QB2OLAP: Enabling OLAP on Statistical Linked Open Data

Jovan Varga*[1], Lorena Etcheverry†[2], Alejandro A. Vaisman‡[3],
Oscar Romero*[4], Torben Bach Pedersen§[5] and Christian Thomsen§[6]
*Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
†Instituto de Computación, Facultad de Ingeniería, UdelaR Montevideo, Uruguay
‡Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina
§Aalborg Universitet, Aalborg, Denmark
[1]jvarga@essi.upc.edu, [2]lorenae@fing.edu.uy, [3]avaisman@itba.edu.ar, [4]oromero@essi.upc.edu, [5]tbp@cs.aau.dk, [6]chr@cs.aau.dk

***Abstract*—Publication and sharing of multidimensional (MD) data on the Semantic Web (SW) opens new opportunities for the use of On-Line Analytical Processing (OLAP). The RDF Data Cube (QB) vocabulary, the current standard for statistical data publishing, however, lacks key MD concepts such as dimension hierarchies and aggregate functions. QB4OLAP was proposed to remedy this. However, QB4OLAP requires extensive manual annotation and users must still write queries in SPARQL, the standard query language for RDF, which typical OLAP users are not familiar with. In this demo, we present *QB2OLAP*, a tool for enabling OLAP on existing QB data. Without requiring any RDF, QB(4OLAP), or SPARQL skills, it allows semi-automatic transformation of a QB data set into a QB4OLAP one via enrichment with QB4OLAP semantics, exploration of the enriched schema, and querying with the high-level OLAP language *QL* that exploits the QB4OLAP semantics and is automatically translated to SPARQL.**

I. Introduction

OLAP analysis [1] is a well-established approach for decision making. Typically used in Data Warehousing (DW), OLAP relies on the MD model which represents data in terms of *facts* and *dimensions*. In short, dimensions conform the axes of an MD space in which a set of *measures* (associated to the fact) are represented. Dimensions provide appropriate contextual meaning to facts, and are organized as *hierarchies*, providing different *levels* of data aggregation. By means of an *MD algebra*, MD data are aggregated and disaggregated (through *roll-up* and *drill-down*, respectively), and filtered (through *slice* and *dice* operations), among other operations.

Initiatives like Open Data[1] are pushing organizations to publish MD data using standards and non-proprietary formats. Two main approaches can be followed for OLAP analysis of SW data. The first one aims at extracting MD data from the Web, and loading them into traditional DWs for OLAP analysis [2]. The second one (that we follow in our work) carries out OLAP-like analysis directly over MD data represented in RDF, following the notion of *self-service BI* [3].

Statistical data have traditionally been accessed and analyzed by means of OLAP [1]. In the SW, statistical data sets are usually published using the *RDF Data Cube Vocabulary*[2] (QB), a W3C recommendation since January, 2014. However, QB does not support the dimension hierarchies and aggregate functions needed for OLAP analysis. To address this challenge, a new vocabulary called QB4OLAP has been proposed [4]. QB4OLAP allows reusing data already published in QB by defining an MD schema containing the hierarchical structure of the dimensions (and the corresponding instances that populate the dimension levels). Once a data cube becomes published using QB4OLAP, we benefit from all the OLAP advances achieved in order to enable users to perform OLAP operations over the cube at a higher level of abstraction by using an OLAP algebra. In the demo, we present the *QB2OLAP* tool that can semi-automatically transform a QB data set into a QB4OLAP data set by enriching it with QB4OLAP semantics, explore the enriched schema (i.e., dimensions' structures and instances), and query the data set using a high-level OLAP language, denoted QL. *QB2OLAP* semi-automatically discovers dimension hierarchies to enrich the original data set, and automatically translates QL queries into SPARQL and executes them on an endpoint. Thus, *QB2OLAP* is a tool that facilitates data analysis, encouraging the use of MD data on the web. To our best knowledge, it is the first tool enabling native OLAP analysis on Statistical Linked Open Data.

