International Journal of Information Management 54 (2020) 102089

Contents lists available at ScienceDirect

International Journal of Information Management

journal homepage: www.elsevier.com/locate/ijinfomgt

Research Note

Efficient querying of multidimensional RDF data with aggregates: Comparing NoSQL, RDF and relational data stores

Franck Ravat[a,*], Jiefu Song[a,c], Olivier Teste[b], Cassia Trojahn[b]

[a] IRIT (CNRS/UMR5505) – Université Toulouse I Capitole, Toulouse France
[b] IRIT (CNRS/UMR5505) – Université Toulouse II Jean-Jaures, Toulouse France
[c] Activus Group, Toulouse France

ARTICLE INFO

Keywords:
Statistical RDF data
Graph aggregation
NoSQL
Data analytics

ABSTRACT

This paper proposes an approach to tackle the problem of querying large volume of statistical RDF data. Our approach relies on pre-aggregation strategies to better manage the analysis of this kind of data. Specifically, we define a conceptual model to represent original RDF data with aggregates in a multidimensional structure. A set of translations rules for converting a well-known multidimensional RDF modelling vocabulary into the proposed conceptual model is then proposed. We implement the conceptual model in six different data stores: two RDF triple stores (Jena TDB and Virtuoso), one graph-oriented NoSQL database (Neo4j), one column-oriented data store (Cassandra), and two relational databases (MySQL and PostGreSQL). We compare the querying performance, with and without aggregates, in these data stores. Experimental results, on real-world datasets containing 81.92 million triplets, show that pre-aggregation allows for reducing query runtime in all data stores. Neo4j NoSQL and relational databases with aggregates outperform triple stores speeding up to 99% query runtime.

1. Introduction

Classically, data analytics focuses on business data managed by a decision support system, with data being mostly stored in relational databases or structured files. In the era of Big Data (de Vasconcelos & Rocha, 2019; Gandomi & Haider, 2015), business analytics must evolve constantly. In particular, with the increasing amount of RDF data being made available on the Linked Open Data (LOD) cloud, the need of considering this kind of data source is more than ever evident. This, however, involves not only dealing with the heterogeneity of representations and granularities, but also dealing with large volume of data.

While data analytics process largely benefits from multidimensional models (supporting, for example, online analytical processing (OLAP) analysis (Ravat, Teste, Tournier, & Zurfluh, 2008) and reporting (Schulz, Winter, & Choi, 2015)), these models have been largely studied in the database community and maturity on optimising multidimensional querying has been mostly reached. In the Semantic Web, exploiting such multidimensional views on RDF data has received attention over the last ten years and a growing number of RDF data sources relying on such view has being published on the Linked Open Data.[1] These data cube models[2] answer the need of analysing statistical RDF data from different perspectives and levels of granularity (Etcheverry, Vaisman, & Zimányi, 2014).

However, querying multidimensional data generates high workloads on RDF triple stores. Despite the different efforts on optimising native RDF triple stores (Ingalalli, Ienco, Poncelet, & Villata, 2016; Neumann & Weikum, 2010; Pérez, Arenas, & Gutierrez, 2009; Tsialiamanis, Sidirourgos, Fundulaki, Christophides, & Boncz, 2012), the scalability of SPARQL queries (the standard language for querying RDF graphs) is still limited owing further development and optimisation (Wang, Staab, & Tiropanis, 2016). Different proposals have addressed querying traditional relational data on the top of SPARQL (the well-known Ontology-based Data Access (Calvanese et al., 2017)), where the problem is shifted to the translation of SPARQL to SQL rather than to the performance of the data store itself. In parallel, a number of new data management systems, broadly known as NoSQL databases, have being proposed, offering a performing alternative to SPARQL engines. Such systems have emerged as an infrastructure for handling large amounts of data outside the RDF space (Cudré-Mauroux et al., 2013).

Different works have addressed the follow-up question on how the

* Corresponding author.
E-mail address: franck.ravat@irit.fr (F. Ravat).
[1] https://www.w3.org/2011/gld/wiki/Data_Cube_Implementations.
[2] https://www.w3.org/TR/vocab-data-cube/.

https://doi.org/10.1016/j.ijinfomgt.2020.102089
Received 29 April 2019; Received in revised form 20 January 2020; Accepted 23 January 2020
Available online 17 June 2020
0268-4012/

performance of NoSQL engines perform compared to RDF engines (Bouhali & Laurent, 2015; Cudré-Mauroux et al., 2013; Hernández, Hogan, Riveros, Rojas, & Zerega, 2016). While most of them scale more gracefully than the RDF stores, the findings in these works also point out weaknesses of NoSQL engines in dealing with complex SPARQL queries involving several joins or containing complex filters on large volume of data (Cudré-Mauroux et al., 2013). As stated in Hernández et al. (2016), NoSQL engines like Neo4j could better isolate queries such that one poorly performing query does not cause a domino effect and benefit from better query algorithms.

With a focus on a particular kind of data (statistical RDF data) which requires specific treatments for facilitating the data analytics processes, one raised question is how triple stores performance would compare with that of relational and NoSQL data stores, when exploiting aggregates (i.e. summarised numeric values with grouping conditions and aggregation functions). For instance, suppose that we want to calculate the SO2 rate by country for a given year. If the data are modelled according to cities without aggregates, this analysis requires a query parsing all triples related to the SO2 rate by city before computing on-the-fly an average for each country. Contrary, including both detailed data and pre-computed aggregates, the analysis consists in a simple projection of SO2 rate for each country. Computing aggregates implies an overload in the data store by the addition of new data into the data store. Hence, the query performance intrinsically depends on the data store. Our hypothesis here are a) aggregating data and b) taking advantage of relational optimisations and NoSQL performances would better scale than triple stores when dealing with large amounts of data. Differently from graph summarisation (Chen, Yan, Zhu, Han, & Yu, 2008; Ghrab, Romero, Skhiri, Vaisman, & Zimányi, 2015; Liu, Dighe, Safavi, & Koutra, 2016; Zhao, Li, Xin, & Han, 2011), which mostly focus on structurally grouping vertices or edges, our aggregation strategy is based on creating new vertices whose content results from calculating numeric values from a set of vertices through aggregation functions. Moreover, while works have studied the problem of aggregating RDF data in RDF triple stores like (Kämpgen & Harth, 2013), to the best of our knowledge, this problem has not been addressed using property-graph and column NoSQL systems. This has been addressed in other NoSQL engines such as MongoDB document-oriented (Botoeva, Calvanese, Cogrel, Rezk, & Xiao, 2016).

The contributions of this paper can be summarised as follows:

- we propose a conceptual model to represent statistical RDF data with aggregates according to a multidimensional structure. This model is generic enough to serve as a basis for representing graph-like multidimensional data with aggregates;
- we present a designing process that takes as input well-known multidimensional RDF data and produces a conceptual graph enriched with pre-computed aggregates;
- we propose a set of rules for converting the logical model into different data representations (RDF, NoSQL graph-property and column, and relational representations);
- we propose a benchmark composed of 24 queries for querying aggregated and non-aggregated data together with the translation of them in different languages (SQL, SPARQL, Cypher and CQL)[3] ;
- we compare the performance of querying data with and without aggregates on a real-world RDF datasets containing 81.92 million triples.

To the best of our knowledge, this is the first systematic study of aggregated graph data including conceptual modelling, translation rules and experimental evaluation on RDF triple stores, property-graph, column NoSQL and relational engines.

The remainder of the paper is organised as follows. Section 2 summarises the related work. Section 3 introduces our conceptual multidimensional model. Section 4 describes the designing process of Multidimensional Graph from statistical RDF data. Section 5 presents different logical and physical implementations of a conceptual Multidimensional Graph. Section 6 describes the querying of Multidimensional Graph with and without pre-computed aggregates. Section 7 presents the experimental results. Finally, Section 9 concludes the paper and discusses future work.

2. Related work

This section presents the main proposals on the different aspects related to our work (i) managing RDF data on relational and NoSQL data stores, (ii) optimising RDF storage and querying; (iii) graph summarisation and RDF aggregates; and (iv) multidimensional analysis over RDF data.

Relational and NoSQL databases for managing RDF data A large body of literature has focused on Ontology-based Data Access (OBDA) as a way of querying relational databases via ontologies and SPARQL queries. The reader can refer to Xiao, Calvanese, et al. (2018) for a survey on these approaches. This involves dealing with the different aspects of translating SPARQL queries into SQL queries, as stated in Xiao, Kontchakov, Cogrel, Calvanese, and Botoeva (2018). In these works, however, the performance of triple-stores is not at all addressed. In the perspective of comparing the performance of different RDF data management solutions, Schmidt, Hornung, Küchlin, Lausen, and Pinkel (2008) have shown that a triple store built on top of a column-store DBMS is competitive, with no approach scaling up to millions of RDF triples, and no approach competing with a purely relational model. In parallel, various works have also investigated the different aspects of using NoSQL databases to manage RDF data. In Cudré-Mauroux et al. (2013), the performance of NoSQL bases (HBase, Couchbase and Cassandra) is evaluated. While most NoSQL systems scale more gracefully than RDF stores, complex SPARQL queries involving several joins, on large volumes of data, or containing complex filters perform poorly on NoSQL systems. In Hernández et al. (2016), the authors compare the performance of triple stores, relational and graph databases for querying Wikidata. Two SPARQL triple stores (Virtuoso and Blazegraph), one relational database (PostgreSQL), and one graph database (Neo4J) have been evaluated. They show that NoSQL could better isolate queries such that one poorly performing query does not cause a domino effect and benefit from better query algorithms. In Michel (2017), using NoSQL MongoDB for RDF data integration is addressed. A set of transformation rules is applied to translate MongoDB documents to SPARQL. A SPARQL query is transformed into a pivot abstract query based on the xR2RML mapping of the target database to RDF. In Thakkar, Punjani, Lehmann, and Auer (2018), the Gremlinator system translates SPARQL queries to path traversals for executing graph pattern matching over graph databases. It provides the foundation for a hybrid use of RDF triple stores and property graph such as Neo4J, Sparksee and OrientDB. Combining different data stores has been further developed in Sun et al. (2015). The proposal combines relational storage (for adjacency information), with JSON storage (for vertex and edge attributes), for storing property graphs. The query translation mechanism translates a subset of Gremlin queries into SQL queries. The proposed system outperformed Titan and Neo4j property graph systems, on query performance.

