Publishing and querying government multidimensional data using QB4OLAP

M. Bouza, B. Elliot, and L. Etcheverry
Instituto de Computación, Facultad de Ingeniería
UdelaR Montevideo, Uruguay
lorenae@fing.edu.uy

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

***Abstract*—The web is changing the way in which data warehouses are designed, used, and queried. With the advent of initiatives such as Open Data and Open Government, organizations want to share their multidimensional data cubes and make them available to be queried online. The RDF data cube vocabulary (QB), the W3C standard to publish statistical data in RDF, presents several limitations to fully support the multidimensional model. The QB4OLAP vocabulary extends QB to overcome these limitations, and provides the distinctive feature of being able to implement several OLAP operations, such as rollup, slice, and dice using standard SPARQL queries. In this paper we present QB4OLAP Engine, a tool that transforms multidimensional data stored in relational DWs into RDF using QB4OLAP, and apply the solution to a real-world case, based on the national survey of housing, health services, and income, carried out by the government of Uruguay.**

***Keywords*-Semantic Web, Data warehouses.**

I. Introduction

OLAP (or, more generally, Business Intelligence) (BI) software, produces reports and interactive interfaces that summarize multidimensional data via basic aggregation functions (e.g., counts and averages) over various hierarchical breakdowns of the data into groups, defined in the dimension hierarchies. A lot of academic research and industrial development was carried out throughout the 1990's related to conceptual modelling, query processing and optimization, aggregate precomputation, etc. Since the mid 90's, data warehouses (DW) and BI applications have been built to consolidate enterprise business data, allowing taking timely and informed decisions based on up-to-date consolidated data.

However, the web is changing the way in which data warehouses are designed, used, and queried [1]. With the advent of initiatives such as Open Data[1] and Open Government, organizations want to publish multidimensional data using standards and non-propietary formats.[2] Traditionally, OLAP and BI solutions are commercial tools with proprietary formats. 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 [2]. In addition, the Linked Data paradigm allows sharing and reusing data in the web by means of semantic web standards [3]. Domain ontologies expressed in RDF (the basic data representation layer) 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.

The RDF data cube vocabulary (QB) is the W3C standard to publish statistical data in RDF, presents several limitations to fully support the multidimensional model [4]. The QB4OLAP vocabulary [4], [5] extends QB by means of a set of RDF terms, organized in an RDF-S ontology. A distinctive feature of QB4OLAP is that several OLAP operations, such as rollup, slice and dice, can be implemented as standard SPARQL queries. This implies that, from multidimensional data published at high granularity levels, users can obtain coarse-grained multidimensional data aggregations without the need of specific infrastructure such as cube servers. This approach is aligned with open government principles, in particular with the *primary* principle which states that data should be published as collected from the source, with the highest possible level of granularity and not in aggregate or modified forms.[3]. In light of the above, in this paper we introduce *QB4OLAP Engine*, a tool that transforms multidimensional data stored in relational DWs into RDF using QB4OLAP, and show how real-world multidimensional data can be published and analyzed over the web, taking the national home and income survey in Uruguay.

After introducing background concepts and a description of QB4OLAP (Section II), we present the QB4OLAP Engine (Section III). In Section IV we describe our application scenario: multidimensional data obtained from surveys in Uruguay, and we show how the resulting RDF model can be queried using SPARQL, the standard query language for RDF. Related work is discussed in Section V. Finally we comment on open challenges and future work (Section VI).

II. Preliminaries

RDF The Resource Description Framework (RDF) [6] is a data model for expressing assertions over resources identified by an universal resource identifier (URI). Assertions are expressed as triples *subject - predicate - object*, where *subject* are always resources, and *predicate* and *object* could

[1] https://okfn.org/opendata/

[2] http://opengovdata.org/

[3] http://opengovdata.org/

978-1-4799-6953-1/14 $31.00 © 2014 IEEE
DOI 10.1109/LAWeb.2014.11

be resources or strings. *Blank nodes* are used to represent anonymous resources or resources without an URI, 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 or *RDF dataset* 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 rdfs-vocabulary) [7] is used to define classes, properties, and to represent hierarchical relationships between them. For example, the triple *(s, `rdf:type`, c)* explicitly states that *s* is an instance of *c* but it also implicitly states that object *c* is an instance of `rdf:Class` since there exists at least one resource that is an instance of *c*. Many formats for RDF serialization exist. The examples presented in this paper use Turtle [8].

SPARQL 1.1 [9] 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. SPARQL supports aggregate functions and the `GROUP BY` clause, which are relevant to OLAP.

R2RML [10] is a language for expressing mappings from relational databases to RDF datasets, 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*, which is 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 `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 brace-enclosed column names as placeholders. Foreign keys are handled through referencing object maps, which use the subjects of another triples map as the objects generated by a predicate-object map . We show examples of R2RML mappings in Section IV-A. R2RML mappings.

Linked Data [3], [11] is a data publication paradigm, based on semantic web standards that aims at publishing and relating data on the web. W3C projects such as the Linking Open Data community (LOD)[4] encourage the publication of open data using the linked data principles, which recommend using RDF as data publication format. LOD site, which consisted of than 500 million RDF links.

Multimensional Data Model In OLAP, data are organized as hypercubes whose axes are *dimensions*. Each point in this multidimensional space is mapped through *facts* into one or more spaces of *measures*. Dimensions are structured in *hierarchies* of *levels* that allow analysis at different levels of aggregation. The values in a dimension level are called *members*, which can also have properties or *attributes*. Members in a dimension level must have a corresponding member in the upper level in the hierarchy, and this correspondence is defined through so-called rollup functions.