Demo Use Case: Mary is a journalist covering the European migration crisis. She wants to analyze historical migration data for the European Union (EU), and knows that these data[3] are provided by the statistical office of the EU (Eurostat) and are also available as Linked Open Data in QB format[4]. Mary wants to compute some basic filtering/ summaries, typical for OLAP, such as aggregate the origin nationality of immigrants per continent. However, due to the limited schema information, she soon realizes that it is not possible to perform OLAP operations. To do so, she would need to enrich the data set (e.g., with dimension hierarchies to roll-up through). Moreover, both enrichment and analysis require the use of SPARQL, a language that she cannot manage although she is quite proficient in OLAP. Fortunately, she knows about *QB2OLAP* and decides to use it to overcome her lack of technical knowledge on RDF, QB, and SPARQL. This demo shows how *QB2OLAP* can be used to achieve OLAP-like analysis over existing QB data sets and enable even wider analysis, e.g., analyze migration data according to the kind

◇This research is funded by the European Commission through the Erasmus Mundus Joint Doctorate IT4BI-DC.

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

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

[3]http://ec.europa.eu/eurostat/statistics-explained/index.php/Asylum_statistics

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

of political organization of the host countries. The original data set contains data about asylum applications from 2008 to 2014. For the demo purposes, we consider the subset of recent observations about asylum applications between 2013 and 2014, comprising approximately 80,000 observations.

II. Background: QB vs. QB4OLAP

A *QB data set* is a collection of so-called *observations* (in OLAP terminology *facts*) whose schema is specified by means of a *Data Structure Definition* (DSD) as an instance of the RDF class `qb:DataStructureDefinition`. This specification comprises a set of *component* properties representing *dimensions*, *measures*, and *attributes*, as shown below for a portion of the Eurostat data cube (RDF prefixes are omitted).

```
dsd:migr_asyappctzm rdf:type qb:DataStructureDefinition ;
    qb:component [ qb:dimension sdmx−dimension:refPeriod] ;
    qb:component [ qb:dimension property:age] ;
    qb:component [ qb:dimension property:citizen] ;
    ...
    qb:component [ qb:measure sdmx−measure:obsValue] .
```

From an OLAP analyst's point of view, QB has the following limitations: (a) *No native support of dimension hierarchies.* OLAP operations rely on the organization of dimension members into hierarchies defined in terms of aggregation levels. QB only allows representing relationships between dimension instances. Thus, for example, although Mary knows that Nigeria aggregates to Africa, there is no way to express that Nigeria is a country, Africa a continent, and that countries aggregate to continents. (b) *No native support to represent aggregate functions.* Most OLAP operations aggregate measure values along dimensions in a cube using the default aggregate function defined for the measure, which is not present in QB. (c) *No support for descriptive attributes.* In the MD model, dimension levels are associated with a set of *attributes* that describe the characteristics of their members. Lack of descriptive attributes is not only awkward from a user's point of view, but also inefficient. For example, if Mary wants to ask only for applications from Nigeria, she would need to know the IRI representing Nigeria[5].

The QB4OLAP[6] vocabulary addresses the drawbacks above by representing the most common features of the MD model as shown in [5]. It is currently being used in several research projects concerning OLAP over RDF data. From a well-formed MD schema we can again automate most of the OLAP processing, as done in traditional DW settings. Importantly, QB4OLAP has been devised to operate over observations published in QB without the need of rewriting them. Typically, observations are the largest part of the data, while dimensions are usually orders of magnitude smaller.

