SPARQLytics: Multidimensional Analytics for RDF

Michael Rudolf[1,2] Hannes Voigt[1] Wolfgang Lehner[1]

Abstract: With the rapid growth of open RDF data in recent years, being able to perform multidimensional analytics with it has become more and more important, in particular for the data analyst performing explorative business intelligence tasks. Existing analytic approaches are often not flexible enough to address the needs of data analysts and enthusiasts with iterative exploratory workflows. In this paper we propose SPARQLytics, a tool that exposes the concepts of multidimensional graph analytics by offering standard OLAP cube operations and generating SPARQL queries. Our evaluation shows that SPARQLytics unburdens data analysts from writing many lines of SPARQL code in iterative data explorations and at the same time it does not impose any overhead to query execution. SPARQLytics fits well with interactive computing tools, such as Jupyter, providing data enthusiasts with a familiar work environment.

1 Introduction

Today, enterprise business intelligence does not rely only on well-controlled in-house data anymore but also makes heavily use of external data sources. Applications, such as marketing research and sentiment analysis, are more and more data-driven. Since they investigate aspects outside of the corporate realm, external data is essential. Hence, the abundance of open data in the web that has become freely available over the last decade is of particular interest. It may allow novel insights that go well beyond the limits of traditional reporting and analysis. 5-star open data,[4] such as the Linked Open Data cloud,[5] follows the RDF data model, links to other open datasets, and is provided as SPARQL endpoints so that it can be directly queried over the web. There is a plethora of such interconnected datasets available.

Open RDF data is particularly useful for data enthusiasts [Mo14], who, on the one hand, do not engage in weekly and monthly reporting but investigate and explore beyond the business segment to discover new trends and business opportunities [Ab13, Ab15]. On the other hand, multidimensional analytics is an important part of the data analyst's toolbox. It is well understood and results can be easily communicated to business colleagues who are familiar with multidimensional reporting. As data enthusiasts explore the unknown, they work iteratively [Bl14]. They play with data, try out initial hypotheses, dismiss some, follow others, dig deeper, refine questions, etc. Every answer can spark new questions.

The traditional OLAP ecosystem offers a wide range of frontend tools for multidimensional analytics. These tools typically speak to an MDX backend provided by a data warehouse

[1] TU Dresden, Database Technology Goup, 01062 Dresden, `firstname.lastname@tu-dresden.de`

[2] SAP SE, Dietmar-Hopp-Allee 16, 69190 Walldorf, `michael.rudolf01@sap.com`

[4] See `http://5stardata.info/`

[5] See `http://lod-cloud.net/`

with pre-loaded data in a schema designed up-front. As this limits the analytical scenarios to what the schema designer or the metadata provider envisions at design-time, it is not a good fit for a data enthusiast. When working in an ad-hoc iterative exploratory manner, every new question quickly requires new dimensions, new facts, new cubes, or even new data sources. Speaking to a schema designer and re-engineering ETL processes for every other iteration is not a solution.

As a remedy, we propose SPARQLytics, providing the following characteristics:

- SPARQLytics introduces a dedicated component between SPARQL endpoints and the user. It exposes the concepts of multidimensional graph analytics explicitly and facilitates their use in an iterative exploratory work with RDF data.
- SPARQLytics modularizes the ad-hoc definition of multidimensional cubes on RDF data and offers standard OLAP cube operators. Building blocks, such as dimensions and measures, can be easily re-used, reducing SPARQL code writing to a minimum.
- SPARQLytics keeps track of all artifacts in a repository and generates the corresponding SPARQL query that computes measures on a specific cube. It also submits the query to a SPARQL endpoint, hiding most of the technical aspects of the endpoint communication from the user.
- SPARQLytics provides a concise DSL that fits very well with interactive computing tools, such as Jupyter.[6] It builds on SPARQL triple patterns, so that the user can leverage existing SPARQL expertise and quickly reach the productivity break-even.

In Sect. 2 of the paper we present and define the SPARQLytics DSL, while Sect. 3 evaluates it based on the business intelligence workload of the Social Network Benchmark proposed by the Linked Data Benchmark Council (LDBC) [Bo13]. We elaborate on related work in Sect. 4 and conclude the paper in Sect. 5.

2 SPARQLytics

For the SPARQLytics DSL we employ a formalization of multidimensional graph analytics [Ru14] based on the more generic property graph model, which we adapted to the RDF data model. For the sake of readability we re-use production rules from the SPARQL grammar (indicated with angle brackets) where possible.

