then
15         if to − oneFromChild / minCard ≥
           to − manyFromParentSet.Size() then
16             cardinality = m : 1
17         else if to − oneFromParent ≥ getInstanceNumber(l) *
           minCompl then
18             cardinality = 1 : 1
19     else
20         cardinality = m : m;      // other cardinality value
```

Algorithm 2 determines a property cardinality using simple SPARQL queries to retrieve the number of property instances related to a subject (line 7) or an object (lines 10 and 12). This algorithm takes as input the level (e.g., the `country` level) and the property (e.g., the region property) for which it must retrieve the cardinality. Moreover, it also needs the minimum completeness and disjointness *minCompl* and the minimum cardinality *minCard* parameters as inputs. The former defines the minimal required percentage of *to − one* relationships for the total number of level instances, e.g., a value 0.90 means that at least 90% of countries need to have one and only one region property. This way, there might be some level instances that have none or more than one property instances. Although non-complete / non-disjoint properties stand against the conditions discussed earlier in the present section, this is needed to identify conceptual FDs that, due to imperfect and /or partial data, do not hold for all the data. Following the idea presented in [29], by means of these two parameters we identify *quasi-FDs* (which is often the case in Linked Data and RDF data sets). We say that a property is a *quasi-FD* if most of the data satisfy the FD (e.g., 98% of level instances are associated to exactly one property instance). The second parameter, *minCard*, defines the minimum average number of child level instances per parent level instance (e.g., *minCard* = 5 meaning at least 5 countries per region). The values for these parameters should be empirically defined depending on the domain and data set quality (see Section 6).

The algorithm proceeds as follows. The local variable *to − oneFromChild* (line 2) holds the number of *to − one* properties from child to parent instances (e.g., the number of cases where there is only one region property per `country`); analogously, *to − oneFromParent* (line 3) holds the number of *to − one* properties from parent to child instances (e.g., the number of cases where for a `region` instance there is only one region property from a `country` instance to that `region` instance), and *to − manyFromParentSet* (line 4) holds the set of parent instances that are in *to − many* relationships (e.g., the `region` instances that are related to more than one `country` via the region property). For all instances of a given level (line 6), e.g., all `country` instances, we count the ones that have only one instance of a given property (line 8), e.g., the region property. In this case, the algorithm retrieves the level instance on the other side of the property (e.g., the `region` instance) and checks its cardinality (lines 10 and 12). Note that this check differs from the first one (line 7), since here the level instance (e.g., the `region` instance) is used as a property object while in the first one, the input level instance is used as a subject. In case of *to − one* property instances (lines 10 and 11), we count them; in case of *to − many* instances, we add them to the set (lines 12 and 13) so that we can count them at the end (line 15), since the property instances will be repeated for child instances with the same parent instance (e.g., several `country` instances are related to the same `region` instance). Finally, we determine the cardinality in lines 14 – 20.

We next show how Algorithm 2 can be implemented with the following SPARQL queries. We consider that the queries use an RDF graph that contains a QB4OLAP level (e.g., `?levelIRI? a qb4o:LevelProperty`) and a set of QB4OLAP level members (e.g., `levelMemberIRI1 a qb4o:LevelMember`) belonging to this level (i.e., `levelMemberIRI1 qb4o:memberOf ?levelIRI?`). Furthermore, we use the following parameter values *minCompl* = 100 and *minCard* = 2. Hence, all properties for the level members can be retrieved with Query 1. The query takes the graph and level IRIs as parameters. Note that the prefixes used in

queries are following:

```
prefix qb: <http://purl.org/linked-data/cube#>
prefix qb4o: <http://purl.org/qb4olap/cubes#>
```

Query 1. *Get properties for level members.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI, and
# ?levelIRI? – the level IRI

SELECT DISTINCT ?p
FROM ?qb4oGraphIRI?
WHERE {
 ?levelMember ?p ?o .
{ SELECT DISTINCT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } } }
```

For a chosen property, we first need to check if it is a *to − one* property, i.e., each level member is related to one and only one instance of the property. Query 2 performs this check. In addition to the previous ones, the query also takes the property IRI as parameter.

Query 2. *Check if the property is to-one.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?levelIRI? – the level IRI, and
# ?propertyIRI? – the property IRI

ASK { {
SELECT (COUNT (?levelMember) AS ?lmWithUniqueObject )
FROM ?qb4oGraphIRI?
WHERE { { #get #unique object per level member for the input property
 SELECT ?levelMember (COUNT (DISTINCT ?obj) AS ?uniqueObjNum)
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember ?propertyIRI? ?obj .
  {SELECT ?levelMember
  FROM ?qb4oGraphIRI?
  WHERE {
   ?levelMember a qb4o:LevelMember .
   ?levelMember qb4o:memberOf ?levelIRI? .}}
 } GROUP BY ?levelMember }
 FILTER ( ?uniqueObjNum = 1) } }
 { #get the total #level members for a level
 SELECT (COUNT (DISTINCT ?lm) AS ?totalLevelMemberNumber)
 FROM ?qb4oGraphIRI?
 WHERE {
  ?lm a qb4o:LevelMember .
  ?lm qb4o:memberOf ?levelIRI? . } }
 FILTER (?lmWithUniqueObject = ?totalLevelMemberNumber) }
```

If Query 2 returns true, the property is *to − one* and we can check if it is 1:1 or m:1 with Queries 3 and 4, respectively. The parameters are the same as for the previous query.

Query 3. *Check if the property is 1:1.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?levelIRI? – the level IRI, and
# ?propertyIRI? – the property IRI

ASK { { #get the #object per level member for the input property
SELECT (COUNT (DISTINCT ?levelMember) AS ?totalLevelMemberNumber)
(COUNT (DISTINCT ?obj) AS ?uniqueObjNum)
FROM ?qb4oGraphIRI?
WHERE {
 {SELECT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } }
 ?levelMember ?propertyIRI? ?obj . } }
 FILTER (?uniqueObjNum = ?totalLevelMemberNumber) }
```

Query 4. *Check if the property is m:1.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?levelIRI? – the level IRI, and
# ?propertyIRI? – the property IRI

ASK { { #get the #object per level member for the input property
SELECT (COUNT (DISTINCT ?levelMember) AS ?totalLevelMemberNumber)
(COUNT (DISTINCT ?obj) AS ?uniqueObjNum)
FROM ?qb4oGraphIRI?
WHERE {
 {SELECT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE {
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } }
 ?levelMember ?propertyIRI? ?obj . } }
FILTER (?uniqueObjNum < ?totalLevelMemberNumber/2)
{ #check that objects are not literals
SELECT (COUNT (DISTINCT ?obj2) AS ?notLiteralObj)
FROM ?qb4oGraphIRI?
WHERE {
 {SELECT ?lm
 FROM ?qb4oGraphIRI?
 WHERE {
  ?lm a qb4o:LevelMember .
  ?lm qb4o:memberOf ?levelIRI? . } }
  ?lm ?propertyIRI? ?obj2 .
 FILTER isIRI(?obj2) } }
FILTER (?uniqueObjNum = ?notLiteralObj) }
```

Our algorithms consider settings where the input QB data set contains implicit MD semantics, i.e., where the levels have properties that link them with coarser granularity levels inside the data set. If this is not the case, we can use existing IRIs or look for external IRIs (e.g., the IRI for Serbia on DBpedia[23]) to search for the necessary semantics from external data sets. If this is not possible, the user should define these IRIs manually. Further, we assume that in the input QB data set, all observations are at the same level of granularity for each dimension which is the case most of the time. Then, on top of these levels we build new dimension hierarchies. Special situations, where there might exist observations at different granularities, must be treated manually in a data preparation step. This situation can be detected with Algorithm 1 if it identifies a rollup property between instances of an initial level.

[23] http://dbpedia.org

5. Enrichment Method

Taking advantage of the QB4OLAP vocabulary and the algorithms introduced in Section 4.3, we now propose a method to enrich an input QB graph[24] with additional MD semantics. This method presents a set of detailed enrichment steps. For the sake of comprehension, each step is described as a SPARQL query showing the precise enrichment and transformations. The queries take the specified parameters, use an input QB graph and incrementally create the new QB4OLAP graph by generating the necessary triples. Since this method requires some user actions, the overall enrichment process is semi-automatized. The method consists of two phases:

1. *Redefinition phase* which syntactically transforms the input QB graph into QB4OLAP constructs and, given the required input (see Section 4.2), specifies aggregate functions for measures.
2. *Enrichment phase* which, given a set of required inputs (see Section 4.3), enriches the QB4OLAP graph generated by the redefinition phase with additional MD semantics.

For the ease of understanding, this section introduces the main ideas for the enrichment tasks to be accomplished. In addition, Appendix A provides a fully formalized, more general, and detailed enrichment methodology, which is agnostic of the implementation decisions made and further specifies the pre-conditions, post-conditions, and transformations to be conducted by each step in terms of set theory. Thus, the method presented in this section can be considered a possible solution to cover the steps defined by the methodology. In this section, we first introduce some preliminaries for understanding the method. Next, each phase is defined in terms of queries to be performed that taking the input parameters produce the output triples. Finally, we provide some additional considerations.

5.1. Method Preliminaries

The method uses two RDF graphs, namely the *QB graph* (i.e., the set of triples defining the QB cube structure and instances) and the *QB4OLAP graph* (analogous to the QB graph definition). These graphs are assumed to be compliant with the QB and QB4OLAP vocabularies, respectively. According to the QB and QB4OLAP definitions, we further identify two sets of RDF triples in each graph: the set of triples describing the QB or QB4OLAP *cube schema* and the set describing the *cube instances*.

According to the QB definition (see Section 2), the *QB cube schema* consists of the triples involving the following classes and related properties: the QB *dataset*[25] (i.e., `qb:DataSet`), *structure* (i.e., `qb:DataStructureDefinition`, *dimensions* (i.e., `qb:DimensionProperty`), and *measures* (i.e., `qb:MeasureProperty`). Following QB's notation, the cube structure is defined as a set of dimensions and measures via the cube components (i.e., `qb:ComponentSpecification`). An example of QB cube schema extracted from our running example (see Section 2.3) is presented in Example 9.

Example 9. *QB cube schema triples.*

```
1 <http://worldbank.270a.info/dataset/world−bank−indicators/structure>
2   a qb:DataStructureDefinition ;
3   qb:component  [ qb:dimension sdmx−dimension:refArea ] ;
4   qb:component [ qb:measure sdmx−measure:obsValue ] .
5 sdmx−dimension:refArea a qb:DimensionProperty .
6 sdmx−measure:obsValue a qb:MeasureProperty .
7 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet ;
8   qb:structure
9     <http://worldbank.270a.info/dataset/world−bank−indicators/structure> .
```

Lines 1 – 4 relate to the QB cube structure, line 5 to dimensions, line 6 to measures, and line 7 to the dataset. The dataset is related to the cube structure in lines 8 – 9.

QB cube instances contain triples related to the QB *dimension instances* (extracted from the observations with Query 9 as explained later) and *observations* (i.e., `qb:Observation`). As discussed before, observations represent measure values for the fixed dimension instances determined by the cube structure. An example of QB cube instances is presented in Example 10.

Example 10. *QB cube instance triples.*

```
1 data:world−bank−indicators/CM.MKT.LCAP.CD/RS/2012
2   a qb:Observation ;
3   sdmx−dimension:refArea country:RS ;
4   sdmx−measure:obsValue 7450560827.04874 ;
5   qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> .
```

Line 1 – 2 define an observation. Line 3 specifies a dimension instance and line 4 defines a measure value of the observation. The observation relates to the cube schema structure indirectly (see lines 7 – 9 of Example 9) via `qb:dataSet` in line 5.

[24] For simplicity of presentation, we assume that all triples related to the input QB data set are in a single RDF graph.

[25] Note that in the present section the term "dataset" refers to qb:DataSet.

Analogously, we next define the QB4OLAP cube schema and instances. The *QB4OLAP cube schema* consists of the triples involving the following classes and related properties:

- *dataset* (i.e., `qb:DataSet`),
- *structure* (i.e., `qb:DataStructureDefinition`),
- *dimensions* (i.e., `qb:DimensionProperty`),
- *measures* (i.e., `qb:MeasureProperty`),
- dimension *levels* (i.e., `qb4o:LevelProperty`),
- dimension *level attributes* (i.e., `qb4o:LevelAttribute`),
- dimension *hierarchies* (i.e., `qb4o:Hierarchy`),
- *hierarchy steps* (i.e., `qb4o:HierarchyStep`),
- predefined set of *aggregate functions* (i.e., `qb4o:AggregateFunction`), and
- predefined set of possible *cardinalities* (i.e., `qb4o:Cardinality`).

The *QB4OLAP cube instances* contain the triples related to the QB4OLAP cube *level instances*, *rollup relationships* between child and parent level instances (represented with `skos:broader`), *observations* (i.e., `qb:Observation`), and *level attribute values* (being literals or IRIs).

Examples of the QB4OLAP cube schema and instances are presented below in the method definition. The examples are based on the running example (see Section 2.3). We consider the scenario where the input graph contains implicit MD semantics (e.g., a country is linked to a region but this is not explicitly stated as a rollup relationship since this cannot be described in QB). Other scenarios are discussed in Section 5.4. For the sake of simplicity, we define the steps as SPARQL INSERT queries assuming that from the original input QB graph we build a *new* QB4OLAP graph (which is implicitly created with the first SPARQL INSERT query). Note that the elements between two '?' in the queries represent parameters that should be replaced with the IRI values specified at each step. Moreover, all the examples of the query results follow up on one another. In addition to the prefixes introduced in the previous section, the queries also use the following prefix:

```
prefix skos: <http://www.w3.org/2004/02/skos/core#>
```

5.2. Redefinition Phase

Redefinition of a cube schema. We start by building the new QB4OLAP cube schema. We proceed incrementally and first we perform a syntactic transformation from QB to QB4OLAP constructs, while the complete QB4OLAP cube schema is formed after the Enrichment phase (see Section 5.3). The QB cube schema triples defining the cube dataset, structure, dimension levels, and measures are added to the QB4OLAP cube schema in the following way. First, dimensions are redefined as levels in the QB4OLAP cube schema. Next, we copy the measures from the QB graph to the QB4OLAP one. Then, we define the new cube schema structure, assign to it both levels and measures, and add it to the QB4OLAP cube structure. Finally, we copy the dataset definition to the QB4OLAP cube schema and assign the new cube schema structure to it. This transformation can be performed with Query 5. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the QB dataset IRI, and the new QB4OLAP structure IRI.

Query 5. *Redefinition of a cube schema.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
#?qbGraphIRI? – the input QB graph IRI,
#?dsIRI? – the data set IRI, and
#?dsdIRI? – the data structure definition IRI

INSERT INTO ?qb4oGraphIRI? {
?dsIRI? a qb:DataSet .
?dsIRI? qb:structure ?dsdIRI? .
?dsdIRI? a qb:DataStructureDefinition .
?dsdIRI? qb:component ?bl . ?bl qb4o:level ?d .
?dsdIRI? qb:component ?bm . ?bm qb:measure ?m .
?d a qb4o:LevelProperty .
?m a qb:MeasureProperty . }
FROM ?qbGraphIRI?
WHERE {
?dsd a qb:DataStructureDefinition .
?dsIRI? qb:structure ?dsd .
?dsd qb:component  ?bl . ?bl qb:dimension ?d .
?dsd qb:component  ?bm . ?bm qb:measure ?m . }
```

Thus, at this point, we have obtained the initial QB4OLAP cube schema. An example of a QB4OLAP cube schema is shown in Example 11 which illustrates the result of Query 5. Note that we use the `newG` namespace for the new QB4OLAP graph.