QB4OLAP The RDF data cube vocabulary (QB),[5] is used to publish statistical data in RDF. The QB4OLAP vocabulary[6] extends QB to enhance the support of the multidimensional model [4]. 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 among levels. This information is needed to automatically obtain SPARQL queries that implement OLAP operations, for example rollup operator [4]. On the other hand, 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, which allows data cubes already 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 (with prefix `qb4o`) 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 (`qb:componentProperty`) for representing dimensions, measures, and attributes, called `qb:dimension`, `qb:measure`, and `qb:attribute`, respectively. Observations (facts) represent points in a multidimensional space. An observation is linked to a value in each dimension of the DSD using instances of `qb:DimensionProperty`. The `qb:concept` property links components to the concept they represent, modeled using the `skos:Concept` class defined in the SKOS vocabulary,[7] also used to define hierarchies by means of `skos:broader` and `skos:narrower`.

[4] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

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

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

[7] http://www.w3.org/2009/08/skos-reference/skos.html

For instance, the triple `country skos:narrower region` represents a hierarchical relationship where region is at a lower level than country. The `skos:hasTopConcept` property provides an entry point to these concept hierarchies.

As mentioned earlier, a key aspect of QB4OLAP is its ability to structure dimensions as hierarchies of levels. The class `qb4o:LevelProperty` models dimension levels. Level members are instances of the class `qb4o:LevelMember`, and relations between them are expressed using the property `skos:broader`. Dimension hierarchies are defined using 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 levels and hierarchies using the `qb4o:levelComponent` and the `qb4o:hierarchyComponent` properties respectively. The class `qb4o:HierarchyStep` represents the parent-child relationship between two levels in a hierarchy using the `qb4o:childLevel` and the `qb4o:parentLevel` properties respectively. The `qb4o:cardinality` property allows to represent the cardinality of this relationship using members of the `qb4o:Cardinality` class, and 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. In Section IV we show extracts of a QB4OLAP representation of a data cube.

III. QB4OLAP Engine: a Tool to Transform Relational Data Cubes into RDF

We now describe *QB4OLAP Engine*, our approach to obtain a QB4OLAP representation of a relational representation (ROLAP) of a data cube. After providing an overview of the architecture of the developed solution, we present the key aspects of the main components followed by some implementation details (Section III-D).

A. Design and architecture

QB4OLAP Engine is a tool that takes as input the specification of a multidimensional data cube, and its relational implementation (a set of relational tables) and produces two RDF graphs that use the QB4OLAP vocabulary. One of these graphs represents the *schema of the cube* and the other one an *instance of the cube*. These graphs are stored in a so-called RDF triplestore, which also implements an SPARQL endpoint, in short, a web interface to write and execute SPARQL queries. This allows publishing the cubes on the web, and offering the capability of performing queries over them.

Since there is no standard and machine-processable format to specify multidimensional data cubes, we have chosen the format provided by Pentaho Mondrian open-source OLAP server.[8] The Mondrian schema[9] is an XML document that defines a multidimensional database. It contains a *logical model*, consisting of cubes, dimensions, hierarchies, and members, and a *mapping* of this model onto a physical model. The physical model is the source of the data which is presented through the logical model, which is typically a star schema, implemented as a set of tables in a relational database. In a *star schema*, a fact table is linked through foreign keys to one or more denormalized dimension tables. In a *snowflake schema*, dimension tables are normalized, and a dimension is represented as a collection of tables linked to each other through foreign keys. Snowflake schemas and mixed approaches (like starflake) are also supported by Mondrian schema.

QB4OLAP engine is composed of several modules that tackle each one of the extraction and transformation processes. Figure **??** depicts the data and control flow between these modules, and the interaction with external components.

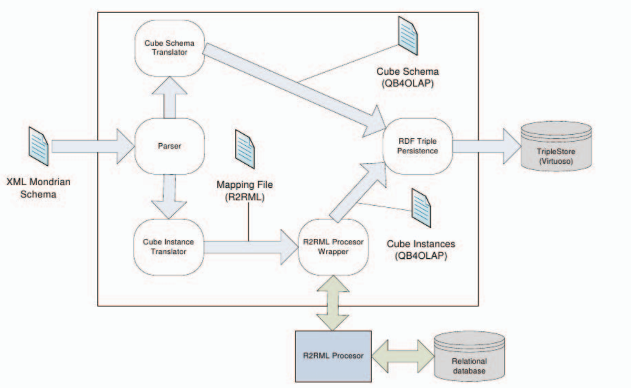

Figure 2: QB4OLAP Engine architecture.

The *parser* component first validates the input XML file that contains the Mondrian schema against its DTD. Then, it extracts the logical model of the cube and the mappings onto the physical model, creating an in-memory representation of this information that will be used throughout all the transformation processes. The *cube schema translator* is responsible for translating the logical model of the data cube