RDF storage and SPARQL query optimisation While performance has been addressed in the works described above, mostly on the perspective of the data and data storage, complementary work has addressed the optimisation of the querying mechanisms themselves. In that perspective, *Characteristic Sets* (CS) have been introduced as a way to provide estimations for join cardinalities for a better optimisation of queries. In Neumann and Moerkotte (2011), optimisation of SPARQL queries with multiple joins relies on the notion of CS as a way for helping estimating the cardinality of general queries. In Meimaris,

[3] https://github.com/iblid/queries.

Papastefanatos, Mamoulis, and Anagnostopoulos (2017), an indexing scheme for RDF data that explores the inherent structure of triples and that relies on extended characteristic set (ECS) has been proposed. ECS are used to classify triples based on the properties of their subjects and objects, with the objective of improving query processing for conjunctive queries. In Meimaris and Papastefanatos (2016a), the notion of CS has been extended to address the chain-star joins by reducing pairs of chain-star patterns that typically involve multiple self-joins. In Yuan et al. (2013) a system optimising both storing and accessing RDF data has been proposed. In terms of storage, it reduces both the size of stored RDF data and the size of its indexes, using a bit matrix storage structure and an encoding-based compression method for storing huge RDF graphs. In terms of querying, a query plan generation algorithm aims at generating an optimal execution plan for a join query, reducing the size of intermediate results. In Bornea et al. (2013), the RDF storage and query evaluation mechanisms rely on relational representations. For storing, the approach takes advantage of indexing, compressing and scalability of native relational stores and an entity-oriented mechanism for converting RDF data to relational data. For querying optimisation, the approach is based on a specialised structure, called a data flow, that captures the query inter-relationships given the sharing of common variables or constants of different query components. The data flow and cost estimations are then used to establish the order with which to optimise the query components. In Peng, Zou, Chen, and Zhao (2019), the optimisation approach relies on distributed systems. The workload of fragmentation and allocation of distributing RDF datasets is addressed, with a focus on the reduction of the communication cost during SPARQL query processing. Based on frequent access patterns expressing the characteristics of the workload, vertical, horizontal, and mixed fragmentation strategies have been proposed. Experiments have shown the performance improvement on the results of fragmentation and allocation over large RDF datasets. Finally, in Husain, McGlothlin, Masud, Khan, and Thuraisingham (2011), the strategy for managing RDF large datasets is based on distributed file systems and injecting reasoning within RDF query processing. They propose an algorithm to generate query plans, and use Hadoop's MapReduce framework to answer the queries. In fact, reasoning can be exploited for pre-compute and materialise implicit triples or to compute the implicit triples on the fly (Kaoudi & Manolescu, 2013). The proposed approach has shown good results when compared to Jena, BigOWLIL and RDF3X systems.

Graph summarisation and aggregates in RDF Graph summarisation has been extensively studied in the literature (Liu et al., 2016). Two main categories of approaches can be distinguished: (1) *aggregation approaches* (Tian, Hankins, & Patel, 2008; Wu, Zhong, Xiong, & Jing, 2014), which rely on strategies for grouping the graph nodes into groups based on diverse functions (e.g. similarity of values of attributes, relationships to adjacent nodes, application of aggregation functions, etc.); and (2) *structural approaches* (Campinas, Perry, Ceccarelli, Delbru, & Tummarello, 2012; Cebiric, Goasdoue, & Manolescu, 2015; Zneika, Lucchese, Vodislav, & Kotzinos, 2016) which rely on extracting a schema representing a summary of the graph, in general, based on equivalence relations of nodes. The approach we propose here falls into the former. This is the same for the work in Tian et al. (2008), which produces a summary graph of K-groups by grouping nodes based on user-selected node attributes and relationships. This approach, however, is limited to graphs describing entities characterised by the same set of attributes. Our approach is not limited to this kind of graphs, given that RDF graphs are heterogeneous by nature. In the second category, (Zneika et al., 2016) summarise graphs following a top-K approximate RDF graph pattern strategy, aiming at guiding the user in the formulation of his queries. A similar approach is adopted in Campinas et al. (2012), where summarising relies on a generic graph model defining the notion of node collections, i.e. set of nodes sharing similar characteristics. In Khatchadourian and Consens (2010), graphs from the LOD cloud are summarised, focusing on the distribution of classes and properties across LOD sources. The summaries are based on a mechanism that combines text labels and bisimulation contractions. The labels assigned to RDF graphs are hierarchical, enabling summarisation at different granularities. In Diao, Manolescu, and Shang (2017) 'interesting insights' in (generic) RDF graph are automatically identified by the systems as RDF aggregate queries. The system ranks such insights and plots the most interesting ones as bar charts, and shows them to the user. Finally, hybrid approaches are proposed in Wu et al. (2014), Zhao et al. (2011), which take into account both attribute aggregation and structure summarisation of graphs. These works rather focus on topological summarisation of graph aiming at helping users to extract and understand main characteristics of a graph. Other works on RDF summarisation include LODeX (Benedetti, Po, & Bergamaschi, 2014), ABSTAT (Spahiu, Porrini, Palmonari, Rula, & Maurino, 2016) and SchemEX (Konrath, Gottron, Staab, & Scherp, 2012). LODeX is a tool that produces a representative summary of a LOD source. Similarly to Khatchadourian and Consens (2010), the summary reports statistical and structural information regarding the LOD dataset (number of instances of classes and attributes). Contrary to them, we apply aggregation on the values of properties. Under a different view, SchemEX extracts a concise LOD schema with a structure to be used as an index, where schema extraction means to abstract RDF instances to RDF schema concepts that represent instances with the same properties. Finally, ABSTAT takes the RDF data summarisation problem as to provide a compact but complete representation of a dataset, where every relation between concepts that is not in the summary can be inferred. It adopts a minimisation mechanism based on minimal type patterns. In Joshi, Hitzler, and Dong (2015), the focus is the removal of semantic and contextual redundancies in linked data, with two techniques that compress RDF datasets begin introduced. The first, *Logical Linked Data Compression* compresses a dataset by generating a set of new logical rules from the dataset and removing triples that can be inferred from these rules. The second one, *Contextual Linked Data Compression* compresses datasets by performing schema alignment and instance matching followed by pruning of alignments based on confidence value and subsequent grouping of equivalent terms. Depending on the structure of the dataset, the first technique was able to prune more than 50% of the triples.

On the perspective of data materialisation, Goasdoué, Karanasos, Leblay, and Manolescu (2011) addresses the problem of selecting a set of RDF views to be materialised in the database, minimising both query processing, such that *workload queries can be answered based solely on the recommended views, with no need to access the database.* The view selection is modelled as search space in a space of states, where each state models a candidate view set together with the rewriting of the workload queries based on these views. They take into account as well the implicit tuples, by saturating the database with them. Finally, Meimaris, Papastefanatos, Vassiliadis, and Anagnostopoulos (2018) addresses the problem of identifying and analysing instance relationships in multidimensional data (i.e. RDF Data Cube sources) and injecting them into the query mechanism. They compare their strategies to traditional query-based and inference-based solutions, with their proposal providing better scalability.

Multidimensional analysis over RDF data Proposals on this topic mainly address different aspects of analytical queries over RDF data. In Wang et al. (2016), the proposal maps typical OLAP operations to SPARQL and a tool named ASPG automatically generates OLAP queries from real-world Linked Data. The works in Gür, Nielsen, Hose, and Pedersen (2017), Ravat, Song, and Teste 2016 focus on generating SPARQL queries based on OLAP analysis, requiring datasets being described in QB or QB4OLAP vocabularies. In Etcheverry and Vaisman (2017), a high-level query language (COL) that operates over cubes is proposed. Using the metadata provided by QB4OLAP, COL queries are translated into SPARQL, exploiting SPARQL query optimisation techniques. To address also performance, the approach in Ravindra, Kim, and Anyanwu (2016) is based on a refactoring of analytical queries expressed in the relational-like SPARQL algebra based on a set of logical

operators. This refactoring enables parallel evaluation of groupings and aggregations, particularly beneficial for scale-out processing on distributed cloud systems. Contrary to these works, we address the performance of analytical queries exploiting materialisation of aggregates. In Jindal, Madden, Castellanos, and Hsu (2015), graph analytics (graph queries involving full scans, joins, and aggregates) are expressed in relational databases. Translation logical query plans of graph queries into relational operators, query optimisation techniques to tune the performance of graph queries, including considering updating vs replacing the nodes table on each iteration, incremental evaluation of queries, and eliminating redundant joins. In Ordonez, Cabrera, and Gurram (2017), the focus is on the impact of query optimisation of recursive queries on large graphs with different shapes and densities, comparing columnar, row and array DBMSs. The performance on three fundamental relational operators has been analysed (join, projection and, selection). While a columnar DBMS with tuned query optimisation outperforms row and array systems, regardless of their shape, density and connectivity, there is no clear best system between the row and array DBMSs.