Example 11. *Resulting triples of Query 5.*

```
<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
                                qb:structure newG:newDSD .
newG:newDSD a qb:DataStructureDefinition ;
  qb:component  [ qb4o:level sdmx−dimension:refArea ] ;
  qb:component  [ qb:measure sdmx−measure:obsValue ] .
sdmx−dimension:refArea a qb4o:LevelProperty .
sdmx−measure:obsValue a qb:MeasureProperty .
```

Lines 1 and 2 illustrate the triples related to the cube dataset. Results in lines 3, 4, and 5 define the new cube schema structure and add a level and a measure as components to it. Line 6 redefines the dimension from Example 9 as a QB4OLAP level, while line 7 illustrates the measure from Example 9 copied to the new QB4OLAP graph.

Specification of an aggregate function. Next, we need to specify an aggregate function per measure.

Note that possible aggregate functions are predefined by QB4OLAP. The inputs for this task are the QB4OLAP graph IRI, the QB dataset IRI, and the *MAggMap* mapping; i.e., the [measure IRI, aggregate function IRI] pair (see Section 4.2). The aggregate function is specified as a triple that relates the aggregate function IRI with the component of the cube schema structure related to the measure. This triple is added to the QB4OLAP cube schema and it can be performed with Query 6. In case of more than one measure, the query should be run for each measure.

Query 6. *Specification of an aggregate function.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?dsIRI? – the data set IRI, and
# ?measureIRI?
# ?aggregateFunctionIRI? – the aggregate function IRI

INSERT INTO ?qb4oGraphIRI? {
?comp qb4o:aggregateFunction ?aggregateFunctionIRI? }
FROM ?qb4oGraphIRI?
WHERE {
?dsd  a  qb:DataStructureDefinition .
?dsIRI? qb:structure  ?dsd .
?dsd  qb:component  ?comp .
?comp qb:measure ?measureIRI? . }
```

An example of the updated QB4OLAP cube schema is presented in Example 12.

Example 12. *Resulting triples of Query 6.*

```
<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> qb:structure
newG:newDSD .
newG:newDSD a qb:DataStructureDefinition ;
  qb:component  [ qb4o:level sdmx−dimension:refArea ] ;
        qb:component  [ qb:measure sdmx−measure:obsValue ;
                        qb4o:aggregateFunction qb4o:sum ] .
sdmx−dimension:refArea a qb4o:LevelProperty .
sdmx−measure:obsValue a qb:MeasureProperty .
```

Line 6 presents the aggregate function that is assigned to a measure by the grouping mechanism via a blank node. In this case, the SUM (i.e., `qb4o:Sum`) aggregate function.

Definition of a dimension. As part of the automatic redefinition, to build QB4OLAP-compliant dimension hierarchies, a new dimension for each initial level needs to be defined. As explained in Section 3, QB4OLAP reuses the `qb:DimensionProperty`, however with different semantics than in QB: while in the latter a dimension represents a point at a fixed granularity, QB4OLAP considers a dimension to contain points at different level granularities. Therefore, in QB4OLAP, a QB dimension becomes a dimension level (see Query 5) and a dimension represents a set of levels that are hierarchically organized. The inputs for this task are the QB4OLAP graph IRI and the dimension IRI, and it can be performed with Query 7 that should be run for each dimension.

Query 7. *Definition of a dimension.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI and
# ?dimensionIRI? – the dimension IRI

INSERT INTO ?qb4oGraphIRI? {
?dimensionIRI? a qb:DimensionProperty . }
```

An example of the triple added to the updated QB4OLAP cube schema is presented in Example 13.

Example 13. *Resulting triple of Query 7.*

```
newG:geoDimension a qb:DimensionProperty .
```

The triple presents a dimension for the `sdmx-dimension:refArea` initial level.

Definition of a hierarchy. Once the dimensions are created, we need to create a hierarchy for each dimension. A hierarchy represents the set of hierarchically ordered levels in the dimension. Once created, it needs to be linked with the corresponding dimension and the initial level. Thus, the inputs for this task are the QB4OLAP graph IRI , the hierarchy IRI, the dimension IRI, and the level IRI. This can be performed with Query 8 that should be run for each hierarchy.

Query 8. *Definition of a hierarchy.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?hierarchyIRI? – the hierarchy IRI,
# ?dimensionIRI? – the dimension IRI, and
# ?levelIRI? – the level IRI

INSERT INTO ?qb4oGraphIRI? {
?hierarchyIRI? a qb4o:Hierarchy .
?dimensionIRI? qb4o:hasHierarchy ?hierarchyIRI? .
?hierarchyIRI? qb4o:inDimension ?dimensionIRI? .
?hierarchyIRI? qb4o:hasLevel ?levelIRI? . }
```

An example of the triples added to the updated QB4OLAP cube schema is presented in Example 14.

Example 14. *Resulting triples of Query 8.*

```
newG:geoHierarchy a qb4o:Hierarchy .
newG:geoDimension qb4o:hasHierarchy newG:geoHierarchy .
newG:geoHierarchy qb4o:inDimension newG:geoDimension .
newG:geoHierarchy qb4o:hasLevel newG:region .
```

Line 1 illustrates a new hierarchy being created. Triples in lines 2 and 3 link the hierarchy with a dimension, and to a level in line 4.

Populating level members of an initial level. Finally, at the end of the redefinition phase, we populate level members for the initial levels of the QB4OLAP graph schema. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the level IRI, and the QB dataset IRI. This can be performed with Query 9 that should be run for each initial level.

Query 9. *Populating level members of an initial level.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?qbGraphIRI? – the input QB graph IRI,
# ?levelIRI? – the level IRI, and
# ?dsIRI? – the data set IRI

INSERT INTO ?qb4oGraphIRI? {
?levelMember a qb4o:LevelMember .
?levelMember qb4o:memberOf ?levelIRI? . }
FROM ?qbGraphIRI?
WHERE { {
 SELECT DISTINCT ?levelMember WHERE {
  ?o a qb:Observation .
  ?o qb:dataSet ?dsIRI? .
  ?o ?levelIRI? ?levelMember . } } }
```

An example of level member triples added to the QB4OLAP graph instances is presented in Example 15.

Example 15. *Resulting triples of Query 9.*

```
country:RS a qb4o:LevelMember .
country:RS qb4o:memberOf sdmx−dimension:refArea .
```

Line 1 illustrates a level member extracted from the observations and line 2 links it to the level it belongs to.

5.3. Enrichment Phase

Once the cube schema is redefined in terms of QB4OLAP, we next focus on its enrichment with new dimensional concepts to construct richer hierarchies. At this point, we assume that a pre-process to discover potential new levels and level attributes has been carried out. For example, this could be done with the algorithms proposed in Section 4.3. Starting from an initial dimension level we explain the construction of two three-level hierarchies. This scenario is illustrated in Figure 4. Starting from the `refArea` level that belongs to the hierarchy H1 we first add a new level - `region`. For this we need to add the `region` level to the hierarchy H1, create the hierarchy step S1, add both levels to S1 (`refArea` as child and `region` as parent), and add S1 to H1. Then, we add one more level, `income-level`, on top of `refArea`. As conceptually the new level belongs to a new hierarchy (i.e., it does not belong to the same rollup path as `region`), we need to create a new hierarchy H2, create a new step S2, add `refArea` (as child) and `income-level` (as parent) to S2, and add S2 to H2. Moreover, H2 needs to be added to the same dimension that H1 already belongs to. This notation is fixed by QB4OLAP. Finally, we create the mandatory `All` level for the dimension and accordingly link the `region` and `income-level` levels to it via two new hierarchy steps S3 and S4 that are created and added to H1 and H2, respectively. The process is as follows.

Creating, populating, and linking a new parent level. First, we need to create a new level (e.g., `region`) and add it as a parent level to the child level (e.g., `refArea`). Moreover, we need to link all the members of the child level with the corresponding members of the new parent level. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the child level IRI, and the new parent level (i.e., property) IRI. This can be performed with Query 10. To build the hierarchy illustrated in Figure 4, the query should be run for both `region` and `income-level` levels that are added as parent levels for the `refArea` level.

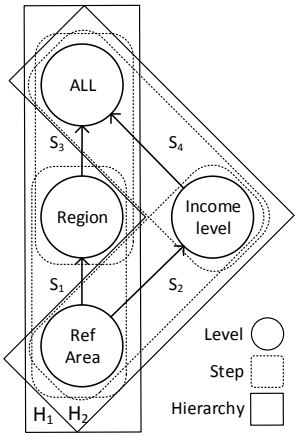

Figure 4: Three-level Hierarchies Construction

Query 10. *Creating, populating, and linking a new parent level.*

```
# INPUT: ?qb4oGraphIRI? – the new QB4OLAP graph IRI,
# ?qbGraphIRI? – the input QB graph IRI,
# ?levelIRI? – the existing level IRI, and
# ?propertyIRI? – the property IRI (i.e., new level)

INSERT INTO ?qb4oGraphIRI? {
?propertyIRI? a qb4o:LevelProperty .
?obj a qb4o:LevelMember .
?obj qb4o:memberOf ?propertyIRI? .
?levelMember2 skos:broader ?obj2 . }
WHERE { {
 SELECT DISTINCT ?obj
 FROM ?qbGraphIRI?
 WHERE { {
  SELECT ?levelMember
  FROM ?qb4oGraphIRI?
  WHERE {
   ?levelMember a qb4o:LevelMember .
   ?levelMember qb4o:memberOf ?levelIRI? . }}
  ?levelMember ?propertyIRI? ?obj . }}
{ SELECT ?levelMember2 ?obj2
  FROM ?qbGraphIRI?
  WHERE {
  {SELECT ?levelMember2
   FROM ?qb4oGraphIRI?
   WHERE {
    ?levelMember2 a qb4o:LevelMember .
    ?levelMember2 qb4o:memberOf ?levelIRI? . }}
  ?levelMember2 ?propertyIRI? ?obj2 .
 } GROUP BY ?levelMember2 ?obj2 } }
```

An example of triples for two new levels, their level members, and linking of the level members of the new parent levels with the level members of the child level added to the QB4OLAP graph is presented in Example 16.

Example 16. *Resulting triples of Query 10.*

```
newG:region a qb4o:LevelProperty .
region:ECS a qb4o:LevelMember .
region:ECS qb4o:memberOf newG:region .
country:RS skos:broader region:ECS .

newG:income−level a qb4o:LevelProperty .
income:UMC a qb4o:LevelMember .
income:UMC qb4o:memberOf newG:income−level .
country:RS skos:broader income:UMC .
```

Line 1 illustrates the new level definition for the `region` *level. Lines 2 and 3 exemplify a new level member and its linking to the* `region` *level, respectively. Then, line 4 links a child level member (i.e.,* `country:RS` *to the new parent level member (i.e., region:ECS) and this way creating a rollup relationship between them. Finally, lines 6-9 reflect the same definition for* `income-level`*.*

Definition of a hierarchy step in an existing hierarchy. Having a new parent level defined and added to the QB4OLAP graph (both schema and instance parts), we next need to create a hierarchy step that determines the order of these levels in a hierarchy. In this context, we explain two particular cases. The first, simpler, case is to add a new parent level to a level that is either the only level in a hierarchy or that is the last (i.e., coarsest) parent level in the hierarchy. The inputs for this task are the QB4OLAP graph IRI, the child level IRI, the new parent level (i.e., property) IRI, the hierarchy IRI, and the hierarchy step IRI. This can be performed with Query 11. To build the hierarchy illustrated in Figure 4, the query should be run for the `region` level that is added as parent level to the `refArea` level.

Query 11. *Definition of a hierarchy step in an existing hierarchy.*

```
# INPUT: ?qb4oGraphIRI? − the new QB4OLAP graph IRI,
# ?levelIRI? − the existing level IRI,
# ?propertyIRI? − the property IRI (i.e., new level),
# ?hierarchyIRI? − the hierarchy IRI, and
# ?hsIRI? − the hierarchy step IRI (typically blank node)

INSERT INTO ?qb4oGraphIRI? {
?hsIRI? a qb4o:HierarchyStep .
?hsIRI? qb:inHierarchy ?hierarchyIRI? .
?hsIRI? qb4o:parentLevel ?propertyIRI? .
?hsIRI? qb4o:childLevel ?levelIRI? .
?hsIRI? qb4o:pcCardinality qb4o:ManyToOne .
?hierarchyIRI? qb4o:hasLevel ?propertyIRI? .}
```

Example triples for creating a hierarchy step in an existing hierarchy are presented in Example 17.

Example 17. *Resulting triples of Query 11.*

```
_:newHierarchyStep a qb4o:HierarchyStep .
_:newHierarchyStep qb4o:inHierarchy newG:geoHierarchy .
_:newHierarchyStep qb4o:parentLevel newG:region .
_:newHierarchyStep qb4o:childLevel sdmx−dimension:refArea .
_:newHierarchyStep qb4o:pcCardinality qb4o:ManyToOne .
newG:geoHierarchy qb4o:hasLevel newG:region .
```

Line 1 defines the new hierarchy step and lines 2 – 5 link the hierarchy step with its hierarchy, parent level, child level, and cardinality, respectively. Finally, line 6 adds the new parent level to the existing hierarchy.

Definition of a hierarchy step while creating a new hierarchy. The second, more complex, case is to add a parent level to a child level that already has a parent level (in one or more hierarchies). In this case, for each hierarchy where there is a parent level, create a new hierarchy. Then, replicate the hierarchy steps and add the corresponding levels such that the child level is the only or the last (i.e., coarsest) parent level in the hierarchy. Finally, add the new parent level to the new hierarchy and create a new hierarchy step with the child level. In case that identical (i.e., duplicate) new hierarchies would be created from different existing hierarchies, only one new hierarchy should be created. Thus, in the context of Figure 4, adding the `income-level` level as parent to the `refArea` level (i.e., S2 in H2) can be performed with Query 12. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, the child level IRI, the new parent level (i.e., property) IRI, the hierarchy IRI, and the hierarchy step IRI. Note that if there was a longer sequence of levels (and hierarchy steps) to be replicated, Query 12 would need to be extended.

Query 12. *Definition of a hierarchy step with creating a new hierarchy.*

```
# INPUT: ?qb4oGraphIRI? − the new QB4OLAP graph IRI,
# ?qbGraphIRI? − the input QB graph IRI,
# ?levelIRI? − the existing level IRI,
# ?propertyIRI? − the property IRI (i.e., new level),
# ?hierarchyIRI? − the hierarchy IRI, and
# ?hsIRI? − the hierarchy step IRI (typically blank node)

INSERT INTO ?qb4oGraphIRI? {
?hierarchyIRI? a qb4o:Hierarchy .
?hierarchyIRI? qb4o:inDimension ?d .
?d qb4o:hasHierarchy ?hierarchyIRI? .
?hsIRI? a qb4o:HierarchyStep .
 ?hsIRI? qb4o:inHierarchy ?hierarchyIRI? .
 ?hsIRI? qb4o:parentLevel ?propertyIRI? .
 ?hsIRI? qb4o:childLevel ?levelIRI? .
 ?hsIRI? qb4o:pcCardinality qb4o:ManyToOne .
?hierarchyIRI? qb4o:hasLevel ?propertyIRI? .
?hierarchyIRI? qb4o:hasLevel ?levelIRI? }
FROM ?qbGraphIRI?
WHERE {
 ?h a qb4o:Hierarchy .
 ?h qb4o:hasLevel ?levelIRI? .
 ?h qb4o:inDimension ?d .
 ?d a qb:DimensionProperty .
}
```

Example triples for creating a hierarchy step in a new hierarchy are presented in Example 18.