[8] http://community.pentaho.com/projects/mondrian/

[9] http://mondrian.pentaho.com/documentation/schema.php

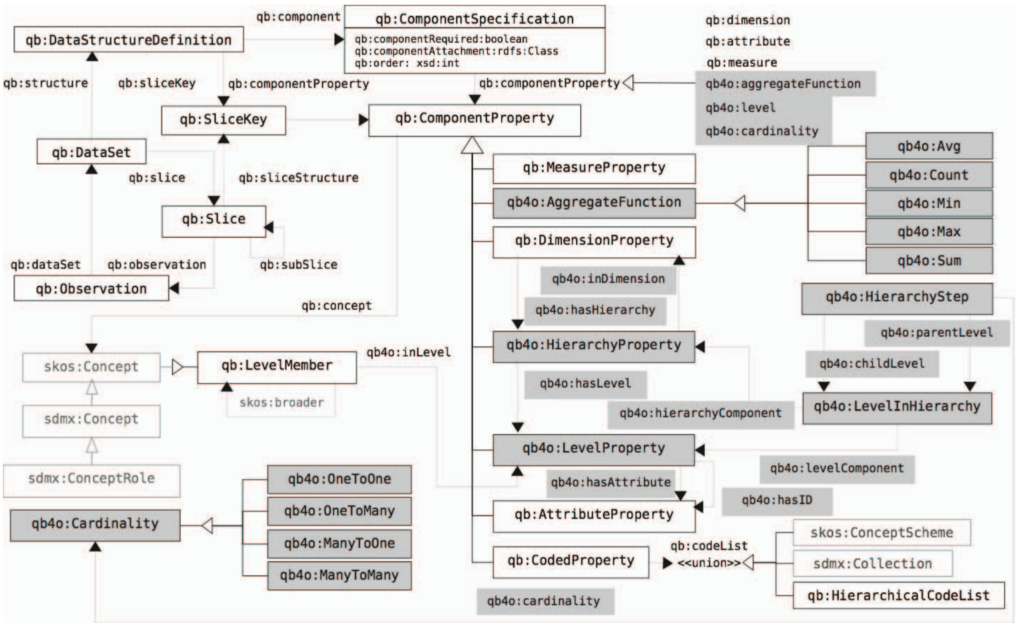

Figure 1: QB4OLAP vocabulary

into RDF, producing as output a set of RDF triples that represent the schema of the cube using QB4OLAP.

The *cube instance translator* is responsible for translating the data in the underlying relational database into RDF triples that represent instances of the data cube. This means generating triples, using QB4OLAP and the terms defined as the schema of the cube, that represent: level members with their corresponding attribute values, parent-child relationships among level members, and fact instances. The task at hand corresponds to the more general problem of providing an RDF view over relational data. Instead of directly generating triples that represent the instances, this component produces a set of R2RML mappings that encode how to produce these instances from the data stored in the physical model of the cube. Then, this 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*). Each of these strategies has a set of well-known advantages and disadvantages. In our current implementation we have chosen to materialize the triples that represent the instance. This decision is mainly based on the static nature of the underlying data (no need for updates). After the mappings are obtained, the *R2RML processor wrapper* interacts with an R2RML processor, which actually builds the RDF triples. Finally, the *RDF triples persistence* module stores all the triples into a triplestore. In the following sections we details the translation process.

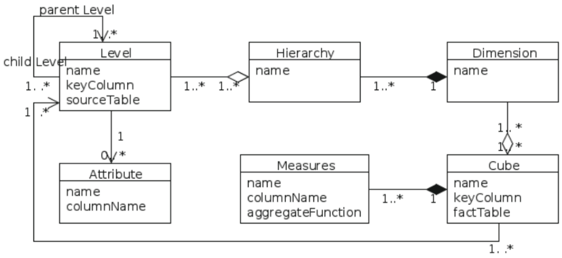

Figure 3: Internal representation of Mondrian schema.

B. Cube Schema Translator

As stated before, the cube schema translator takes as input an in-memory representation of the information contained in the Mondrian schema XML file. Figure 3 shows a diagram of the classes involved in this representation. A *cube* is composed of a set of *dimensions*, each dimension contains one or more *hierarchies*, composed of *levels* which have a set of *attributes*. A cube is also related to a set of levels, one in each of the dimensions involved in the cube. These levels define the granularity of the facts in the cube, and therefore the granularity of the values of the *measures* in the cube. Levels are related to other levels through parent-child relationships. The following restriction applies, and is enforced by XML schema definition. For every pair of levels (l_1, l_2) related via a parent-child relationship, assuming that H_1 and H_2 are the sets of hierarchies to which l_1 and l_2 belong respectively, then $H_1 \cap H_2 \neq \emptyset$ and $|H_1 \cap H_2| = 1$.

Table I: Auxiliary functions

Function signature	Description
getURI(*type*,*name*)	Given the type (ex: level, dimension, etc.) and name of a multidimensional concept, generates or retrieves an absolute URI to identify it.
getPairURI(*URI1*, *URI2*)	Given a pair of URIs, generates or retrieves an absolute URI that identifies the pair.
getTripleMapURI(*name*)	Given the name of a multidimensional concept, generates or retrieves an absolute URI to identify the R2RML TripleMap that will populate the concept.
getTemplate(*name*, *keyColumn*)	Returns a string to be used as a template to build an URI for each level member or fact instance
templateTermMap(*mapType*, *template*, *termType*)	According to *mapType* returns an instance i of rr:SubjectMap, rr:PredicateMap or rr:ObjectMap, where i.rr:template = *template* and i.rr:termType = *termType*
columnTermMap(*mapType*, *column*, *termType*)	According to *mapType* returns an instance i of rr:SubjectMap, rr:PredicateMap or rr:ObjectMap, where i.rr:column=*column* and i.rr:termType = *termType*
constantTermMap(*mapType*, *constant*)	According to *mapType* returns an instance i of rr:SubjectMap, rr:PredicateMap or rr:ObjectMap, where i.rr:constant = *constant*
predicateObjectMap(*predicateMap*, *objectMap*)	Returns an instance i of rr:predicateObjectMap, where i.rr:predicateMap = *predicateMap* and i.rr:objectMap = *objectMap*

Table I presents a list of auxiliary functions used to improve the presentation of the algorithms included in this work. Also, the "." operator is used to retrieve the values of the properties of an object (i.e., if ex: l is a Level, then $l.keyColumn$ retrieves the value of property $keyColumn$). Algorithm 1 describes the schema translation process, which can be decomposed in three steps: (i) processing the dimensions and its structure (lines 1-43), (ii) processing the measures (lines 44-50), and (iii) processing the cube definition (lines 51-57).

C. Cube Instance Translator

The instance translator module also takes as input the internal representation that adheres to the metamodel depicted in Figure 3, but instead of producing a set of RDF triples that represent the instances of the cube, it builds a set of R2RML mappings that will generate those instances. It is worth noting that R2RML mappings are also RDF triples. Algorithm 2 describes the mapping generation process, which can be decomposed in two steps: (i) building R2RML mappings to populate dimensions, which means to create level members and parent-child relationships within them (lines 1-32), and (ii) building R2RML mappings to build facts and measure values (lines 33-57). Auxiliary functions are defined in Table I.

