Interacting with Statistical Linked Data via OLAP Operations

Benedikt Kämpgen[1(✉)], Seán O'Riain[2], and Andreas Harth[1]

[1] Institute AIFB, Karlsruhe Institute of Technology, Karlsruhe, Germany
{benedikt.kaempgen,harth}@kit.edu
[2] Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland
sean.oriain@deri.org

Abstract. Online Analytical Processing (OLAP) promises an interface to analyse Linked Data containing statistics going beyond other interaction paradigms such as follow-your-nose browsers, faceted-search interfaces and query builders. Transforming statistical Linked Data into a star schema to populate a relational database and applying a common OLAP engine do not allow to optimise OLAP queries on RDF or to directly propagate changes of Linked Data sources to clients. Therefore, as a new way to interact with statistics published as Linked Data, we investigate the problem of executing OLAP queries via SPARQL on an RDF store. First, we define projection, slice, dice and roll-up operations on single data cubes published as Linked Data reusing the RDF Data Cube vocabulary and show how a nested set of operations lead to an OLAP query. Second, we show how to transform an OLAP query to a SPARQL query which generates all required tuples from the data cube. In a small experiment, we show the applicability of our OLAP-to-SPARQL mapping in answering a business question in the financial domain.

Keywords: OLAP · Query · Linked data · Statistics · XBRL · SPARQL

1 Introduction

Linked Data provides easy access to large amounts of interesting statistics from many organizations for information integration and decision support, including financial information from institutions such as the UK government[1] and the U.S. Securities and Exchange Commission.[2] However, interaction paradigms for Linked Data such as follow-your-nose browsers, faceted-search interfaces, and query builders [12,14] do not allow users to analyse large amounts of numerical data in an exploratory fashion of "overview first, zoom and filter, then details-on-demand" [22]. Online Analytical Processing (OLAP) operations on data cubes for viewing statistics from different angles and granularities, filtering for specific features, and comparing aggregated measures fulfil this information seeking

[1] http://data.gov.uk/resources/coins.
[2] http://edgarwrap.ontologycentral.com/.

E. Simperl et al. (Eds.): ESWC 2012 Satellite Events, LNCS 7540, pp. 87–101, 2015.
DOI: 10.1007/978-3-662-46641-4_7

mantra and provide interfaces for decision-support from statistics [2,5,19]. However, OLAP on statistical Linked Data imposes two main challenges:

- OLAP requires a model of data cubes, dimensions, and measures. Automatically creating such a multidimensional schema from generic ontologies such as described by Linked Data is difficult, and only semi-automatic methods have proved applicable [18]. Although the RDF Data Cube vocabulary (QB)[3] is a Linked Data vocabulary to model statistics in RDF and several publishers have already used the vocabulary for statistical datasets, there is yet no standard to publish multidimensional models as Linked Data.[4]
- OLAP queries are complex and require specialised data models, e.g., star schemas in relational databases, to be executed efficiently [11]. The typical architecture of an OLAP system consists of an ETL pipeline that extracts, transforms and loads data from the data sources into a data warehouse, e.g., a relational or multidimensional database. OLAP clients such as JPivot allow users to built OLAP queries and display multidimensional results in pivot tables. An OLAP engine, e.g., Mondrian, transforms OLAP queries into queries to the data warehouse, and deploys mechanisms for fast data cube computation and selection, under the additional complexity that data in the data warehouse as well as the typical query workload may change dynamically [10,17].

As a first effort to overcome these challenges, in previous work [13], we have presented a proof-of-concept to automatically transform statistical Linked Data that is reusing the RDF Data Cube vocabulary (QB) into a star schema and to load the data into a relational database as a backend for a common OLAP engine. OLAP queries are executed not on the RDF directly but by a traditional OLAP engine after automatically populating a data warehouse which results in drawbacks: (1) although the relational star schema we adopted is a quasi-standard logical model for data warehouses, our approach requires an OLAP engine to execute OLAP queries and does not allow to optimise OLAP queries to RDF; (2) if statistical Linked Data is updated, e.g., if a single new statistic is added, the entire ETL process has to be repeated, to have the changes propagated; (3) integration of additional data sources for more expressive queries is difficult, since multidimensional models have a fixed schema.

Therefore, in this work, we approach the following problem as illustrated in Fig. 1: At the backend, given statistical Linked Data reusing QB, crawled into a triple store, and accessible via a SPARQL endpoint. SPARQL is a standard query language for issuing queries to Linked Data. On the frontend, given a common OLAP client capable of running OLAP operations on data cubes, and capable of communicating those OLAP operations as an OLAP query via a common OLAP query language such as MDX. The Multidimensional Expression Language (MDX), as far as we know, is the most widely used OLAP query language, adopted by OLAP engines such as Microsoft SQL Server, the Open Java API for OLAP, XML for

[3] http://www.w3.org/TR/2012/WD-vocab-data-cube-20120405/.
[4] http://wiki.planet-data.eu/web/Datasets.

