Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP

Lorena Etcheverry[1], Alejandro Vaisman[2], and Esteban Zimányi[3]

[1] Universidad de la República, Uruguay
lorenae@fing.edu.uy
[2] Instituto Tecnológico de Buenos Aires, Argentina
avaisman@itba.edu.ar
[3] Université Libre de Bruxelles, Belgium
ezimanyi@ulb.ac.be

Abstract. The web is changing the way in which data warehouses are designed and exploited. Nowadays, for many data analysis tasks, data contained in a conventional data warehouse may not suffice, and external data sources, like the web, can provide useful multidimensional information. Also, large repositories of semantically annotated data are becoming available on the web, opening new opportunities for enhancing current decision-support systems. Representation of multidimensional data via semantic web standards is crucial to achieve such goal. In this paper we extend the QB4OLAP RDF vocabulary to represent balanced, recursive, and ragged hierarchies. We also present a set of rules to obtain a QB4OLAP representation of a conceptual multidimensional model, and a procedure to populate the result from a relational implementation of the multidimensional model. We conclude the paper showing how complex real-world OLAP queries expressed in SPARQL can be posed to the resulting QB4OLAP model.

1 Introduction

The web is changing the way in which data warehouses (DW) are designed, used, and exploited [4]. For some data analysis tasks (like worldwide price evolution of some product), the data contained in a conventional data warehouse may not suffice. The web can provide useful multidimensional information, although usually too volatile to be permanently stored [1]. Further, the advent of initiatives such as Open Data[1] and Open Government promotes publishing multidimensional data using standards and non-proprietary formats[2]. Also, the Linked Data paradigm allows sharing and reusing data on the web by means of semantic web (SW) standards [8]. Domain ontologies expressed in RDF[3], or in languages built on top of RDF like RDF-S or OWL, define a common terminology for the concepts involved in a particular domain. In spite of the above, although in the last decade several open-source BI platforms have emerged, they still do not provide an open format to publish and share cubes among organizations [7], and the most popular commercial Business Intelligence (BI) tools are still proprietary.

[1] https://okfn.org/opendata/
[2] http://opengovdata.org/
[3] http://www.w3.org/TR/rdf-concepts/

Usually, in a relational DW representation, a conceptual model is implemented at the logical level as a collection of tables organized in specialized structures, basically star and snowflake schemas, which relate a fact table to several dimension tables through foreign keys. In a semantic web DW scenario, the logical model becomes the RDF data model. The first proposal of an RDF vocabulary to cover many of the multidimensional model components was the QB4OLAP vocabulary [5,6]. In this paper, after a brief introduction to semantic web concepts (Section 2) and related work (Section 3), we present an extension of QB4OLAP that supports the most used model characteristics, like balanced, recursive, ragged, and many-to-many hierarchies (Section 4). We then propose a mechanism to translate a conceptual multidimensional model into a logical RDF model using the QB4OLAP vocabulary (Section 5), and show how we can transform an existent relational implementation of a DW into QB4OLAP via an R2RML mapping. Finally, we show how QB4OLAP cubes can be queried using SPARQL (Section 6), and discuss open challenges and future work (Section 7) .

2 Preliminary Concepts

RDF and SPARQL. The Resource Description Framework (*RDF*) allows expressing assertions over resources identified by an Internationalized Resource Identifier (IRI) as triples of the form *subject - predicate - object*, where *subject* are always resources, and *predicate* and *object* could be resources or strings. *Blank nodes* are used to represent anonymous resources or resources without an IRI, typically with a structural function, e.g., to group a set of statements. Data values in RDF are called *literals* and can only be *objects*. A set of RDF triples can be seen as a directed graph where *subject* and *object* are nodes, and *predicates* are arcs. Usually, triples representing schema and instance data coexist in RDF datasets. A set of reserved words defined in RDF Schema (called the RDF-S vocabulary) is used to define classes, properties, and hierarchical relationships. Many formats for RDF serialization exist. In this paper we use Turtle [4].

SPARQL 1.1[5] is the current 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, composed by *basic graph patterns (BGP)*. The '.' operator represents the conjunction of graph patterns. Relevant to our study, SPARQL supports aggregate functions and the `GROUP BY` clause.

R2RML[6] is a language for expressing mappings from relational databases to RDF datasets, allowing representing relational data in RDF using a customized structure and vocabulary. Both, R2RML mapping documents (written in Turtle syntax) and mapping results, are RDF graphs. The main object of an R2RML mapping is the *triples map*, a collection of triples composed of a *logical table*, a *subject map*, and one or more *predicate object maps*. A logical table is either a base table or a view (using the predicate

[4] `http://www.w3.org/TeamSubmission/turtle/`
[5] `http://www.w3.org/TR/sparql11-query/`
[6] `http://www.w3.org/TR/r2rml/`

`rr:tableName`), or an SQL query (using the predicate `rr:sqlQuery`). A predicate object map is composed of a predicate map and an object map. Subject maps, predicate maps, and object maps are either constants (`rr:constant`), column-based maps (`rr:column`), or template-based maps (`rr:template`). Templates use column names as placeholders. Foreign keys are handled referencing object maps, which use the subjects of another triples map as the objects generated by a predicate-object map. A set of R2RML mappings can either be used to generate a static set of triples that represent the underlying relational data (*data materialization*) or to provide a non-materialized RDF view of the relational data (*on-demand mapping*).