In the first part of the process, for each level in the schema we build mappings that create an RDF representation of the level members stored in the database. Each level member is transformed into an RDF resource. To guarantee the uniqueness of level member URIs, a template is used that takes into account the column stated as the key of each level. Also, each level member is related to the values of each of its attributes using the RDF properties defined in Algorithm 1 (lines 17-22). Finally, we represent parent-child relationships between level members using the `skos:broader` property. In the second part of the process we build a mapping that creates an RDF representation of fact instances. Each fact instance relates a set of level members, one for each dimension in the cube, with a set of measure values.

D. Implementation Details

QB4OLAP Engine has been implemented in Java, using Jena libraries to deal with RDF in-memory representation. MySQL (version 5.5) is used to store the relational representation of the cube, although other RDBMS are easily supported. R2RML Parser[10] is used to process the generated R2RML mappings. Finally, generated RDF triples are stored in Virtuoso Open-Source Edition (version 7.1.0).[11] QB4OLAP Engine source code is available under CC 3.0 BY license.[12]

IV. Case Study: Data from Surveys in Uruguay

In Uruguay, the *Encuesta Continua de Hogares (ECH)* is a continuous survey commissioned by the *Instituto Nacional de Estadística (INE)*, the national office for statistics. It collects information about people's housing, access to health services, employment and income indicators, among others. Since 1968 the EHC is carried out in the capital city (Montevideo); in 1981 it started to cover all the cities in the country, and since 2006 it covers all the territory, including rural areas. This survey is applied over a sample of the target population and INE publishes the sets of records containing information on individual respondents and metadata about ECH, which can be found in Uruguay's government open data catalog.[13]

Among the many data cubes are available, we will focus on a cube that allows to analyze two measures: the number of people (#people) and the number of homes (#homes), according to three dimensions that represent: (a) the comfort level of the home (ComfortLevel), (b) its geographical location (Geography), and (c) the time of the survey (Time). The comfort level (LOW, MEDIUM-LOW, MEDIUM-HIGH, and HIGH) summarizes the presence or absence of different services and appliances such as: internet connection, computer, car, among others. The Geography dimension is organized into levels Neighborhood, City, State, and Region; the Time dimension in Month, Quarter and Year.

[10] https://github.com/nkons/r2rml-parser

[11] http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/

[12] https://code.google.com/p/publishing-multidimensional-data/

[13] https://catalogodatos.gub.uy/dataset?tags=INE

Algorithm 1 Creating the data structure definition of a cube in QB4OLAP from a cube in Mondrian Schema

Input: C_M: internal representation of cube C obtained from Mondrian schema.
Output: C_{RDF}: data structure definition of cube C in QB4OLAP.

```
$C_{RDF} \leftarrow \emptyset$
$c_U = getURI(cube, C.name)$
$C_{RDF} \leftarrow C_{RDF} \cup \{c_U$ a $\texttt{qb:DataStructureDefinition}\}$
let $D$ be the set of dimensions in $C_M$
for all $d \in D$ do
  $d_U = getURI(dimension, d.name)$
  $C_{RDF} \leftarrow C_{RDF} \cup \{d_U$ a $\texttt{qb:DimensionProperty}\}$
  let $H$ be the set of hierarchies in dimension $d$
  for all $h \in H$ do
    $h_U = getURI(hierarchy, h.name)$
    $C_{RDF} \leftarrow C_{RDF} \cup \{h_U$ a $\texttt{qb4o:HierarchyProperty}\}$
    $C_{RDF} \leftarrow C_{RDF} \cup \{h_U$ $\texttt{qb4o:inDimension}$ $d_U\}$
    $C_{RDF} \leftarrow C_{RDF} \cup \{d_U$ $\texttt{qb4o:hasHierarchy}$ $h_U\}$
    let $L$ be the set of levels in hierarchy $h$
    for all $l \in L$ do
      $l_U = getURI(level, l.name)$
      $C_{RDF} \leftarrow C_{RDF} \cup \{l_U$ a $\texttt{qb4o:LevelProperty}\}$
      $C_{RDF} \leftarrow C_{RDF} \cup \{l_U$ $\texttt{qb4o:inDimension}$ $d_U\}$
      $C_{RDF} \leftarrow C_{RDF} \cup \{l_U$ $\texttt{qb4o:inHierarchy}$ $h_U\}$
      $C_{RDF} \leftarrow C_{RDF} \cup \{h_U$ $\texttt{qb4o:hasLevel}$ $l_U\}$
      let $A$ be the set of attributes in level $l$
      for all $a \in A$ do
        $a_U = getURI(attribute, a.name)$
        $C_{RDF} \leftarrow C_{RDF} \cup \{a_U$ a $\texttt{qb:AttributeProperty}\}$
        $C_{RDF} \leftarrow C_{RDF} \cup \{l_U$ $\texttt{qb4o:hasAttribute}$ $a_U\}$
      end for
      $lh_U = getPairURI(l_U, h_U)$
      $C_{RDF} \leftarrow C_{RDF} \cup \{lh_U$ a $\texttt{qb4o:LevelInHierarchy}\}$
      $C_{RDF} \leftarrow C_{RDF} \cup \{lh_U$ $\texttt{qb4o:levelComponent}$ $l_U\}$
      $C_{RDF} \leftarrow C_{RDF} \cup \{lh_U$ $\texttt{qb4o:hierarchyComponent}$ $h_U\}$
      let $P$ be the set of parent levels of level $l$ according to $h$
      for all $p \in P$ do
        $p_U = getURI(level, p.name)$
        $ph_U = getPairURI(p_U, h_U)$
        $step_U = getPairURI(lh_U, ph_U)$
        $C_{RDF} \leftarrow C_{RDF} \cup \{step_U$ a $\texttt{qb4o:HierarchyStep}\}$
        $C_{RDF} \leftarrow C_{RDF} \cup \{step_U$ $\texttt{qb4o:childLevel}$ $lh_U\}$
        $C_{RDF} \leftarrow C_{RDF} \cup \{step_U$ $\texttt{qb4o:parentLevel}$ $ph_U\}$
        $C_{RDF} \leftarrow C_{RDF} \cup \{step_U$ $\texttt{qb4o:cardinality}$ $\texttt{qb4o:OneToMany}\}$
      end for
    end for
  end for
end for
```

```
for all $m \in M$ do
  $m_U = getURI(m.name)$
  $aggFunc_{RDF} = getAggregateFunction(m.aggregateFunction)$
  $C_{RDF} \leftarrow C_{RDF} \cup \{m_U a$ $\texttt{qb:MeasureProperty}\}$
  $aux$ = [$\texttt{qb:measure}$ $m_U$;$\texttt{qb:aggregateFunction}$ $aggFunc_{RDF}$]
  $C_{RDF} \leftarrow C_{RDF} \cup \{c_U$ $\texttt{qb:component}$ $aux\}$
end for
```

```
let $FactLevels$ be the set of levels that participate in cube $C$
for all $fl \in FactLevels$ do
  $fl_U = getURI(fl.name)$
  $aux$ = [$\texttt{qb4o:level}$ $fl_U$; $\texttt{qb4o:cardinality}$ $\texttt{qb4o:ManyToOne}$]
  $C_{RDF} \leftarrow C_{RDF} \cup \{c_U$ $\texttt{qb:component}$ $aux\}$
end for
let $M$ be the set of measures in cube $C$
```