Example 18. *Resulting triples of Query 12.*

```
newG:incomeHierarchy a qb4o:Hierarchy .
newG:incomeHierarchy qb4o:inDimension newG:geoDimension .
newG:geoDimension qb4o:hasHierarchy newG:incomeHierarchy .
_:newHierarchyStep2 a qb4o:HierarchyStep .
_:newHierarchyStep2 qb4o:inHierarchy newG:incomeHierarchy .
_:newHierarchyStep2 qb4o:parentLevel newG:income−level .
_:newHierarchyStep2 qb4o:childLevel sdmx−dimension:refArea .
_:newHierarchyStep2 qb4o:pcCardinality qb4o:ManyToOne .
newG:incomeHierarchy qb4o:hasLevel sdmx−dimension:refArea
newG:incomeHierarchy qb4o:hasLevel newG:income−level .
```

Lines 1 – 3 define the new hierarchy and link it with the existing dimension. Then, lines 4 – 8 create a new hierarchy step and link it with its hierarchy, parent level, child level, and cardinality, respectively. Finally, lines 9 and 10 add both levels to the hierarchy.

In any case, when adding a hierarchy step, cycles need to be avoided in the hierarchy definition and this can be achieved following the rationale of Algorithm 1.

Definition of the all level and its level member. Finally, the mandatory `All` level with its `all` level member must top all the dimension hierarchies. Thus, we first need to add the `All` level to each dimension. The inputs for this task are the QB4OLAP graph IRI, the `All` level IRI, and the `all` level member IRI. This can be performed with Query 13.

Query 13. *Definition of the all level and its level member.*

```
# INPUT: ?qb4oGraphIRI? − the new QB4OLAP graph IRI,
# ?allLevelIRI? − the all level IRI, and
# ?allLevelMemberIRI? − the all level member IRI

INSERT INTO ?qb4oGraphIRI? {
?allLevelIRI? a qb4o:LevelProperty .
?allLevelMemberIRI? a qb4o:LevelMember .
?allLevelMemberIRI? qb4o:memberOf ?allLevelIRI? .}
```

Example triples for creating the `All` level and its `all` level member are presented in Example 19.

Example 19. *Resulting triples of Query 13.*

```
newG:geoALL a qb4o:LevelProperty .
newG:geoALLmember a qb4o:LevelMember .
newG:geoALLmember qb4o:memberOf newG:geoALL .
```

Line 1 illustrates the new level definition of the all level for the geo dimension. Line 2 defines its level member and line 3 links the member to the level.

Linking of the `all` level member with the lower level members. After creating the `All` level and its `all` level member for a dimension, all the coarsest levels of each hierarchy belonging to the dimension, must be linked to the `All` level. Furthermore, their level members must be related to the `all` level member. The inputs for this task are the QB4OLAP graph IRI, the `All` level IRI, and the child level IRI. This can be performed with Query 14. In the context of the hierarchy illustrated in Figure 4, the query should be run for both `region` and `income-level` that need to be linked to the `All` level.

Query 14. *Linking of the `all` level member with the lower level members.*

```
# INPUT: ?qb4oGraphIRI? − the new QB4OLAP graph IRI,
# ?levelIRI? − the child level IRI, and
# ?allLevelMemberIRI? − the all level member IRI

INSERT INTO ?qb4oGraphIRI? {
?levelMember skos:broader ?allLevelMemberIRI? . }
FROM ?qb4oGraphIRI?
WHERE {
?levelMember a qb4o:LevelMember .
?levelMember qb4o:memberOf ?levelIRI? . }
```

Example triples that link the child level members with the `all` level member are presented in Example 20.

Example 20. *Resulting triples of Query 14.*

```
region:ECS skos:broader newG:geoALLmember .
income:UMC skos:broader newG:geoALLmember .
```

Lines 1 and 2 illustrate linking of the `region` and `income-level` level members with the `all` level member in the geo dimension, respectively.

Once the `All` levels (one per dimension) and their `all` level member are created and linked, we can use Query 11 to link both `region` and income-level to the `All` level. This way we can create the S3 and S4 hierarchy steps in Figure 4. The addition of the `All` levels for the remaining two dimensions is analogous.

Definition of a level attribute and its linking to a level. Additionally to the construction of the dimension hierarchies, the levels may have level attributes that can also be discovered with Algorithm 1. Thus, level attributes can be added to the QB4OLAP graph. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, level IRI, and the attribute IRI. This can be performed with Query 15.

Query 15. *Definition of a level attribute and its linking to a level.*

```
# INPUT: ?qb4oGraphIRI? − the new QB4OLAP graph IRI,
# ?qbGraphIRI? − the input QB graph IRI,
# ?levelIRI? − the child level IRI, and
# ?propertyIRI? − the property IRI (i.e., the attribute)
```

```
INSERT INTO ?qb4oGraphIRI? {
 ?propertyIRI? a qb4o:LevelAttribute .
 ?propertyIRI? qb4o:inLevel ?levelIRI? .
 ?levelIRI? qb4o:hasAttribute ?propertyIRI? .
 ?levelMember ?propertyIRI? ?obj . }
FROM ?qbGraphIRI?
WHERE {
 {SELECT ?levelMember
 FROM ?qb4oGraphIRI?
 WHERE{
  ?levelMember a qb4o:LevelMember .
  ?levelMember qb4o:memberOf ?levelIRI? . } }
 ?levelMember ?propertyIRI? ?obj . }
```

Example triples of a level attribute, its linking with a level, and specifying the value for a level member are presented in Example 21.

Example 21. *Resulting triples of Query 15.*

```
skos:prefLabel a qb4o:LevelAttribute .
skos:prefLabel qb4o:inLevel sdmx−dimension:refArea .
sdmx−dimension:refArea qb4o:hasAttribute skos:prefLabel .
country:RS skos:prefLabel "Serbia"\@en .
```

Line 1 defines a level attribute and lines 2 and 3 link the level attribute and the corresponding level. Then, line 4 exemplifies the value of the attribute for `country:RS`.

Copying of observations. The output of the previous steps (i.e., queries) is the complete QB4OLAP schema graph and partial QB4OLAP instance graph. To complete the latter one, the observations need to be copied to the QB4OLAP graph. The inputs for this task are the QB4OLAP graph IRI, the QB graph IRI, and the QB dataset IRI. This can be performed with Query 16.

Query 16. *Copying of observations.*

```
# INPUT: ?qb4oGraphIRI? − the new QB4OLAP graph IRI,
# ?qbGraphIRI? − the input QB graph IRI, and
# ?dsIRI? − the data set IRI

INSERT INTO ?qb4oGraphIRI? {
?oIRI a qb:Observation .
?oIRI ?prop ?val . }
FROM ?qbGraphIRI?
WHERE {
 ?oIRI a qb:Observation .
 ?oIRI qb:dataSet ?dsIRI? .
 ?oIRI ?prop ?val . }
```

Example triples of a QB observation copied to the QB4OLAP graph are presented in Example 22.

Example 22. *Resulting triples of Query 16.*

```
<http://worldbank.270a.info/dataset/world−bank−indicators/
                              CM.MKT.LCAP.CD/RS/2012>
  a qb:Observation ;
  qb:dataSet newG:newDS ;
  property:indicator indicator:CM.MKT.LCAP.CD ;
  sdmx−dimension:refArea country:RS ;
  sdmx−measure:obsValue 7450560827.04874 ;
```

Line 1 defines an observation and the rest of the triples link observation with the data set, levels, and measure value for `country:RS`.

5.4. Additional considerations

We want to position our approach with respect to two QB concepts that might be considered relevant in our context, namely `qb:AttributeProperty` and `qb:Slice`. The former refers to the attributes describing the observations (e.g., the type of the measure). We consider these attributes as metadata rather than elements of the schema. Our proposal is to capture this information, by means of analytical metadata (e.g., the SM4AM analytical metadata [30]). For instance, different attributes can be defined as instances (via `rdf:type`) of `sm4am:DataProperty` and kept as additional metadata. Regarding the latter, `qb:Slice`, we focus on the base cuboid, without considering higher level cuboids, since they are derivable from the base cuboid.

For the creation of the IRIs of the new objects, we recommend to use the W3C best practices.[26]

A final consideration relates to whether or not the QB4OLAP enrichment should entail the construction of a new graph. For simplicity of presentation, we assume so far that the method steps always build a new graph. Nevertheless, the newly created QB4OLAP data cube schema can be used for the exploration of the existing observations. As the existing QB dimension properties used in the observations are now redefined as QB4OLAP dimension level properties, the new schema that defines dimension structures and includes the aggregate functions for measures can be used to interpret observations and aggregate measure values. Moreover, the existing graph can be changed such that its `qb:DataSet` points only to the newly defined `qb:DataStructureDefinition` representing the QB4OLAP data cube schema.

6. Evaluation

To evaluate our approach, we have built the tool called *QB2OLAP Enrichment Module* (QB2OLAPem) that redefines a QB data set in terms of the QB4OLAP vocabulary. The resulting QB4OLAP data set can then be enriched as discussed in the previous section by relating measures with aggregate functions and by discovering new potential dimensional concepts that can then

[26] http://www.w3.org/2011/gld/wiki/223_Best_Practices_URI_Construction

be used to extend the MD knowledge described in the QB4OLAP data set. To do so, QB2OLAPem follows the steps described in the method. Our tool was used in a set of experiments aimed at validating our approach and in this section we present the results. We first briefly introduce QB2OLAPem. Then, we explain the evaluation setting and rational. Finally, we present the evaluation results for the applicable quality characteristics of the Quality in use and the Product quality models of the ISO/IEC 25000 [31].

6.1. QB2OLAP Enrichment Module

QB2OLAPem is developed in Java using JRE 1.8.0_60, Apache Jena 2.13.0 for working with RDF graphs, SWT[27] for the GUI, and Windows 8.1 as OS. It implements the algorithm to discover implicit dimensional concepts from Section 4 and the enrichment method from Section 5. The enrichment steps are performed in an iterative and interactive fashion. QB2OLAPem automatically retrieves potentially new dimension levels and level attributes, lists them to the user who can choose the ones of her interest, and automatically enriches the data set based on the user choices. The user can also configure an aggregate function for each measure. QB2OLAPem visualizes the cube structure for the user and enables her to automatically generate the QB4OLAP triples once the enrichment is finished. This way, QB2OLAPem enables user-friendly and (semi-)automatic enrichment of QB data sets with additional QB4OLAP semantics. Furthermore, the output triples produced by QB2OLAPem can be straightforwardly loaded into an SW-based OLAP engine and allow traditional OLAP analysis (i.e., by means of a high-level interface in terms of an MD algebra that automatically translates these high-level operators in terms of SPARQL). Thus, it lowers the entry barrier for data analysis on SW data for non-expert users (e.g., traditional OLAP users). For further details, [9] reports on how to connect QB2OLAPem with an OLAP engine. From here on we focus on QB2OLAPem, whose process flow is presented in Figure 5.

The figure illustrates three kinds of external processes, as well as the QB2OLAPem internal process flow. The first kind of external process is the user – system interaction, where the user is guided by the Interface element to iteratively perform the semi-automatic enrichment using the QB2OLAPem GUI. The second kind of external process is the system – SPARQL endpoint interaction where QB2OLAPem queries external

[27] https://www.eclipse.org/swt/

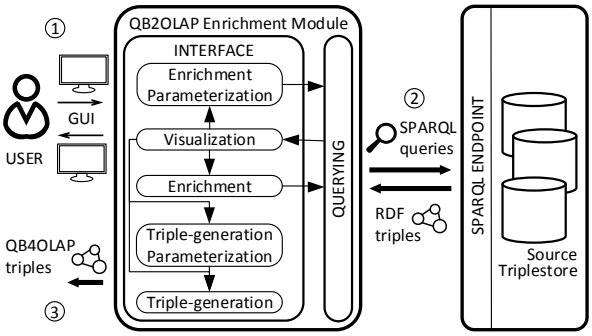

Figure 5: QB2OLAP Enrichment Module Process Flow

triplestores to identify implicit potential MD concepts that could be used for the enrichment. The last kind of external process is the generation of the QB4OLAP triples that can then be used in other tools (e.g., an OLAP engine, a SPARQL endpoint, etc.). These three kinds of external processes are coordinated by the QB2OLAPem internal process flow as follows. In the user – system interaction, the user initiates the enrichment process by specifying the SPARQL endpoint, the data set IRI, and possibly additional fine-tuning parameters in the Enrichment Parameterization activity. Then, the Querying element triggers the SPARQL queries to retrieve the RDF triples specifying the initial QB cube structure. This structure is then redefined in terms of QB4OLAP (see Section 5.2). The results are then visualized to the user in the Visualization activity. Moreover, the Querying element also runs the queries to discover candidate enrichment concepts (see Section 4.3). The user can then again set the fine-tuning parameters and optionally restart the process, perform the enrichment in the Enrichment activity, or decide to generate triples in the Triple-generation activity, where she can optionally first set the fine-tuning parameters for triple generation in the Triple-generation Parameterization activity. After each enrichment, the Querying element again runs the necessary SPARQL queries to discover new candidate enrichment concepts if any. The enrichment process finishes when the user decides to generate the QB4OLAP triples as the final result. The QB2OLAPem GUI is illustrated in Figure 6 and more details about the tool can be found in [9].

6.2. Evaluation Setting and Rational

For the evaluation of our approach, we adopt quality characteristics from the Quality Model of the ISO/IEC 25000 – System and Software Quality Requirements and Evaluation (SQuaRE) series of standards [31] that are applicable to our usage scenario. It includes the

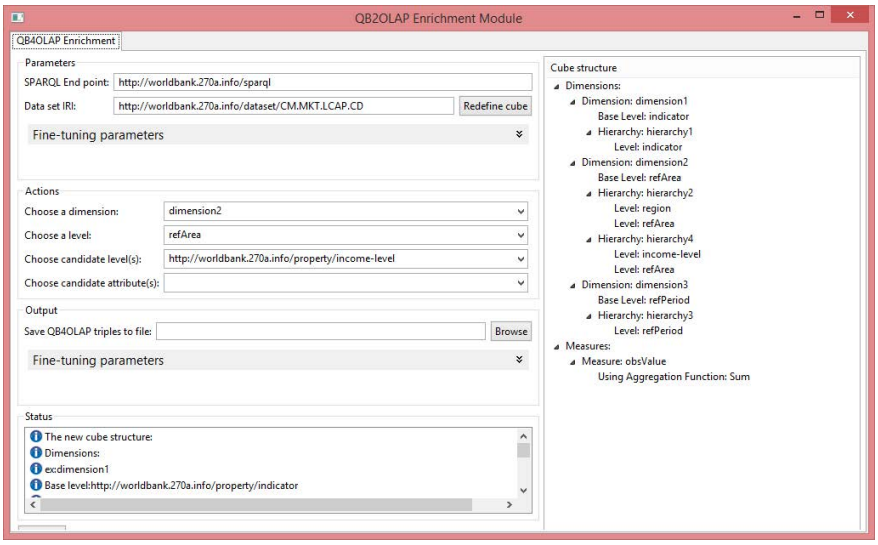

Figure 6: QB2OLAP Enrichment Module Screenshot

Quality in use model and the Product quality model that specify quality characteristics and subcharacteristics for which we define metrics specific for our tool. Table 1 presents the details.