A key difference between QB and QB4OLAP is that, in the latter, facts represent relationships *between dimension levels* and fact instances (observations) that map *level members* to measure values. The *dimension levels* are represented in the same way as *dimensions*, i.e., as *component* properties, and they can be linked to the DSD via the `qb4o:level` property. Similarly, *aggregate functions* are also *component* properties that are linked to the DSD via the `qb4o:aggregateFunction` property associating measures with aggregate functions. Moreover, QB4OLAP defines the `qb4o:cardinality` property that represents the cardinality of the relationship between a fact and a *dimension level*. Finally, *level attributes* can be linked to a *dimension level* via the `qb4o:hasAttribute` property. Below, we show the cube structure of the Eurostat data set, represented in QB4OLAP.

```
schema:migr_asyappctzmQB4O rdf:type qb:DataStructureDefinition;
    qb:component [ qb4o:level sdmx−dimension:refPeriod ;
                   qb4o:cardinality qb4o:ManyToOne ];
    qb:component [ qb4o:level property:citizen ;
                   qb4o:cardinality qb4o:ManyToOne ] ;
    ...
    qb:component [ qb:measure sdmx−measure:obsValue;
                   qb4o:aggregateFunction qb4o:sum ] ;
```

Furthermore, we also show a portion of the structure of the *Citizenship* dimension, discovered by the tool that we present later. We show the definition of the dimension, the dimension levels (as part of the hierarchies), and hierarchy steps that represent roll-up relationship between levels. We point the interested reader to the QB4OLAP project's wiki[7] for details.

```
schema:citizenshipDim a qb:DimensionProperty ;
   qb4o:hasHierarchy schema:citizenshipGeoHier,
schema:citizenshipGeoHier a qb4o:Hierarchy ;
   qb4o:inDimension schema:citizenshipDim ;
   qb4o:hasLevel property:citizen, schema:continent, schema:citAll .
_:ih45 a qb4o:HierarchyStep ;
   qb4o:inHierarchy schema:citizenshipGeoHier ; qb4o:ChildLevel property:citizen ;
   qb4o:parentLevel schema:continent ; qb4o:pcCardinality qb4o:ManyToOne .
```

III. *QB2OLAP* Overview

QB2OLAP is organized in three main modules, *Enrichment*, *Exploration*, and *Querying*, as illustrated in Figure 1. By using the *Enrichment* module, the user generates the QB4OLAP graph. Then, this semantics is exploited by the *Exploration* module enabling the user to explore the QB4OLAP schema and by the *Querying* module enabling OLAP analysis. The schema and level instance enrichment triples are loaded into a local SPARQL endpoint. All modules provide graphical interfaces. *QB2OLAP* automatically generates and triggers the necessary SPARQL queries and handles the result triples. The Querying module also gives the possibility to manually formulate SPARQL queries. Next, modules' details are explained.

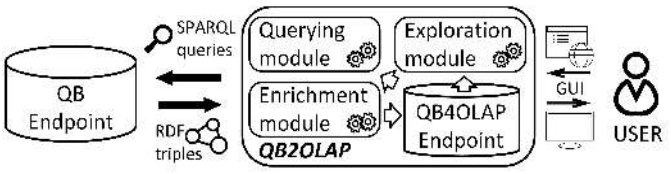

Fig. 1. *QB2OLAP* Architecture

A. Enrichment module

The enrichment of the QB data set is a labor-intensive task that is semi-automatized in the Enrichment module [6]. The user is released of the burden to manually explore the data set, define dimension levels and hierarchies, and generate the corresponding QB4OLAP triples. Instead, the Enrichment module triggers the queries, performs the necessary processing,

[5]Some data sets include a Label, although there is no guarantee about this.
[6]http://purl.org/qb4olap/cubes
[7]https://github.com/lorenae/qb4olap/wiki

makes suggestions for the user, and based on her choices enriches the schema. Thus, even an ordinary OLAP user can perform the enrichment on her own. The workflow of the Enrichment module is presented in Figure 2.