Algorithm 2 Creating a cube instance in QB4OLAP from a cube in Mondrian Schema

Input: C_M: internal representation of cube C obtained from Mondrian schema.
Output: C_{R2RML}: a set of R2RML mappings that transform relational data in cube C into RDF and QB4OLAP.

```
$C_{R2RML} \leftarrow \emptyset$
let $D$ be the set of dimensions in $C_M$
for all $d \in D$ do
  let $L$ be the set of levels in dimensions $d$
  for all $l \in L$ do
    $ltm_U = getTripleMapURI(l.name)$
    $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ a $\texttt{rr:TriplesMap}\}$
    $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:logicalTable}$ [$\texttt{rr:tableName}$ "$l.tableName$"]$\}$
    $template = getTemplate(l.name, l.keyColumn)$
    $sm = templateTermMap(subject, template,$ $\texttt{rr:IRI}$)
    $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:subjectMap}$ $sm\}$
    $pm = constantTermMap(predicate,$ $\texttt{qb4o:inLevel}$ )
    $om = constantTermMap(object, getURI(l.name))$
    $pom = predicateObjectMap(pm, om)$
    $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:predicateObjectMap}$ $pom\}$
    let $A$ be the set of attributes in level $l$
    for all $a \in A$ do
      $pm = constantTermMap(predicate, getURI(a.name))$
      $om = columnTermMap(object, a.columnName,$ $\texttt{rr:Literal}$)
      $pom = predicateObjectMap(pm, om)$
      $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:predicateObjectMap}$ $pom\}$
    end for
    let $P$ be the set of parent levels for level $l$
    for all $p \in P$ do
      $template = getTemplate(p.name, p.keyColumn)$
      $pm = constantTermMap(predicate,$ $\texttt{skos:broader}$ )
      $om = templateTermMap(object, template,$ $\texttt{rr:IRI}$)
      $pom = predicateObjectMap(pm, om)$
      $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:predicateObjectMap}$ $pom\}$
    end for
  end for
end for
```

```
$ctm_U = getTripleMapURI(C.name)$
$template = getTemplate(c.name, c.factKey)$
$C_{R2RML} \leftarrow C_{R2RML} \cup \{ctm_U$ a $\texttt{rr:TriplesMap}\}$
$C_{R2RML} \leftarrow C_{R2RML} \cup \{ctm_U$ $\texttt{rr:logicalTable}$ [$\texttt{rr:tableName}$ "$c.factTable$"]$\}$
$sm = templateTermMap(subject, template,$ $\texttt{rr:IRI}$)
$C_{R2RML} \leftarrow C_{R2RML} \cup \{ctm_U$ $\texttt{rr:subjectMap}$ $sm\}$
$pm = constantTermMap(predicate,$ $\texttt{qb:dataSet}$ )
$om = constantTermMap(object, getURI(c.dataSet))$
$pom = predicateObjectMap(pm, om)$
$C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:predicateObjectMap}$ $pom\}$
let $FactLevels$ be the set of levels that participate in cube $C$
for all $fl \in FactLevels$ do
  $template = getTemplate(fl.name, fl.keyColumn)$
  $pm = constantTermMap(predicate, getURI(fl.name))$
  $om = templateTermMap(object, template,$ $\texttt{rr:IRI}$)
  $pom = predicateObjectMap(pm, om)$
  $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:predicateObjectMap}$ $pom\}$
end for
let $M$ be the set of measures in cube $C$
for all $m \in M$ do
  $pm = constantTermMap(predicate, getURI(m.name))$
  $om = columnTermMap(object, m.columnName,$ $\texttt{rr:Literal}$)
  $pom = predicateObjectMap(pm, om)$
  $C_{R2RML} \leftarrow C_{R2RML} \cup \{ltm_U$ $\texttt{rr:predicateObjectMap}$ $pom\}$
end for
```

Figure 4 shows the conceptual design of the cube, using the MultiDim model [12].