3 Related Work

At least two approaches are found concerning OLAP analysis of SW data. In a nutshell, the first one consists on extracting multidimensional data from the SW and loading them into traditional OLAP repositories. The second one consists in performing OLAP-like analysis directly over SW data, e.g., over multidimensional data represented in RDF.

Along the first line of research are the works by Nebot et al. [14] and Kämpgen et al. [9]. The former proposes a semi-automatic method for on-demand extracting semantic data into a multidimensional database, where data will be exploited. Kämpgen et al. follow a similar approach, although restricted to a particular kind of SW data. They explore the extraction of statistical data, published using the RDF Data Cube vocabulary (QB)[7], into a multidimensional database. Thus, these two approaches allow using the existent knowledge in this area, and reusing available tools, at the expense of requiring the existence of a local DW to store the semantic web data extracted. This constraint clashes with the autonomous and high volatile nature of web data sources.

The second line of research tries to overcome this restriction, exploring data models and tools that allow publishing and analyzing multidimensional data directly over the SW. The aim of this models is to support concepts like *self-service BI*, *situational BI* [12], and *on-demand BI*, in order to take advantage of web data to enrich decision-making processes. Abello et al. [1] envision a framework to support self-service BI, based on the notion of *fusion cubes*, i.e., multidimensional cubes that can be dynamically extended both in their schema and their instances. Beheshti et al. [2] propose to extend SPARQL to express OLAP queries. However, we believe that not working with standard languages and established data models limits the applicability of the approach. With a different approach, Kämpgen et al. [10,11] proposed an OLAP data model on top of QB and other related vocabularies, and a mechanism to implement OLAP operators over these extended cubes, using SPARQL queries. This approach inherits the drawbacks of QB, that is, multidimensional modeling possibilities are limited.

4 RDF Representation of Multidimensional Data

The QB4OLAP vocabulary[8] extends the QB vocabulary mentioned in Section3, to enhance the support to the multidimensional model, overcoming several limitations of

[7] `http://www.w3.org/TR/vocab-data-cube/`
[8] `http://purl.org/qb4olap/cubes`

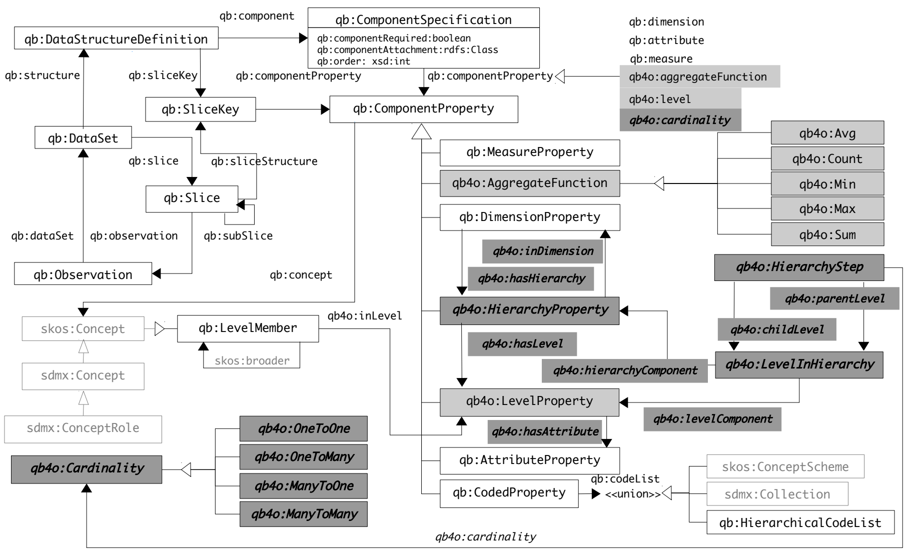
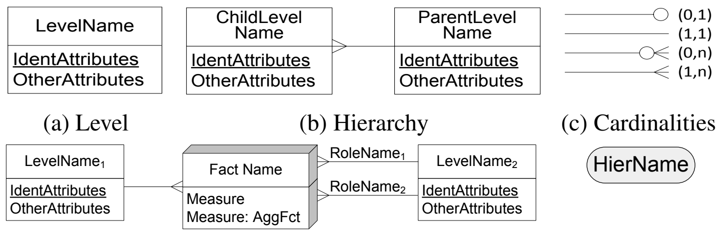


Fig. 1. A new version of the QB4OLAP vocabulary

QB [6]. Unlike QB, QB4OLAP allows implementing the main OLAP operations, such as rollup, slice, and dice, using standard SPARQL queries. Two different kinds of sets of RDF triples are needed to represent a data cube in QB4OLAP: (i) the *cube schema*, and (ii) the *cube instances*.The former defines the structure of the cube, in terms of dimension levels and measures, but also defines the hierarchies within dimensions and the parent-child relationships between levels. These metadata can then be used to automatically produce SPARQL queries that implement OLAP operations [6], which is not possible in QB. Cube instances are sets of triples that represent level members, facts and measured values. Several cube instances may share the same cube schema. Figure 1 depicts the QB4OLAP vocabulary. We can see that QB4OLAP embeds QB, allowing data cubes published using QB to be represented using QB4OLAP without affecting existing applications. Original QB terms are prefixed with `qb:`. Capitalized terms represent RDF classes and noncapitalized terms represent RDF properties. Classes in external vocabularies are depicted in light gray font. QB4OLAP classes and properties are prefixed with `qb4o`.