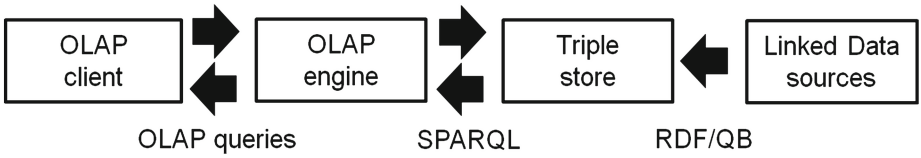

Fig. 1. Data flow for OLAP queries on statistical Linked Data in a triple store

Analysis (XMLA), and Mondrian. The question is how an OLAP engine can map MDX queries to SPARQL queries.

This paper presents a new way to interact with statistical Linked Data:

- We define common OLAP operations on data cubes published as Linked Data reusing QB and show how a nested set of OLAP operations lead to an OLAP query.
- We show how to transform an OLAP query to a SPARQL query which generates all required tuples from the data cube.

In the remainder of the paper, we first present a motivational scenario from the financial domain in Sect. 2. As a prerequisite for our contribution, in Sect. 3, we formally define a multidimensional model of data cubes based on QB. Then, in Sect. 4, we introduce OLAP operations on data cubes and present a direct mapping of OLAP to SPARQL queries. We apply this mapping in a small experiment in Sect. 5 and discuss some lessons learned in Sect. 6. In Sect. 7, we describe related work, after which, in Sect. 8, we conclude and describe future research.

2 Scenario: Analysing Financial Linked Data

In this section we describe a scenario of analysing Linked Data containing financial information. XBRL is an XML data format to publish financial information.[5] The U.S. Securities and Exchange Commission (SEC) requires companies to provide financial statement information in the XBRL format. The Edgar Linked Data Wrapper[6] provides access to XBRL filings from the SEC as Linked Data reusing QB. Those filings disclose balance sheets of a large number of US organizations, for instance that *RAYONIER INC* had a *sales revenue net* of 377,515,000 USD from 2010-07-01 to 2010-09-30.[7]

Using LDSpider, we crawled Linked Data from the Edgar wrapper and stored a data cube *SecCubeGrossProfitMargin* into an Open Virtuoso triple store. The data cube contains single disclosures from financial companies such as *RAYONIER INC*. Each disclosure either discloses cost of goods sold (CostOfGoodsSold) or sales revenue net (Sales) as measures. The two measures have unit USD and an

[5] http://www.xbrl.org/Specification/XBRL-RECOMMENDATION-2003-12-31+Corrected-Errata-2008-07-02.htm.
[6] http://edgarwrap.ontologycentral.com/.
[7] http://edgarwrap.ontologycentral.com/archive/52827/0001193125-10-238973#ds.

Columns (issuer) / Rows (dtstart, dtend)		RAYONIER INC	WEYERHAEUSER CO
2009-01-01	2009-3-31	*1,100,335 USD*	*0 values*
2009-04-01	2009-06-30	*2 values*	*2,300,800 USD*
...	...	...	...

Filters: CostOfGoodsSold

Fig. 2. Pivot table to be filled in our scenario

aggregation function that returns the number of disclosures, or – if only one – the actual number. Any disclosure is fully dependent on the following dimensions: the disclosing company (Issuer), the date a disclosure started (Dtstart) and ended (Dtend) to be valid, and additional meta information (Segment).

In our scenario, a business analyst wants to compare the number of disclosures of cost of goods sold for two companies. He requests a pivot table with issuers *RAYONIER INC* and *WEYERHAEUSER CO* on the columns, and the possible periods for which disclosures are valid on the rows, and in the cells showing the number of disclosed cost of goods sold, or – if only one – the actual number. Figure 2 shows the needed pivot table.

3 A Multidimensional Model Based on QB

In this section, as a precondition for OLAP queries on Linked Data, we formally define the notion of data cubes in terms of QB. The definition is based on a common multidimensional model, e.g., used by Gómez et al. [9], Pedersen et al. [20] and the Open Java API for OLAP.[8] Also, we base our definition on an RDF representation reusing QB, as well as other Linked Data vocabularies for publishing statistics, e.g., SKOS[9] and skosclass.[10]

Definition 1. (Linked Data store with RDF terms and triples). *The set of RDF terms in a triple store consists of the set of IRIs $\mathcal{I}$, the set of blank nodes $\mathcal{B}$ and the set of literals $\mathcal{L}$. A triple $(s, p, o) \in \mathcal{T} = (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object.*

Given a triple store with statistical Linked Data, we use basic SPARQL triple patterns on the store to define elements of a multidimensional model. Given a multidimensional element x, $id(x) \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ returns its RDF identifier:

[8] http://www.olap4j.org/.
[9] http://www.w3.org/2004/02/skos/.
[10] http://www.w3.org/2011/gld/wiki/ISO_Extensions_to_SKOS.

Member defines the set of members as $Member = \{?x \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}) | ?x$ a *skos:Concept* $\vee x \in \mathcal{L}\}$. Let $\mathcal{V} = 2^{Member}$, $V \in \mathcal{V}$, $ROLLUPMEMBER \subseteq Member \times Member$, $rollupmember(V) = \{(v_1, v_2) \in V \times V | (id(v_1)$ *skos:broader* $id(v_2) \vee id(v_2)$ *skos:narrower* $id(v_1))\}$. Note, in case of literal members $rollupmember(V)$ is empty.