As presented in Section III our solution generates an RDF representation of a multidimensional data cube implemented as ROLAP, specifically implemented over Pentaho Mondrian. A ROLAP implementation of a multidimensional model consists of a set of relational tables. The *fact table* stores the measured values. Each tuple in the fact table is related to one level member in each of the dimensions that participate in the cube. Dimensions can be represented in different ways: one denormailzed table for each dimension (*star schema*), several normalized tables for each dimension (*snowflake schema*) and mixed approaches. Figure 5 presents the star schema that implements the conceptual model presented in Figure 4, while Figure 6 shows some instances for each of the tables in the star schema. Figure 6d only shows the columns in the Geography table that are involved in the level path Home→City→State.

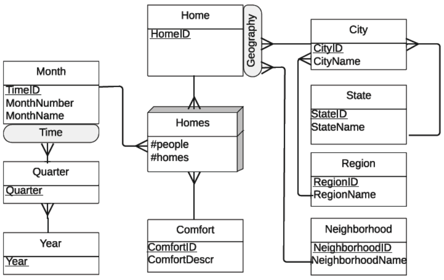

Figure 4: Conceptual multidimensional schema (Homes cube).

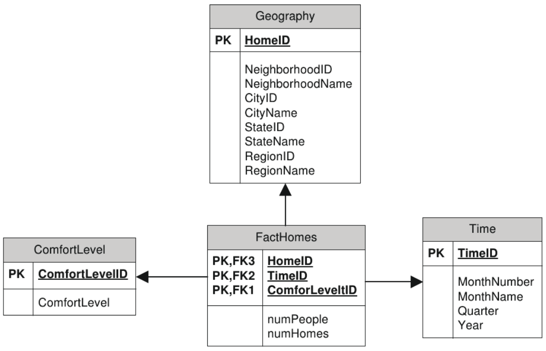

Figure 5: Logical design (Homes cube).

A. Using QB4OLAP Engine on the Homes Cube

We next show some excerpts of the result of using QB4OLAP Engine to transform the Homes cube. We begin showing the triples generated by the *cube schema translator*, followed by a sample of the R2RML mappings produced by the *cube instance translator*. We also show the triples resulting of applying these R2RML mappings to the relational representation of the cube.

1) Representation of the Cube Structure: Figure **??** shows a sample of the result of applying Algorithm 1 to the data cube presented in Figure 4. A set of prefixes (Lines 1-7) are followed by the cube structure (Lines 9-17). The rest of the triples define the Geography dimension. First, the dimension itself is defined (lines 20-22), followed by the the hierarchies in the dimension (lines 23-28). Notice that in this case there is only one hierarchy that contains all the levels. Due to space restrictions we only show the definition of levels Home and City (lines 29-35). Since levels may belong to different hierarchies, and in each hierarchy each level may have different parents, QB4OLAP represents parent-child relationships between levels as relationships of pairs (level,hierarchy). These pairs are modeled as instances of `qb4o:LevelInHierarchy`. Lines 36-42 define these instances for the levels Home and City. The relationships between pairs are represented as instances of tttqb4o:HierarchyStep, which allows to add properties to each step, such as cardinality restrictions. For example, Lines 43-47 tell that the City level is the parent of the Home level.

(a) FactHomes tableName

HomeID	TimeID	ComfortLevelID	numPeople	numHomes
2011000016	201109	1	2	1
2011000434	201103	2	1	1
2011002342	201102	3	2	1

(b) ComfortLevel table

ComfortLevelID	ComfortLevel
1	LOW
2	MEDIUM-LOW
3	MEDIUM-HIGH
4	HIGH

(c) Time table

TimeID	MonthNumber	MonthName	Quarter	Year
201102	2	February	Q12011	2011
201103	3	March	Q12011	2011
201109	9	September	Q32011	2011

(d) Geography table

HomeID	CityID	CityName	StateID	StateName
2011000016	05320	Colonia del Sacramento	5	Colonia
2011002342	10024	Punta del Este	10	Maldonado
2011000434	10024	Punta del Este	10	Maldonado

Figure 6: Sample instances of the star schema of the Homes cube

2) Representation of a Cube Instance: As presented in Section III-C, QB4OLAP Engine generates a set of R2RML mappings to transform the relational representation of the data cube into RDF. In Figure **??** we show, as an example, the R2RML mappings that populate levels Home and City in the Geography dimension. When these mappings are applied to relational data, a set of RDF triples are obtained. Figure **??** shows the result of applying the R2RML mappings from Figure **??** to the relational instances described in Figure 6.