The main target group of users for our tool are OLAP practitioners that are well familiar with the data cube abstraction paradigm (see Section 2.1). Typical OLAP users are not proficient in, for example, SQL or other query languages. Instead, they analyze data using graphical user interfaces of different OLAP engines. Such high-level automation is enabled by the definition of a correct MD schema (see Section 2.1), which allows OLAP users to abstract their actions from data manipulation languages (such as SQL or SPARQL) and use high-level interfaces based on an MD algebra not demanding IT expertise [10]. In the same spirit, our tool supports OLAP users to enrich the QB schema with a graphical environment. Thus, more than just analyzing data as in traditional OLAP settings, QB2OLAPem enables OLAP users to participate in discovering interesting analysis perspectives and construct the MD schema according to their requirements / needs. Moreover, although the users with both SW and OLAP skills could perform this manually, our tool lowers the barrier such that the users non-familiar with SW technologies and even non-technical users can perform the enrichment. Therefore, we conduct the experiments with 25 users including 4 OLAP experts, 4 SW experts, and 17 users having some knowledge of both. The expert users are PhD students[28] and a master student[29] doing research in the OLAP and SW areas. The non-expert users include a PhD student familiar with both areas and students of a UPC-BarcelonaTech master[30] that includes a Data Warehousing / OLAP course and a Semantic Web course. All of them had a 20 minutes tutorial in QB / QB4OLAP and some recall on DW / OLAP to facilitate the understanding of the template SPARQL queries provided to perform the manual enrichment as well as the overall objective.

For the experiments with the users we consider the running example data set. The users should perform the enrichment according to a predefined scenario. Currently, the alternative to using our tool is that the user manually explores a QB data set, discovers enrichment concepts, and constructs the QB4OLAP graph by using SPARQL while, also manually, storing and defining RDF triples. Thus, we compare this case as an alternative to using the tool for the same tasks. The users are asked to both use the tool and follow a guided tutorial that provides template SPARQL queries to manually perform the enrichment. They are provided with detailed guidelines in both cases. Moreover, the order of the cases is alternated among users. After the enrichment, the users deliver the results and their feedback. Later, for the evaluation of the tool performance, we consider two more data sets from another Linked Data source and of different sizes. All the data sets are

[28] https://it4bi-dc.ulb.ac.be/
[29] http://it4bi.univ-tours.fr/it4bi/
[30] http://www.fib.upc.edu/en/masters/miri.html

Table 1: Quality Characteristics, Subcharacteristics, and Metrics

Model	Characteristics	Subcharacteristics	Metrics
Quality in use - The evaluation of the interaction with a system	**Effectiveness** - Accuracy and completeness with which users can enrich a schema	/	The percentage of users producing (not) complete and (not) sound schemata according to a predefined scenario, both with and without the tool
	Efficiency - The actions that the user needs to perform to enrich the schema	/	The number of user actions and time contributing to redefinition/enrichment tasks according to a predefined scenario, and the total number of user actions and total time
	Satisfaction - How much is the user satisfied with using the tool	**Usefulness** - How useful does the user considers the tool	Survey statistics
Product quality - The software product quality properties	**Functional suitability** - The functions specified by the tool	**Functional completeness** - Functions that cover the specified tasks	The percentage of cases where the user produces a complete schema according to the underlying QB data set and a predefined scenario
		Functional correctness - Correctness of the tool results	The percentage of cases where the user produces a sound schema according to the MD model
	Performance efficiency - The amount of resources used for the enrichment	**Time-behavior** - The processing time	(i) Redefinition time to: a) Retrieve the initial cube structure, b) Retrieve the level members, and (ii) Enrichment time: a) Properties / Level members retrieval time
		Capacity - The limits of the tool	The number of observations (characterizing the data set size) that can be handled by the tool
	Usability - How much can a tool be used for the enrichment with effectiveness, efficiency, and satisfaction	**Learnability** - Degree to which specified users can learn to use the system with effectiveness, efficiency, and satisfaction	Statistics about how many different types of user are able to use all (or partial) tool functions (e.g., redefinition, adding new level, etc.)
		Operability - The attributes of the system that make it easy to operate and control the enrichment	The time that the user takes for performing the actions in the tool

loaded into a local endpoint after a cleaning process in which observations with pre-aggregated and zero measure values are removed. Moreover, we applied additional transformations, e.g., conversion from literals to IRIs to enable further search for new levels. The local SPARQL endpoint is provided by a Virtuoso 7 server running on an Intel Core i5 CPU at 2.6 GHz and 8 GB of RAM machine with Windows 8.1 OS. The details and results of the experiments are presented in the following subsections.

6.3. *Quality in Use Evaluation*

The quality in use metrics are evaluated over the "Market Capitalization" QB data set from our running example. It is an indicator data set of 1.9 MB in size extracted from the WBLD source. All the WBLD indicator data sets have the same cube structure and the average size is of approx. 1.15 MB. Thus, "Market Capitalization" is considered as a representative data set for WBLD. Table 2 includes its main features such as the number of dimensions in the schema (#dimensions), the number of observations which in fact characterizes the size of the data set (#observations), and the total number of dimension members for all dimensions (#members), along with other data sets and results relevant for the next subsection.

The scenario that the users need to perform is to redefine the schema according to the QB4OLAP vocabulary, check if certain properties can be considered as candidates for the enrichment, create three-level hierarchies (i.e., hierarchies consisting of the initial QB level, one new level on top of the initial one, plus the all level) for two different dimensions and optionally add a level attribute to a level. Figure 4 summarizes the enrichment to be done for each of these dimensions: define levels, a hierarchy for each rollup path, hierarchy steps for each adjacent pair of nodes in the same path, and a dimension wrapping all of them (see Section 5). The evaluation guidelines provide a predefined scenario and thus, ask the participants to enrich the data set with some specific levels. The quality in use characteristics specified in Table 1 focus on the user efforts to perform these tasks with and without the tool. Detailed guidelines for both i) the manual and ii) tool enrichment were given[31]. The guidelines include i) the set of predefined SPARQL queries with the parameters that need to be used and ii) the specification of tool actions with the parameters that need to be used, respectively. The users have one hour and a half to perform the manual enrichment and half an hour with the tool.

[31]The guidelines given to the evaluation participants can be found at: http://www.essi.upc.edu/~jvarga/qb2olapem.html

Table 3: Effectiveness Results

Option / Case	**Manual**		**Tool**	
Sound & Complete	16%		84%	
Incomplete and/or Not Sound	84%		16%	
	Incomplete	100%	Incomplete	100%
	Not sound	47.62%	Not sound	0%

Effectiveness. The results of the metrics for the effectiveness characteristics are presented in Table 3. The table presents the percentage of users producing sound and complete schemata with and without the tool. In this context, completeness refers to whether the user managed to complete all the tasks required to fulfill the requirements in the guidelines within the time available, while soundness indicates whether the schema produced is correct in terms of MD modeling (see Section 6.4 for details). The results show that, even with very detailed guidelines, only 4 (including just 2 SW experts) out of 25 users managed to manually create a sound and complete schema. The rest of users did not manage to manually complete all the tasks and almost half of them also created errors in this process undermining the schema soundness. These errors are typically a consequence of copy/paste actions and include the same concept defined as both level and level attribute, or hierarchy and hierarchy step, adding the same level to different dimensions (and their hierarchies), adding the same level members to several levels, adding duplicate hierarchies, etc. Oppositely, 4 users did not complete the task but the partial output produced was correct (i.e., generated sound but incomplete schemata). The results are much better when using the tool. 21 participants created sound and complete schemata. Out of the 4 not completing the task, none produced an incorrect schema. This shows that in practice *the tool guarantees the schema soundness*.

Efficiency. Regarding the efficiency characteristic of the Quality in use model, the results of our experiments with users are presented in Table 4. The table represents the average number of effective actions (i.e., queries or tool interactions) contributing to the task (i.e., redefinition/enrichment) and the total number of actions (which include unnecessary actions to complete the task, such as errors). Moreover, the table illustrates the average effective and total time. At the bottom, the table shows the ratio between the effective and total actions, and it is used for the calculation of the effective time with re-

Table 2: The Schema Transformation

Data set	#dimensions	t_{schema}	#observations	#members	$t_{members}$	#properties	$t_{properties}$
Market Capitalization	3	0.91 s	2360	146	0.04 s	2364	1.08 s
Renewable Energies	6	1.18 s	537447	155	1.05 s	786	1.05 s
Asylum seekers	7	0.94 s	499369	286	1.4	796	1.90 s

Table 4: Efficiency Results

Task / Parameter	Redefinition		Enrichment	
	Manual	Tool	Manual[32]	Tool
Effective Actions	11.2	5.6	14	9.28
Total Actions	17	6.32	17	9.92
Effective Time (min)	30.02	4.18	43.32	7.15 (3.75)
Total Time (min)	45.5	4.72	52.38	7.64 (3.75)
Ratio of Effective and Total Actions	65%	89%	82%	94%

spect to the total time. Furthermore, note that the time for the manual enrichment only refers to the 4 users that finished (thus, it does not consider the 21 users that did not finish the enrichment task). In this context, we added in brackets the values referring to these four users when performing the enrichment with the tool (since they were among the most skilled ones). Thus, Table 4 illustrates that *the tool reduces the necessary time* for the redefinition and enrichment *by at least 7 times* even when users are provided with detailed guidelines for manually performing these tasks.

Satisfaction. Finally, we discuss the results of a survey answered by the users that provide insights about the satisfaction characteristic of the Quality in use model[33]. The average user rating of how much they like the tool, how easy it is to use the tool, how useful the tool is, and if they would use it in the future, are all over 4 in a scale from 1 to 5. The average user rating on the helpfulness of the manual and tool guidelines are 3.98 and 4.1, respectively. However, only 5 users (including 2 SW experts) consider that they would be able to formulate the queries without guidelines with a certainty of 4 or 5. However, this is proven wrong since even with the guidelines only 4 users were able to generate a sound and complete schema as discussed earlier. In an open-ended question section, more than 10 users, including an SW expert, stated that the manual part is too repetitive, error-prone, and hard to perform even with the guidelines. Thus, *QB2OLAPem was appreciated and considered needed and helpful*. The main reason for this is that generating a QB4OLAP data set from an available QB data set does not consist of purely syntactical transformations but requires triggering queries to identify and validate potential new dimensional concepts (i.e., levels). Such queries need to guarantee the MD integrity constraints and also require a solid OLAP knowledge to avoid making mistakes. Additionally, the users also pointed out some improvements to be done in our tool. Mainly interface enhancements, e.g., changing the way of choosing new concepts and adding the capacity to delete / remove concepts (which is not currently present). We plan to make the changes / add these features to QB2OLAPem in the future.

[32]The values refer to the 4 users finishing the manual enrichment. The other 21 users did not finish.

[33]The survey can be found at: http://www.essi.upc.edu/~jvarga/qb2olapem.html

6.4. Product Quality Evaluation

The product quality evaluation focuses on the tool properties. In the following paragraphs we discuss the results for each of the characteristics with its subcharacteristics.

Functional suitability. The functional suitability is measured in terms of the *completeness* and *correctness* subcharacteristics. In our context, the functional completeness refers to the tool capacity to transform a QB data set into a QB4OLAP one as well as its capacity for extending the QB4OLAP data set with relevant dimensional data from all the available potential dimensional concepts. The functional correctness refers to the correctness of the QB4OLAP data set produced. In the MD context, if the underlying MD schema produced satisfies the MD integrity constraints discussed in Section 4.3. Next, we explain how both criteria are met via QB2OLAPem.

The ***correctness*** of an MD schema has been exhaustively studied in the DW/OLAP community (e.g., [24]). Based on these findings, the MD integrity constraints were identified and, in turn, several methods to automatically model diverse data (following a given data model) in terms of the MD model have been proposed (the reader is addressed to [32] for a detailed survey on this topic). The automatic identification of factual data (i.e., facts and measures) still results challenging for many domains. However, there is a clear consensus on discovering dimensional data based on FDs. Arbitrarily

looking for FDs in the MD context is known to be computationally expensive [27]. In our case, QB2OLAPem exploits the QB semantics and use the QB dimension concept as valid starting point of analysis from where to look for FDs. To do so, QB2OLAPem follows a traditional approach but adapted to the SW technologies [33]: it looks for many-to-one relationships by applying the algorithms presented in Section 4.3. However, like most automatic modeling approaches, instead of detecting and constructing all possible dimension hierarchies, the detection of new dimension levels and attributes is performed in an iterative fashion each time the user selects a new level to be added. In MD terms, QB2OLAPem applies a hybrid *data-driven* and *requirement-driven* approach [32]; i.e., effectively combining the discovery of MD knowledge hidden in the available data set (i.e., FDs are identified for all the initial QB dimensions) with the interest of the user (from there on, the user is able to enrich the MD schema with dimensional data that are semantically meaningful and of her interest). This approach is widely acknowledged as the most appropriate way to proceed in automatic MD modeling. For example, in our running example, the `region` level is meaningful to be added on top of the `reference area` (referring to country) level to conform a richer hierarchy within the same dimension (and thus, the user most probably would choose this enrichment), while this is probably not the case for the `creator` level even though it is identified as a FD and proposed by our tool as a potential rollup relationship from `reference area`. With the default settings (next we discuss how to relax them), possible new levels are only available for the `reference area`. Out of the total 20 outbound distinct properties, 6 satisfy the many-to-one cardinality and only two are a meaningful choice for the construction of hierarchies, namely, `region` and `income level`. Although there are some properties that are candidates for the next coarser granularity level for these two new levels, none of them would be meaningful from the user point of view. These enrichment possibilities are based on the data available in the data set and more meaningful enrichment concepts can be found by exploring external sources (e.g., DBpedia can provide more concepts that are meaningful for constructing coarser levels for countries).

Additionally, due to imperfect data (see Section 4.1) it might happen that some properties are not proposed as FDs (e.g., due to incorrect data) when conceptually they should be. This issue, typical from the SW and similar scenarios, has also been studied in the recent past (e.g., [33]). To deal with these situations, we also consider *quasi FDs* (see Section 4.3). In any case, selecting a quasi FD to enrich the schema requires cleaning the data set to meet the FD. Otherwise, the resulting MD schema would not guarantee a correct data summarization (see [25]). In the case of our running example, Figure 7 illustrates the effect of reducing the percentage of instances that must satisfy the many-to-one cardinality for the `reference area`. When the percentage is reduced to 80% (i.e., the *minCompl* parameter in Algorithm 2), there is an additional candidate property (`lending type`), that is also meaningful for the construction of a new hierarchy. This is a consequence of imperfect data as not all members of the `reference area` level are countries. Removing these errors, `lending type` turns out to be a functional property and thus a rollup candidate. Accordingly, these errors must be addressed to guarantee correct data aggregations if this level is chosen. The next potentially meaningful level is found at 40% but it surely does not make sense to build a hierarchy where massive cleaning should be done. Therefore, the value of 80% represents an empirically based threshold for the discovery of new levels in the case of our running example. Overall, the running example shows a good quality for the construction of dimension hierarchies as the first two properties completely satisfy the many-to-one cardinality and the third one does so for at least 80% of the cases.