DiscussionContrary to the works on OBDA, the study here concentrates on how other kinds of data stores scale when dealing with aggregates in RDF data. Moreover, none of the above-mentioned approach has systematically studied the efficiency of queries computing observation values according to different attributes at different granularity levels organised in different dimensions. While the performance of RDF triple stores have been largely studied, as discussed below, here we state that one can take advantage of the performance of relational (as for OBDA) and NoSQL engines for better dealing with larger RDF datasets. As Cudré-Mauroux et al. (2013) and Hernández et al. (2016), we address the evaluation of NoSQL systems as Neo4j (differently from Michel, 2017) for storing RDF data, but here we focus on a specific kind of data (multidimensional statistical data). While they have pointed out some weakness of these systems, we argue here that materialising aggregates can provide better querying performance. As Thakkar et al. (2018), we consider both storage supports (triple stores and NoSQL databases) but do not combine them together. This is what has been done in Sun et al. (2015). Differently from the works on query optimisation and storage, we do not deal with the different optimising mechanisms. These are complementary works to our study that has to be taken into account in the future. With respect to graph summarisation, unlike most proposals, our approach differs from those by nature, since it works on aggregating numeric values within vertices and not on reducing the graph. Differently from works exploiting reasoning for query optimisation or graph summarisation, we do not exploit yet this mechanism. As Goasdoué et al. (2011), where RDF views are materialised in the database, minimising query processing our idea is to materialise aggregations and query the dedicated graph, reducing the number of comparisons to be done in the query time. Here, we analyse the performance of this approach in several data stores. Close to ours, in particular in multidimensional analysis, Kämpgen and Harth (2013) compares the performance of SPARQL and of ROLAP SQL queries and measures the gain of RDF aggregate views that materialise parts of the RDF data cube. While RDF aggregate views show the capability to optimise query execution, yet, overall still take six times longer for preprocessing and not nearly reach the performance gain of aggregate tables in ROLAP. Here, we could observe that globally the aggregates can have a positive impact in the querying performance. It is important to note, however, that our experiments compared to Kämpgen and Harth (2013) have not been executed on the same basis. On one hand, we propose a new conceptual modelling solution to include pre-computed aggregates in a multidimensional graph. On the other hand, we base our experiments on a new technical environment including triple stores, NoSQL databases and relational databases.

3. Conceptual modelling

Our objective is to propose a conceptual multidimensional model for modelling statistical RDF data. The proposed model is characterised as follows: (i) it should be independent of a specific storage structure (e.g. triple store, DBMS, etc.), (ii) it should represent data in the graph form and (iii) it should include different aggregation levels useful for multidimensional analyses. To do so, we introduce a conceptual multidimensional model based on the concepts of graph structure and aggregates. In this section, we present the model named *Multidimensional Graph* and illustrate it through an example.

A multidimensional graph enriches classical conceptual graph modelling with pre-computed aggregates. It contains individuals organised according to three multidimensional concepts, namely numeric indicators (*measures*), descriptive properties (*attributes*) and summarised numeric indicators with grouping conditions and an aggregate function (*aggregates*).

Definition 1. A ***multidimensional graph*** is a bipartite graph composed of measures, attributes and aggregates. It is defined as (V, E) where

- $\mathtt{V}$ is a set of vertices such that $\mathtt{V} = \mathtt{M} \cup \mathtt{A} \cup \mathtt{G}$, where $\mathtt{M}$ is a set of vertices corresponding to measures, $\mathtt{A}$ is a set of vertices corresponding to attributes (parameters) and $\mathtt{G}$ is a set of vertices corresponding to aggregates. $\mathrm{M} = \{e_i^{\mathrm{M}}\}_{1 \le i \le |\mathrm{M}|}$ is the set of measure values, $\mathrm{A} = \{e_j^{\mathrm{A}}\}_{1 \le j \le |\mathrm{A}|}$ is the set of attribute values, $\mathrm{G} = \{e_k^{\mathrm{G}}\}_{1 \le k \le |\mathrm{A}|}$ is the set of aggregates.
- $\mathtt{E} \subseteq \mathtt{V} \times \mathtt{V}$ is a set of edges such that $\mathtt{E} = \mathtt{E}_{MA} \cup \mathtt{E}_{AA} \cup \mathtt{E}_{MG} \cup \mathtt{E}_{AG} \cup \mathtt{E}_{GG}$. $\mathtt{E}_{MA}$ is an edge between a measure value e_i^{M} and an attribute value e_j^{A}, $\mathtt{E}_{AA}$ is an edge between two attribute values e_i^{A} and e_j^{A}, $\mathtt{E}_{MG}$ is an edge between a measure value e_i^{M} and an aggregate e_i^{G}, $\mathtt{E}_{AG}$ is an edge between an attribute value e_j^{A} and an aggregate e_i^{G}, $\mathtt{E}_{GG}$ is an edge between two aggregates e_i^{G} and e_j^{G}.

The relationships between different types of vertices and edges are presented in Fig. 1.

Example. Fig. 2 gives an example of a multidimensional graph. It is composed of one measure denoted *cost*, and one *geographical* analysis axis. The measure contains 3 values (i.e. e_1^{M}, e_2^{M}, e_3^{M}), while the analysis axis is composed of 4 attributes (*city*, *region*, *country* and *geography*) with 7 instances (i.e. e_1^{A}, e_2^{A}, e_3^{A}, e_4^{A}, e_5^{A}, e_6^{A}, e_7^{A}). The last attribute *geography* represents the maximal (the most general) granularity, called *ALL*. The aggregates of the measure according to different attributes are stored into one measure denoted *cost_sum*. The formal representation of this multidimensional graph is as follows:

- $\mathtt{V} = \mathtt{M} \cup \mathtt{A} \cup \mathtt{G}$ where $\mathrm{M} = \{e_1^{\mathrm{M}}, e_2^{\mathrm{M}}, e_3^{\mathrm{M}}\}$, $\mathrm{A} = \{e_1^{\mathrm{A}}, e_2^{\mathrm{A}}, e_3^{\mathrm{A}}, e_4^{\mathrm{A}}, e_5^{\mathrm{A}}, e_6^{\mathrm{A}}, ,$ $\mathrm{G} = \{e_1^{\mathrm{G}}, e_2^{\mathrm{G}}, e_3^{\mathrm{G}}, e_4^{\mathrm{G}}\}$.
- $\mathtt{E} = \mathtt{E}_{MA} \cup \mathtt{E}_{AA} \cup \mathtt{E}_{AG}$ where $\mathrm{E_{MA}} = \{(e_1^{\mathrm{M}}, e_1^{\mathrm{A}}), (e_2^{\mathrm{M}}, e_2^{\mathrm{A}}), (e_3^{\mathrm{M}}, e_3^{\mathrm{A}})\}$, $\mathrm{E_{AA}} = \{(e_1^{\mathrm{A}}, e_4^{\mathrm{A}}), (e_2^{\mathrm{A}}, e_4^{\mathrm{A}}), (e_3^{\mathrm{A}}, e_5^{\mathrm{A}}), (e_4^{\mathrm{A}}, e_6^{\mathrm{A}}), (e_5^{\mathrm{A}}, e_6^{\mathrm{A}}), (e_6^{\mathrm{A}}, e_7^{\mathrm{A}})\}$, $\mathrm{E_{MG}} = \{(e_1^{\mathrm{M}}, e_1^{\mathrm{G}}), (e_2^{\mathrm{M}}, e_1^{\mathrm{G}}), (e_3^{\mathrm{M}}, e_2^{\mathrm{G}})\}$, $\mathrm{E_{AG}} = \{(e_4^{\mathrm{A}}, e_1^{\mathrm{G}}), (e_5^{\mathrm{A}}, e_2^{\mathrm{G}}),$

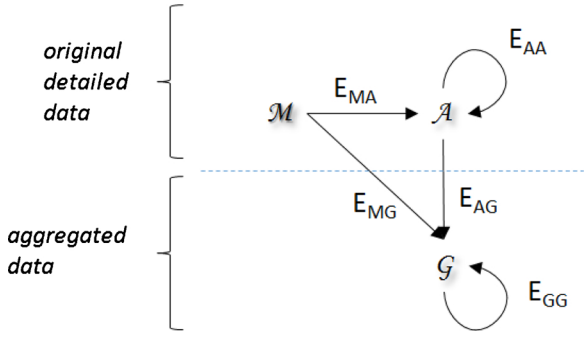


Fig. 1. Relationships between different types of vertices and edges.

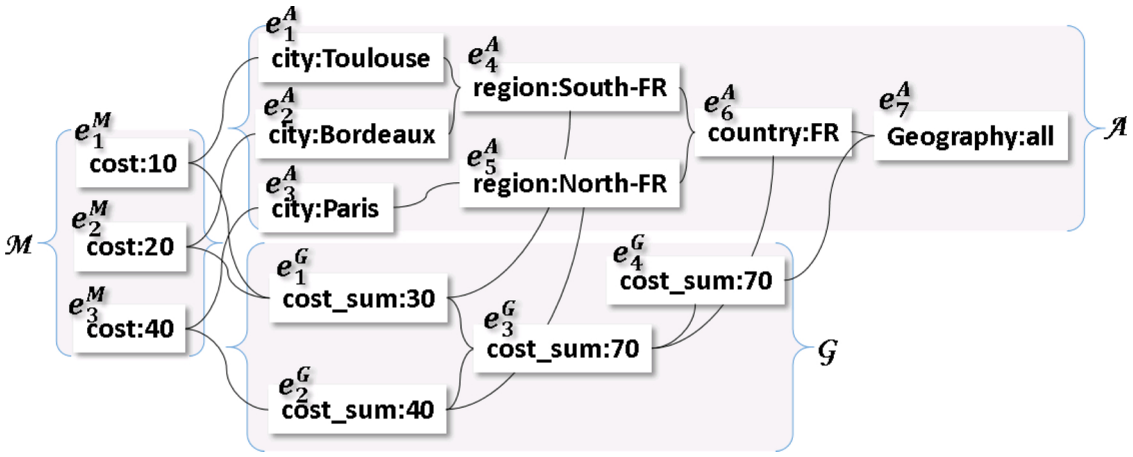


Fig. 2. Example of multidimensional graph.