B. Querying a Cube in QB4OLAP

QB4OLAP cubes can be analyzed *à la* OLAP using standard SPARQL queries. Due to the fact that dimension hierarchies and parent-child relationships between levels are explicitly represented using RDF triples, QB4OLAP allows to compute aggregated data from data published at a certain granularity level. In other approaches, like QB, this is not possible and data at different granularities need to be computed before publishing it as RDF. Take for instance Example 1, where the Geography dimension is traversed following the Home→City→State level path. This query corresponds to performing a rollup operation over the cube, from level Home to level State, and a slice operation to keep only the dimensions Geography and ComfortLevel. It is important to remark that the information encoded in the cube schema allows to obtain this query automatically.

```
@prefix qb: <http://purl.org/linked-data/cube#>.
@prefix qb4o: <http://purl.org/qb4olap/cubes#>.
@prefix cube: <http://www.fing.edu.uy/inco/cubes#> .
@prefix dims: <http://www.fing.edu.uy/inco/cubes/dimensions#> .
@prefix hier: <http://www.fing.edu.uy/inco/cubes/hierarchies#> .
@prefix ms: <http://www.fing.edu.uy/inco/cubes/measures#> .
@prefix att: <http://www.fing.edu.uy/inco/cubes/attributes#> .

# Cube definition (Data structure)
cube:Homes a qb:DataStructureDefinition ;
qb:component [qb4o:level dims:geography;
  qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level dims:time;
  qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb4o:level dims:comfort;
  qb4o:cardinality qb4o:ManyToOne] ;
qb:component [qb:measure ms:numPeople;
  qb4o:aggregateFunction qb4o:sum] ;
qb:component [qb:measure ms:numHomes;
  qb4o:aggregateFunction qb4o:sum] .

# Geography dimension
dims:geography a qb:DimensionProperty;
          rdfs:label ”Geography dimension”@en;
          qb4o:hasHierarchy hier:geographyHier.
# Geography dimension hierarchy
hier:geographyHier a qb4o:HierarchyProperty;
          rdfs:label ”Geography Hierarchy”@en;
          qb4o:inDimension dims:geography;
          qb4o:hasLevel dims:geo-home,dims:geo-neighborhood,
          dims:geo-state,dims:geo-city, dims:geo-region.
# Geography dimension levels
dims:geo-home a qb4o:LevelProperty;
          rdfs:label ”Home Level”@en;
          qb4o:inHierarchy hier:geographyHier.
dims:geo-city a qb4o:LevelProperty;
          rdfs:label ”City Level”@en;
          qb4o:inHierarchy hier:geographyHier.
# Organize levels in hierarchies
_:geo-home-geographyHier a qb4o:LevelInHierarchy ;
          qb4o:levelComponent dims:geo-home ;
          qb4o:hierarchyComponent hier:geographyHier.
_:geo-city-geographyHier a qb4o:LevelInHierarchy ;
          qb4o:levelComponent dims:geo-city ;
          qb4o:hierarchyComponent hier:geographyHier.
_:hs1 a qb4o:HierarchyStep;
          qb4o:childLevel _:geo-home-geographyHier ;
          qb4o:parentLevel _:geo-city-geographyHier ;
          qb4o:cardinality qb4o:OneToMany.
```

(a) Sample of the Homes cube schema

```
<#geography-HomesMap> a rr:TriplesMap;
rr:logicalTable [ rr:tableName ”Geography” ];
rr:subjectMap [rr:termType rr:IRI ;
 rr:template ”http://www.fing.edu.uy/inco/cubes/dic/geoHome#{HomeID}”];
rr:predicateObjectMap [ rr:predicate qb4o:inLevel;
 rr:object dims:geo-home];
rr:predicateObjectMap [ rr:predicate skos:broader;
 rr:objectMap [ rr:termType rr:IRI ;
  rr:template ”http://www.fing.edu.uy/inco/cubes/dic/geoCity#{CityID}”];].
```

(b) Sample of the R2RML mappings that populate the Geography dimension

```
@prefix ns1:<http://www.fing.edu.uy/inco/cubes/dic/geoHome#>.
@prefix ns2:<http://www.fing.edu.uy/inco/cubes/dic/geoCity#>.
@prefix ns3:<http://www.fing.edu.uy/inco/cubes/dic/geoState#>.
@prefix dims:<http://www.fing.edu.uy/inco/cubes/dimensions#>.

ns1:2011000016 qb4o:inLevel dims:geo-home;
  skos:broader ns2:05320.
ns2:05320 qb4o:inLevel dims:geo-city;
  rdfs:label ”Colonia del Sacramento”;
  skos:broader ns3:10.
```

(c) Sample of the RDF triples obtained applying R2RML mappings

Figure 7: Sample RDF triples

Example 1. *Total number of homes and people surveyed, by state and comfort level.*

```
SELECT ?stateName ?comfLevel
  (sum(?numHomes) as ?totalHomes) (sum(?numPeople) as ?totalPeople)
WHERE { ?o a qb:Observation; dims:geo-home ?home;
  dims:comfort-comfort ?comfort ; ms:numPeople ?numPeople;
  ms:numHomes ?numHomes.
  ?home skos:broader ?city. ?city qb4o:inLevel dims:geo-city .
  ?city skos:broader ?state. ?state qb4o:inLevel dims:geo-state .
  ?state rdfs:label ?stateName .?comfort atts:comfortLevel ?comfLevel }
GROUP BY ?stateName ?comf ?comfName
```

Example 2 also performs an aggregation over the Geography dimension, and shows how easy is to implement a Top-k query using SPARQL `ORDER BY` and `LIMIT` clauses.

Example 2. *Top 10 cities with most surveyed homes.*

```
SELECT ?cityName (SUM(?numHomes) AS ?totalHomes)
WHERE {
  ?o a qb:Observation; dims:geo-home ?home; ms:numHomes ?numHomes.
  ?home skos:broader ?city. ?city qb4o:inLevel dims:geo-city .
  ?city atts:cityName ?cityName }
GROUP BY ?cityName
ORDER BY DESC (?totalHomes) LIMIT 10
```

Example 3 combines aggregation and Top-k with filtering. A BGP is used to keep only the facts (observations) that refer to LOW comfort level homes (line 7).

Example 3. *Top 3 states with the largest number of LOW comfort level homes.*

```
SELECT ?stateName (SUM(?lowHomes) AS ?lowComHomes)
WHERE {
  ?o a qb:Observation; dims:geo-home ?home; ms:numHomes ?lowHomes.
  ?home skos:broader ?city. ?city qb4o:inLevel dims:geo-city .
  ?city skos:broader ?state. ?state qb4o:inLevel dims:geo-state .
  ?state atts:stateName ?stateName.
  ?o dims:comfort-comfort
      <http://www.fing.edu.uy/inco/cubes/dic/comfort#LOW> }
GROUP BY ?stateName
ORDER BY DESC (?lowComHomes) LIMIT 3
```

Example 4 compares aggregated values that correspond to different periods of time. SPARQL `FILTER` clause is used to state the comparison criteria.