2.1 Cube Definition

In order to perform OLAP operations, the observations (i.e. facts) to be analyzed and the dimensions and measures of interest need to be defined first. In the remainder of this section we illustrate the various commands provided in our language by means of a running example. To that end we use RDF data generated by the LDBC [Bo13] Social Network Benchmark.[7]

[6] See `http://jupyter.org/`

[7] See `http://ldbcouncil.org/benchmarks/snb`

2.1.1 Fact Selection

In an RDF graph a fact can be an attribute of a vertex or the presence of one or several edges—thus, we can generalize a fact to be a subgraph. The selection of subgraphs from a larger graph can be achieved in various ways, but to us the most convenient one seems to be with the help of graph pattern matching, as it is sufficiently abstract and can be used with a graphical specification or with a dedicated textual language.

In order to facilitate the selection of facts in our language, we re-use certain concepts from SPARQL, because pattern matching is a central building block of it: the prologue steers the resolution of names with the help of prefixes for shortening identifiers and a group graph pattern combines basic graph patters, which consist of triple blocks. The following listing shows the syntax for selecting facts.

```
<Prologue>
SELECT FACTS <GroupGraphPattern>
```

In the following example first three namespace prefixes are defined.[8] Then people who are at least 18 years old (or who will turn 18 this year) are selected as facts for subsequent multidimensional analyses.

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX snvoc:   <http://www.ldbc.eu/ldbc_socialnet/1.0/vocabulary/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
SELECT FACTS {
 ?person rdf:type snvoc:Person ;
         snvoc:birthday ?birthday .
 FILTER (YEAR(NOW()) - YEAR(?birthday) >= 18)
}
```

2.1.2 Dimension Specification

The most elementary concept in multidimensional analytics is that of a dimension. It is an aspect of the facts and as such a set of values. In order to better understand the aspect of the facts, the dimensional values are often structured into hierarchies, so that groups of facts can be subsumed by navigating through the levels of the hierarchy. A dimension specification consists of an ordered set of levels and a seed pattern:

```
DEFINE DIMENSION <String> FROM (<TriplesBlock>)
WITH ( LEVEL <String> AS <NumericExpression>, ... )
```

A *seed pattern* is a graph pattern (denoted by the production rule `<TriplesBlock>` from the SPARQL grammar) that is matched against the facts and connects them with the level expressions. This works by binding the variables in the level expression to the values of

[8] The `dbpedia` namespace prefix is not used in the example, it is needed later on.

the equally-named variables in the match of the seed pattern—it is an error if the level expression contains variables not present in the seed pattern. When a level expression is then applied to a match of the seed pattern in a fact, it yields the level member as its value. Note that a seed pattern is not actually necessary for connecting dimensions to facts, because level expressions could be applied directly to the facts. We deliberately introduce them for decoupling facts and dimension specifications in order to enable the re-use of the latter in other analytical scenarios. By encoding in the seed pattern only what is absolutely required from the level expressions, a single dimension specification can easily be applied to a multitude of different facts, provided that the seed pattern matches.

Note the use of names for levels and dimensions—they are required for univocally identifying those constructs within OLAP operations. Although the name of the SPARQL grammar production rule `<NumericExpression>` suggests a numeric result, the type of level expressions is not restricted to that. It can be used to derive any kind of expression, for which an equivalence relation is defined (i.e., equality comparison has to be supported). The following example specifies the two dimensions `Location` and `Birth Date`.

```
DEFINE DIMENSION "Location" FROM (
  ?person  snvoc:isLocatedIn ?city .
  ?city    snvoc:isPartOf ?country .
  ?country snvoc:isPartOf ?continent
) WITH (
  LEVEL "City" AS ?city,
  LEVEL "Country" AS ?country,
  LEVEL "Continent" AS ?continent
)
```

```
DEFINE DIMENSION "Birth Date" FROM (
  ?person snvoc:birthday ?birthday
) WITH (
  LEVEL "Day" AS DAY(?birthday),
  LEVEL "Month" AS MONTH(?birthday),
  LEVEL "Year" AS YEAR(?birthday)
)
```

In practice it can be desirable to support multiple hierarchies within a single dimension, such as time by calendar year and fiscal year or location by political and topographic division. That would require a partial ordering of dimension levels, but as the same can be achieved with multiple dedicated dimensions sharing the same seed pattern, we deliberately simplify our model for the sake of understanding.

2.1.3 Measure Definition

Measures are numerical values derived from sets of facts and constitute the result of a multidimensional analysis. As with dimension specifications, measure definitions also contain a seed pattern that decouples the derivation of a numerical value from the underlying facts. The following listing shows the syntax for defining a measure with the given name, based on a numeric expression over the variable(s) bound in the specified triples block, with the aggregation performed by the function with the given name.