Level defines the set of levels as $Level = \{(?x, V, rollupmember(V)) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{V} \times ROLLUPMEMBER | (?x$ a *skosclass:ClassificationLevel* $\wedge \forall v \in V(id(v)$ *skos:member* $?x)\}$. Let $\mathcal{L} = 2^{Level}$, $L \in \mathcal{L}$, $ROLLUPLEVEL \subseteq Level \times Level$, $rolluplevel(L) = \{(l_1, l_2) \in L \times L | (id(l_1)$ *skosclass:depth* $x) \wedge (id(l_2)$ *skosclass:depth* $y) \wedge x \leq y))\}$

Hierarchy defines the set of hierarchies as $Hierarchy = \{(?x, L, rolluplevel(L)) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{L} \times ROLLUPLEVEL | (?x$ a *skos:ConceptScheme* $) \wedge \forall l \in L(id(l)$ *skos:inScheme* $?x)\}$. Let $\mathcal{H} = 2^{Hierarchy}$.

Dimension defines the set of dimensions as $Dimension = \{(?x, H) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{H} | (?x$ a *qb:DimensionProperty* $) \wedge \forall h \in H(?x$ *qb:codeList* $id(h))\}$. Let $\mathcal{D} = 2^{Dimension}$.

Measure defines the set of measures as $Measure = \{(?x, aggr) \in (\mathcal{I} \cup \mathcal{B}) \times \{UDF\} | (?x$ a *qb:MeasureProperty* $)\}$ with $UDF : 2^{\mathcal{L}} \rightarrow \mathcal{L}$ a default aggregation function since QB so far does not provide a standard way to represent typical aggregation functions such as *SUM*, *AVG* and *COUNT*. If the input set of literals only contains one literal, *UDF* returns the literal itself, otherwise *UDF* returns a literal describing the number of values. *UDF* is an algebraic aggregation function in that it can be computed by distributive functions *COUNT* and *SUM* [10]. Conceptually, measures are treated as members of a dimension-hierarchy-level combination labelled "Measures". Let $\mathcal{M} = 2^{Measure}$.

DataCubeSchema defines the set of data cube schemas as $\{(?x, D, M) \in (\mathcal{I} \cup \mathcal{B}) \times \mathcal{D} \times \mathcal{M} | (?x$ a *qb:DataStructureDefinition* $\wedge \forall d \in D(?x$ *qb:component* $?comp \wedge ?comp$ *qb:dimension* $id(d)) \wedge \forall m \in M(?x$ *qb:component* $?comp \wedge ?$ *comp qb:measure* $id(m)))\}$.

Fact defines the set of possible statistical facts as $Fact = \{(?x, C, E) \in (\mathcal{I} \cup \mathcal{B}) \times 2^{Dimension \times Member} \times 2^{Measure \times Literal} | (?x$ a *qb:Observation* $) \wedge \forall (d_1, m_1), (d_2, m_2) \in C, id(m_1) \neq id(m_2) (?x id(d_1) id(m_1) \wedge id(d_1) \neq id(d_2)) \wedge \forall (m_1, v_1), (m_2, v_2) \in E, id(v_1) \neq id(v_2) (?x id(m_1) v_1 \wedge id(m_1) \neq id(m_2))$. Note that each fact is restricted to have for each dimension and measure at maximum one member and value, respectively. Let $\mathcal{F} = 2^{Fact}$.

DataCube defines the set of data cubes as $DataCube = \{(cs, F) \in DataCubeSchema \times \mathcal{F} | cs = (?x, D, M) \wedge \forall (?obs_1, C_1, E_1), (?obs_2, C_2, E_2) \in \mathcal{F}, ?obs_1 \neq ?obs_2 (?obs_1$ *qb:dataSet* $?ds \wedge ?ds$ *qb:structure* $?x) \wedge C_1 \neq C_2 \wedge \{d : \exists (d, m) \in C_1\} = D \wedge \forall (d, m) \in C_1 (id(m)$ *skos:member* $?l.?l$ *skos:inScheme* $?h.id(d)$ *qb:codeList* $?h \vee ?m$ *skos:notation* $id(m).?m$ *skos:member* $?l.?l$ *skos:inScheme* $?h.id(d)$ *qb:codeList* $?h)\}$ Note, the measure value is fully dependent on the dimension members, thus, any two facts need to have a different member on one of their dimensions. Also, any fact needs to have a member for each dimension mentioned in the schema. As a last requirement, each member needs to be contained in a level of a hierarchy of the dimension. A data

cube may be sparse and not containing facts for each possible combination of dimension members. If the member is a literal, there will be a concept representing this member and linking to its literal value via *skos:notation*. Similar to data in a fact table of a star schema, we assume that all facts of the data cube are on the lowest granularity level, since then, all measures on any higher aggregation level can be computed from these facts.