Example 4. *Total number of surveyed homes by comfort level compared to those of the previous month.*

```
SELECT ?comfLevel ?yearNo ?monthNo ?tHomes ?tHomes1
WHERE {
{ SELECT ?comfort ?yearNo ?monthNo (SUM(?numHomes) AS ?tHomes)
  WHERE { ?o a qb:Observation; dims:geo-home ?home;
   dims:comfort-comfort ?comfort ; dims:time-month ?month;
   ms:numHomes ?numHomes. ?comfort atts:comfortName ?comfName .
   ?month qb4o:inLevel dims:time-month. ?month skos:broader ?quarter.
   ?month atts:monthNumber ?monthNo .
   ?comfort atts:comfortLevel ?comfLevel.
   ?quarter qb4o:inLevel dims:time-quarter.?quarter skos:broader ?year.
   ?year qb4o:inLevel dims:time-year . ?year atts:year ?yearNo.}
  GROUP BY ?comfLevel ?yearNo ?monthNo }
  OPTIONAL {
  {SELECT ?comfort ?yearNo1 ?monthNo1 (SUM(?numHomes) AS ?tHomes1)
   WHERE { ?o a qb:Observation; dims:geo-home ?home;
    dims:comfort-comfort ?comfort ; dims:time-month ?month1;
    ms:numHomes ?numHomes. ?comfort atts:comfortName ?comfName .
    ?month1 qb4o:inLevel dims:time-month. ?month1 skos:broader ?quarter1.
    ?month1 atts:monthNumber ?monthNo1 .
    ?comfort atts:comfortLevel ?comfLevel.
    ?quarter1 qb4o:inLevel dims:time-quarter.?quarter1 skos:broader ?year1.
    ?year1 qb4o:inLevel dims:time−year . ?year1 atts:year ?yearNo1.}
   GROUP BY ?comfLevel ?yearNo1 ?monthNo1 } }
FILTER(
( (?monthNo = ?monthNo1 + 1) && (?yearNo = ?yearNo1) ) ||
( (?monthNo = 1) && (?monthNo1 = 12) && (?yearNo = ?yearNo1+1) ) ) }
GROUP BY ?comfLevel ?yearNo ?monthNo
ORDER BY ?comfLevel ?yearNo ?monthNo
```

V. Related Work

Several organizations publish statistical data using the RDF Data Cube (QB) vocabulary[14], although few information is available on the tools and processes used to build these data cubes. Some tools have been developed in the context of the Eurostat-Linked Data project[15], but they take as input statistical data in SDMX format.

There exist tools for analyzing and querying data cubes in QB. CubeViz[16] is a faceted browser on QB data cubes. OLAP4LD [13] proposes to implement OLAP operations as SPARQL queries over cubes expressed in QB. However, hierarchies and levels must be added to the cubes in order to implement the operations that need these concepts. The authors analyze the performance of their proposal, and suggest using aggregate views to materialize aggregations and speed-up queries.

VI. Conclusion and Future Work

In this paper we have presented QB4OLAP Engine, a prototype of a tool that allows to transform multidimensional ROLAP data into RDF, using QB4OLAP. This prototype shows the feasibility of our approach, while further testing of the tool is needed to reveal the scalability and performance characteristics of our implementation. Future work regarding QB4OLAP Engine includes allowing the reuse of already generated QB4OLAP data, and studying how to deal with changes in the underlying relational data. We also plan on studying mechanisms to obtain QB4OLAP data cubes using other data sources, not necessarily multidimensional data sources.

References

[1] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton, “MAD Skills: New analysis practices for big data,” *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1481–1492, 2009.

[2] M. Golfarelli, “Open source BI platforms: A functional and architectural comparison,” in *Data Warehousing and Knowledge Discovery*, ser. LNCS. Springer, 2009, vol. 5691, pp. 287–297.

[3] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, ser. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[4] L. Etcheverry and A. Vaisman, “QB4OLAP: A vocabulary for OLAP cubes on the semantic web,” in *Proc. of the 3rd. International Workshop on Consuming Linked Data, COLD 2012*. Boston, USA: CEUR-WS.org, 2012.

[5] L. Etcheverry and A. A. Vaisman, “Enhancing OLAP analysis with web cubes,” in *Proc. of the 9th Extended Semantic Web Conference, ESWC 2012*, ser. LNCS 7295, Crete, Greece, 2012, pp. 469–483.

[6] G. Klyne, J. J. Carroll, and B. McBride, “Resource Description Framework (RDF): Concepts and Abstract Syntax,” 2004. [Online]. Available: http://www.w3.org/TR/rdf-concepts/

[7] D. Brickley, R. Guha, and B. McBride, “RDF Vocabulary Description Language 1.0: RDF Schema,” 2004. [Online]. Available: http://www.w3.org/TR/rdf-schema/

[8] D. Beckett and T. Berners-Lee, “Turtle - Terse RDF Triple Language,” 2011. [Online]. Available: http://www.w3.org/TeamSubmission/turtle/

[9] E. Prud’hommeaux and A. Seaborne, “SPARQL 1.1 Query Language for RDF,” 2011. [Online]. Available: http://www.w3.org/TR/sparql11-query/

[10] S. Das, S.Sundara, and R. Cyganiak, “R2RML: RDB to RDF Mapping Language,” 2012. [Online]. Available: http://www.w3.org/TR/r2rml/

[11] C. Bizer, T. Heath, and T. Berners-Lee, “Linked data-the story so far,” *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, pp. 1–22, 2009.

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

[13] B. Kämpgen and A. Harth, “No size fits all - running the star schema benchmark with SPARQL and RDF aggregate views,” in *The Semantic Web: Semantics and Big Data*, ser. LNCS. Springer, 2013, vol. 7882, pp. 290–304.

[14] http://www.w3.org/2011/gld/wiki/Data_Cube_Implementations

[15] http://eurostat.linked-statistics.org/

[16] http://aksw.org/Projects/CubeViz.html