The previous version of QB4OLAP had some limitations regarding the representation of dimension hierarchies. For example, it did not allow more than one hierarchy per dimension, or to represent cardinalities in the relationships between level members. To overcome these limitations we have introduced new classes and properties, depicted in Fig. 1 in dark gray background and bold italics; existent QB4OLAP classes and properties are depicted in light gray background.

A data structure definition (DSD) specifies the schema of a data set of the class `qb:DataSet`. The DSD can be shared among different data sets, and has properties for representing dimensions, dimension levels, measures, and attributes, called



(a) Level (b) Hierarchy (c) Cardinalities

(d) Fact with measures and associated levels (e) Hierarchy name

Fig. 2. Notation of the MultiDim model

qb:dimension, qb:measure, qb4o:level, and qb:attribute, respectively. Observations (facts) represent points in a multidimensional space. An observation is linked to a value in each dimension level of the DSD using instances of qb4o:LevelProperty. This class models dimension levels. Instances of the class qb4o:LevelMember represent level members, and relations between them are expressed using the skos:broader property. Dimension hierarchies are defined via the class qb4o:HierarchyProperty. The relationship between dimensions and hierarchies is represented via the property qb4o:hasHierarchy and its inverse qb4o:inDimension. A level may belong to different hierarchies, and in each hierarchy it may have a different parent level. Also, the relationships between level members may have different cardinalities (e.g. one-to-many, many-to-many, etc.). The qb4o:LevelInHierarchy class represents pairs of hierarchies and levels, and properties are provided to relate a pair with its components: qb4o:hierarchyComponent and qb4o:levelComponent. A parent-child relationship between two levels is represented using the class qb4o:HierarchyStep and the properties qb4o:childLevel and qb4o:parentLevel. The cardinality of this relationship is represented via the qb4o:cardinality property and members of the qb4o:Cardinality class. This property can also be used to represent the cardinality of the relationship between a fact and a level. QB4OLAP also allows to define level attributes via the qb4o:hasAttribute property and aggregate functions via the qb4o:AggregateFunction class. The association between measures and aggregate functions is represented using the property qb4o:aggregateFunction. This property, together with the concept of component sets, allows a given measure to be associated with different aggregate functions in different cubes.

5 QB4OLAP Implementation of Multidimensional Data Cubes

We next show that we can translate both, data cube schema and instances, into an RDF representation using QB4OLAP. To represent the cube schema we will use the MultiDim model [15,13], whose main components are depicted in Fig. 2. Of course, any conceptual model could be used instead. A *schema* is composed of a set of dimensions and a set of facts. A *dimension* is composed of either one *level*, or one or more hierarchies. Instances of a level are called *members*. A level has a set of *attributes* that describe the characteristics of their members (Fig. 2a), and one or more *identifiers*, each identifier being composed of one or several attributes. A *hierarchy* is composed of a set of levels (Fig. 2b). Given two related levels in a hierarchy, the lower level is called the *child* and the higher one the *parent*; the relationships between them are called *parent-child relationships*, whose *cardinalities* are shown in Fig. 2c. A dimension may contain several hierarchies identified by a *hierarchy name* (Fig. 2e). The name of the leaf level in a hierarchy defines the dimension name, except when the same level participates several times in a fact, in which case the role name defines the dimension name. These are called *role-playing dimensions*. A *fact* (Fig. 2d) relates several levels. Instances of a fact are called *fact members*. A fact may contain attributes called *measures*. The aggregation function associated to a measure can be specified next to the measure name (Fig. 2d), the default being the SUM function.

5.1 QB4OLAP Implementation of a Cube Schema

We first present an algorithm to obtain a QB4OLAP representation of a cube schema, from a conceptual schema. We assume that we have a conceptual schema that represents a cube C, with a fact F composed of a set M of measures, and a set D of dimensions. Each dimension $d \in D$ is composed of a set L of levels, organized in hierarchies $h \in H$. Each level $l \in L$ is described by a set of attributes A. Figure 3 depicts a simplified version of the *MultiDim* representation of the well-known Northwind DW, which we will use as our running example. The algorithm comprises seven steps described next. We call CS_{RDF} the RDF graph that represents the cube schema, which is built incrementally.

Step 1 (Dimensions). For each dimension $d \in D$, $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple stating that there exists a resource d_{RDF} of type `qb:DimensionProperty`. Triples indicating the name of each dimension can be added using property `rdfs:label`.

The triples below show how some dimensions in Fig. 3 are represented (`@en` indicates that the names are in English, and `nw:` is a prefix for the cube schema graph).

```
nw:employeeDim a qb:DimensionProperty ; rdfs:label ”Employee Dimension”@en .
nw:orderDateDim a qb:DimensionProperty ; rdfs:label ”OrderDate Dimension”@en .
```

Step 2 (Hierarchies). For each hierarchy $h \in H$, $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple stating that h_{RDF} is a resource with type `qb4o:HierarchyProperty`. Triples indicating the name of each hierarchy can be added using property `rdfs:label`.