We distinguish *metadata queries* and *OLAP queries* on data cubes. Whereas metadata queries return multidimensional objects such as the cube schema, the dimensions, and the measures, OLAP queries on a certain data cube return tuples. The number of tuples that possibly can be queried from a data cube is exponentially growing with the number of dimensions, as the following definition shows:

Definition 2. (Data Cube Tuples). *Adopting the concept of Gray et al. [10], we can compute all tuples* $(c_1, \ldots, c_{|D|}, t_1, \ldots, t_j)$, $j \leq |M|$ *with* $c_i \in Member \cup ALL$ *and* $t_i \in T$ *with* T *a numeric domain including the special* null *value in case of cube sparsity from a data cube represented in QB as follows: We extract a relational table containing measures for each possible combination of dimension members from the data cube* (cs, F) *with data cube schema* $cs = (?x, D, M)$*, dimensions* $D = \{D_1, \ldots, D_N\}$ *and facts on the lowest granularity level* $(?obs, C, E) \in F$*. We compute* 2^N *aggregations of each measure value by the measure's aggregation function over all possible select lists of dimensions* $sl \in 2^D$ *with a standard* GROUP BY*. Then, we merge all aggregation results with a standard* UNION*, substituting the special* ALL *value for the aggregation columns. If the number of possible members of a dimension is* $card(D_i)$*, then the number of resulting facts in the materialised data cube is* $\prod(card(D_i) + 1)$*. The extra value for each dimension is* ALL*.*

Subqueries and aggregation functions in SPARQL 1.1 make easily possible to apply the relational concept of Gray et al. [10] to a data cube represented as Linked Data reusing QB. However, such a SPARQL query would have an exponential number of subqueries and would take a long time to execute. Also, the query would fully materialise the data cube, i.e. compute all possible tuples, although OLAP queries may require only a small subset. Therefore, in the next section, we show how to evaluate OLAP queries directly without subqueries and without fully materialising the cube.

4 Mapping OLAP Operations to SPARQL on QB

In this section we show how to issue OLAP queries on a multidimensional model. We define common OLAP operations on single data cubes [19–21]. A nested set of OLAP operations lead to an OLAP query. We describe how to evaluate such an OLAP query using SPARQL on QB. Figure 3 illustrates the effect of common OLAP operations, with inputs and outputs.

Note, this paper focuses on direct querying of single data cubes, the integration of several data cubes through *Drill-Across* or set-oriented operations such as

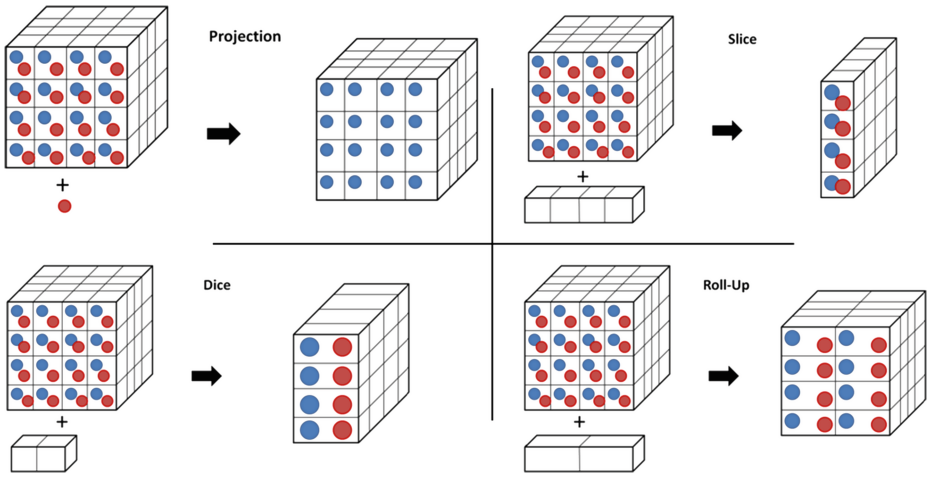

Fig. 3. Illustration of common OLAP operations with inputs and outputs (adapted from [21])

union, intersection, difference is out-of-scope. Multiple datasets can already be queried together if they are described by the same *qb:DataStructureDefinition*.

Each OLAP operation has as input and output a data cube. Therefore, operations can be nested. A nested set of OLAP operations lead to an OLAP query. For interpreting a set of OLAP operations as an OLAP query and evaluating the query using SPARQL on QB, we use and slightly adapt the notion of *subcube queries* [15].

Definition 3. (OLAP Query). *We define an OLAP query on a certain cube* $c = (cs, C)$, $cs = (?x, D, M)$ *as a subcube query* $Q = (c, q)$ *with* $c \in DataCube$ *and* q *a subcube query tuple* $(q_1, ..., q_{|D|}, m_1, ..., m_j)$, $j \leq |M|$ *[15]. The subcube query tuple contains for each dimension a tuple element* q_i, $Dimension(q_i) = D_i \in D$, $Hierarchy(q_i) = H_i \in Hierarchy$, $Level(q_i) = L_i \in Level$ *with* $dom(q_i) = \{?, ALL, x\}$. *The element* ? *marks a dimension as* inquired, *the* ALL *as aggregated, and* $x \in \mathcal{V}$ *fixes a dimension to a specific set of members. Also, for any queried measure the subcube query tuple contains an* $m_i \in M$. *For each dimension a granularity in the form of a hierarchy and level is specified. Note, for simplicity reasons we assume a fixed ordering of dimensions in the cube and in the subcube query tuple. An OLAP query returns a set of tuples from a data cube as defined by Definition 2.*

As examples, we describe three distinguishable subcube queries:

Full-Cube Query $(?, ?, ?, ..., m_1, ..., m_{|M|})$ returns the tuples on the highest granularity, i.e., the lowest level of each dimension, inside a data cube. //returns the tuples resulting from an aggregation over all dimensions. It contains all tuples described by the facts inside a data cube, plus any tuples not explicitly contained in the cube due to sparsity.