Accordingly, we say that *QB2OLAPem is functionally correct because all dimensional data proposed is based on FDs*. This is guaranteed by the SPARQL queries internally triggered by QB2OLAPem (see Section 4.3). This is shown in Table 3 where no incorrect schema is produced using the tool. Note this is one of the main advantages of our tool in front of the manual enrichment. The tool guarantees the overall correctness of all the enrichment steps (queries) triggered, which cannot be guaranteed in the manual enrichment.

The ***completeness*** of the MD schema produced is guaranteed by several means. The first step is a syntactical transformation from QB to QB4OLAP constructs (i.e., the redefinition phase). There, the QB dimension concept is redefined in terms of QB4OLAP constructs, i.e., as a level within a hierarchy and belonging to a dimension. The basic QB4OLAP structure is then enriched by relating each measure with an aggregate function (see Section 5.2) and with relevant dimensional data selected by the user from that automatically discovered by the tool. The additional dimensional data added is conformed according to QB4OLAP and thus, to the good MD modeling practices (in terms of levels, hierarchies, dimensions and dimension attributes).

We say that *QB2OLAPem is functionally complete because all relevant MD data from the QB data set is*

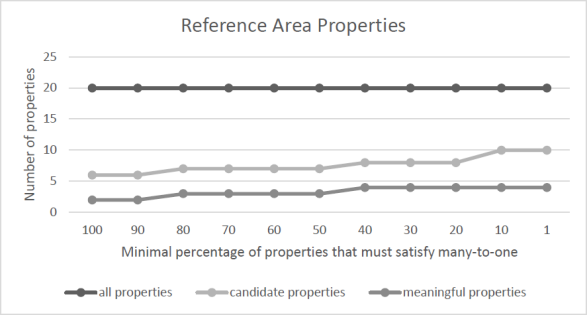

Figure 7: Reference Area Candidate Properties

included in the QB4OLAP one, all measures are related to an aggregate function and all the available FDs are discovered and proposed to the user as potential dimensional enrichment. As discussed in Section 5.2, for each QB data structure QB2OLAPem retrieves all the initial QB measures and dimensions and all of them are redefined in terms of QB4OLAP. Also, QB2OLAPem guarantees that each QB4OLAP measure is associated with an aggregated function (if none is selected, SUM is used by default). About the enrichment phase (see Section 5.3), the dimensional enrichment is based on the detection of many-to-one cardinalities based on the analysis of instances (see Section 4.3). QB2OLAPem guarantees that all the instances are covered by the newly defined levels (i.e., every level instance is member of a level) and that all potential functional properties, i.e., all potential new levels, are identified and proposed to the user. This is done in two steps, QB2OLAPem exhaustively finds all FDs for the basic dimension levels (i.e., the former QB dimensions). Later, for each additional level chosen by the user, QB2OLAPem exhaustively searches for new potential FDs starting from it.

Thus, we say *QB2OLAP is complete with regard to the available data and the user requirements stated* (as hybrid automatic MD modeling tools do). Note that the manual enrichment proposed guarantees the same degree of completeness since the SPARQL template queries provided exhaustively search for FDs and it depends on the user to properly use them. The only difference in this respect is that QB2OLAPem guarantees an aggregate function will be linked to each measure, whereas the manual enrichment cannot guarantee so. Also, that in a given time frame the user is able to complete more tasks (this is shown in Table 3 where the degree of incompleteness with regard to the requirements given in the guidelines is lower when using the tool).

Performance efficiency. To evaluate the tool's performance, we conducted the experiments considering two more data sets in addition to the one used in the previous section. In particular, we include "Asylum statistics"[34](of 1.2 GB of size) and "Renewable energies (wind, solar, hydro, tidal, wave and ocean, geothermal energy, energy from biomass)"[35](660 MB) from Eurostat Linked Data. In addition to the characteristics explained in the previous section, Table 2 shows the following characteristics relevant for the performance measurement: the time to retrieve the QB schema (t_{schema}), the time to retrieve level members ($t_{members}$), the total number of properties retrieved for level members (#properties), and the time to retrieve these properties ($t_{properties}$).

The ***time-behavior*** subcharacteristic relates to the tool performance for two main tasks, the schema redefinition and enrichment. Thus, for the schema redefinition (see Section 5.2) we measure the QB schema retrieval time and the time for acquiring dimension members. As shown in Table 2, the schema retrieval time is approximately the same for all data sets and it does not depend on the number of observations (i.e., the data set size). This advantage originates from the fact that the QB schema typically represents a small part of the entire data set. The schema can be retrieved with a simple query where the central schema node (i.e., schema structure definition) is linked to a dimension or a measure via a component node, i.e., with only two hops (see Query 5). This maps to a path join [34] and can be efficiently solved with indexing and index-based join operations [22]. Furthermore, acquiring the level members depends on the number of observations from where they are retrieved (see Query 9). Nevertheless, as shown in Table 2, *even for large data sets* (e.g., half a million observations and 286 level members) *the retrieval time is still small* (around 1 sec), demonstrating the scalability. Additionally, Table 5 shows that, *inside a single data set, the time for acquiring level members is similar regardless of their number*. Again, this is the result of indexing techniques used by triple stores.

Furthermore, regarding the schema enrichment (see Section 5.3), Table 2 shows the total number (#properties) and the retrieval time ($t_{properties}$) for all the properties related to the level members of all the initial levels of the previously introduced data sets. Moreover, Table 5 provides more details about the number of properties related to the level members of a level in each data set. All values refer to the initial schemata. From the tables

[34] http://eurostat.linked-statistics.org/data/migr_asyappctzm

[35] http://eurostat.linked-statistics.org/data/nrg_107a.rdf

Table 5: Level Member Statistics

Data set	Level	#Members	$t_{members}$	#Properties	$t_{properties}$
Market Capitalization	indicator	1	0.01	16	0.01
Market Capitalization	refArea	120	0.02	2348	0.91
Market Capitalization	refPeriod	25	0.01	0	0.16
Renewable Energies	indic_nrg	81	0.15	486	0.54
Renewable Energies	freq	1	0.22	0	0.01
Renewable Energies	product	15	0.18	90	0.08
Renewable Energies	geo	34	0.15	204	0.25
Renewable Energies	unit	1	0.14	6	0.01
Renewable Energies	refPeriod	23	0.21	0	0.16
Asylum seekers	citizen	157	0.25	620	1.00
Asylum seekers	freq	1	0.14	0	0.01
Asylum seekers	geo	34	0.12	132	0.23
Asylum seekers	asyl_app	2	0.28	8	0.02
Asylum seekers	age	6	0.25	24	0.04
Asylum seekers	refPeriod	83	0.13	0	0.59
Asylum seekers	sex	3	0.25	12	0.02

we can note that *the retrieval time does not depend on the number of observations but on the number of level members and their properties*. Our tool runs a query for each level member to retrieve its properties. Thus, the complexity of such queries is that of *adjacency* queries [35, 36] and in our case, in the worst case, it directly depends on the out degree of each level member (LevelMemberOutDegree) of each level; i.e.:

$$\Sigma_1^{levels}\Sigma_1^{members} LevelMemberOutDegree$$

When retrieving the properties, the tool acquires the property range objects that automatically become new level members (if the property is chosen by the user). The tool then automatically iterates in the same way for these new level members. Thus, there is no need for additional queries to retrieve new level members. In the worst case, if the user chooses all potential functional properties identified by the tool, the total number of properties for the 3 data sets would be 47, 23, and 18, respectively. Nevertheless, the user is the last responsible for choosing relevant properties for creating new levels. Table 6 shows a real scenario for the running example data set. There, the user chooses to enrich the data set with three new levels (below the double line) out of the seven proposed by the tool as functional properties.

The ***capacity*** subcharacteristic is measured by the number of observations (characterizing the size of the data set) that the tool can handle. Table 2 illustrates that *the tool can process data sets of different sizes*. Thus, the schema transformation is feasible and efficient even for large data sets by benefiting from indexing techniques to deploy efficient access on SPARQL endpoints [37].

Table 7: QB2OLAPem Usability Results

Tasks/User Group	SW	OLAP	Non-experts
Redefinition	100%	100%	100%
Enrichment	100%	75%	81.25%
Redefinition	1.88 min	1.5 min	6.15 min
Enrichment	2.75 min	4.75 min	9.47 min

Usability. In this context, we focus on the learnability and operability subcharacteristics. The results come from the experiments with users explained in Section 6.3. The ***learnability*** is measured with the percentage of different users that managed to use different tool functionalities. The first two rows of Table 7 show the percentage of users that successfully performed the redefinition and enrichment tasks with the tool, respectively. All the users successfully performed the redefinition. One OLAP expert and three non-expert users missed to perform the complete enrichment, although all of them performed at least one enrichment action. Thus, *all the users were capable of using both types of tool functionalities*. Moreover, the ***operability*** subcharacteristic is shown in the bottom two rows of the table that illustrate the average time that the different user types took for redefinition and enrichment, respectively. We can note that with higher expertise the necessary time for performing the tasks is reduced. Nevertheless, in all cases the redefinition and enrichment is at least 7 times faster than with the manual approach (see Section 6.3) showing *a high operability degree*.

Overall, the evaluation showed that QB2OLAP facilitates the enrichment and it is appreciated and considered needed and helpful by different users. The tool speeds up the enrichment process and guarantees the schema

Table 6: Market Capitalization Levels Statistics

Level	#Members	Members t	Avg Out-degree	#Properties	Properties t
indicator	1	0.01	16.00	16	0.01
refArea	120	0.02	19.57	2348	0.91
refPeriod	25	0.01	0.00	0	0.16
income-level	5	0.07	0.00	0	0.03
lending-type	4	0.07	0.00	0	0.02
region	7	0.07	0.00	0	0.04

soundness and completeness in practice.

7. Related Work

As mentioned in Section 1, 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 (and also in some sense related to the methodology we present in this paper) are the works by Nebot and Llavori [38] and Kämpgen and Harth [39]. The former proposes a semi-automatic method for on-demand extraction of semantic data into an 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 an MD model with such facts. They assume that data are represented as an OWL 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 [39] 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. 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 metadata 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 an MD model to allow OLAP operations over the underlying relational representation. Such model is expressed using XML for Analysis (XMLA)[36], 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 autonomous 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 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* [40], *Situational BI* [41], *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. [40] present a framework to support self-service BI, based on the notion of *fusion cubes*,

[36] http://xmlforanalysis.com

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 [14] aims at representing, using RDF, 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 [7] 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 (RollUp, Dice, and Slice), using SPARQL queries over data cubes specified using QB4OLAP. These two approaches have been thoroughly discussed in Sections 2 and 3. In [42], Ibragimov et al. 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.

Kämpgen et al. [43, 44] attempt to override the lack of structure in QB, discussed in Section 3, defining an OLAP data model on top of QB and other related vocabularies, e.g., some proposed ISO extensions to SKOS.[37]. They propose to represent each level as an instance of a class `skosclass:ClassificationLevel` and organize levels in hierarchies via stating the depth of each level in the hierarchy using the `skosclass:depth` property. The proposed representation restricts levels to participate in only one hierarchy and does not provide support for level attributes. They also propose a mechanism for implementing some OLAP operators over these extended cubes, using SPARQL queries, but restricting to data cubes with only one hierarchy per dimension.

Related to the SKOS extensions mentioned above, and realizing that SKOS is insufficient to represent the needs of statistical classifications and concept management, a new proposal to address those needs was issued, and denoted XKOS.[38] XKOS attenpts to capture the main semantic relationships like generalization, specialization, part-of, among other ones, with properties like `xkos:generalizes`, `xkos:generalizes`, `xkos:isPartOf`, respectively. At the time of writing this paper, this proposal is in a preliminary status.

For an exhaustive study of the possibilities of using SW technologies for OLAP, we refer the reader to the survey by Abelló et al. [33].

[37] http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS

[38] http://rdf-vocabulary.ddialliance.org/xkos.html

8. Conclusion

The approach presented in this paper opens new possibilities for performing OLAP analysis in Linked Data and SW contexts. After thoroughly elaborating on the significant benefits that QB4OLAP brings in terms of additional schema constructs that are necessary for the state-of-the-art OLAP analysis, we have elaborated on how to, as automatically as possible, introduce these enhancements into an existing QB data set. We have proposed the use of metadata to automate the association between measures and aggregate functions, and the algorithm for the detection of implicit MD semantics to automate the discovery of dimension hierarchy schema and instances, since these are the two most relevant semantic enhancements of QB4OLAP. The enrichment task is formalized in a semi-automatic method that defines steps described as SPARQL queries to create a new enriched QB4OLAP graph. Moreover, we have presented QB2OLAPem implementing the method and the algorithm for the detection of implicit MD semantics. QB2OLAPem enables the user to enrich a QB data set with minimal user efforts. Finally, the evaluation of our approach using three real-world QB data sets of different sizes demonstrates its feasibility in practice. Furthermore, the experiments conducted with 25 users show that, in practice, QB2OLAPem facilitates, speeds up, and guarantees the correct results of the enrichment process.

In the future, we plan to extend our approach to automatically identify the data heterogeneity cases and inspired by [45] explore the possibilities to integrate different QB schemata into a single QB4OLAP schema.

Acknowledgments

This research has been funded by the European Commission through the Erasmus Mundus Joint Doctorate "Information Technologies for Business Intelligence - Doctoral College" (IT4BI-DC) and it has been partially supported by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya. Alejandro Vaisman was partially supported by PICT-2014 Project 0787, funded by Argentina's Scientific Agency.

References

[1] L. I. Gómez, S. A. Gómez, A. A. Vaisman, A generic data model and query language for spatiotemporal OLAP cube analysis, in: EDBT, 2012, pp. 300–311.

[2] C. A. Hurtado, A. O. Mendelzon, A. A. Vaisman, Maintaining data cubes under dimension updates, in: ICDE, 1999, pp. 346–355.

[3] P. Vassiliadis, Modeling multidimensional databases, cubes and cube operations, in: SSDBM, 1998, pp. 53–62.

[4] T. Heath, C. Bizer, Linked Data: Evolving the Web into a Global Data Space, Morgan & Claypool Publishers, 2011.

[5] R. Cyganiak, D. Wood, M.Lanthaler, Resource description framework (RDF): Concepts and abstract syntax, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (2014).

[6] D. Brickley, R. Guha, RDF schema 1.1, http://www.w3.org/TR/rdf-schema/ (2014).

[7] L. Etcheverry, A. Vaisman, QB4OLAP: A vocabulary for OLAP cubes on the semantic web, in: COLD, 2012.

[8] E. Prud'hommeaux, A. Seaborne, SPARQL 1.1 Query Language for RDF (2011).
URL http://www.w3.org/TR/sparql11-query/

[9] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, C. Thomsen, QB2OLAP: Enabling OLAP on statistical linked open data, in: ICDE, 2016, In Press.

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

[11] C. Ciferri, R. Ciferri, L. Gómez, M. Schneider, A. Vaisman, E. Zimányi, Cube algebra: A generic user-centric model and query language for OLAP cubes, IJDWM 9 (2) (2013) 39–65.

[12] D. Beckett, T. Berners-Lee, Turtle - Terse RDF Triple Language (2011).
URL http://www.w3.org/TeamSubmission/turtle/

[13] SDMX, SDMX standards: Information model, http://sdmx.org/wp-content/uploads/2011/08/SDMX_2-1-1_SECTION_2_InformationModel_201108.pdf (2011).