Applying Step 2 to the hierarchies in the Employee dimension we obtain:

```
nw:supervision a qb4o:HierarchyProperty ; rdfs:label ”Employee Supervision Hierarchy”@en .
nw:territories a qb4o:HierarchyProperty ; rdfs:label ”Employee Territories Hierarchy”@en .
```

Step 3 (Levels and Attributes). For each level $l \in L$, $CS_{RDF} = CS_{RDF} \cup \{t\}$, where t is a triple stating that l_{RDF} is a resource with type `qb4o:LevelProperty`. For each attribute $a \in A$, add to CS_{RDF} a triple stating that there exists a resource a_{RDF} with type `qb4o:AttributeProperty`. Finally, add triples relating a level l_{RDF} with its corresponding attribute a_{RDF}, using the property `qb4o:hasAttribute`. Triples indicating the names of levels and attributes can be added using property `rdfs:label`.

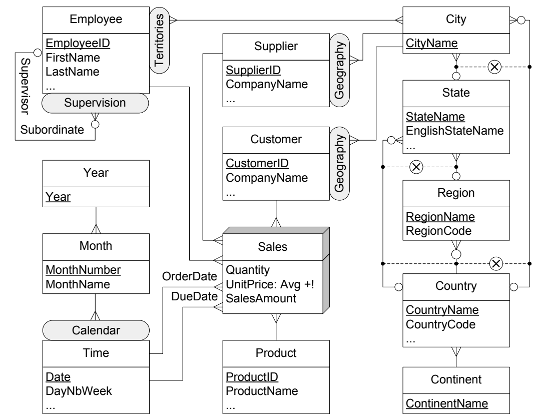


Fig. 3. A simplified version of the conceptual schema of the NorthwindDW

Applying Step 3 to level Employee in the Employee dimension we obtain:

```
nw:employee a qb4o:LevelProperty ; rdfs:label "Employee Level"@en ;
    qb4o:hasAttribute nw:firstName ; qb4o:hasAttribute nw:lastName .
nw:firstName a qb:AttributeProperty ; rdfs:label "First Name"@en .
nw:lastName a qb:AttributeProperty ; rdfs:label "Last Name"@en .
```

Step 4 (Dimension-Hierarchy Relationships). For each $h \in H$ in $d \in D$, relate d_{RDF} in CS_{RDF} to h_{RDF}, and h_{RDF} to d_{RDF}. Then, $CS_{RDF} = CS_{RDF} \cup \{d_{RDF}$ `qb4o:hasHierarchy` $h_{RDF}\} \cup \{h_{RDF}$ `qb4o:inDimension` $d_{RDF}\}$.

Applying Step 4 to the Employee dimension and its hierarchies we obtain:

```
nw:employeeDim qb4o:hasHierarchy nw:Supervision ; qb4o:hasHierarchy nw:territories .
nw:supervision qb4o:inDimension nw:employeeDim. nw:territories qb4o:inDimension nw:employeeDim.
```

Step 5 (Hierarchy Structure). For each hierarchy $h \in H$ composed of a level $l \in L$, relate h_{RDF} in CS_{RDF} to l_{RDF} as $CS_{RDF} = CS_{RDF} \cup \{h_{RDF}$ `qb4o:hasLevel` $l_{RDF}\}$. Also, create a blank node lh_{RDF} of type `qb4o:LevelInHierarchy` that represents the pair (l_{RDF}, h_{RDF}) and add triples as $CS_{RDF} = CS_{RDF} \cup \{lh_{RDF}$ `qb4o:hierarchyComponent` $h_{RDF}\} \cup \{lh_{RDF}$ `qb4o:levelComponent` $l_{RDF}\}$. Let (l, l') be a pair of levels in the C, such that $l, l' \in h$, and $parentLevel(l, h) = l'$ with cardinality car. Also, let l_{RDF}, l'_{RDF}, and h_{RDF} be the representations of l, l' and h in CS_{RDF}, and lh_{RDF}, lh'_{RDF} be the resources of type `qb4o:LevelInHierarchy` that represent the pairs (l_{RDF}, h_{RDF}) and (l'_{RDF}, h_{RDF}) respectively. Then add to CS_{RDF} a blank node hs_{RDF} of type `qb4o:HierarchyStep`, and the triples hs_{RDF} `qb4o:childLevel` lh_{RDF}, hs_{RDF} `qb4o:parentLevel` lh'_{RDF}. Finally, add a triple hs_{RDF} `qb4o:cardinality` car_{RDF}, car_{RDF} being the relationship's cardinality.