```
DEFINE MEASURE <String> AS <NumericExpression>
WHERE (<TriplesBlock>) WITH <String>
```

The following example illustrates the definition of two measures: the first measure counts the number of languages a person speaks and is aggregated for groups of people by computing the average, whereas the second measure returns only the maximum length of comments for a single person and also for groups of people.

```
DEFINE MEASURE "Avg. No. Languages"
 AS COUNT(?language) WHERE (
  ?person snvoc:speaks ?language
 ) WITH "AVG"
```

```
DEFINE MEASURE "Max. Comment Length"
 AS MAX(?length) WHERE (
  ?comment snvoc:hasCreator ?person ;
           rdf:type snvoc:Comment ;
           snvoc:length ?length
 ) WITH "MAX"
```

2.1.4 Cube Creation

After the facts, dimensions, and measures have been specified individually and before OLAP operations can be executed, they have to be assembled into a cube. Formally, a graph cube $c := (F, D, M)$ is a triple, where $F := \text{match}(G, p)$ is the set of facts matched by applying the fact selection pattern p to the graph G, D is a set of dimensions, and M is a set of measures. The following listing shows the syntax and an example invocation for creating a cube from the previously specified fact pattern and the dimensions and measures referenced by their names.

```
CREATE CUBE <String> FROM <String>, ... WITH <String>, ...

CREATE CUBE "QB" FROM "Location", "Birth Date"
WITH "Avg. No. Languages", "Max. Comment Length"
```

2.2 OLAP Operations

Similar to SQL, SPARQL is stateless: in between two queries no state is maintained. As a consequence, each multidimensional SPARQL query must contain all information related to the involved facts, dimensions, and measures, which makes them complex and error-prone if written by hand. To simplify multidimensional analytics, we therefore introduce the concept of an *analytical session*, which maintains state in between OLAP operations.

To initiate an analytical session, the data cube that should be subject to the analysis has to be selected from the repository. Since the graph cube is merely a dataset description, the SPARQL endpoint to execute the generated queries against must be specified. The optional dataset clause permits the selection of a specific RDF dataset in case multiple are present:

```
USING CUBE <String> OVER <IRIREF> <DatasetClause>
```

Slicing and dicing are two well-known ways of filtering the facts in a cube. For slicing (i.e. cutting a slice off) a cube, only facts for a specific level member are preserved. The dice operation accepts a more general predicate selecting a range of level members. The following listing shows the syntax for the two operations with the first two parameters being the names of the dimension and level, respectively.

```
SLICE(<String>, <String>, <PrimaryExpression>)
DICE(<String>, <String> AS <Var>, <ConditionalOrExpression>)
```

An analytical session associates a granularity with the cube, initialized to the lowest level of each dimension. The roll-up operation decreases the granularity associated with the graph cube for the specified dimension whereas the drill-down operation increases it:

```
ROLLUP(<String>, <Integer>)
DRILLDOWN(<String>, <Integer>)
```

Note that similar to the slice/dice operations, the grouping of facts is only modified conceptually by the roll-up and drill-down operations, as the cube's granularity and stored filters are only evaluated when actually computing values for measures.

2.3 Query Generation

To actually have values computed for the measures of interest with the analytical session's current granularity and filters taken into consideration, the user has to issue a corresponding statement listing the names of the desired measures (as in SQL, a star can be specified for all measures). That statement can optionally be accompanied by solution modifiers imposing a sort order on dimensions or measures as well as a limit and an offset, which are helpful for pagination or top-*k* queries:

```
COMPUTE(* | <String>, ...) ORDER BY <String> ASC | DESC ...
LIMIT <Integer> OFFSET <Integer>
```

This will cause the SPARQL query implementing the fact selection, grouping and aggregation operations and projecting to the given measures to be generated. Note that there are no round trips to the underlying triple store or SPARQL endpoint required for looking up metadata, as all necessary information has been specified before. A typical workflow consists of several measure computations interleaved with changes to the granularity of the cube and its contained facts.

In the following example the first 10 values for the two previously defined measures are computed, ordered by the `Birth Date` dimension:

```
COMPUTE("Avg. No. Languages", "Max. Comment Length")
ORDER BY "Birth Date" ASC LIMIT 10
```

A measure value by itself is meaningless, it needs to be seen in context—the facts that were used in the computation. The group of facts is represented by level members: for each dimension of the graph cube, the corresponding member is returned by applying the expression of the level indicated by the cube's granularity to the fact.

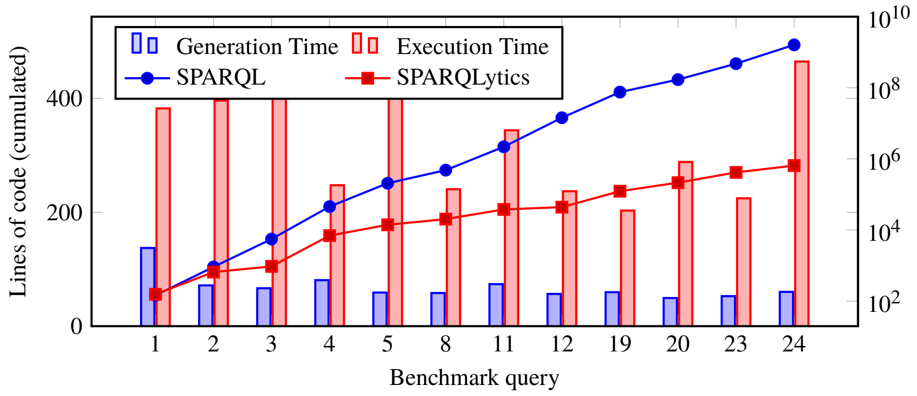

Fig. 1: Accumulated lines of SPARQL and SPARQLytics code for the sequential execution of the implemented benchmark queries (left y-axis) as well as generation and execution times (right y-axis).

3 Evaluation

We have implemented the SPARQLytics DSL described in the previous section in a Java-based prototype using Apache Jena[9] and made it freely available online.[10] A parser translates the commands entered by the user into operations executed in the context of an analytical session, which is backed by a repository of multidimensional artifacts. Whenever the calculation of measures is requested, the corresponding SPARQL query is generated and executed against the chosen endpoint and the results are returned.

As explained in the introduction, the challenge in the big data era lies in use cases that have the human in the loop, and to the best of our knowledge, no benchmark covering such workloads exists today. The Social Network Benchmark proposed by the LDBC [Bo13], is still under development, and at the time of this writing the most recently published version consists only of an interactive workload. However, this benchmark is currently being extended with query descriptions that are typical for business intelligence on graph-structured data.[11] Whereas half of the 24 queries currently proposed express classical data mining tasks, the other half qualify as traditionally multidimensional, identifying dimensions and computing measures. We have implemented those using the SPARQLytics DSL and to that end identified 10 graph cubes, 23 dimensions, and 9 measures. There are at most 8 (on average 3.5) dimensions per cube with 3 (1.5) levels per dimension and 4 (1.6) measures per cube. Dimension and measure artifacts are re-used in up to 3 (on average 1.6) and 5 (1.8) queries, respectively.

Fig. 1 contrasts the cumulated lines of code for SPARQL and SPARQLytics when executing these 12 queries one after another. The plain SPARQL queries are actually not that large—assuming a line width of 120 characters, they average at about 23 lines. However, if a user were to write SPARQL queries for multidimensional analyses by hand, line breaks and indentation would be used to improve readability. Therefore, Fig. 1 shows the number

[9] See `http://jena.apache.org/`.
[10] See `https://github.com/javaprog/sparqlytics`.
[11] See `https://github.com/ldbc/ldbc_snb_implementations`.

of lines of the generated SPARQL queries after having them formatted by Apache Jena. Comparing that to the few lines of code that are needed for executing OLAP operations, such as changing the cube's granularity by rolling up or drilling down, and then invoking the measure computation, it is easy to see the benefit of the abstractions for multidimensional analytics, regardless of whether in a programming interface or on a language level.

The focus of our approach is the generation of SPARQL queries from user-specified RDF cubes modified by OLAP operations, rather than their execution in a SPARQL engine. Therefore we evaluated the effectiveness and efficiency of the generation. Moreover, to verify their correctness, the generated queries were executed on an LDBC dataset with scale factor 1 (3 million entities, 21 million relations) using the Open Source edition of OpenLink Virtuoso[12] in version 7.2.1. We conducted our experiments running both our prototype and the Virtuoso server on a 64 bit Windows 7 workstation with 12 hardware threads and 48 GB RAM. On the right y-axis of Fig. 1 we contrast the query generation and execution times depicted with bars on a logarithmic scale. We assume that the query execution times can be reduced by one order of magnitude with the help of expert performance tuning of Virtuoso. But even then, they would strongly dominate query generation times. Thus, the overhead SPARQLytics imposes on the execution of analytical queries on RDF data is negligible.