Point Query $(a_1, a_1, ..., a_{|D|}, m_1, ..., m_{|M|})$ with $a_i \in \mathcal{V}, |a_i| = 1, Dimension(a_i) = D_i$ returns one specific tuple from a data cube.

Fully-Aggregated Query $(ALL, ALL, \ldots, ALL, m_1, \ldots, m_{|M|})$ returns one single fact with measures aggregated over an empty select list.

In the following we describe how to evaluate each OLAP operation in terms of this query model and how a nested set of OLAP operations results in one specific OLAP subcube query. Given a data cube as input to an OLAP operation, the tuples from the cube are given as a full-cube query tuple $(?, ?, ?, ..., m_1, ..., m_{|M|})$.

Projection is defined as $Projection : DataCube \times Measure \rightarrow DataCube$ and removes a measure from the input cube and allows to query only for specific measures. We evaluate *Projection* by removing a measure from the subcube query tuple.

Slice is defined as $Slice : DataCube \times Dimension \rightarrow DataCube$ and removes a dimension from the input cube, i.e., removes this dimension from all selection lists over which to aggregate. We evaluate *Slice* by setting the tuple element of the dimension to *ALL*.

Dice is defined as $Dice : DataCube \times Dimension \times \mathcal{V} \rightarrow DataCube$ and allows to filter for and aggregate over certain dimension members. We evaluate *Dice* by setting the tuple element of that dimension to this particular set of members and aggregate over the set. Note, we regard dice not as a selection operation but a combined filter and slice operation.

Roll-Up is defined as $Roll-Up : DataCube \times Dimension \rightarrow DataCube$ and allows to create a cube that contains instance data on a higher aggregation level. We evaluate $Roll-Up$ on a dimension by specifying the next higher level of the specified hierarchy. Note, $Drill-Down$ can be seen as an inverse operation to $Roll-Up$.

As an example, consider an OLAP query on our *SecCubeGrossProfitMargin* cube for the cost of goods sold (*edgar:CostOfGoodsSold*) for each issuer (*edgar:issuer*) and each date until when each disclosure is valid (*edgar:dtend*), filtering by disclosures from two specific segments (*edgar:segment*). A nested set of OLAP operations that queries the requested facts can be composed as follows. In all our queries, we use prefixes to make URIs more readable:

```
Slice(
    Dice(
            Projection(
        edgar:SecCubeGrossProfitMargin,
        edgar:CostOfGoodsSold),
    edgar:segment,
    {edgar:segmentAHealthCareInsuranceCompany,
    edgar:segmentAResidentialRealEstateDeveloper}),
edgar:dtstart)
```

This query can then be represented as a subcube query with dimensions Issuer, Dtstart, Dtend, Segment:

```
(?, ALL, ?, {edgar:segmentAHealthCareInsuranceCompany,
edgar:segmentAResidentialRealEstateDeveloper}, edgar:
    CostOfGoodsSold)
```

Next, we describe how to evaluate such an OLAP query using a SPARQL query on QB. Since QB does not yet fully specify how to represent OLAP hierarchies, and since dimensions in our scenario were flat, in this paper, we simplify the problem of translating a subcube query to SPARQL and assume a queried data cube with only one hierarchy and level per dimension. A $Roll-Up$ has a similar effect to a *Slice* operation and can be added to our concept by a group-by not on the members of the lowest level of a dimension but on members of a higher level, specified via their $ROLLUPMEMBER$ and $ROLLUPLEVEL$ relations. An OLAP query $Q = (c, q)$ with $c \in DataCube$ and query tuple $q = (q_1, \ldots, q_{|D|}, m_1, ..., m_j)$, $j \leq |M|$ can be translated into a SPARQL query using the following steps:

1. We initialise the SPARQL query using the URI of the data cube. We query for all instance data from the data cube, i.e., observations linking to datasets which link to the data structure definition.
2. For each selected measure, we incorporate it in the SPARQL query by selecting additional variables for each measure and by aggregating them using the aggregation function of that measure, using *OPTIONAL* patterns for cases of cube sparsity.
3. For each inquired dimension, we add query patterns and selections for all the instances of *skos:Concept* in the specified level and hierarchy of the dimension. We query for the observations showing property-value pairs for each of these variables, either directly using concepts or using literals linked from the concepts via *skos:notation*. We use *OPTIONAL* patterns for cases of cube sparsity. To display inquired dimensions in the result and correctly aggregating the measures, we *GROUP BY* each inquired dimension variable.
4. For each fixed dimension, we filter for those observations that exhibit for each dimension one of the listed members.