A part of the Employee dimension structure obtained is shown below. Note the relationship `_:ih3; qb4o:parentLevel _:ih6` supporting a ragged hierarchy.

```
nw:supervision qb4o:hasLevel nw:employee .
nw:territories qb4o:hasLevel nw:employee, nw:city, nw:state, nw:country, nw:continent .
# nw:supervision levels
_:ih1 a qb4o:LevelInHierarchy ; qb4o:levelComponent nw:employee ; qb4o:hierarchyComponent nw:supervision .
```

```
# nw:territories levels
_:ih2 a qb4o:LevelInHierarchy ; qb4o:levelComponent nw:employee ; qb4o:hierarchyComponent nw:territories .
_:ih3 a qb4o:LevelInHierarchy ; qb4o:levelComponent nw:city ; qb4o:hierarchyComponent nw:territories .
_:ih4 a qb4o:LevelInHierarchy ; qb4o:levelComponent nw:state ; qb4o:hierarchyComponent nw:territories .
_:ih5 a qb4o:LevelInHierarchy ; qb4o:levelComponent nw:region ; qb4o:hierarchyComponent nw:territories .
_:ih6 a qb4o:LevelInHierarchy ; qb4o:levelComponent nw:country ; qb4o:hierarchyComponent nw:territories .
# nw:supervision hierarchy structure
_:pl1 a qb4o:HierarchyStep ; qb4o:childLevel _:ih1 ; qb4o:parentLevel _:ih1 ; qb4o:cardinality qb4o:OneToMany .
# nw:territories hierarchy structure
_:pl2 a qb4o:HierarchyStep ; qb4o:childLevel _:ih2 ; qb4o:parentLevel _:ih3 ; qb4o:cardinality qb4o:ManyToMany .
_:pl3 a qb4o:HierarchyStep ; qb4o:childLevel _:ih3 ; qb4o:parentLevel _:ih4 ; qb4o:cardinality qb4o:OneToMany .
_:pl4 a qb4o:HierarchyStep ; qb4o:childLevel _:ih3 ; qb4o:parentLevel _:ih6 ; qb4o:cardinality qb4o:OneToMany .
```

Step 6 (Measures). For each measure $m \in M, CS_{RDF} = CS_{RDF} \cup \{t\}$, such that t is a triple that states that m_{RDF} is a resource with type `qb4o:MeasureProperty`. The range of each m_{RDF} can be defined using the rdfs:range predicate.

The following triples are the result of the application of Step 6 to our example.

```
nw:quantity a qb:MeasureProperty ; rdfs:label "Quantity"@en ; rdfs:range xsd:integer .
nw:unitPrice a qb:MeasureProperty ; rdfs:label "UnitPrice"@en ; rdfs:range xsd:decimal .
nw:salesAmount a qb:MeasureProperty ; rdfs:label "SalesAmount"@en ; rdfs:range xsd:decimal .
```

Step 7 (Cube). For each fact F, $CS_{RDF} = CS_{RDF} \cup \{t\}$, such that t is a triple stating that c_{RDF} is a resource with type `qb:DataStructureDefinition`. For each measure $m \in M$, $CS_{RDF} = CS_{RDF} \cup \{c_{RDF}$ `qb:component` [`qb:measure` m_{RDF}; `qb4o:aggregateFunction` $f_{RDF}]\}$, where f_{RDF} is an aggregation function. Also, for each of the levels $l \in L$ related to a fact F in the conceptual schema, $CS_{RDF} = CS_{RDF} \cup \{c_{RDF}$ `qb:component` [`qb:level` l_{RDF}; `qb4o:cardinality` $car_{RDF}]\}$, where car_{RDF} represents the cardinality of the relationship between facts and level members and is one of cardinality restrictions defined in QB4OLAP (`qb4o:OneToOne`, `qb4o:OneToMany`, `qb4o:ManyToOne`, `qb4o:ManyToMany`).

The following triples are the result of the application of Step 7.

```
# Cube definition (Data structure)
nw:Northwind a qb:DataStructureDefinition ;
# Lowest level for each dimension in the cube
qb:component [qb4o:level nw:employee ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:orderDate ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:dueDate ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:supplier ; qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level nw:customer ; qb4o:cardinality qb4o:ManyToOne] ;
# Measures in the cube
qb:component [qb:measure nw:quantity ; qb4o:aggregateFunction qb4o:sum] ;
qb:component [qb:measure nw:unitPrice ; qb4o:aggregateFunction qb4o:avg] ;
qb:component [qb:measure nw:salesAmount ; qb4o:aggregateFunction qb4o:sum] .
```

5.2 QB4OLAP Implementation of Cube Instances

We now show a procedure to obtain a QB4OLAP implementation from a relational cube instance, starting from: (a) The conceptual schema of a cube C; (b) CS_{RDF}, the RDF representation of the schema of C; (c) the relational implementation of the cube C, which we denote CI_{ROLAP}. The procedure produces an R2RML mapping file CI_{RDF} that generates an RDF representation of the data stored in CI_{ROLAP}, using the schema CS_{RDF}. A collection of IRI-safe strings P are used to generate unique level members

IRIs with R2RML rr:template, such that each level $l_i \in L$ has its corresponding $p_i \in P$. Analogously, f is used to generate unique IRIs for fact instances. The procedure comprises two parts: (1) Define mappings to generate level members; (2) Define mappings to generate facts (observations). We define a procedure for each relational representation of a multidimensional model construct as follows. We assume the reader is familiar with the usual kinds of dimension hierarchies.

Step 1 (**Balanced hierarchies**.) These hierarchies can be represented as *snowflake schemas* or as *star schemas*. If $h \in H$ is a balanced hierarchy composed of a set of levels L, represented as a **snowflake schema**, there exists a set of tables $T_h \in CI_{ROLAP}$, where each table $t_i \in T_h$ represents a level in $l_i \in L$, and contains a key attribute pk_i and one attribute a_i for each level attribute $at_i \in l_i$. For each pair $l_i, l_{i+1} \in h$, represented as $t_i, t_{i+1} \in T$, such that $parentLevel(l_i, h) = l_{i+1}$, there exists a foreign key attribute $fk_i \in t_i$ referencing $pk_{i+1} \in t_{i+1}$.