[14] R. Cyganiak, D. Reynolds, The RDF Data Cube Vocabulary (W3C Recommendation), http://www.w3.org/TR/vocab-data-cube/ (January 2014).

[15] SDMX, Content Oriented Guidelines, http://sdmx.org/?page_id=11 (2009).

[16] A. A. Vaisman, Publishing OLAP cubes on the semantic web, eBISS 2015, LNBIP 253 (2016) 1–30.

[17] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis, Fundamentals of data warehouses, Springer, 2013.

[18] A. Y. Halevy, Why your data won't mix, ACM Queue 3 (8) (2005) 50–58.

[19] C. Batini, C. Cappiello, C. Francalanci, A. Maurino, Methodologies for data quality assessment and improvement, ACM Comput. Surv. 41 (3).

[20] H. Lenz, A. Shoshani, Summarizability in OLAP and statistical data bases, in: SSDBM, 1997, pp. 132–143.

[21] J. Varga, O. Romero, T. B. Pedersen, C. Thomsen, Towards next generation BI systems: The analytical metadata challenge, in: DaWaK, 2014, pp. 89–101.

[22] H. Garcia-Molina, J. D. Ullman, J. Widom, Database systems - the complete book (2. ed.), Pearson Education, 2009.

[23] O. Romero, D. Calvanese, A. Abelló, M. Rodriguez-Muro, Discovering functional dependencies for multidimensional design, in: DOLAP, 2009, pp. 1–8.

[24] J. Mazón, J. Lechtenbörger, J. Trujillo, A survey on summarizability issues in multidimensional modeling, Data Knowl. Eng. 68 (12) (2009) 1452–1469.

[25] O. Romero, A. Abelló, A framework for multidimensional design of data warehouses from ontologies, Data Knowl. Eng. 69 (11) (2010) 1138–1157.

[26] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The DL-Lite family and relations, J. Artif. Intell. Res. (JAIR) 36 (2009) 1–69.

[27] M. R. Jensen, T. Holmgren, T. B. Pedersen, Discovering multidimensional structure in relational data, in: DaWaK, 2004, pp. 138–148.

[28] T. B. Pedersen, C. S. Jensen, C. E. Dyreson, A foundation for capturing and querying complex multidimensional data, Inf. Syst. 26 (5) (2001) 383–423.

[29] F. M. Suchanek, S. Abiteboul, P. Senellart, PARIS: probabilistic alignment of relations, instances, and schema, PVLDB 5 (3) (2011) 157–168.

[30] J. Varga, O. Romero, T. B. Pedersen, C. Thomsen, SM4AM: A semantic metamodel for analytical metadata, in: DOLAP, 2014, pp. 57–66.

[31] I. ISO, Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models, International Organization for Standardization (2011) 34.

[32] O. Romero, A. Abelló, A survey of multidimensional modeling methodologies, IJDWM 5 (2) (2009) 1–23.

[33] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, A. Simitsis, Using semantic web technologies for exploratory OLAP: A survey, IEEE Trans. Knowl. Data Eng. 27 (2) (2015) 571–588.

[34] M. Saleem, A. N. Ngomo, Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation, in: The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings, 2014, pp. 176–191.

[35] R. Angles, A comparison of current graph database models, in: Workshops Proceedings of the IEEE 28th International Conference on Data Engineering, ICDE 2012, Arlington, VA, USA, April 1-5, 2012, 2012, pp. 171–177.

[36] L. Kowalik, Adjacency queries in dynamic sparse graphs, Inf. Process. Lett. 102 (5) (2007) 191–195.

[37] P. A. Boncz, O. Erling, M. Pham, Experiences with virtuoso cluster RDF column store, in: Linked Data Management., 2014, pp. 239–259.

[38] V. Nebot, R. B. Llavori, Building data warehouses with semantic web data, Decision Support Systems 52 (4) (2012) 853–868.

[39] B. Kämpgen, A. Harth, Transforming statistical linked data for use in OLAP systems, in: I-SEMANTICS, 2011, pp. 33–40.

[40] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J. Mazón, F. Naumann, T. B. Pedersen, S. Rizzi, J. Trujillo, P. Vassiliadis, G. Vossen, Fusion cubes: Towards self-service business intelligence, IJDWM 9 (2) (2013) 66–88.

[41] A. Löser, F. Hueske, V. Markl, Situational business intelligence, in: BIRTE, 2008, pp. 1–11.

[42] D. Ibragimov, K. Hose, T. Pedersen, Z. E, Towards exploratory OLAP over linked open data: A case study, in: BIRTE, 2014, pp. 114–132.

[43] B. Kämpgen, S. O'Riain, A. Harth, Interacting with Statistical Linked Data via OLAP Operations, in: ESWC (Satellite Events), 2012, pp. 87–101.

[44] B. Kämpgen, A. Harth, No size fits all - running the star schema benchmark with SPARQL and RDF aggregate views, in: ESWC, 2013, pp. 290–304.

[45] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, D. Mayorova, A requirement-driven approach to the design and evolution of data warehouses, Inf. Syst. 44 (2014) 94–119.

Appendix A. Enrichment Methodology

Here, we provide a fully formalized, more general, and detailed enrichment methodology, which is agnostic of the implementation decisions made. The methodology specifies the pre-conditions, post-conditions, and transformations to be conducted by each step in terms of set theory. The method presented in Section 5 is an instantiation of it.

The methodology considers the inputs listed below. These inputs can be generated from the outputs of Algorithm 1 (i.e., *allL*, *allP*, *allHS*, and *allLLA*) as follows. We can construct the dimension hierarchies (i.e., build structures like the one in Figure 2) and generate the inputs needed by the methodology by following the detected hierarchy steps (the *allHS* output of Algorithm 1). Each of these structures is considered a dimension (from here on, *dimension structure*) that has a name assigned to it, e.g., derived from the initial level name. A bottom-up traversal of the paths in this structure defines the hierarchies. On top of each dimension structure, the `All` level should be automatically added as the highest granularity level for all hierarchies. Thus, we obtain the inputs required for the enrichment methodology as follows.

- *NewLevSet*, the set of the new level IRIs (e.g., `schema:region`), generated as the difference between the set of all levels and the set of initial levels.
- *DimSet*, the set of dimension IRIs (e.g., `schema:geoDim`), generated to include an element for each dimension structure.
- *HierSet*, the set of hierarchy IRIs (e.g., `schema:geoHier`), generated for each distinct traversal path in the dimension structures.
- *LAttSet*, the set of new level attribute IRIs (e.g., `schema:capital`), extracted from the *allLLA* output of Algorithm 1.
- *HStepSet*, the set of hierarchy step IDs, i.e., IRIs or blank node identifiers (e.g., `_:hs1`), generated for each occurrence of a hierarchy step in hierarchies.
- *MapLAttribute2L*, a mapping of level attribute IRIs to level IRIs (e.g., (`schema:capital`, `sdmx-dimension:refArea`)), generated from the *allLLA* output of Algorithm 1, as pairs of IRIs.
- *MapH2D*, a mapping of hierarchy IRIs to dimension IRIs (e.g., (`schema:geoHier`, `schema:geoDim`)), extracted from the dimension structure as pairs of IRIs.
- *MapH2L*, a mapping of hierarchy IRIs to level IRIs (e.g., (`schema:geoHier`, `schema:region`)), extracted from the dimension structure as pairs of IRIs.
- *MapHStep2ParentL*, a mapping of hierarchy step IDs to parent level IRIs (e.g., (`_:hs1`, `schema:region`)), extracted from the dimension structure as pairs of IRIs.
- *MapHStep2ChildL*, a mapping of hierarchy step IDs to child level IRIs (e.g., (`_:hs1`, `sdmx-dimension:refArea`)), extracted from the dimension structure as pairs of IRIs.
- *MapHStep2H*, a mapping of hierarchy step IDs to hierarchy IRIs (e.g., (`_:hs1`, `schema:geoHier`)), extracted from the dimension structure as pairs of IRIs.
- *MapHStep2C*, a mapping of hierarchy step IDs to cardinality IRIs (e.g., (`_:hs1`, `qb4o:ManyToOne`)), extracted from the dimension structure as pairs of IRIs.
- *MapChild2Parent*, a mapping of child level member IRIs to parent level member IRIs (e.g., (`country:RS`, `region:ECS`)), extracted from the data set(s) as pairs of IRIs according to the hierarchy step structures.
- *MapLInstance2L*, a mapping of level member IRIs to level IRIs (e.g., (`country:RS`, `sdmx-dimension:refArea`)), extracted from the data set(s) as pairs of IRIs for all levels.
- *MapLInstance2LAInstance*, a mapping of level member IRIs to the level attribute IRI – level attribute value (that is IRI or literal) pairs (e.g., (`country:RS`, (`schema:capital`, `''Belgrade''xsd:string`))), extracted from the data set(s) based on the *MapLAttribute2L* mapping as, IRI – a pair of IRIs, or, IRI – an IRI and literal pair, pairs.

Taking advantage of the QB4OLAP vocabulary, we next propose a methodology to enrich a QB data set. The methodology defines the steps that need to be performed for the input QB data set in order to produce an output data set that is enriched with QB4OLAP semantics (e.g., dimension hierarchies). The methodology steps are fully automatized considering that the inputs discussed above are provided. Taking into account that the inputs generation involves some user actions, the overall enrichment process is semi-automatized. The methodology steps are:

1. Redefinition of the cube schema.
2. Specification of the aggregate functions.
3. Definition of the dimension hierarchies.
4. Annotation of the cube instances.

In this section, we first introduce preliminaries for the formal definition of each step. Then, each step is defined in terms of the input, tasks to be performed, and output that it produces.

Appendix A.1. Methodology Preliminaries

We first formally define a QB graph.

Definition 1. *A QB graph G_{qb} is a set of RDF triples, i.e., an RDF graph, defined as follows.*

- $G_{qb} = S_{qb} \cup I_{qb}$, *where* S_{qb} *and* I_{qb} *are sets of triples that define the QB cube schema and instances, respectively.*
- $S_{qb} = DS_{qb} \cup DSD_{qb} \cup D_{qb} \cup M_{qb}$, *where* DS_{qb}, DSD_{qb}, D_{qb}, *and* M_{qb} *are subsets of triples that define the cube data set, the cube structure, the cube dimensions, and the cube measures, respectively. Following QB's notation, the cube structure (i.e.,* DSD_{qb}*) is defined as a set of dimensions and measures (in QB terminology, the cube components).*
- $I_{qb} = DI_{qb} \cup O_{qb}$, *where* DI_{qb} *is the subset of triples that defines all dimension instances, while* O_{qb} *is the subset of triples that defines the observations. As discussed before, observations represent measure values for the fixed dimension instances determined by* DSD_{qb} *related to the data set* DS_{qb}.

The considered elements of the QB graph are the ones needed in our approach to create a QB4OLAP graph while the other ones are omitted. Triple examples of S_{qb} that are extracted from the running example (see Section 2.3) are presented in Example 23.

Example 23. *S_{qb} triples.*

```
1  <http://worldbank.270a.info/dataset/world−bank−indicators/structure>
2    a qb:DataStructureDefinition ;
3    qb:component  [ qb:dimension sdmx−dimension:refArea ] ;
4    qb:component [ qb:measure sdmx−measure:obsValue ] .
5  sdmx−dimension:refArea a qb:DimensionProperty .
6  sdmx−measure:obsValue a qb:MeasureProperty .
7  <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet ;
8    qb:structure
9      <http://worldbank.270a.info/dataset/world−bank−indicators/structure> .
```

Lines 1 - 4 belong to DSD_{qb}*, line 5 to* D_{qb}*, line 6 to* M_{qb}*, and line 7 to* DS_{qb}*. The data set* DS_{qb} *is related to the cube structure* DSD_{qb} *in lines 8 – 9 and this triple belongs to* DS_{qb}.

Triple examples of I_{qb} are presented in Example 24.

Example 24. *I_{qb} triples.*

```
1  data:world−bank−indicators/CM.MKT.LCAP.CD/RS/2012
2    a qb:Observation ;
3    sdmx−dimension:refArea country:RS ;
4    sdmx−measure:obsValue 7450560827.04874 ;
5    qb:dataSet <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> .
```

Lines 1 – 5 define an observation O_{qb}*. Line 3 specifies a dimension instance that belongs to* DI_{qb} *and line 4 defines a measure value of* O_{qb}*. Note that this observation relates to the cube schema structure (i.e.,* DSD_{qb}*) indirectly (see lines 7 – 9 of Example 23) via* `qb:dataSet` *in line 5.*

Analogously, we next define the output QB4OLAP graph G_{qb4o} in terms of sets of triples.

Definition 2. *A QB4OLAP graph G_{qb4o} is defined as follows.*

- $G_{qb4o} = S_{qb4o} \cup I_{qb4o}$, *where* S_{qb4o} *and* I_{qb4o} *are sets of triples that define the QB4OLAP cube schema and instances, respectively.*
- $S_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup D_{qb4o} \cup M_{qb4o} \cup L_{qb4o} \cup LA_{qb4o} \cup H_{qb4o} \cup HS_{qb4o} \cup AF_{qb4o} \cup C_{qb4o}$, *where the subsets of triples are:*
  - DS_{qb4o} *defining the cube data set;*
  - DSD_{qb4o} *defining the cube structure;*
  - D_{qb4o} *defining the cube dimensions;*
  - M_{qb4o} *defining the cube measures;*
  - L_{qb4o} *defining the dimension levels;*
  - LA_{qb4o} *defining the dimension level attributes;*
  - H_{qb4o} *defining the dimension hierarchies;*
  - HS_{qb4o} *defining the hierarchy steps;*
  - AF_{qb4o} *a predefined set of aggregate functions; and*
  - C_{qb4o} *a predefined set of possible cardinalities.*
- $I_{qb4o} = LI_{qb4o} \cup O_{qb4o} \cup LAI_{qb4o}$, *where* LI_{qb4o} *is the subset of triples that defines level instances and roll-up relationship instances between child and parent level instances,* O_{qb4o} *is the subset of triples that defines the observations, and* LAI_{qb4o} *is the subset of triples that defines level attribute values.*

Patterns of triples and examples of QB4OLAP sets are presented below in the methodology step definitions. The examples are based on the running example (see Section 2.3). By using this formalization we define our methodology considering the scenario where the input data set contains implicit MD semantics (e.g., a country is linked to a region) that is not explicitly stated, i.e., without the semantics that this is a rollup relationship in an MD hierarchy between a country level and a region level. Other scenarios are discussed in Section 5.4.

Appendix A.2. Redefinition of the cube schema

We start by building the new cube schema S_{qb4o}. For simplicity of presentation, when defining the steps we assume that from the input QB graph G_{qb}, we build a *new* QB4OLAP graph G_{qb4o}. We proceed incrementally and in the first step we build the schema S^1_{qb4o} (the complete cube schema S_{qb4o} is the output of Step 3). S^1_{qb4o} contains the sets DS_{qb4o} (defining the new cube data set), DSD_{qb4o} (defining the new cube schema structure), L_{qb4o} (defining the QB4OLAP dimension levels), and M_{qb4o} (defining the measures). We populate these sets from S_{qb}, starting from the dimensions in D_{qb} that are redefined as levels in L_{qb4o}. Next, we copy the measures from M_{qb} to M_{qb4o}. Then, we define the new cube

schema structure, assign it both levels and measures, and add it to the DSD_{qb4o}. Finally, we copy the data set definition to DS_{qb4o} and assign the new cube schema structure to it. The details of the step are presented below.