Note that the 12-query workload we consider here is just a set of loosely related queries on the same dataset. A real workload induced by a data analyst performing an interactive data exploration is likely to consist of a higher number of queries, which will be considerably more related with a significantly higher re-use of multidimensional artifacts. Every re-use of an artifact directly translates into higher work productivity of the data enthusiast. Also, a stateful interface for interactive multidimensional graph analytics significantly reduces verbosity (i.e. requires less information transfer), because the user has to specify only those aspects of the session's state that should be changed. Avoiding the unnecessary and error-prone repetition leads to a more fluent interaction with the system.

4 Related Work

Several approaches allow feeding RDF graph data often by means of ETL processes into a data warehouse [IAK13, JFL14, KH11, NL12]. A different strategy pursued by more recent approaches is to annotate the RDF data with additional metadata to mark the multidimensional schema within the RDF data or its schema. This metadata enables interpreting the RDF data in a multidimensional model and generating SPARQL queries accordingly [EV12, KOH12, MCG13, Ib14]. All these approaches require the data to be annotated by the provider, who has to fix its intension (i.e., the schema and thereby its meaning) up-front. This works well for homogeneous datasets, such as census data published by government or publicly-owned agencies. Only few organisations do that today, but they limit the available fact bases, dimensions, and measures to what they see fit. As a result, users are not able to turn a measure into a dimension or analyze facts not designated as such, nor can they easily combine different datasets with overlapping semantics.

[12] See `http://virtuoso.openlinksw.com/`.

Recently Colazzo et al. [Co14] proposed to abstract an RDF graph into an analytical schema, which is itself an RDF graph. Each vertex in that schema graph represents a class of facts and has a unary pattern (i.e. one rooted in a single variable) associated to it, whereas every edge stands for instances of concepts captured by binary patterns. Employing the global-as-view metaphor of information integration, the analytical schema can be understood as a “lens” through which the underlying data can be seen. Every vertex then represents a fact base through the pattern attached to it and the reachable vertices represent the available measures and dimensions. This approach is well-suited for exploratory and impromptu analysis: the analytical schema for an RDF dataset has to be constructed by a user and maintained to accommodate for the dynamic nature of graphs, but can also be tailored to a specific analysis task and only extended on demand. A multidimensional query is formulated against the analytical schema but is then automatically rewritten using the attached patterns and executed on the original RDF data. In contrast to our approach, this intentionally does not make multidimensional concepts explicit: the definition of a cube requires writing two queries starting from the same vertex of the schema graph (representing the fact base) and navigating along edges to other vertices (representing the dimensions and measures, respectively). Also, since facts are encoded as unary patterns, they are limited to vertices instead of arbitrary subgraphs (e.g., paths). Nevertheless, the schema graph could be used as a visual guide in graphical user interfaces for exploratory computing scenarios.

5 Conclusion

Using open data sources from the internet for analytical applications is gaining importance for discovering trends and business opportunities outside of the corporate realm—a task typically performed by data enthusiasts. Open RDF data is particularly useful as it offers a plethora of interconnected datasets on nearly every aspect of human life. The traditional approaches for multidimensional analytics on graph-structured data fix the available analytical perspectives up-front and therefore are unsuited for iterative data exploration.