To generate the instances of a *balanced hierarchy represented as a snowflake schema*, for each level $l_i \in h$, $CI_{RDF} = CI_{RDF} \cup \{t\}$, where t is an R2RML `rr:TripleMap` that generates the members of l_i. The components of t are: the `rr:logicalTable` t_i, a `rr:subjectMap` which is an IRI built using the rr:template $p_i\{pk_i\}$, and one or more `rr:predicateObjectMap` that express: (1) To which level $l_{i_{RDF}} \in CS_{RDF}$ the members generated by t belong; (2) The value of each attribute $a_{RDF} \in l_{i_{RDF}}$, which is obtained from the attributes in t_i, specified using `rr:column`; (3) The associated members in other levels l_j, using `skos:broader` and the `rr:template` $p_j\{fk_j\}$.

The R2RML mapping that generates the members in level Region in Fig. 3 is:

```
<#TriplesMapRegion > a rr:TriplesMap ;
rr:logicalTable [ rr:tableName ”State” ] ;
rr:subjectMap [ rr:termType rr:IRI ;
    rr:template ”http://www.fing.edu.uy/inco/cubes/instances/northwind/Region#{RegionCode}” ; ] ;
rr:predicateObjectMap [ rr:predicate qb4o:inLevel ; rr:object nw:region ; ] ;
rr:predicateObjectMap [ rr:predicate nw:regionCode ; rr:objectMap [ rr:column ”RegionCode” ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:regionName ; rr:objectMap [ rr:column ”RegionName” ] ; ] ;
rr:predicateObjectMap [ rr:predicate skos:broader;
    rr:objectMap [ rr:termType rr:IRI ;
        rr:template ”http://www.fing.edu.uy/inco/cubes/instances/northwind/Country#{CountryKey}” ] ; ] .
```

If $h \in H$ is a balanced hierarchy composed of a set of levels L, represented as a **star schema**, there exists a table $t_h \in CI_{ROLAP}$ representing all levels in $l_i \in L$. For each l_i there exists an attribute $pk_i \in t_h$ which identifies each level member and for each level attribute $at_i \in l_i$ there exists an attribute $a_i \in t_h$.

The mapping for a *balanced hierarchy represented as a star schema* is similar to the one in Step 1, except that the `rr:logicalTable` is the same for all levels. The R2RML mapping that produces the members in levels Month and Year is:

```
<#TriplesMapMonth> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName ”Time” ] ;
rr:subjectMap [ rr:termType rr:IRI ; rr:template ”http://www.fing.edu.uy/instances/nw/Month#{MonthName}{Year}” ; ] ;
rr:predicateObjectMap [ rr:predicate qb4o:inLevel ; rr:object nw:month; ] ;
rr:predicateObjectMap [ rr:predicate nw:monthNumber ; rr:objectMap [ rr:column ”MonthNumber” ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:monthName ; rr:objectMap [ rr:column ”MonthName” ] ; ] ;
rr:predicateObjectMap [ rr:predicate skos:broader;
    rr:objectMap [ rr:termType rr:IRI ; rr:template ”http://www.fing.edu.uy/instances/nw/Year#{Year}” ] ;] .
<#TriplesMapYear> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName ”Time” ] ;
rr:subjectMap [ rr:termType rr:IRI ; rr:template ”http://www.fing.edu.uy/instances/nw/Year#{Year}” ; ] ;
rr:predicateObjectMap [ rr:predicate qb4o:inLevel ; rr:object nw:year ; ] ;
rr:predicateObjectMap [ rr:predicate nw:yearNumber ; rr:objectMap [ rr:column ”Year” ] ; ] .
```

Step 2 (**Parent-child (Recursive) hierarchies**.) These are represented as a table containing all attributes in a level, and a foreign key to the same table, relating child members to their parent. If $h \in H$ is a parent-child hierarchy, composed of a pair of levels $l_i, l_{i+1} \in h$ such that $parentLevel(l_i, h) = l_{i+1}$, there exists a table $t_h \in CI_{ROLAP}$ which contains a key attribute pk_i that identifies the members of l_i and an attribute $fk_i \in t_h$, that identifies the members of l_{i+1} and is a foreign key referencing $pk_i \in t_h$.

The mapping for level members in a parent-child hierarchy is similar to the one presented for a star representation of a balanced hierarchy, since all hierarchy levels are populated from the same `rr:logicalTable`. For the Supervision hierarchy we have:

```
<#TriplesMapEmployee> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName "Employee" ] ;
rr:subjectMap [ rr:termType rr:IRI ; rr:template "http://www.fing.edu.uy/instances/nw/Employee#{EmployeeKey}" ; ];
rr:predicateObjectMap [ rr:predicate qb4o:inLevel ; rr:object nw:employee ; ] ;
rr:predicateObjectMap [ rr:predicate nw:firstName ; rr:objectMap [ rr:column "FirstName" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:lastName ; rr:objectMap [ rr:column "LastName" ] ; ] ;
rr:predicateObjectMap [ rr:predicate skos:broader;
    rr:objectMap [ rr:termType rr:IRI ;
        rr:template "http://www.fing.edu.uy/instances/nw/Employee#{SupervisorKey}" ] ; ].
```