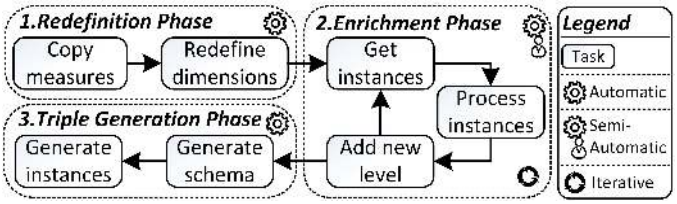

Fig. 2. The Enrichment Module Workflow

The first phase is the *Redefinition Phase* where the input schema of the QB graph is adjusted according to the QB4OLAP semantics, i.e., dimensions are redefined as levels (e.g., `[qb:dimension property:citizen]` is redefined to `[qb4o:level property:citizen; qb4o:cardinality qb4o:ManyToOne]`) while measures are copied and an aggregate function is assigned to them (e.g., `[qb:measure sdmx-measure:obsValue]` to `[qb:measure sdmx-measure:obsValue; qb4o:aggregateFunction qb4o:sum]`). Starting from the levels of this redefined schema, the *Enrichment Phase* collects the level instances and their properties. A query is run for each level instance and the results are processed to discover the properties that represent *functional dependencies* (FD) which are typically used in MD modeling to automatically discover potential roll-up relationships [7]. Therefore, such properties are automatically suggested to the user as sound candidates for coarser granularity level(s) (e.g., `schema:continent` for `property:citizen`). The user then chooses out of the automatically discovered candidate properties the roll-up relationships of her interest and by doing so, we drastically prune the search space guided by the user preferences. The tasks of the *Enrichment Phase* are iteratively repeated until the user has added all desired levels and conformed the dimension hierarchies. When a new level is added, the dimension hierarchies are automatically constructed or updated (e.g., a portion of triples related to the previous levels `schema:citizenshipGeoHier a qb4o:Hierarchy; qb4o:hasLevel property:citizen, schema:continent`) and the new *candidate hierarchy levels* for the added level are again discovered. Finally, once the *Enrichment Phase* is over, the RDF triples are automatically generated for both the schema and schema instances in the *Triple Generation Phase*. The generated triples are then exploited in the Exploration and Querying modules. Additionally, the Enrichment module also enables configuring fine-tuning parameters for the aggregate function, level detection, and triple generation. In the Linked Data dynamic context involving external and non-controlled data sources, the fine-tuning parameters that *QB2OLAP* offers are essential to deal with data quality issues, e.g., by searching for *quasi FDs* (i.e., an FD with an allowed error threshold).

The Enrichment module is implemented in Java 8. The Jena 2.13.0 library is used to manipulate RDF. QB and QB4OLAP graphs and the SPARQL endpoint are stored and run on Virtuoso 7 that is shared with the Exploration and Querying modules. The module interface is implemented in SWT.

B. Exploration and Querying modules

The *Exploration* module[8] allows to choose a data cube (represented in QB4OLAP) among a collection of cubes stored in an endpoint and, in a user-friendly fashion, navigate its dimension structures and instances. Graphics allow to explore the dimension instances and group them in many ways (e.g., hierarchies, dimensions, etc.).

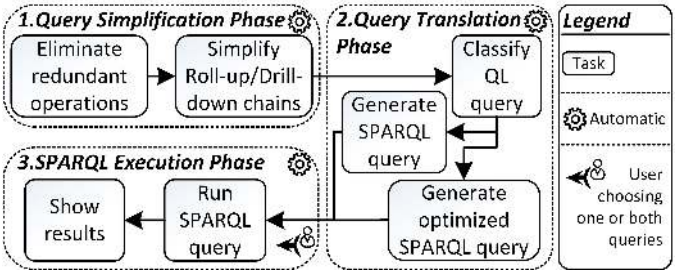

Fig. 3. The Querying Module Workflow

The *Querying* module lets the user write QL queries (or load predefined example queries) in a query editor. Its workflow is presented in Figure 3. QL follows the ideas introduced in the work by Ciferri et al. [8]. Basically, a QL program is a sequence of operations of the form (ROLLUP | SLICE | DRILLDOWN)* (DICE)*. Thus, we impose (for simplicity of processing) that dicing must always be written at the end of the QL program. In the *Query Simplification Phase* QL queries are then automatically simplified to produce better ones (e.g., the user may have included unnecessary operations, or written them in a non-optimal ordered sequence). The current implementation applies the following typical OLAP processing optimization rules: (a) perform SLICE operations as soon as possible, to reduce the size of intermediate results; and (b) group all the ROLLUP and DRILLDOWN operations over the same dimension, and replace them with a single ROLLUP *from the dimension's bottom level* to the latest level reached by the sequence of ROLLUP/DRILLDOWN operation(s).