With SPARQLytics we have proposed a tool that exposes the concepts of multidimensional graph analytics for their utilization on RDF data. By modularizing the ad-hoc definition of multidimensional cubes, SPARQLytics makes it simple to define cubes and re-using multidimensional artifacts in modified cube definitions. With standard OLAP cube operators, SPARQLytics allows data analysts to work with the RDF cubes—manipulate them and computing measures on them—with minimal need for SPARQL code writing. For measure computation, SPARQLytics generates the corresponding SPARQL query and posts it to an endpoint, hiding most of the technical aspects of the endpoint communication from the user. SPARQLytics provides a concise DSL that fits very well with interactive computing tools, such as Jupyter, which allows data enthusiasts to leverage SPARQLytics in a familiar environment. The DSL builds on SPARQL triple pattern making it very easy to learn for anyone familiar with RDF, SPARQL, and the concept of multidimensional analytics. Our evaluation shows that SPARQLytics saves data analysts many lines of SPARQL code in iterative data explorations. At the same time, SPARQLytics’s query generation in less than 5 ms imposes no overhead to query execution and is practically not recognizable by humans.

References

[Ab13] Abelló, Alberto; Darmont, Jérôme; Etcheverry, Lorena; Golfarelli, Matteo; Mazón, Jose-Norberto; Naumann, Felix; Pedersen, Torben; Rizzi, Stefano Bach; Trujillo, Juan; Vassiliadis, Panos; Vossen, Gottfried: Fusion Cubes: Towards Self-Service Business Intelligence. Int. J. Data Warehous. Min., 9(2):66–88, 2013.

[Ab15] Abelló, Alberto; Romero, Oscar; Bach Pedersen, Torben; Berlanga, Rafael; Nebot, Victoria; Aramburu, María José; Simitsis, Alkis: Using Semantic Web Technologies for Exploratory OLAP: A Survey. IEEE Trans. Knowl. Data Eng., 27(2):571–588, 2015.

[Bl14] Blas, Nicoletta Di; Mazuran, Mirjana; Paolini, Paolo; Quintarelli, Elisa; Tanca, Letizia: Exploratory computing: a draft Manifesto. In: Proc. DSAA. IEEE, pp. 577–580, 2014.

[Bo13] Boncz, Peter: LDBC: Benchmarks for Graph and RDF Data Management. In: Proc. IDEAS. ACM, pp. 1–2, 2013.

[Co14] Colazzo, Dario; Goasdoué, François; Manolescu, Ioana; Roatiş, Alexandra: RDF Analytics: Lenses over Semantic Graphs. In: Proc. WWW. ACM, pp. 467–478, 2014.

[EV12] Etcheverry, Lorena; Vaisman, Alejandro A.: QB4OLAP: A Vocabulary for OLAP Cubes on the Semantic Web. In: Proc. COLD. volume 905 of CEUR Workshop Proceedings. CEUR-WS.org, 2012.

[IAK13] Inoue, Hiroyuki; Amagasa, Toshiyuki; Kitagawa, Hiroyuki: An ETL Framework for Online Analytical Processing of Linked Open Data. In: Proc. Conf. Web-Age Information Management. Springer, pp. 111–117, 2013.

[Ib14] Ibragimov, Dilshod; Hose, Katja; Pedersen, Torben Bach; Zimanyi, Esteban: Towards Exploratory OLAP over Linked Open Data – A Case Study. In: Business Intelligence for the Real-Time Enterprise. volume 206 of LNBIP. Springer, pp. 1–16, 2014.

[JFL14] Jakawat, Wararat; Favre, Cécile; Loudcher, Sabine: OLAP on Information Networks: A New Framework for Dealing with Bibliographic Data. In: Proc. ADBIS, volume 241 of Advances in Intelligent Systems and Computing, pp. 361–370. Springer, 2014.

[KH11] Kämpgen, Benedikt; Harth, Andreas: Transforming Statistical Linked Data for Use in OLAP Systems. In: Proc. International Conference on Semantic Systems. ACM, pp. 33–40, 2011.

[KOH12] Kämpgen, Benedikt; O'Riain, Sean; Harth, Andreas: Interacting with Statistical Linked Data via OLAP Operations. In: Interacting with Linked Data. volume 913 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 36–49, 2012.

[MCG13] Matei, Adriana; Chao, Kuo-Ming; Godwin, Nick: OLAP for Multidimensional Semantic Web Databases. In: Business Intelligence for the Real-Time Enterprise. volume 206 of LNBIP. Springer, pp. 81–96, 2013.

[Mo14] Morton, Kristi; Balazinska, Magdalena; Grossman, Dan; Mackinlay, Jock: Support the Data Enthusiast: Challenges for Next-generation Data-analysis Systems. PVLDB, 7(6):453–456, 2014.

[NL12] Nebot, Victoria; Llavori, Rafael Berlanga: Building Data Warehouses with Semantic Web Data. Decis. Support Syst., 52(4):853–868, 2012.

[Ru14] Rudolf, Michael; Voigt, Hannes; Bornhövd, Christof; Lehner, Wolfgang: SynopSys: Foundations for Multidimensional Graph Analytics. In: Business Intelligence for the Real-Time Enterprise. volume 206 of LNBIP. Springer, pp. 159–166, 2014.