We transform our example from above to the following SPARQL query. Note, *UDF* represents the standard aggregation function from our scenario:

```
select ?dimMem0 ?dimMem1 UDF(?measureValues0) where {
?obs qb:dataSet ?ds.
?ds qb:structure edgar:SecCubeGrossProfitMargin.
?dimMem0 skos:member edgar:issuerRootLevel.
    OPTIONAL {?obs edgar:issuer ?dimMem0. }
?concept1 skos:member edgar:dtendRootLevel.
?concept1 skos:notation ?dimMem1.
    OPTIONAL {?obs edgar:dtend ?dimMem1. }

?obs edgar:segment ?slicerMem0.
    Filter(?slicerMem0 = edgar:
        segmentAHealthCareInsuranceCompany
```

```
    OR ?slicerMem0 = edgar:
        segmentAResidentialRealEstateDeveloper)

OPTIONAL {?obs egar:CostOfGoodsSold ?measureValue0. }
} group by ?dimMem0 ?dimMem1
```

5 Experiment: Evaluating an OLAP Query on Financial Linked Data

In this section, we demonstrate in a small experiment the applicability of our OLAP-to-SPARQL mapping to our scenario from the financial domain. In this experiment, we have a triple store with around 148,426 triples. The triple store describes a data cube *edgar:SecCubeGrossProfitMargin* that contains 17,448 disclosures that either disclose cost of goods sold or sales revenue net. The values of the measures fully depend on one of 625 different issuers (dimension *edgar:issuer*), the date a disclosure started (27 members of dimension *edgar:dtstart*) and ended (20 members of *edgar:dtend*) to be valid, and additional information (21,227 members of *edgar:segment*). The two measures (*edgar:CostOfGoodsSold* and *edgar:Sales*) have the unit USD and an aggregation function that returns the number of disclosures, or – if only one – the actual number. If fully materialised according to Definition 2, the cube contains $626 \cdot 28 \cdot 21 \cdot 21,228 = 7,813,772,064$ facts. To compute all of its facts, $2^4 = 16$ SPARQL subqueries would be needed.

In order to answer the OLAP question of our scenario, we use an OLAP client such as Saiku to compose OLAP operations to an OLAP query in MDX:

```
SELECT
{edgar:cik1417907idConcept, edgar:cik106535idConcept} ON
    COLUMNS,
CrossJoin(edgar:dtstartRootLevel.Members, edgar:
    dtendRootLevel.Members) ON ROWS
FROM [edgar:SecCubeGrossProfitMargin]
WHERE {edgar:CostOfGoodsSold}
```

The MDX query is sent to the OLAP engine, the resulting tuples will be visualised using a pivot table, a compact format to display multidimensional data [6]. Multidimensional elements are described in the MDX query using their unique URIs.[11] For an introduction to MDX, see its website.[12] A more detailed description of how to transform an MDX query into an OLAP query due to space constraints we leave for future work when we evaluate our OLAP-to-SPARQL mapping more thoroughly.

Now, we show that an MDX query can be transformed into an OLAP subcube query according to Definition 3 and evaluate the subcube query using SPARQL.

[11] Note, URIs need to be translated to an MDX-compliant format that does not use reserved MDX-specific characters, which is why we use the prefixed notation of URIs.

[12] http://msdn.microsoft.com/en-us/library/aa216770.

The result is a subset of all possible tuples from a data cube. The pivot table determines what dimensions to display on its columns and rows.

A nested set of OLAP operations to compose our OLAP query is as follows:

```
Slice(Projection(
    edgar:SecCubeGrossProfitMargin,
    edgar:CostOfGoodsSold),
edgar:segment)
```

This query can then be represented as a subcube query with dimensions Issuer, Dtstart, Dtend, Segment: $(?, ?, ?, ALL, CostOfGoodsSold)$. The resulting SPARQL query is as follows:

```
select ?dimMem0 ?dimMem1 ?dimMem2 count(xsd:decimal(?
    measureValue0)) sum(xsd:decimal(?measureValue0))
where {
?obs qb:dataSet ?ds.
?ds qb:structure edgar:SecCubeGrossProfitMargin.

?dimMem0 skos:member edgar:issuerRootLevel.
    OPTIONAL {?obs edgar:issuer ?dimMem0. }

?values1 skos:member edgar:dtstartRootLevel.
?values1 skos:notation ?dimMem1.
    OPTIONAL {?obs edgar:dtstart ?dimMem1. }

?values2 skos:member edgar:dtendRootLevel.
?values2 skos:notation ?dimMem2.
    OPTIONAL {?obs edgar:dtend ?dimMem2. }

OPTIONAL {?obs edgar:CostOfGoodsSold ?measureValue0. }
} group by ?dimMem0 ?dimMem1 ?dimMem2
```

UDF, our default aggregation function is algebraic, therefore, we had to compute the *SUM* and *COUNT* for the measure. We run the query after a reboot of the triple store. The query took 18sec and returned 58 tuples to be filled into the requested pivot table. The number of $7,813,772,064$ potential tuples in the cube does not have a strong influence on the query since the cube is very sparse, for instance, the triple store contains observations only for a fraction of segment members.

6 Discussion

In our experiment, we show the applicability of our mapping between OLAP and SPARQL queries. We correctly aggregate data on one specific granularity, defined by the mentioned inquired and fixed dimensions. Dimensions that are not mentioned will be automatically handled as having an *ALL* value [10], representing all

possible values of the dimension. The aggregation results in correct calculations, since we assume that a QB data cube only contains facts on the lowest granularity level. Only an aggregation of observations from different granularities would result in incorrect numbers, e.g., a *SUM* over gender *male*, *female*, and *total*.