After simplifying and optimizing the QL query, it is automatically translated into a single SPARQL expression in the *Query Translation Phase* as explained next. ROLLUPs are implemented navigating the roll-up relationships between members, guided by the dimension hierarchy representation provided by the QB4OLAP metadata, and aggregations are performed using GROUP BY clauses. Navigation is performed through SPARQL graph patterns (corresponding to joins). Since SLICE removes dimensions, this requires measure values to be aggregated up to a single value in the dimension being sliced out. The mechanism for this is the same used as to compute a ROLLUP. Lastly, a DICE operation is associated with a condition over measures and/or attribute values, and its result filters out of cells in the cube that do not satisfy the condition. We implemented these conditions using SPARQL FILTER clauses[9]. This way, the QL query is classified according to the existing query patterns and two SPARQL queries are generated. Both are semantically equivalent and one represents the direct translation while the other is an alternative query generated using optimization heuristics thought to deal with

[8]https://www.fing.edu.uy/inco/grupos/csi/apps/qb4olap/explorer

[9]Details and examples of the translation process can be found in http://cs.ulb.ac.be/conferences/ebiss2015/files/slides/vaisman_ebiss2015.pdf

some of the typical limitations of SPARQL endpoints. Finally, the user can choose to run either one or both queries and see the results in the *SPARQL Execution Phase*. The resulting cube is computed on-the-fly.

The Exploration and Querying modules are implemented in JavaScript and run on the Node.js platform. The interface of both modules is implemented with D3.js.

IV. Demonstration

In the on-site demonstration, we will show how Mary, our journalist, can use the three modules of *QB2OLAP* to do her work. The following scenarios will be demonstrated.

Enrichment. Starting from the QB data set loaded into the endpoint, Mary can use the Enrichment module to interactively retrieve the cube structure and candidate properties pointing to the possible higher dimension levels. Using the graphical interface in Figure 4, she is able to add new hierarchy levels to the cube. The cube structure is visualized as a tree that is updated after every change. Once all the levels are added, the triples representing the schema and level instances are loaded into the endpoint and used by the *Exploration and Querying* modules. We will also show that, in the presence of linked data sets, our tool is able to extract dimensional information (schema and instances) from other data sets (e.g., DBpedia).

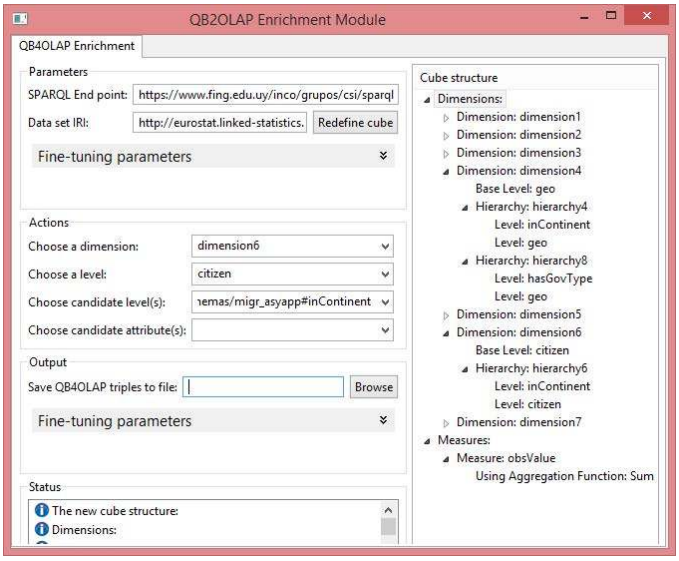

Fig. 4. The Enrichment Example

Exploration and Querying. With the enriched data, Mary can now explore the cube dimensions, hierarchies, attributes, etc., through the graphical interface in Figure 5. In the figure, Mary explores the dimensional cube data by clustering the instances according to their level value. Nodes represent level members (e.g., Syria) and edges represent roll-up relationships.