Step 1. ***Redefinition of the cube schema:***
INPUT: S_{qb}
OUTPUT: S^1_{qb4o}

Step 1.1. *Redefine input dimensions as levels*:

- $L_{qb4o} \cup= \{createLevel(d), \quad d \in D_{qb}\}$, where *createLevel* is a function redefining a dimension d as a level l, i.e., taking as argument a triple defining a *qb:DimensionProperty*, and producing a triple defining a *qb4o:LevelProperty*.

Triples pattern added to L_{qb4o}:
`qbDimensionIRI a qb4o:LevelProperty`, where `qbDimensionIRI` is the existing IRI of the QB dimension redefined as a QB4OLAP level. For instance, a triple related to the running example:

```
sdmx−dimension:refArea a qb4o:LevelProperty .
```

Step 1.2. *Copy input measures*:

- $M_{qb4o} \cup= \{m, \ m \in M_{qb}\}$, where m is a measure triple in M_{qb} that is added to M_{qb4o}. These triples are defining instances of the class *qb:MeasureProperty*.

Triples pattern added to M_{qb4o}:
`qbMeasureIRI a qb:MeasureProperty`, where `qbMeasureIRI` is the existing IRI of the QB measure copied. For instance, a triple related to the running example:

```
sdmx−measure:obsValue a qb:MeasureProperty .
```

Step 1.3. *Define the new cube schema structure and add to it levels and measures as components*:

- $DSD_{qb4o} \cup= \{createDSD()\}$, where *createDSD* is a function that creates a new cube schema structure triple, i.e., defining a new *qb:DataStructureDefinition*.

Triples pattern added to DSD_{qb4o}:
`dsdIRI a qb:DataStructureDefinition`, where `dsdIRI` is the newly defined IRI of the new schema structure definition. For instance, a triple related to the running example:

```
newG:newDSD a qb:DataStructureDefinition .
```

- $DSD_{qb4o} \cup= \{createComponent(lm), \ lm \in L_{qb4o} \cup M_{qb4o}\}$, where *createComponent* is a function that creates a schema structure component (in this case, a blank) node to which a level or measure is related. It receives a triple *lm* defining a *qb4o:LevelProperty* or a *qb:MeasureProperty* and produces a triple c defining a *qb:ComponentProperty*.

Triples pattern added to DSD_{qb4o}:
`dsdIRI qb:component [qb4o:level qbDimensionIRI]; dsdIRI qb:component [qb:measure qbMeasureIRI]`. Here, `dsdIRI`, `qbDimensionIRI`, and `qbMeasureIRI` are the IRIs previously introduced. Note that we include dsdIRI for better understanding. For instance, triples related to the running example:

```
newG:newDSD qb:component [ qb4o:level sdmx−dimension:refArea ] ;
            qb:component [ qb:measure sdmx−measure:obsValue ] .
```

Step 1.4. *Create the QB4OLAP cube schema*:

- $S^1_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup L_{qb4o} \cup M_{qb4o}$. Initially, we add to DS_{qb4o} the data set definition triple and the triple linking the data set to the DSD. Then, S^1_{qb4o} represents a union of DS_{qb4o} and the previous sets (i.e., DSD_{qb4o}, L_{qb4o}, and M_{qb4o}) with no additional triples pattern.

Triples pattern added to DS_{qb4o}:
`dsdIRI a qb:DataSet` and `dsdIRI qb:structure dsdIRI`, where `dsdIRI` and `dsdIRI` are the IRIs of the data set and the new schema structure definition, respectively. For instance, triples related to the running example:

```
<http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
                                    qb:structure newG:newDSD .
```

Thus, at this point, we have obtained S^1_{qb4o}.

Triple examples of S^1_{qb4o} are summed up in Example 25 to illustrate the overall result of Step 1. Note that we use the `newG` namespace for the new graph G_{qb4o}.

Example 25. *Resulting triples of Step 1.*

```
1 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2                                     qb:structure newG:newDSD .
3 newG:newDSD a qb:DataStructureDefinition ;
4   qb:component [ qb4o:level sdmx−dimension:refArea ] ;
5   qb:component [ qb:measure sdmx−measure:obsValue ] .
6 sdmx−dimension:refArea a qb4o:LevelProperty .
7 sdmx−measure:obsValue a qb:MeasureProperty .
```

*Lines 1 and 2 illustrate the output of **Step** 1.4. **Step** 1.3. results in lines 3, 4, and 5 define the new cube schema structure and add a level and a measure as components to it. Line 6 redefines the dimension from Example 23 as a QB4OLAP level (result of **Step** 1.1.), while line 7 illustrates the measure from Example 23 copied in **Step** 1.2.*

Appendix A.3. Specification of the aggregate functions

In this step we perform the first QB4OLAP enrichment by specifying an aggregate function for each measure. Note that possible aggregate functions are predefined by QB4OLAP. The inputs of this step are S^1_{qb4o}

and a mapping *MAggMap* from a measure $m \in M_{qb4o}$ to an aggregate function $a \in AF_{qb4o}$ (introduced in Section 4.2). The aggregate function is specified as a triple that relates a with the component of the cube schema structure related to m. This triple is added to DSD_{qb4o} and this enrichment is represented as the updated cube schema S^2_{qb4o} that is the step output. The details of the step are presented below.

Step 2. ***Specification of the aggregate functions***
INPUT: S^1_{qb4o}, *MAggMap*
OUTPUT: S^2_{qb4o}

Step 2.1. *Specify an aggregate function for each measure component of the cube schema structure*:

- $DSD_{qb4o} \cup = \{addAggFunction(getComponent(m), MAggMap(m)), m \in M_{qb4o}\}$, where:

– *getComponent* is a function that, given a measure m, returns the schema structure component c related to it (i.e., m is a triple defining a *qb:MeasureProperty* and c is a triple defining a *qb:ComponentProperty* (typically a blank node)).

– *MAggMap* is a mapping function from an input measure m (same as above) to the aggregate function a (i.e., a is a triple defining a *qb4o:AggregateFunction* (a predefined QB4OLAP aggregate function)).

– *addAggFunction* is a function that links an aggregate function a to the corresponding component c (a and c are the ones defined above).

Triples pattern added to DSD_{qb4o}:
`dsdIRI qb:component [qb:measure qbMeasureIRI; qb4o:aggregateFunction afIRI]`, where `dsdIRI` and `qbMeasureIRI` are the IRIs previously introduced and `afIRI` is the IRI of the aggregate function. Note that only `qb4o:aggregateFunction afIRI` is new in the pattern while the rest of the pattern refers to the earlier defined triples. For instance, triples related to the running example:

```
1 newG:newDSD qb:component
2       [ qb:measure sdmx−measure:obsValue ;
3       qb4o:aggregateFunction qb4o:sum ] .
```

Step 2.2. *Create new partial cube schema*:

- $S^2_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup L_{qb4o} \cup M_{qb4o}$. S^2_{qb4o} represents a union of updated DSD_{qb4o} and the DS_{qb4o}, L_{qb4o}, and M_{qb4o} sets with no additional triples pattern.

Triple examples of S^2_{qb4o} are presented in Example 26. It is a follow-up of the previous example (i.e., Example 25).

Example 26. *Resulting triples of Step 2.*

```
1 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> a qb:DataSet;
2 <http://worldbank.270a.info/dataset/CM.MKT.LCAP.CD> qb:structure
  newG:newDSD .
3 newG:newDSD a qb:DataStructureDefinition ;
4   qb:component  [ qb4o:level sdmx−dimension:refArea ] ;
5         qb:component  [ qb:measure sdmx−measure:obsValue ;
6                 qb4o:aggregateFunction qb4o:sum ] .
7 sdmx−dimension:refArea a qb4o:LevelProperty .
8 sdmx−measure:obsValue a qb:MeasureProperty .
```

Line 6 presents the aggregate function that is assigned to a measure by the grouping mechanism via a blank node. We use the SUM (i.e., `qb4o:sum`) aggregate function as an example, while in general it depends on the input mapping (i.e., MAggMap).

Appendix A.4. Definition of the dimension hierarchies

We now construct the dimension hierarchies. As explained in Section 3, QB4OLAP reuses the `qb:DimensionProperty`, however with different semantics than in QB: while in the latter, a dimension represents a point at a fixed granularity, QB4OLAP considers a dimension to contain points at different granularities. Therefore, in QB4OLAP, a QB dimension becomes a dimension level (see Step 1) and a dimension represents a set of levels that are hierarchically organized. Thus, *the final cube schema* S_{qb4o} is created by updating S^2_{qb4o} and adding of additional sets of triples defining the new semantics. For their definition we use certain additional inputs about the structure of dimensions (e.g., about new dimension levels and dimension hierarchies). In Section 4 we have shown how to compute the inputs to this step.

The general process is as follows. First, we update the set of levels L_{qb4o} obtained in Step 1, with the new levels that will define the dimension hierarchies. These new levels are specified in *NewLevSet* as a set of IRIs. Each level can have one or more level attributes defined in the LA_{qb4o} set of triples (see Definition 2) added to the final cube schema S_{qb4o}. They are specified by the input set of IRIs *LAttSet*. Once we have all the levels and their attributes, we use the input *DimSet* (a set of dimension IRIs) to create the new set of dimensions D_{qb4o} in the complete cube schema S_{qb4o}. Then, the set of hierarchies H_{qb4o} that represents the hierarchically ordered levels in the dimensions is added to S_{qb4o}. For this, the input *HierSet* (a set of hierarchy IRIs) is used. Finally, we add to S_{qb4o} the set of hierarchy steps HS_{qb4o} that represents parent-child relationships between two levels. For this, we use the input *HStepSet*, a set of hierarchy step IDs.

Once levels, level attributes, dimensions, hierarchies, and hierarchy steps are defined, we must correlate them using the QB4OLAP properties. For this, we use the

mappings *MapLAttribute2L* through *MapLInstance2L* in the input list (see Section 4.3). The *MapLAttribute2L* mapping associates level attributes with levels (defining both, the "direct" association, and its "inverse", e.g., telling that a level has an attribute, and that an attribute belongs to a dimension level, respectively); *MapH2D* links hierarchies with dimensions; *MapH2L* links hierarchies with levels; *MapHStep2ParentL* associates a hierarchy step with its parent level, while *MapHStep2ChildL* does the same with its child level. Then, *MapHStep2H* associates a hierarchy step with the hierarchy and *MapHStep2C* links a hierarchy step with its corresponding cardinality. Finally, *MapChild2Parent* associates child level members with their parent level members while *MapLInstance2L* associates level members to their levels.

The output of this step is *the complete cube schema* S_{qb4o}. The details of the step are presented below.

Step 3. **Definition of the dimension hierarchies**
INPUT: S^2_{qb4o}, *NewLevSet*, *DimSet*, *HierSet*, *LAttIN*, *HStepSet*, *MapLAttribute2L*, *MapH2D*, *MapH2L*, *MapHStep2ParentL*, *MapHStep2ChildL*, *MapHStep2H*, *MapHStep2C*, *MapChild2Parent*, *MapLInstance2L*
OUTPUT: S_{qb4o}

Step 3.1. *Define new levels*:

- $L_{qb4o} \cup= \{createNewLevel(newL), \ newL \in NewLevSet\}$, where *createNewLevel* is a function that, for each level IRI in *NewLevSet*, produces the corresponding triple using the *qb4o:LevelProperty*.

Triples pattern added to L_{qb4o}:
`lIRI a qb4o:LevelProperty`, where `lIRI` is the IRI of the new level. For instance, a triple related to the running example:

```
newG:region a qb4o:LevelProperty .
```

Step 3.2. *Define level attributes*:

- $LA_{qb4o} \cup= \{createLevelAttribute(attr), \ attr \in LAttSet\}$, where *createLevelAttribute* is a function that, for each attribute IRI in *LAttSet*, produces the corresponding triple using the *qb4o:LevelAttribute*.

Triples pattern added LA_{qb4o}:
`laIRI a qb4o:LevelAttribute`, where `laIRI` is the IRI of the new level attribute. For instance, a triple related to the running example:

```
newG:label a qb4o:LevelAttribute .
```

Step 3.3. *Define dimensions*:

- $D_{qb4o} \cup= \{createDimension(dim), \ dim \in DimSet\}$, where *createDimension* is a function that, for each dimension IRI in *DimSet*, produces the corresponding triple using the *qb:DimensionProperty*.

Triples pattern added D_{qb4o}:
`dIRI a qb:DimensionProperty`, where `dIRI` is the IRI of the new dimension. For instance, a triple related to the running example:

```
newG:geoDimension a qb:DimensionProperty .
```

Step 3.4. *Define hierarchies*:

- $H_{qb4o} \cup= \{createHierarchy(hier), \ hier \in HierSet\}$, where *createHierarchy* is a function that, for each hierarchy IRI in *HierSet*, produces the corresponding triple using the *qb4o:Hierarchy*.

Triples pattern added H_{qb4o}:
`hIRI a qb4o:Hierarchy`, where `hIRI` is the IRI of the new hierarchy. For instance, a triple related to the running example:

```
newG:geoHierarchy a qb4o:Hierarchy .
```

Step 3.5. *Define hierarchy steps*:

- $HS_{qb4o} \cup= \{createHStep(hStep), \ hStep \in HStepSet\}$, where *createHStep* is a function that, for each hierarchy step ID in *HStepSet*, produces the corresponding triple using the *qb4o:HierarchyStep*. Since QB uses blank nodes for representing reification, we follow the same principle and use blank node identifiers for hierarchy steps in the examples.

Triples pattern added HS_{qb4o}:
`hsID a qb4o:HierarchyStep`, where `hsID` is the ID of the new hierarchy step. For instance, a triple related to the running example:

```
_:newHierarchyStep a qb4o:HierarchyStep .
```

Step 3.6. *Associate dimension levels with level attributes*:

- $L_{qb4o} \cup= \{linkLevelWithAttr(MapLAttribute2L(la), la), \ la \in LA_{qb4o}\}$, where *linkLevelWithAttr* is a function that receives a pair (l, la), where l is an instance of *qb4o:LevelProperty* and *la* is an instance of *qb4o:LevelAttribute*, and produces a triple *lla* telling that the level l has the level attribute *la*. *MapLAttribute2L* is a mapping that, given a level attribute, returns the level to which the level attribute belongs.

Triples pattern added L_{qb4o}:
`lIRI qb4o:hasAttribute laIRI`, where `lIRI` and `laIRI` are the IRIs of the level and level attribute, respectively. For instance, a triple related to the running example:

```
newG:region qb4o:hasAttribute newG:label .
```

• $LA_{qb4o} \cup= \{linkAttrWithLevel(la, MapLAttribute2L(la)), la \in LA_{qb4o}\}$, where *linkAttrWithLevel* is a function that receives a pair (la, l), where *la* and *l* have the same meaning as above, and produces a triple *lal* telling that the level attribute *la* belongs to the level *l*.

Triples pattern added LA_{qb4o}:
`laIRI qb4o:inLevel lIRI`, where `laIRI` and `lIRI` are the IRIs of the level attribute and level, respectively. For instance, a triple related to the running example:

```
newG:label qb4o:inLevel newG:region .
```

Step 3.7. *Associate dimensions with hierarchies*:

• $D_{qb4o} \cup= \{linkDimWithHier(MapH2D(h), h), h \in H_{qb4o}\}$, where *linkDimWithHier* is a function that receives a pair (d, h), where *d* is an instance of *qb:DimensionProperty* and *h* is an instance of *qb4o:Hierarchy*, and produces a triple *dh* telling that the dimension *d* has the hierarchy *h*. *MapH2D* is a mapping that, given a hierarchy, returns the dimension to which the hierarchy belongs.