Step 3 (**Nonstrict hierarchies).** Here, each level is represented as a separate table and a *bridge table* is used to represent the many-to-many relationship between level members. If $h \in H$ is a nonstrict hierarchy, composed of a set of levels L, there exists a set of tables $T_h \in CI_{ROLAP}$, one table $t_i \in T_h$ with a key attribute pk_i, for each level $l_i \in L$. For each pair of levels $l_i, l_{i+1} \in h$, represented as $t_i, t_{i+1} \in T$, such that $parentLevel(l_i, h) = l_{i+1}$ and members of l_i have exactly one associated member in l_{i+1}, the mapping is the same as for the snowflake representation of balanced hierarchies. If members of l_i have more than one associated member in l_{i+1}, there exists a bridge table $b_i \in T$ that contains two attributes fk_i, fk_{i+1} referencing $pk_i \in t_i$ and $pk_{i+1} \in t_{i+1}$ respectively. Thus, each pair of levels is populated by three `rr:TriplesMap`: two of them generate level members, while the third uses the bridge table as `rr:logicalTable` to generate parent-child relationships between level members.

The R2RML mapping that generates the parent-child relationship between members in the Employees and City levels, in the Territories hierarchy is:

```
<#TriplesMapTerritories>
rr:logicalTable [ rr:tableName "Territories" ] ;
    rr:subjectMap [ rr:template "http://www.fing.edu.uy/instances/nw/Employee#{EmployeeKey}" ; ] ;
rr:predicateObjectMap [ rr:predicate skos:broader ;
    rr:objectMap [ rr:termType rr:IRI ; rr:template "http://www.fing.edu.uy/instances/nw/City#{CityKey}" ] ; ] .
```

Step 4 (**Facts**.) For each fact F, $CI_{RDF} = CI_{RDF} \cup \{t\}$; t is an R2RML `rr:TripleMap` that generates fact instances (observations). The components of t are as follows: one `rr:logicalTable`, one `rr:subjectMap`, which is an IRI built using the rr:template $f\{F\_KEY\}$, one `rr:predicateObjectMap` stating the dataset the observation belongs to, one `rr:predicateObjectMap` for each level related to the fact, and one `rr:predicateObjectMap` for each measure. F_KEY provides a unique value for each fact, and can be obtained from a fact table column, or concatenating the keys of all the level members that participate in the fact.

The R2RML mapping that generates the members in the Sales facts is as follows. Note the representation of the role-playing dimensions, and the key in rr:template.

```
<#TriplesMapSales> a rr:TriplesMap ;
rr:logicalTable [ rr:tableName "Sales" ] ;
rr:subjectMap [ rr:termType rr:IRI ;
    rr:template "http://www.fing.edu.uy/instances/nw/Sale#{OrderNo}_{OrderLineNo}"; rr:class qb:Observation ; ] ;
rr:predicateObjectMap [ rr:predicate qb:dataSet ; rr:object nwi:dataset1 ; ] ;
rr:predicateObjectMap [ rr:predicate nw:customer ;
    rr:objectMap [ rr:termType rr:IRI ;
        rr:template "http://www.fing.edu.uy/instances/nw/Customer#{CustomerKey}" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:employee ;
    rr:objectMap [ rr:termType rr:IRI ;
        rr:template "http://www.fing.edu.uy/instances/nw/Employee#{EmployeeKey}" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:orderDate ;
    rr:objectMap [ rr:termType rr:IRI ; rr:template "http://www.fing.edu.uy/instances/nw/Time#{OrderDateKey}" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:dueDate ;
    rr:objectMap [ rr:termType rr:IRI ; rr:template "http://www.fing.edu.uy/instances/nw/Time#{DueDateKey}" ] ; ];
rr:predicateObjectMap [ rr:predicate nw:supplier ;
    rr:objectMap [ rr:termType rr:IRI ; rr:template "http://www.fing.edu.uy/instances/nw/Supplier#{SupplierKey}" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:quantity ; rr:objectMap [ rr:column "Quantity" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:unitPrice ; rr:objectMap [ rr:column "UnitPrice" ] ; ] ;
rr:predicateObjectMap [ rr:predicate nw:salesAmount ; rr:objectMap [ rr:column "SalesAmount" ] ; ] .
```

6 Querying a QB4OLAP Cube

The representation produced by the procedures described in Section 5 supports expressing in SPARQL commonly-used real-world OLAP queries. We next give the intuition of this, showing two typical OLAP queries expressed in MDX, the *de facto* standard language for OLAP, and its equivalent SPARQL query. We assume the reader is familiar with MDX. We start with the query "*Three best-selling employees*", which reads in MDX:

```
SELECT Measures.[Sales Amount] ON COLUMNS,
    TOPCOUNT(Employee.[Full Name].CHILDREN, 3,Measures.[Sales Amount]) ON ROWS
FROM Sales
```

The query above reads in SPARQL(LIMIT 3 keeps the first three results):

```
SELECT ?fName ?lName (SUM(?sales) AS ?totalSales)
WHERE { ?o qb:dataSet nwi:dataset1 ; nw:employee ?emp ; nw:salesAmount ?sales .
    ?emp qb4o:inLevel nw:employee ; nw:firstName ?fName ;nw:lastName ?lName . }
GROUP BY ?fName ?lName
ORDER BY DESC (?totalSales) LIMIT 3
```