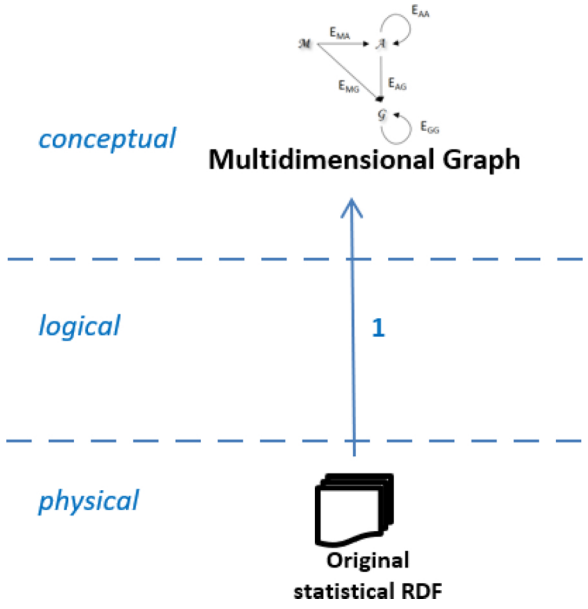


Fig. 3. Designing process of Multidimensional Graph from physical level to conceptual level.

(e_6^A, e_3^G), $(e_7^A, e_4^G)\}$, $E_{GG} = \{(e_1^G, e_3^G), (e_2^G, e_3^G), (e_3^G, e_4^G)\}$.

Let $\{\preccurlyeq_1, \ldots \preccurlyeq_d\}$ a set of binary relations over $\mathbb{A}$. Each $\preccurlyeq_i$ is a binary relation over $D^i_{AA} \subseteq \mathbb{A}$ defining an ordered set. The subsets D^i_{AA} are disjoints; $\forall\, i \in [1..d], j \neq i \in [1..d]$, $D^i_{AA} \cap D^j_{AA} = \emptyset$.

Attributes within a multidimensional graph can be organised according to analysis axes (i.e. dimensions):

Definition 2. A dimension is defined by $(D^i_{AA}, \preccurlyeq_i)$ where

- $D^i_{AA} \subseteq A$ is a set of attributes,
- $\preccurlyeq_i$ is an ordered set over D^i_{AA} satisfying
  - –irreflexivity: $\nexists e^A_j, e^A_j \preccurlyeq_i e^A_j$;
  - –transitivity: if $e^A_{j1} \preccurlyeq_i e^A_{j2}$ and $e^A_{j2} \preccurlyeq_i e^A_{j3}$, then $e^A_{j1} \preccurlyeq_i e^A_{j3}$
  - –asymmetry: if $e^A_{j1} \preccurlyeq_i e^A_{j2}$ then not $e^A_{j2} \preccurlyeq_i e^A_{j1}$.

Attributes associated together through binary relations form one or several aggregation paths (i.e. hierarchies) within a dimension. A measure can be summarised along a hierarchy by zooming in (drilling down) or zooming out (rolling up).

Example. In the previous example, the multidimensional database is composed of one dimension with one hierarchy of four granularities (city, region, country and geography), such as

- $D^{Geography}_{AA} = \{e^A_1, e^A_2, e^A_3, e^A_4, e^A_5, e^A_6, e^A_7\}$,
- $\preccurlyeq_{Geography}$ is an ordered set such as $e^A_1 \preccurlyeq_{Geography} e^A_4$; $e^A_2 \preccurlyeq_{Geography} e^A_4$; $e^A_3 \preccurlyeq_{Geography} e^A_5$; $e^A_4 \preccurlyeq_{Geography} e^A_6$; $e^A_5 \preccurlyeq_{Geography} e^A_6$; $e^A_6 \preccurlyeq_{Geography} e^A_7$.

It is worth noticing that our proposed model is situated at the conceptual level. We introduce a generic modelling solution independent of any implementation environment. Our proposed model can be freely implemented in any data store, for instance in a RDF triple stores through QB and QB4OLAP vocabularies, or in a relational database through relations. We apply the multidimensional modelling principles to our proposed model to make a distinction between raw statistical RDF data and pre-computed aggregates through two types of vertices `M` and `G`. Specifically, the raw data vertices `M` are directly deduced the from data source. Since QB or QB4OLAP which do not require including pre-computed aggregates, not all datasets include all possible aggregates. Unlike these previous solutions, our proposed model systematically browses all the raw data and discovers all possible aggregates. These aggregates are computed before analyses and materialised through aggregate vertices `G` to accelerate multidimensional analyses.

4. Designing a multidimensional graph from statistical RDF data

Based on the conceptual modelling of a multidimensional graph, we propose a designing process to pre-compute and materialise aggregates from statistical RDF data (cf. Fig. 3). Contrary to the work from Bouakkaz, Ouinten, Loudcher, and Strekalova (2017) that proposes textual aggregations for OLAP analysis, here we focus on numeric aggregation. Our design process begins with converting statistical RDF data without aggregates at the physical level into the conceptual Multidimensional Graph (arrow 1). During this step, aggregates are computed and combined with original data in a conceptual Multidimensional Graph.

4.1. Translating algorithm

In order to convert original RDF data into a conceptual Multidimensional Graph and enrich original data with all possible aggregates, we depict an algorithm which (a) takes a statistical RDF dataset in QB or QB4OLAP vocabulary as input and (b) produces a conceptual Multidimensional Graph including original data and aggregates at output (cf. Algorithm 1).

Algorithm 1. Designing a MG based on original statistical RDF data

Algorithm 1: Designing a MG based on original statistical RDF data

input : Original RDF data

output: Conceptual aggregate multidimensional graph

foreach *dimension* `?dim` *in original RDF data, such as* `?dim a qb:DimensionProperty.` **do**

  create a dimension D_i *in the multidimensional graph*;

  if *there exists* `?dim qb:codeList ?lst` **then**

    identify each attribute `?att` *within* `?dim` *, such as*

    - `?lst skos:hasTopConcept ?att.` or
    - `?lst qb:hierarchyRoot ?att.`

  else

    identify each attribute `?att` *within?dim* `(?level a qb4o:LevelProery. ?level qb4o:inDimension ?dim)`*, such as* `?att qb4o:inLevel ?level` ;

  end

  foreach *identified attribute* `?att` *in original RDF data* **do**

    create an attribute vertex a_i *such as* $a_i \in A$, $A \subseteq V$;

  end

  create an edge E_{AA} *between* `?att` *and* `?attUp`*, such as*

  - `?attUp skos:inSchema ?lst; skos:narrower ?att.` or
  - `?lst a qb:HierarchicalCodeList; qb:parentChildProperty ?p2c. ?attUp skos:inSchema ?lst; ?p2c ?att.`

end

foreach *measure* `?m` *in original RDF data, such as* `?m a qb:Observation` **do**

  create an measure vertex m_i *such as* $m_i \in M$, $M \subseteq V$;

  identify attribute `?att` *on dimension* `?dim (?dim a qb:DimensionProperty)` *associated with* `?m`*, such as*

  - `?m ?dim ?att.` or
  - `?level a qb4o:LevelPropery; qb4o:inDimension ?dim. ?att qb4o:inLevel ?level. ?m ?dim ?att.`

  create an edge E_{MA} *between* `?att` *and* `?m` ;

end

foreach *measure* $m_i \in M$ **do**

  calculate aggregated values of m_i *according to the related attributes on each dimension* $2^{A_{D_1} \times \ldots \times A_{D_n}}$*, where* $A_{D_i} \subseteq A$ *is the set of related attribute vertices on* D_i*, such as* $\forall a_j \in A_{D_i}$, a_j *is associated with* m_i *through one* E_{MA} *edge or through a set of* E_{AA} *edges and one* E_{MA} *edge* ;

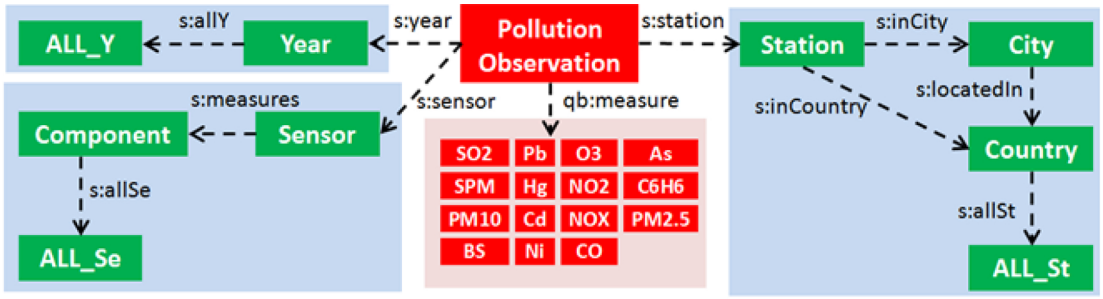
  create an aggregate vertex g_i *such as* $g_i \in G, G \subseteq V$;

  create an edge E_{MG} *between* m_i *and* g_i. ;

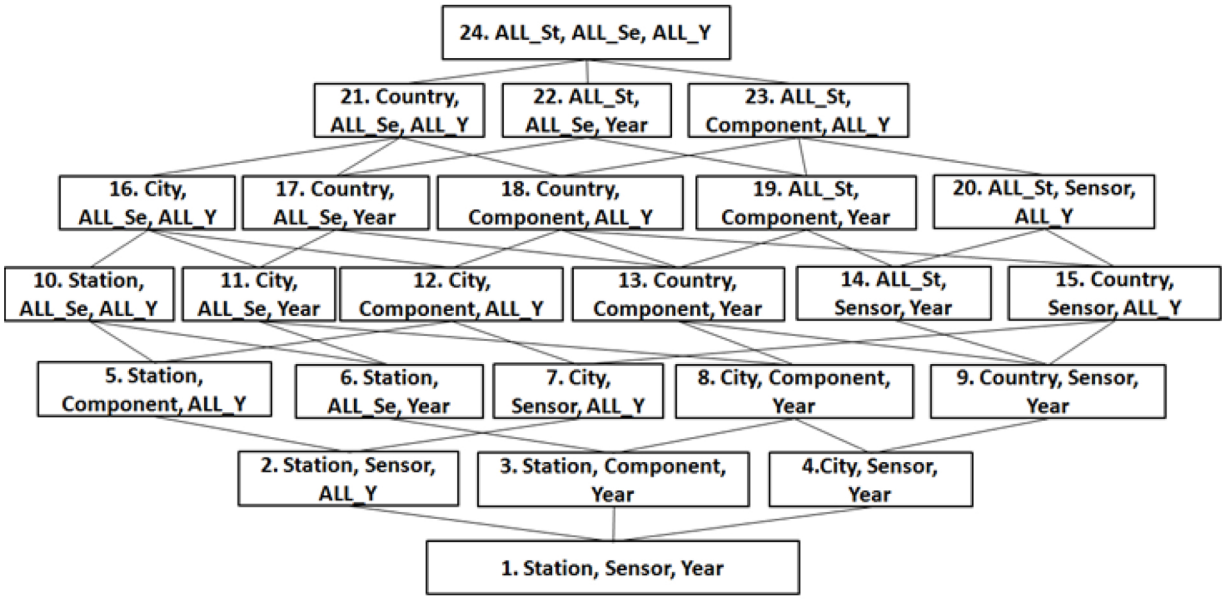
  create an edge E_{AG} *between each attribute* a_{g_i} *in the grouping conditions and* g_i. ;

  create an edge E_{GG} *between related aggregate vertices* g_i *and* g_j*, such as there exists an attribute* a_{g_i} *associated with* g_i *and an attribute* a_{g_j}, $a_{g_i} \preccurlyeq_{D_k} a_{g_j}$;

end



(a) Multidimensional structure of the QBOAirbase dataset



(b) The set of aggregates identified in the QBOAirbase dataset

Fig. 4. QBOAribase dataset.