Triples pattern added D_{qb4o}:
`dIRI qb4o:hasHierarchy hIRI`, where `dIRI` and `hIRI` are the IRIs of the dimension and hierarchy, respectively. For instance, a triple related to the running example:

```
newG:geoDimension qb4o:hasHierarchy newG:geoHierarchy .
```

• $H_{qb4o} \cup= \{linkHierWithDim(h, MapH2D(h)), h \in H_{qb4o}\}$, where *linkHierWithDim* is a function that receives a pair (h, d), where *h* and *d* have the same meaning as above, and produces a triple *hd* telling that the hierarchy *h* belongs to the dimension *d*.

Triples pattern added H_{qb4o}:
`hIRI qb4o:inDimension dIRI`, where `hIRI` and `dIRI` are the IRIs of the hierarchy and dimension, respectively. For instance, a triple related to the running example:

```
newG:geoHierarchy qb4o:inDimension newG:geoDimension .
```

Step 3.8. *Associate hierarchies with levels*:

• $H_{qb4o} \cup= \{linkHierWithL(h, MapH2L(h)), h \in H_{qb4o}\}$, where *linkHierWithL* is a function that receives a pair (h, l), where *h* is an instance of *qb4o:Hierarchy* and *l* is an instance of *qb4o:LevelProperty*, and produces a triple *hl* telling that the hierarchy *h* has the level *l*. *MapH2L* is a mapping that, given a hierarchy, returns the level(s) it contains.

Triples pattern added H_{qb4o}:
`hIRI qb4o:hasLevel lIRI`, where `hIRI` and `lIRI` are the IRIs of the hierarchy and level, respectively. For instance, triples related to the running example:

```
newG:geoHierarchy qb4o:hasLevel newG:region .
newG:geoHierarchy qb4o:hasLevel sdmx−dimension:refArea .
```

Step 3.9. *Associate hierarchy steps with the hierarchy, levels, and cardinalities*:

• $HS_{qb4o} \cup= \{linkHStepWithH(hs, MapHStep2H(hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithH* is a function that receives a pair (hs, h), where *hs* is an instance of *qb4o:HierarchyStep* and *h* is an instance of *qb4o:Hierarchy*, and produces a triple *hsh* telling that the hierarchy step *hs* belongs to the hierarchy *h*. *MapHStep2H* is a mapping that, given a hierarchy step, returns the related hierarchy.

Triples pattern added HS_{qb4o}:
`hsID qb4o:inHierarchy hIRI`, where `hsID` and `hIRI` are the hierarchy step ID and the hierarchy IRI, respectively. For instance, a triple related to the running example:

```
:newHierarchyStep qb4o:inHierarchy newG:geoHierarchy .
```

• $HS_{qb4o} \cup= \{linkHStepWithParentL(hs, MapHStep2ParentL(hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithParentL* is a function that receives a pair (hs, pl), where *hs* is an instance of *qb4o:HierarchyStep* and *pl* is an instance of *qb4o:LevelProperty*, and produces a triple *hspl* telling that the hierarchy step *hs* has *pl* as parent level. *MapHStep2ParentL* is a mapping that, given a hierarchy step, returns the related parent level.

Triples pattern added HS_{qb4o}:
`hsID qb4o:parentLevel plIRI`, where `hsID` and `plIRI` are the hierarchy step ID and the parent level IRI, respectively. For instance, a triple related to the running example:

```
:newHierarchyStep qb4o:parentLevel newG:region .
```

• $HS_{qb4o} \cup= \{linkHStepWithChildL(hs, MapHStep2ChildL(hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithChildL* is a function that receives a pair (hs, cl), where *hs* is an instance of *qb4o:HierarchyStep* and *cl* is an instance of *qb4o:LevelProperty*, and produces a triple *hscl* telling that the hierarchy step *hs* has *cl* as child level. *MapHStep2ChildL* is a mapping that, given a hierarchy step, returns the related child level.

Triples pattern added HS_{qb4o}:
`hsID qb4o:childLevel clIRI`, where `hsID` and

`clIRI` are the hierarchy step ID and the child level IRI, respectively. For instance, a triple related to the running example:

```
_:newHierarchyStep qb4o:childLevel sdmx−dimension:refArea .
```

- $HS_{qb4o} \cup= \{linkHStepWithC(hs, \quad MapHStep2C(hs)), hs \in HS_{qb4o}\}$, where *linkHStepWithC* is a function that receives a pair (hs, c), where *hs* is an instance of *qb4o:HierarchyStep* and *c* is an instance of *qb4o:Cardinality*, and produces a triple *hsc* telling that the hierarchy step *hs* has *c* as cardinality. *MapHStep2C* is a mapping that, given a hierarchy step, returns the related cardinality.

Triples pattern added HS_{qb4o}: `hsID qb4o:pcCardinality cIRI`, where `hsID` and `cIRI` are the hierarchy step ID and the cardinality IRI, respectively. For instance, a triple related to the running example:

```
_:newHierarchyStep qb4o:pcCardinality qb4o:ManyToOne .
```

Step 3.10. *Create the complete cube schema*:

- $S_{qb4o} = DS_{qb4o} \cup DSD_{qb4o} \cup D_{qb4o} \cup M_{qb4o} \cup L_{qb4o} \cup LA_{qb4o} \cup H_{qb4o} \cup HS_{qb4o} \cup AF_{qb4o} \cup C_{qb4o}$. S_{qb4o} represents a union of all its subsets with no additional triples pattern. Note that AF_{qb4o} and C_{qb4o} are predefined by QB4OLAP.

As we have already said, the output of this step is the complete cube schema S_{qb4o}. Triple examples of new triples in S_{qb4o} are summed up in Example 27.

Example 27. *Resulting triples of Step 3.*

```
1   #Create a level, a level attribute, a dimension, a hierarchy,
2   #and a hierarchy step
3   newG:region a qb4o:LevelProperty .
4   newG:label a qb4o:LevelAttribute .
5   newG:geoDimension a qb:DimensionProperty .
6   newG:geoHierarchy a qb4o:Hierarchy .
7   _:newHierarchyStep a qb4o:HierarchyStep .
8
9   #Link the level and level attribute
10  newG:region qb4o:hasAttribute newG:label .
11  newG:label qb4o:inLevel newG:region .
12
13  #Link dimensions and hierarchies
14  newG:geoDimension qb4o:hasHierarchy newG:geoHierarchy .
15  newG:geoHierarchy qb4o:inDimension newG:geoDimension .
16
17  #Link the hierarchy and levels, and hierarchy step
18  # with all related instances
19  newG:geoHierarchy qb4o:hasLevel newG:region .
20  newG:geoHierarchy qb4o:hasLevel sdmx−dimension:refArea .
21  _:newHierarchyStep qb4o:inHierarchy newG:geoHierarchy .
22  _:newHierarchyStep qb4o:parentLevel newG:region .
23  _:newHierarchyStep qb4o:childLevel sdmx−dimension:refArea .
24  _:newHierarchyStep qb4o:pcCardinality qb4o:ManyToOne .
```

Line 3 shows a triple defining a new level, as a result of ***Step*** *3.1. Analogously, line 4 shows a triple defining a new level attribute as a result of* ***Step*** *3.2. Lines 5 and 6, define a dimension and a hierarchy, resulting from applying* ***Step*** *3.3. and* ***Step*** *3.4., respectively. Then, line 7 defines a hierarchy step as a result of* ***Step*** *3.5. Lines 10 – 11 link a level and a level attribute as the product of* ***Step*** *3.6., and lines 14 – 15 link a dimension and a hierarchy showing the result of* ***Step*** *3.7. Finally, lines 19 – 20 associate a hierarchy with its levels as results of* ***Step*** *3.8. while lines 21 – 24 link a hierarchy step and its hierarchy, levels, and cardinality as the triples produced in* ***Step*** *3.9.*

Appendix A.5. Annotation of the cube instances

Once the new cube schema is defined, we need to link the level members with their schema definitions, i.e., populate the dimension level instances. Moreover, we link the level members with their level attribute instances and copy the observations to the new QB4OLAP graph. The first input of the step is *MapChild2Parent* that maps child level members to their parent level members. Next input is *MapLInstance2L* that maps level members to their levels. The *MapLInstance2LAInstance* input maps level members and level attribute instances. Then, the set of cube instances $I_{qb} = DI_{qb} \cup O_{qb}$, where DI_{qb} defines dimension instances and O_{qb} defines observations. The last input is the complete new cube schema S_{qb4o}. The output of the step is the set of new cube instances $I_{qb4o} = LI_{qb4o} \cup O_{qb4o} \cup LAI_{qb4o}$. The details of the step are presented below.

Step 4. **Annotation of the cube instances**
INPUT: *MapChild2Parent*, *MapLInstance2L*, *MapLInstance2LAInstance*, I_{qb}, S_{qb4o}
OUTPUT: I_{qb4o}

Step 4.1. *Copy dimension instances from* G_{qb} *as base level instances in* G_{qb4o}:

- $LI_{qb4o} \cup= \{copyAsLevelInstance(di), \quad di \in DI_{qb}\}$, where *copyAsLevelInstance* is a function that receives a triple *di* representing a dimension instance in QB and returns a triple *li* defining the subject (i.e., an IRI) of the triple *di* as a level instance in QB4OLAP, using the *qb4o:LevelMember* property.

Triples pattern added LI_{qb4o}: `liIRI a qb4o:LevelMember`, where `liIRI` is the IRI of the level instance obtained from *di*. For instance, a triple related to the running example:

```
country:RS a qb4o:LevelMember .
```

Step 4.2. *Add coarser granularity level instances*:

• $LI_{qb4o} \cup = \{MapChild2Parent(li), \quad li \in LI_{qb4o}\}$, where *MapChild2Parent* is a mapping that, for a given dimension level instance, returns its corresponding parent level member. This object is then defined as a level instance, using the *qb4o:LevelMember* property.

Triples pattern added LI_{qb4o}:
`liIRI a qb4o:LevelMember`, where `liIRI` is the IRI of the new level instance, returned by *MapChild2Parent*. For instance, a triple related to the running example:

```
region:ECS a qb4o:LevelMember .
```

Step 4.3. *Copy observations*:

• $O_{qb4o} \cup = \{o, o \in O_{qb}\}$, where o is an observation from O_{qb} that is added to O_{qb4o}.

Triples pattern added O_{qb4o}:
`oIRI a qb:Observation`, `oIRI qb:dataSet dsIRI`, `oIRI lIRI liIRI`, and `oIRI mIRI mvalue`, where `oIRI`, `dsIRI`, `lIRI`, `liIRI`, and `mIRI` are the IRIs of the observation, data set, level, level instance, and measure, respectively, while `mvalue` is a literal representing measure value. For instance, triples related to the running example:

```
<http://worldbank.270a.info/dataset/world−bank−indicators/
                                        CM.MKT.LCAP.CD/RS/2012>
  a qb:Observation ;
  qb:dataSet newG:newDS ;
  property:indicator indicator:CM.MKT.LCAP.CD ;
  sdmx−dimension:refArea country:RS ;
  sdmx−measure:obsValue 7450560827.04874 ;
```

Step 4.4. *Specify the level for each level instance*:

• $LI_{qb4o} \cup = \{linkToLevel(li, MapLInstance2L(li)), \ li \in LI_{qb4o}\}$, where *linkToLevel* is a function that receives a pair (li, l), where li is an instance of *qb4o:LevelMember* and l is an instance of *qb4o:LevelProperty*, and produces a triple *lil* telling that the level member *li* belongs to the level *l*. *MapLInstance2L* is a mapping that, given a level instance, returns the level it belongs to.

Triples pattern added LI_{qb4o}:
`liIRI qb4o:memberOf lIRI`, where `liIRI` and `lIRI` are the IRIs of the level member and level, respectively. For instance, a triple related to the running example:

```
country:RS qb4o:memberOf sdmx−dimension:refArea .
```

Step 4.5. *Specify rollup (i.e., parent–child) relations between level instances*:

• $LI_{qb4o} \cup = \{linkRollUps(MapChild2Parent(li), \ li), \ li \in LI_{qb4o}\}$, where *linkRollUps* is a function that receives a pair (pli, cli), where *pli* and *cli* are instances of *qb4o:LevelMember*, and produces a triple *pcli* telling that *cli* rolls-up to *pli* using the *skos:broader* property.

Triples pattern added LI_{qb4o}:
`cliIRI skos:broader pliIRI`, where `cliIRI` and `pliIRI` are the IRIs of the child and parent level instances, respectively. For instance, a triple related to the running example:

```
country:RS skos:broader region:ECS .
```

Step 4.6. *Add level attribute instances*:

• $LAI_{qb4o} \cup = \{addLevelAttInstance(li, MapLInstance2LAInstance(li)), \ li \in LI_{qb4o}\}$, where *addLevelAttInstance* is a function that receives *li* and a pair (la, lai), where *li* is an instance of *qb4o:LevelMember*, *la* is an instance of *qb4o:LevelAttribute*, and *lai* is a level attribute value (IRI or literal), and produces a triple *lilalai* telling that *li* has an attribute *la* with the value *lai*. *MapLInstance2LAInstance* is a mapping that, for a given dimension level instance, returns its level attribute – level attribute value pair(s).

Triples pattern added LAI_{qb4o}:
`liIRI laIRI laiIRI` or `liIRI laIRI laiLiteral`, where `liIRI`, `laIRI`, and `laiIRI` are the IRIs of the level instance, level attribute, and level attribute value, respectively, and `laiLiteral` is a literal representing level attribute value. For instance, a triple related to the running example:

```
country:RS schema:capital "Belgrade"xsd:string .
```

Step 4.7. *Create new cube instances*:

• $I_{qb4o} = LI_{qb4o} \cup O_{qb4o} \cup LAI_{qb4o}$. I_{qb4o} represents a union of LI_{qb4o} (i.e., the level members), O_{qb4o} (i.e., observations), and LAI_{qb4o} (i.e., the level attribute instances) with no additional triples pattern.

The output of this step is I_{qb4o}. Triple examples of I_{qb4o} are summed up in Example 28. This example follows our running example and is an extension of previous ones.

Example 28. *Resulting triples of Step 4.*

```
country:RS a qb4o:LevelMember .
region:ECS a qb4o:LevelMember .
<http://worldbank.270a.info/dataset/world−bank−indicators/
                                        CM.MKT.LCAP.CD/RS/2012>
  a qb:Observation ;
  qb:dataSet newG:newDS ;
  property:indicator indicator:CM.MKT.LCAP.CD ;
  sdmx−dimension:refArea country:RS ;
  sdmx−measure:obsValue 7450560827.04874 ;

country:RS qb4o:memberOf sdmx−dimension:refArea .
country:RS skos:broader region:ECS .
country:RS schema:capital "Belgrade"xsd:string .
```

Result examples of ***Step*** *4.1. and* ***Step*** *4.2. are illustrated in lines 1 and 2, respectively. Lines 3 – 9 illustrate copying of the part of observation from Example 2 as result example of* ***Step*** *4.3. Then, line 11 presents the result example of* ***Step*** *4.4. Finally, line 12 illustrates the result example of* ***Step*** *4.5. and line 13 the result example of* ***Step*** *4.6.*