Consider now: "*Total sales and average monthly sales by employee and year*"

```
WITH MEMBER Measures.[Avg Monthly Sales] AS
AVG(DESCENDANTS([Order Date].Calendar.CURRENTMEMBER,
    [Order Date].Calendar.Month),Measures.[Sales Amount]),FORMAT_STRING = '$###,##0.00'
SELECT {Measures.[Sales Amount], Measures.[Avg Monthly Sales] } ON COLUMNS,
Employee.[Full Name].CHILDREN * [Order Date].Calendar.Year.MEMBERS ON ROWS FROM Sales
```

Below we show the equivalent SPARQL query. The inner query computes the total sales by employee and month; the outer query aggregates this result to the Year level, and computes the total yearly sales and the average monthly sales.

```
SELECT ?fName ?lName ?yearNo (SUM(?monthlySales)) (AVG(?monthlySales) )
WHERE { {
    SELECT ?fName ?lName ?month (SUM(?sales) AS ?monthlySales)
    # Montly sales by employee
    WHERE { ?o qb:dataSet nwi:dataset1 ; nw:employee ?emp ;
        nw:orderDate ?odate ; nw:salesAmount ?sales .
        ?emp qb4o:inLevel nw:employee ; nw:firstName ?fName ; nw:lastName ?lName .
        ?odate qb4o:inLevel nw:orderDate ; skos:broader ?month . ?month qb4o:inLevel nw:month . }
    GROUP BY ?fName ?lName ?month }
    ?month skos:broader ?year . ?year qb4o:inLevel nw:year ; nw:yearNumber ?yearNo . }
GROUP BY ?fName ?lName ?yearNo ORDER BY ?fName ?lName ?yearNo
```

7 Discussion and Open Challenges

In this paper we focused in studying modeling issues, and in showing that writing real-world OLAP queries in SPARQL based on an appropriate model is a plausible approach. We presented a new version of QB4OLAP that allows representing most of the concepts in the multidimensional model, and a procedure to obtain a QB4OLAP representation of such model. We also proposed a set of steps that produce a QB4OLAP representation of a relational implementation of a data cube. It would be possible to automatically generate queries like the ones presented in Section 6, starting from a high-level algebra like the one proposed in [3], and this is the approach we will follow in future work. We will also study mechanisms for obtaining QB4OLAP data cubes using other data sources, not necessarily multidimensional ones.

References

1. Abelló, A., Darmont, J., Etcheverry, L., Golfarelli, M., Mazón, J.N., Naumann, F., Pedersen, T.B., Rizzi, S., Trujillo, J., Vassiliadis, P., Vossen, G.: Fusion cubes: Towards Self-Service Business Intelligence. IJDWM 9(2), 66–88 (2013)
2. Beheshti, S.-M.-R., Benatallah, B., Motahari-Nezhad, H.R., Allahbakhsh, M.: A Framework and a Language for On-Line Analytical Processing on Graphs. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 213–227. Springer, Heidelberg (2012)
3. Ciferri, C., Ciferri, R., Gómez, L., Schneider, M., Vaisman, A., Zimányi, E.: Cube Algebra: A Generic User-Centric Model and Query Language for OLAP Cubes. IJDWM 9(2), 39–65 (2013)
4. Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J.M., Welton, C.: Mad skills: New Analysis Practices for Big Data. PVLDB 2(2), 1481–1492 (2009)
5. Etcheverry, L., Vaisman, A.A.: Enhancing OLAP Analysis with Web Cubes. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 469–483. Springer, Heidelberg (2012)
6. Etcheverry, L., Vaisman, A.: QB4OLAP: A Vocabulary for OLAP Cubes on the Semantic Web. In: Proc. of COLD 2012. CEUR-WS.org, Boston (November 2012)
7. Golfarelli, M.: Open source BI platforms: A functional and architectural comparison. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2009. LNCS, vol. 5691, pp. 287–297. Springer, Heidelberg (2009)
8. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Morgan & Claypool Publishers (2011)
9. Kämpgen, B., Harth, A.: Transforming statistical linked data for use in OLAP systems. In: Proceedings of the 7th International Conference on Semantic Systems, I-Semantics 2011, pp. 33–40. ACM, New York (2011)
10. Kämpgen, B., O'Riain, S., Harth, A.: Interacting with Statistical Linked Data via OLAP Operations. In: ESWC Workshops, Heraklion, Crete, Greece (May 2012)
11. Kämpgen, B., Harth, A.: No size fits all – running the star schema benchmark with SPARQL and RDF aggregate views. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 290–304. Springer, Heidelberg (2013)
12. Löser, A., Hueske, F., Markl, V.: Situational Business Intelligence. In: Castellanos, M., Dayal, U., Sellis, T. (eds.) BIRTE 2008. LNBIP, vol. 27, pp. 1–11. Springer, Heidelberg (2009)
13. Malinowski, E., Zimányi, E.: Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications. Springer (2008)
14. Nebot, V., Llavori, R.B.: Building data warehouses with semantic web data. Decision Support Systems 52(4), 853–868 (2011)
15. Vaisman, A., Zimányi, E.: Data Warehouse Systems: Design and Implementation. Springer (2014)