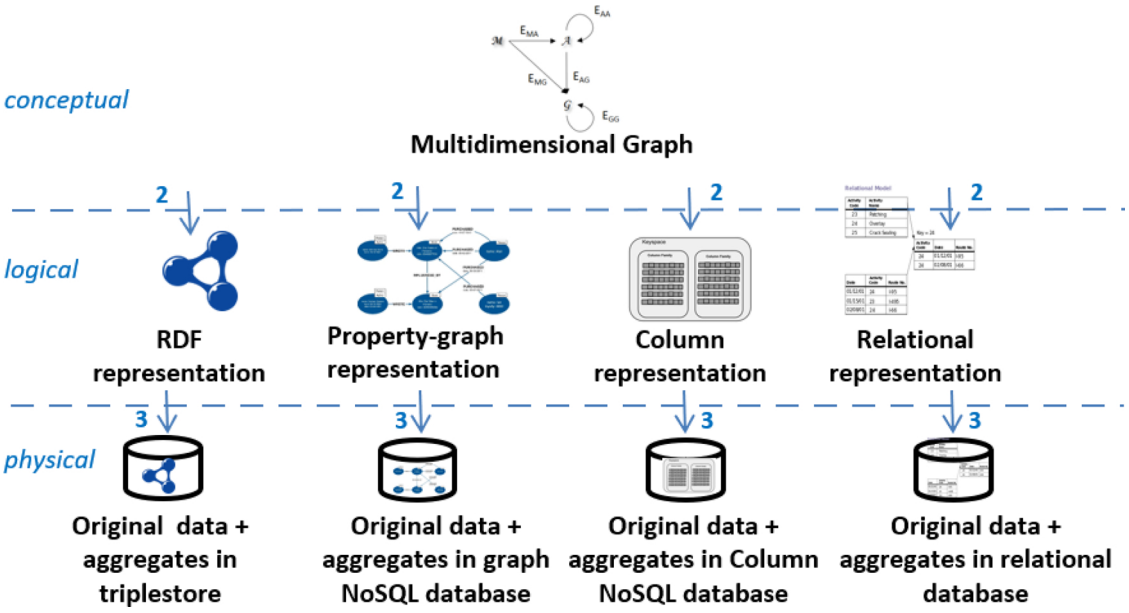


Fig. 5. Logical and physical implementations of a conceptual Multidimensional Graph.

```
aggregate eg:agg1 a qb:Observation;
vertex      MDGraph:value 96.3^^xsd:float;
E_AG edge   MDGraph:dimGeo "Paris"^^xsd:String;
            MDGraph:dimYear "2018"^^xsd:String;
E_GG edge   MDGraph:GG eg:agg2;
E_MG edge   MDGraph:MG eg:m1, eg:m2, eg:m3.
```

Fig. 6. An example of aggregate in RDF model.

Algorithm 1 first identifies attributes on each dimension according to QB (lines 3 and 4) and QB4OLAP (line 6) vocabularies and creates attribute vertices in the Multidimensional Graph accordingly (lines 8–10). Then, it creates edges E_{AA} between attributes linked together through standardised (`skos:narrower`) or customised (`?p2c`[4]) relations (line 11). Next, the algorithm creates measure vertices in the Multidimensional Graph (lines 13–16). At last, it calculates measure values according to attributes of different granularities on each dimension and adds pre-computed aggregates in the Multidimensional Graph (lines 17–22).

4.2. Use case

In order to illustrate the feasibility of our proposed translating process, in this section we describe a use case based on a real-world dataset named QBOAirbase.[5] QBOAirbase corresponds to an RDF dataset including 5.07×10^6 RDF triples describing the European air quality database on air pollution and climate change mitigation.[6] This statistical RDF dataset includes triples describing air pollution *observations* and corresponding descriptive *attributes* organised according to different *analysis axes* and *granularity levels*. The multidimensional structure of the QBAirbase dataset is shown in Fig. 4(a) (without triple individuals).

First, we execute Algorithm 1 to build an intermediate conceptual Multidimensional Graph without aggregate (cf. steps 1-16). The obtained Multidimensional Graph includes a set of *Measures* `M` composed of 1.6×10^6 measure vertices. Each measure vertices corresponds to an observation (`qb:Observation`) in the source. Each measure vertex describes one air pollution indicator such as `SO2`, `Pb`, `O3` (cf. Fig. 4(a)), and it can be analysed according to three *dimensions*, namely *s:year*, *s:station* and *s:sensor*. The set of attributes `A` is composed of 2.43×10^5 attribute vertices. Each attribute vertex represents an instance from the source, and it describes one granularity within a dimension. 4.8×10^6 E_{MA} edges are created to associate an attribute vertex e_j^A with a measure vertex e_i^M. Relationships between attribute vertices are represented through 2.43×10^5 E_{AA} edges.

Then, Algorithm 1 completes the intermediate conceptual Multidimensional Graph with pre-computed aggregates `G` by rolling up on different *dimensions* at different granularities (cf. steps 17–23). To perform the group-by aggregation, attribute vertices are used. All 24 combinations of attribute vertices that can be used in group-by aggregation as well as their relationships can be found in Fig. 4(b).

Note that the first type of aggregate *Station, Sensor, Year* corresponds to the original data, while the 16th types of aggregate compute measure values according to three distinct granularities, namely `City`, `ALL_SE` and `ALL_Y` (1.62×10^6 vertices and 3.23×10^6 edges). Consequently, the obtained Multidimensional Graph contains pre-computed aggregates represented by 3.57×10^{10} aggregate vertices, 1.045×10^{11} E_{AG} edges, 7.63×10^{10} E_{MG} edges and 3.57×10^{10} E_{GG} edges.

[4] `?lst a qb:HierarchicalCodeList; qb:parentChildProperty ?p2c.`

[5] http://qweb.cs.aau.dk/qboairbase/.

[6] https://www.eea.europa.eu/data-and-maps/data/airbase-the-european-air-quality-database-2.

The final conceptual Multidimensional Graph contains different types of vertices. Original raw data are modelled through `M` and `A` vertices, while pre-computed aggregates are represented through `G` vertices. These vertices can be implemented in different technical environments. For instance, in a relational OLAP context, they can be implemented through materialised views; in a multidimensional OLAP system, they can be implemented through cubes; in the RDF context, they can be implemented through new specific triples.

5. Logical and physical implementation

The second aim of our approach is to transform a conceptual Multidimensional Graph including both original data and aggregates into different logical and physical implementations (RDF, property-graph, column and relational models at logical and physical levels (cf. Fig. 5)). In the case of an RDF model, we have defined an underlying RDF vocabulary supporting the translation of our Multidimensional Graph model into the corresponding RDF representation. Note that a pre-computed aggregate can be represented as a `qb:Observation` and its related edges can be represented through predicates (e.g. Fig. 6).

5.1. MG to property-graph representation

In order to transform a conceptual Multidimensional Graph into a property-graph representation, we define the following set of mapping rules.

1. A dimension vertex *dim* is transformed into

   `(dim) - [:a] > - (:'qb:DimensionProperty')`

2. An attribute vertex *att* is transformed into
   (a) `(lst) - [:a] - > (:'skos:ConceptSchema'), (lst) - [:'skos:hasTopConcept'] - > (att)` or
   (b) `(lst) - [:a] - > (:'qb:HierarchicalCodeList'), (lst) - [:'qb:hierarchyRoot'] - > (att)` or
   (c) `(level) - [:a] - > (:'qb4o:LevelProperty'), (level) - [:'qb4o:inDimension'] - > (dim), (att) - [:'qb4o:inLevel'] - > (level)`
3. An edge E_{AA} is transformed into a standard relationship `[:'skos:narrower']` or a customized relationship `[:'e_aa']` such as:
   `(lst) - [:a] - > (:'qb:HierarchicalCodeList'), (lst) - [:'qb:parentChildProperty'] - > (e_aa)`
4. An edge E_{MA} is transformed into a relationship `[:dim]` whose label corresponds to the identifier of a dimension, such as:
   (a) `(dim) - [:a] - > (:'qb:DimensionProperty'), (m) - [:a] - > (:'qb:Observation'), (m) - [:dim] - > (att)` or
   (b) `(dim) - [:a] - > (:'qb:DimensionProperty'), (level) - [:a] - > (:'qb4o:LevelProery'), (level) - [:'qb4o:inDimension'] - > (dim), (att) - [:'qb4o:inLevel'] - > (level), (m) - [:a] - > (:'qb:Observation'), (m) - [:dim] - > (att)`
5. An aggregate vertex agg_i is transformed into
   (agg_i:'qb:Observation' {'MDGraph:value': *value*, 'MDGraph:dim1Name': 'att1Name', …, 'MDGraph:dimNNname': 'attNNname'}),
   (agg_i) < - ['aggDim1': 'MDGraph:dim1'] - (att1),
   …
   (agg_i) < - ['aggDimN': 'MDGraph:dimN'] - (attN),
   (agg_i) - [:'MDGraph:GG'] - > (agg_j)
   …
   (agg_i) - [:'MDGraph:MG'] - > (m_i: 'qb:Observation')