The SPARQL query created by our approach shows sufficiently fast in our small experiment but may not scale for larger datasets for the following reasons: First, data from the data cube is queried on demand, and no materialisation is done. Every OLAP query is evaluated using a SPARQL query without caching and reusing of previous results. Second, a *Dice* operation currently always includes a *Slice* operation; thus, all member combinations of inquired dimensions are calculated, even though only specific combinations might be required, as in the case of the two issuers in the OLAP query of our scenario. Third, OLAP clients and pivot tables require multidimensional data, i.e., data cubes containing facts linking to specific members of dimensions, but our SPARQL query returns relational tuples. Using the unique identifiers of dimensions, members, and measures, query result tuples need to be joined with the multidimensional data points as required by the OLAP client for filling the pivot table. Another possibility would be to use SPARQL *CONSTRUCT* queries to first materialise data cubes resulting from OLAP operations as RDF [8]. Populating the pivot table could then be done by simple SPARQL *SELECT* queries on this resulting multidimensional view. However, the applicability and performance of this approach to answer OLAP queries still needs to be evaluated.

In summary, though our OLAP algebra to SPARQL mapping may not result in the most efficient way to answer an business question and require additional efforts for usage in OLAP clients, it correctly computes all required tuples from the data cube with one SPARQL query, without the need for explicitly introducing the non-relational *ALL* member or using sub-queries [10].

7 Related Work

Kobilarov and Dickinson [14] have combined browsing, faceted-search, and query-building capabilities for more powerful Linked Data exploration, similar to OLAP, but not focusing on statistical data. Though years have passed since then, current literature on Linked Data interaction paradigms does not seem to expand on analysing large amounts of statistics.

OLAP query processing generally distinguishes three levels [4]: On the conceptual level, algebras of OLAP operations over data cubes are defined that are independent from a logical representation [1,3,9,19,21]. On the logical level, query processing mainly depends on the type of data structure on which to perform the computations and in which to store the results. Data structures can roughly be grouped into ROLAP, using relational tables and star or snowflake schemas, and MOLAP, using multidimensional arrays for directly storing and querying of data cubes [2,10,15,23]. The physical level is concerned with efficient execution of low-level executions such as index lookup or sorting over the data stored given a specific hardware and software.

The execution of OLAP operations mainly is concerned with the computation of the data cube and with storing parts of the results of that computation to

efficiently return the results, to require few disk or memory space, and to remain easy to update if data sources change [15,17].

In this work we use the graph-based RDF data model and the QB vocabulary for querying and storing of multidimensional data. Both schema information and actual data is accessed using the Linked Data principles and managed using SPARQL on a triple store.

Other authors recognise the reduced initial processing and update costs of using triple stores and other dedicated query engines for query processing [18], but do not present approaches for executing OLAP queries directly over such Semantic Data Warehouses.

Etcheverry and Vaisman [8] present an algorithm to translate OLAP operations such as Roll-Up and Slice to SPARQL CONSTRUCT queries. Since results are cubes, one can nest operations. Different from our work, OLAP operations are executed for preprocessing of cubes from the Web that are then exported to a data warehouse for query processing.

Although there may be more efficient querying approaches such as special indexing and caching, to the best of our knowledge, this is the first work on executing OLAP queries on data cubes represented as RDF using SPARQL.

Several authors [20,24] motivate the integrating of data from the Web in OLAP systems. Yin and Pedersen [24] present a federated approach to "decorate" Data Cubes with virtual dimensions built from external data, which means that XML data can also be used in filter operations. For instance, Members of a level "Nation" in a Data Cube are linked to nations in an XML document providing additional information such the population which then can be used to filter for certain nations. Different from this approach, we directly execute multidimensional queries using SPARQL over RDF.

Diamantini and Potena [7] enrich a data cubes with a domain ontology represented using OWL as well as a mathematical ontology represented in XML standards for mathemtical descriptions such as MathML. Their goal is to provide analysts with useful background information and to possibly allow novel types of analyses, e.g., drill-down into single compound measures.

Mazón et al. [16] also motivate the use of external data to enhance Data Cubes; they propose the use of semantic relations such as hypernymy ("is-a-kind-of", generalization, e.g., cake is kind of baked goods) and meronymy ("is-a-part-of", aggregation, e.g., wheel is a part of car) between concepts provided by WordNet to enrich Dimension Hierarchies.

Although we so far only translate the common analytical operations, our approach could be extended with operations allowing filtering over decorations described in RDF. Also, OLAP client interfaces can be extended to display additional information related to analysed data cubes.

8 Conclusions and Future Work

We have presented an approach to interact with statistical Linked Data using Online Analytical Processing operations of "overview first, zoom and filter, then

details-on-demand". For that, we define projection, slice, dice and roll-up operations on single data cubes in RDF reusing the RDF Data Cube vocabulary, map nested sets of OLAP operations to OLAP subcube queries, and evaluate those OLAP queries using SPARQL. Both metadata and OLAP queries are directly issued to a triple store; therefore, if the RDF is modified or updated, changes are propagated directly to OLAP clients. Though, our OLAP-to-SPARQL mapping may not result in the most efficient SPARQL query and require additional efforts in populating requested pivot tables, we correctly calculate required tuples from a data cube without inefficient full materialisation and without the need for explicitly introducing the non-relational *ALL* member or for using subqueries.