Once explored, Mary can write her own queries in QL (in the demo we include some predefined queries, which the audience can modify). For example, she can find the number of applications submitted by year by citizens from African countries whose destination is France, a query that could not be supported by the Eurostat site. The query (already simplified and rewritten) reads in QL:

```
PREFIX data: <http://eurostat.linked−statistics.org/data/>;
PREFIX schema: <http://www.fing.edu.uy/inco/cubes/schemas/migr_asyapp#>;
QUERY
$C1 := SLICE (data:migr_asyappctzm, schema:asylappDim);
$C2 := ROLLUP ($C1, schema:citizenshipDim,schema:continent);
$C3 := ROLLUP ($C2, schema:timeDim, schema:year);
$C4 := DICE ($C3, (schema:citizenshipDim|schema:continent|
                   schema:continentName = "Africa"));
$C5 := DICE ($C4, schema:destinationDim|property:geo|
                   schema:countryName = "France");
```

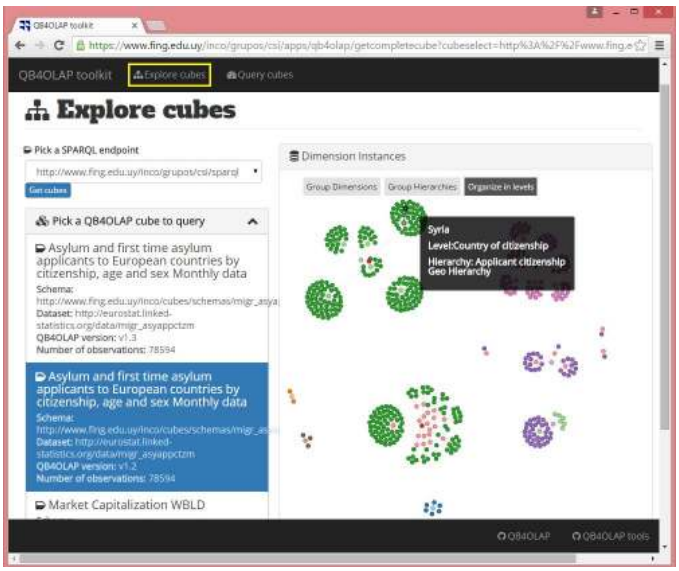

Fig. 5. The Exploration Example

A key feature to promote the use of our proposal is to relieve OLAP users from the need of learning a new and complex language like SPARQL. QL provides a higher abstraction level that is more intuitive to typical OLAP users that only need to write relatively simple QL programs (e.g., the above query translates to more than 30 lines of SPARQL) using OLAP algebra operations. Thus, they have the flexibility to analyze data cubes on-the-fly, since QB4OLAP provides the metadata needed to automatically translate QL into SPARQL. Further, graphical OLAP tools can be developed, and translated first into a mediator language like QL, and then to SPARQL (we omit the SPARQL translation here, for space reasons).

References

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

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

[3] A. Abelló et al., "Fusion cubes: Towards self-service business intelligence," *IJDWM*, vol. 9, no. 2, pp. 66–88, 2013.

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

[5] L. Etcheverry et al., "Modeling and querying data warehouses on the semantic web using QB4OLAP," in *DaWaK*, 2014, pp. 45–56.

[6] J. Varga et al., "Dimensional enrichment of statistical linked open data," *In submission*, 2015.

[7] O. Romero and A. Abelló, "A framework for multidimensional design of data warehouses from ontologies," *Data Knowl. Eng.*, vol. 69, no. 11, pp. 1138–1157, 2010.

[8] C. Ciferri et al., "Cube algebra: A generic user-centric model and query language for OLAP cubes," *IJDWM*, vol. 9, no. 2, pp. 39–65, 2013.