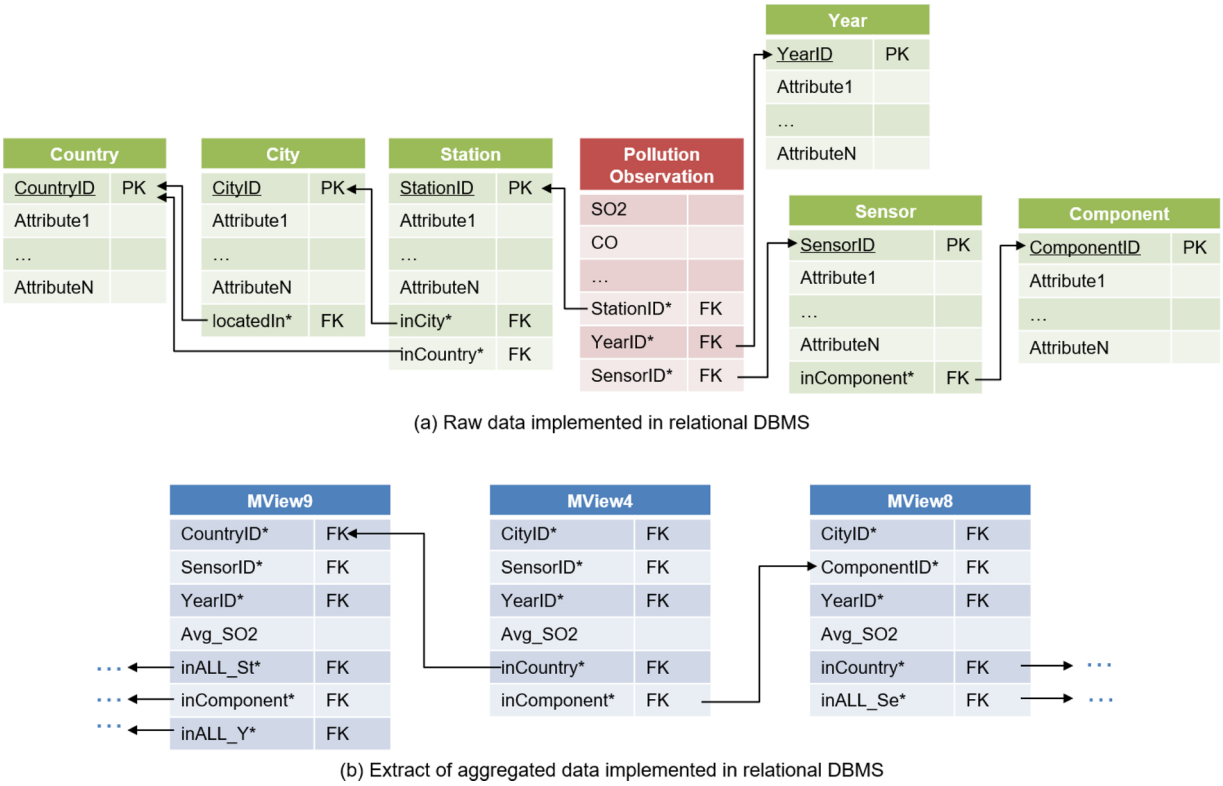


Fig. 7. Relational implementation of the conceptual Multidimensional Graph.

5.2. MG to column-oriented data store

A column-oriented data store manages data by columns rather than by rows like in relational DBMS. More precisely, a column-oriented data store manages each database table column separately, with attribute values belonging to the same column stored contiguously, and densely packed (Abadi, Boncz, & Harizopoulos, 2009). For our experimental assessments, we use the Cassandra data store. This data store is based on the following concepts. A keyspace closely corresponds to a relational database. A column family is a container for an ordered collection of rows, each of which is itself an ordered collection of columns (Hewitt, 2010). A column is the basic component defined by a name, a value, and a timestamp. A partition key correspond to a primary key in a relational database.

Our objective is to translate all the components of a multidimensional graph into a keyspace. In Cassandra, the primary key concept is different from relational databases. A key is used for grouping and organising data into columns and rows. The order in which columns are defined for the primary key matters and it is possible to have a key composed of multiple columns. Note that only the first column of the primary key is considered the partition key and the rest of columns are clustering keys. To apply different grouping queries in CQL, it is necessary to create different tables (or column families) by combining all the possibilities for partition key and clustering keys. Specifically, for a MG including N dimensions, we should create $N!$ tables to enable all grouping conditions during analyses. Each table has a *rowkey* made up of a combination of dimension instances and dimension names, such as such asrowkey(instance$_1^{Dim_1}$, ...,instance$_m^{Dim_n}$, name^{Dim_1}, ...,name^{Dim_n}). For instance, in Fig. 4, we have 3 dimensions (i.e. year, sensor station) and we define 6 tables in order to combine these different possibilities:

```
Table 1: rowkey (StationDimInstance,
SensorDimInstance,
YearDimInstance, station, sensor, year)
Table 2: rowkey (SensorDimInstance, Station-
DimInstance,
YearDimInstance, sensor, station, year)
Table 3: rowkey (SensorDimInstance, YearDimInstance,
StationDimInstance, sensor, year, station)
Table 4: rowkey (StationDimInstance,
YearDimInstance,
SensorDimInstance, station, year, sensor)
Table 5: rowkey (YearDimInstance, Station-
DimInstance,
SensorDimInstance, year, station, sensor)
Table 6: rowkey (YearDimInstance, SensorDimInstance,
StationDimInstance, year, sensor, station)
```

The tables are implemented through CQL queries.

5.3. MG to relational DBMS

Our objective is to implement all the components of the multidimensional graph through relational tables. For the raw data, all the nodes of M (Measures) are implemented through a fact table. The primary key of the fact table is composed of the URI of the subject. All the nodes of A (Attributes) are implemented through different tables and each table represents an aggregation level (the primary key is the URI of the subject and the foreign key is the URI of the predicate related to the following level). In Fig. 7, we can identify one fact table related to the observations and 6 tables for the dimensions related to the attributes.

For the aggregated data, all the nodes G (Aggregates) are

```
SELECT CITYID, COMPONENTID, YEARID, AVG(SO2)
FROM POLLUTION_OBSERVATION, STATION, CITY, YEAR, SENSOR, COMPONENT
WHERE POLLUTION_OBSERVATION.STATIONID = STATION.STATIONID
AND POLLUTION_OBSERVATION.YEARID = YEAR.YEARID
AND POLLUTION_OBSERVATION.SENSORID = SENSOR.SENSORID
AND STATION.INCITY = CITY.CITYID
AND SENSOR.INCOMPONENT = COMPONENT.COMPONENTID
GROUP BY CITYID, COMPONENTID, YEARID
```

(a) query applied to detailed data without pre-computed aggregate

```
SELECT CITYID, COMPONENTID, YEARID, AVG_SO2
FROM MVIEW8
```

(b) query applied to a Multidimensional Graph with aggregates

Fig. 8. Querying with and without pre-computed aggregate in a relational data store.

implemented through different tables. Each tuple in an aggregate table corresponds to an aggregate vertex. A set of aggregate vertices sharing the same attributes are grouped in one aggregate table according to the lattice shown in Fig. 4(b) (excluding the lowest one representing the levels of raw data). The final schema is composed of 7 tables implementing raw data (cf. Fig. 7(a)) and 23 materialised views implementing pre-computed aggregates (e.g. the three materialised views in Fig. 7(b)).

6. Multidimensional graph querying

From the previous discussion (Section 3), we highlight the differences between our approach (graph aggregation) and graph summarisation. Specifically, instead of grouping and then reducing vertices and edges in a graph as the graph summarisation does, we aggregate numeric values within measure vertices according to values of attribute vertices from different analysis axes (dimensions) on different granularities (analytical levels) prior to querying. All pre-computed are materialised in a Multidimensional Graph to help reducing the volume of data involved in queries involving aggregates during analyses.

As a matter of fact, querying aggregated values in a Multidimensional Graph is not carried out in the same way as in original statistical RDF dataset without pre-computed aggregates. In original statistical RDF dataset, a query computes aggregates on-the-fly according to some grouping conditions and an aggregation function. In a Multidimensional Graph, aggregated measure values are pre-computed and can directly be extracted from the data store. In this way, a Multidimensional Graph allows avoiding computing aggregates data on-the-fly from detailed data.

For instance, in a relational data store (cf. Fig. 7), a decision-maker wants to carry out an analysis of average SO2 by city, component and year. Without a Multidimensional Graph, the decision-maker should write the SQL query as shown in Fig. 8(a) to compute average SO2 by city, by component and by year from the detailed data. With a Multidimensional Graph, the decision-maker can directly extract pre-computed aggregated value through the SQL query, as shown in Fig. 8(b).

Specifically, the first query (i.e. Fig. 8(a)) involves complex relational operations such as five joins between six tables as well as an aggregation including one average function and a group-by predicate with three attributes. In total, 41 billion tuples are involved in this query to generate the final result. The second query (i.e. Fig. 8(b)) includes only one projection operation which is much less costly than other complex relational operations in a DBMS. This query directly extracts pre-computed aggregates managed by a materialised view. During its execution, only 2.3 million tuples are involved (i.e. 0.0055% of the first query in Fig. 8). The same principle applies to all the other data stores.