Future work may be conducted in three areas: 1) extending our current approach with OLAP hierarchies and OLAP operations over multiple cubes, e.g., drill-across; 2) implementing an OLAP engine to more thoroughly evaluate our current OLAP-to-SPARQL mapping and to investigate more efficient OLAP query execution plans, e.g., using RDF views; 3) investigating more Linked-Data-specific OLAP clients that allow external information to be used in queries and displayed.

Acknowledgements. The work presented in this paper has been funded in part by the German Ministry of Education and Research (BMBF) within the SMART research project (Ref. 02WM0800), the European Community's Seventh Framework Programme FP7/2007-2013 within the PlanetData project (Grant 257641) and MONNET project (Grant 248458), and the Science Foundation Ireland (Lion-2, Grant SFI/08/CE/I1380).

References

1. Agrawal, R.:, Gupta, A., Sarawagi, S.: Modeling multidimensional databases. In: Proceedings of the Thirteenth International Conference on Data Engineering (1997)
2. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. ACM SIGMOD Record **26**, 65–74 (1997)
3. Chen, L., Ramakrishnan, R., Barford, P., Chen, B.C., Yegneswaran, V.: Composite subset measures. In: VLDB2006 Proceedings of the 32nd International Conference on Very Large Data Bases (2006)
4. Ciferri, C., Ciferri, R., Gómez, L., Schneider, M.: Cube algebra: a generic user-centric model and query language for OLAP cubes. IJDWM **9**, 39–65 (2012)
5. Codd, E., Codd, S., Salley, C.: Providing OLAP to user-analysts: an IT mandate. Technical report (1993)
6. Cunningham, C., Galindo-Legaria, C.A., Graefe, G.: PIVOT and UNPIVOT: optimization and execution strategies in an RDBMS. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, vol. 30 (2004)
7. Diamantini, C., Potena, D.: Semantic enrichment of strategic datacubes. In: Proceedings of the ACM 11th International Workshop on Data Warehousing and OLAP (2008)
8. 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)

9. Gómez, L.I., Gómez, S.A., Vaisman, A.A.: A Generic data model and query language for spatiotemporal OLAP cube analysis categories and subject descriptors. In: EDBT 2012 (2012)
10. Gray, J., Bosworth, a., Lyaman, a., Pirahesh, H.: Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS. in: Proceedings of the Twelfth International Conference on Data Engineering (1995)
11. Harinarayan, V., Rajaraman, A.: Implementing data cubes efficiently. ACM SIGMOD Rec. **25**, 205–216 (1996)
12. Harth, A.: Visinav: a system for visual search and navigation on web data. J. Web Seman. **8**(4), 348–354 (2010)
13. Kämpgen, B., Harth, A.: Transforming statistical linked data for use in OLAP systems. In: I-Semantics 2011 (2011)
14. Kobilarov, G., Dickinson, I.: Humboldt: exploring linked data. In: Linked Data on the Web Workshop (LDOW2008) at WWW2008 (2008)
15. Li, X., Han, J., Gonzalez, H.: High-dimensional OLAP: a minimal cubing approach. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, vol. 30 (2004)
16. Mazón, J.-N., Trujillo, J., Serrano, M., Piattini, M.: Improving the development of data warehouses by enriching dimension hierarchies with WordNet. In: Collard, M. (ed.) Ontologies-Based Databases and Information Systems. LNCS, vol. 4623. Springer, Heidelberg (2007)
17. Morfonios, K., Konakas, S., Ioannidis, Y., Kotsis, N.: ROLAP implementations of the data cube. ACM Comput. Surv. **39**, 12 (2007)
18. Nebot, V., Berlanga, R.: Building data warehouses with semantic web data. Decis. Support Syst. **52**, 853–868 (2012)
19. Pardillo, J., Mazón, J.N., Trujillo, J.: Bridging the semantic gap in olap models: platform-independent queries. In: Proceedings of the ACM 11th International Workshop on Data Warehousing and OLAP (2008)
20. Pedersen, T.B., Gu, J., Shoshani, A., Jensen, C.S.: Object-extended OLAP querying. Data Knowl. Eng. **68**, 453–480 (2009)
21. Romero, O., Marcel, P., Abelló, A., Peralta, V., Bellatreche, L.: Describing analytical sessions using a multidimensional algebra. In: Cuzzocrea, A., Dayal, U. (eds.) Data Warehousing and Knowledge Discovery. LNCS, pp. 224–239. Springer, Heidelberg (2011)
22. Shneiderman, B.: The Eyes Have It: A task by data type taxonomy for information visualizations. In: Information Visualization (1996)
23. Vassiliadis, P., Sellis, T.: A survey of logical models for OLAP databases. ACM Sigmod Rec. **28**, 64–69 (1999)
24. Yin, X., Pedersen, T.B.: Evaluating XML-extended OLAP queries based on a physical algebra. Proceedings of the 7th ACM international workshop on Data warehousing and OLAP - DOLAP '04 p. 73 (2004)