A comparison of the different systems considered in this paper is presented in Table 1. We first present the data model and the query language of the different data stores used in this paper. Second, we provide an empirical comparison to evaluate the capacity of each data store to implement and query graph data. Specifically, implementing a graph in a column-oriented data store is complex, since using nested column structure to manage graph data requires a careful choice of design solutions. In the same manner, querying graph data involving a long path in relational and column data stores needs advanced skills in the corresponding querying languages.

Table 2
QBOAirbase datasets of different volumes.

Dataset	Included countries	Size (GB)	No. of observations	No. of triples
DS1	England	0.7	147,256	1,876,222
DS2	England, France	2.5	489,035	6,331,624
DS3	England, France, Spain	4.6	891,632	11,631,570
DS4	England, France, Spain, Germany	7.7	1,497,442	19,343,025
DS5	All countries	15	2,842,759	36,920,734

7. Experimental evaluation

In this section, we present our experimental evaluation on querying statistical RDF data using the aggregate-based model described above. We aim at studying the benefits of graph aggregation for querying efficiency according to different scenarios:

Table 1
Similarities and differences across the compared systems.

	RDF triplestore		*NoSQL*		*Relational databases*	
	Jena TDB	**Virtuoso**	**Neo4j**	**Cassandra**	**MySQL**	**PostGreSQL**
Model	RDF	RDF	Property Graph	Column	Relational	Relational
Query language	SPARQL	SPARQL	Cypher	CQL	SQL	SQL
Graph implementation	Easy	Easy	Easy	Complex	Medium	Medium
Graph querying	Easy	Easy	Easy	Complex	Complex	Complex

	Query	Remark
1	SO_2 by station, sensor and year	raw data without computing aggregated value
2	Average SO_2 by station, component, year	aggregate by 3 attributes on 3 dimensions
3	Average SO_2 by station and sensor	aggregate by 2 attributes on 2 dimensions
4	Average SO_2 by station and component	aggregate by 2 attributes on 2 dimensions
5	Average SO_2 by station	aggregate by 1 attribute on 1 dimension
6	Average SO_2 by city and component	aggregate by 2 attributes on 2 dimensions
7	Average SO_2 by city	aggregate by 1 attribute on 1 dimension
8	Average SO_2 by country and component	aggregate by 2 attributes on 2 dimensions
9	Average SO_2 by country	aggregate by 1 attribute on 1 dimension
10	Average SO_2	aggregate by extreme granularity on all analysis axes
11	Average SO_2 by city and year	aggregate by 2 attributes on 2 dimensions
12	Average SO_2 by country and year	aggregate by 2 attributes on 2 dimensions
13	SO_2 by station, sensor and year from 2006 to 2010	raw data without computing aggregated value with one condition
14	Average SO_2 recorded by one specified station	one condition on 1 attribute of low granularity on 1 dimension
15	Average SO_2 recorded by three specified stations	several conditions on 1 attribute of low granularity on 1 dimension
16	Average of SO_2 in one specified city	one condition on 1 attribute of middle granularity on 1 dimension
17	Average of SO_2 in one specified country	one condition on 1 attribute of high granularity on 1 dimension
18	Average of SO_2 in one specified city in 2010	conditions on 2 attributes of 2 dimensions
19	Average of SO_2 in one specified city from 2006 to 2010	conditions (with time interval) on 2 attributes of 2 dimensions
20	Average of SO_2 in one specified city (in 2010)	conditions on a displayed and a non-displayed attribute of 2 dimensions
21	Average of SO_2 recorded by one specified station and one specified component in 2010	conditions on 3 attributes of 3 dimensions
22	Average of SO_2 greater than 150 by city and year	conditions on a displayed measure
23	Average of SO_2 greater than 150 by year in three specified cities	conditions on a displayed measure and 1 attribute of 1 dimension
24	Average of SO_2 in three specified cities whose average NO_2 is greater than 150	conditions on a non-displayed measure

Fig. 9. Benchmark queries.

Table 3
%selectivity: how many triples/observations are retrieved.

Query	%selectivity	Triples/observations
13	29.36%	104,199/354,944
14	0.02%	1/4479
15	0,07%	3/4479
16	0.04%	1/2305
17	2.94%	1/36
18	0.00%	1/22,097
19	0.02%	5/22,097
20	0.04%	1/2305
21	0,00%	1/36,515
22	3.56%	787/22,097
23	0.09%	2/2305
24	0.05%	1/2046

1. querying multidimensional graphs without aggregates, using Jena TDB2 and Virtuoso triple stores;
2. querying multidimensional graphs with pre-computed *aggregates*, using Jena TDB2 and Virtuoso triple stores;
3. querying multidimensional graphs without aggregates, in MySQL and PostGre relational databases;
4. querying multidimensional graphs with pre-computed *aggregates*, in MySQL and PostgreSQL relational databases;
5. querying multidimensional graphs without aggregates, in Cassandra data store;
6. querying multidimensional graphs with pre-computed *aggregates*, in Cassandra data store;
7. querying multidimensional graphs without aggregates, in Neo4j database;
8. querying multidimensional graphs with pre-computed *aggregates*, in Neo4j database;

7.1. Material and methods

Datasets. We built five datasets based on the QBOAirbase dataset. Each dataset includes data related to one or several European countries and weights from 0.7 GB (about 4.14 million RDF triples) to 15 GB (over 81.92 million RDF triples). (cf. Table 2). The obtained logical representations are implemented at the physical level in different data stores: Jena TDB2 (v.3.6.0), Virtuoso (v.7.2.4.2), Cassandra (3.11.2), MySQL (8.0.11), PostgreSQL (10.4) and Neo4j (v.3.3.5).

Our experiments are carried out in 60 datasets: 5 different data volumes (DS1 − DS5) × 2 implementations (with and without aggregates) × 6 different data stores at the physical level.

It is worth noticing that we rely on the default tuning of each data store without any customised optimisation technique. By doing so, our objective is to avoid introducing any bias during analyses and find out the best native implementation for querying pre-computed aggregates.

Benchmark queries. We propose 24 queries covering a large range of operations during statistical RDF analyses. (cf. Fig. 9). Specifically, the first 12 queries involve all data ranging from the most detailed granularity to the most general granularity, while the last 12 queries extract a sub-set of data according to a selection criteria on different vertex of different granularities. Specifically, queries Q1 and Q13 involve only original data without computing aggregated measure values. It takes the same form in both *orig.* and *ag.* datasets. Queries Q2–Q12 (without selection conditions) and Q14-Q24 (with selection conditions) involve aggregated measures values upon different granularities on different analysis axes. They (i) compute aggregated values on-the-fly according to some grouping conditions and an aggregation function in *orig.* datasets and (ii) directly extract pre-computed and materialised aggregates in *ag.* datasets. For the selection and range queries, Table 3 shows the selectivity, i.e. how many triples/observations are retrieved). Each query has been run on the original data, without the filtering conditions and with the filtering conditions (i.e. for the query 16, calculating the number of cities of interest with respect to the number of cities in the data store). Each query has been written with SPARQL (Jena and Virtuoso triple stores), SQL (MySQL and PostgreSQL relational databases), CQL (Cassandra NoSQL) and Cypher (Neo4j NoSQL database).

In fact, different SPARQL query benchmarks exist, as for instance, the Berlin SPARQL (Bizer & Schultz, 2009), which aims at comparing the performance of RDF stores with the performance of relational database management systems; the LUBM dataset,[7], dedicated to OWL benchmarking (querying, reasoning, etc.); the EvoGen Benchmark (Meimaris & Papastefanatos, 2016b), for benchmarking of versioning RDF systems (temporal querying, queries on changes, longitudinal queries across versions, etc.); or still the SPB benchmark (Semantic Publishing Benchmark) (Kotsev et al., 2016), a set of queries expressing the requirements each RDF store needs to address in order to satisfy in real use cases (aggregations, join ordering, complex filter conditions, etc.). While these benchmarks cover aggregation queries, as we propose here, their RDF datasets are not expressed in the dedicated vocabulary for representing statistical RDF data. Our motivation for constructing our own dataset is that is dedicated to the multidimensional view and original data is modelled as a data cube using QB4OLAP.

[7] swat.cse.lehigh.edu/projects/lubm/.

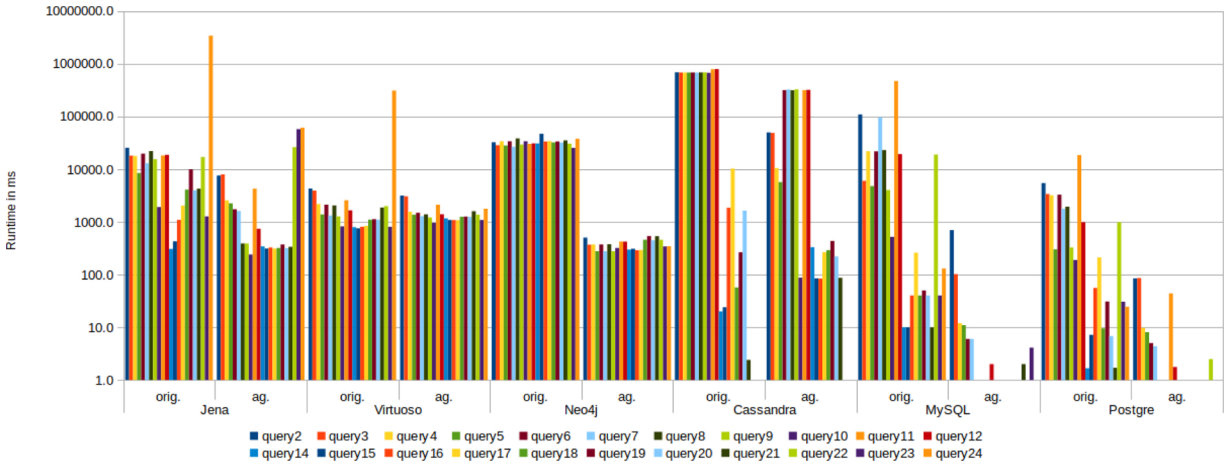


Fig. 10. Execution time (ms) of each query for the dataset D5.

Technical environment and evaluation metrics The hardware configuration is as follows: OS MAC OS 10.12.5, 2 × 2.4 GHz Quad-Core Intel Xeon, 48 GB 1066 MHz DDR3, 1 TB SATA Disk. We willingly choose a RAM with enough space to integrate all the datasets to avoid the different impacts of disk I/O when dealing with different sizes of datasets. In this case, the only influence factor consists of the dataset size.

For each query, we record its execution time (in millisecond) in both implementations, for ten runs. We clear the querying engine's cache before each execution, so that a previously executed query does not serve as warm-up run for the following one. In this way, during each query execution, data are read from a secondary storage instead of system cache. The final execution time of a query corresponds to the mean time of all runs.

7.2. Experimental results

In the following the results of our experiments are detailed by different kinds of queries.

Results per query. In a first analysis, we focus on the performance of each of the 24 queries for the largest dataset (D5) (Fig. 10). For the overall results discussed in the following subsections, we present the results per dataset. As we would expect, the best performance for most queries have been reported for the *ag.* datasets, with the worst performances were observed for the *orig.* datasets. For the best cases, while relational databases outperform the other data stores, for most worst cases Cassandra (queries Q2–Q12, as detailed below) is the one that performs poorly.

For queries Q23 and Q24, in particular, the worst performance was observed for Jena (57,600 ms in the *ag.*, for Q23, and 3,422,447 ms in *orig.* for Q24). In fact, in Jena TDB2, queries are always more efficiently computed in datasets with pre-computed aggregates. Queries with a single selection condition on measures and attributes (Q22, Q23 and Q24), however, are slightly more efficiently computed in datasets without aggregates. For Virtuoso, Q14–Q17 and Q23 perform worst in the *ag.* datasets. For Cassandra, a similar behaviour has been observed where several queries involving a selection perform worst in the *ag.* datasets.

Querying without aggregated data. As the graph aggregation brings additional triples (i.e. materialised aggregates) into a dataset (about 1.8% in all datasets from DS1 to DS5), we study if queries extracting original data without computing aggregated values require longer runtime. To do so, we execute the queries Q1 (without grouping condition or selection condition) and Q13 (including selection condition without grouping condition) in both *orig.* and *ag.* datasets. Figs. 11 and 12 show the execution time for queries Q1 and Q13 over the different datasets.

For query Q1 which contains neither grouping condition nor selection condition, pre-computed aggregates have impacts over query runtime in Jena TDB2 and Virtuoso. We can also notice that the impact of pre-computed aggregate over Q1 increases as the data volume increases (from DS1 to DS5) for Jena (214 ms in D1 up to 3557 ms in D5), Virtuoso (119 ms up to 1804 ms) and Neo4j (25 ms up to 45 ms), while decreasing for Cassandra (−26,744.5 in D1 down to −400,322.5 in D5), MySQL (−997.0 down to −21,760) and PostGres (−62.4 ms down to −4105 ms). The greatest gap (up to 3557 ms) of Q1's runtime is found in dataset DS5 of Jena TDB (with an absolute runtime of 58,790 ms in *orig.* DS5 dataset and 62,347 ms in *ag.* DS5 dataset), opposite to Cassandra (down to −400,322 ms, with an absolute runtime of 789,978 in *orig.* DS5 dataset and 389,655 ms in *ag.* DS5 dataset).

For query Q13 which contains only selection conditions without any grouping condition, its execution is greatly influenced by pre-computed aggregates in Jena TDB2 and Virtuoso. Notably, from DS1 to DS5, the gap is widened nearly 40 times in Virtuoso. The greatest gap (up to 2913 ms) of Q13's runtime is found in dataset DS5 of Virtuoso (with an absolute runtime of 7492 ms in *orig.* DS5 dataset and 10,405 ms in *ag.* DS5 dataset). For the other data stores, the behaviour is similar than Q1, were the biggest differences are observed for D5 with Cassandra (−21,296.4, with an absolute runtime of 171,726 ms in the *orig.* and 150,429.6 ms in the *ag.*).

Overall, Neo4j and Cassandra are the least affected by the materialisation of aggregates. For Neo4j there is practically no difference (max. 45 ms) while querying without summarising data before and after adding pre-computed aggregates, with or without selection condition.

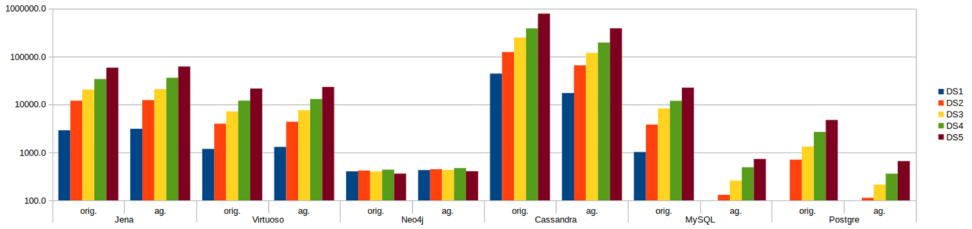


Fig. 11. Execution time (in ms) of query Q1 with and without pre-computed aggregates.

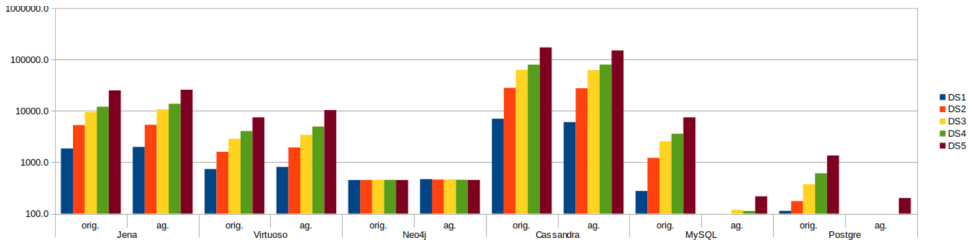


Fig. 12. Execution time (in ms) of query Q13 with and without pre-computed aggregates.

For Cassandra, a positive impact of the aggregation is observed.

Querying aggregated numeric values without selection condition. The second part of the experiments focuses on the impact of graph aggregation over queries Q2–Q12 which summarise data according to different grouping conditions without selection.

As shown in Fig. 13, overall, the average execution time in most data stores increases as the dataset size becomes larger. From DS1 to DS5, in the *orig.* datasets, the execution time increases by 8.87 times for Jena, 3.80 for Virtuoso, 9.98 for Neo4j, 8.90 for Cassandra, 33.33 for MyQSL and 40.61 for PostGres. For the *ag.*, this runtime decreases for all data stores: 4.38 for Jena, 6.14 for Virtuoso, 0.94 for Neo4j, 19.14 for Cassandra, 12.44 for MyQSL and 21.13 for PostGres. While Virtuoso is less affected by the increasing data volume of datasets without aggregates, Neo4j is less affected with the *ag.* datasets. Cassandra is the most affected with the increasing of the datasets for both *orig.* and *ag.*. Overall, the shortest average runtime is observed for PostGres with the largest for Cassandra.

As already stated above when analysing the individual queries, with respect to the pre-computed aggregates, it allows reducing query execution time in all data stores.

Specifically, looking for the smallest and largest datasets, in the dataset DS1. The same query requires less execution time in the *ag.* implementations than the *orig.* implementation of the same dataset in all data stores. Relational databases greatly benefit from the aggregations (MySQL, for instance, decreases from 2130.6 ms to 6.1 ms for D1, and from 71,022.7 ms to 76.4 ms for D5). We notice that the gain in Jena TDB2 and Neo4j with pre-computed aggregates becomes greater as the data volume increases; it reaches up to 83.4% and 98.8% respectively in the dataset DS5.

Querying aggregated numeric values with selection condition. In the third part of our experiments, we study if queries with selection conditions are efficiently computed in datasets with aggregates. We analyse the runtime of queries Q14-Q24 in all datasets with and without aggregates. From Fig. 14, we can observe that all data stores, pre-computed aggregates allow decreasing the execution time of queries with selection conditions. As observed for the first group of queries, overall runtime increases with the increasing in the volume of the datasets (with exception of Neo4j *ag*, Cassandra *orig.* and MySQL *orig.*). Among all data stores, for the largest dataset the lowest average runtime is recorded in relational databases with aggregates (Postgres and MySQL, respectively).

Table 4 shows the gain in terms of runtime for queries Q14-Q24 (with condition). We can observe that overall the gain is more important for this group of queries with respect to the first one. As observed above, all data stores benefit from the aggregation (no negative gain). Again, relational databases are the stores that better benefit from the aggregation, with MySQL with a gain of 100% for the largest dataset (from 1784.5ms to 0.7ms).

8. Discussion

8.1. Synthesis of results

With the increasing amount of statistical RDF data made available on the LOD cloud, there is a clear need for manipulating this kind of data taking advantage from multidimensional models as a way of facilitating the data analytic tasks. However, from the conceptual modelling point of view, existing work focuses rather on detailed data at the lowest levels of granularity. Consequently, querying efficiency based on such modelling solutions is low when detailed data should be summarised on-the-fly according to different levels of granularity on different analysis axes. Moreover, from the implementation point of view, existing work mainly focuses on optimising native RDF triple stores. Our work is a fist attempt to address the problem of querying aggregated data using different data stores.

In particular, the results of our experiments are opposite to what the authors of Kämpgen and Harth (2013) have observed: provided that aggregates are properly modelled in an RDF dataset, we highly recommend pre-computing aggregates in statistical RDF datasets to reduce query execution time. In our empirical evaluation, our findings were that pre-computing aggregates is a promising approach to improving querying performance in large RDF datasets.

In terms of findings, more specifically: (i) pre-aggregates improve the performance of all data stores; (ii) relational databases outperform all other data stores; (iii) Neo4j NoSQL is more recommended than Cassandra NoSQL for manipulating multidimensional data, as it scales better for larger datasets; (iv) Neo4j NoSQL performs better than Jena TDB2 and Virtuoso triple stores. This evaluation was based on a

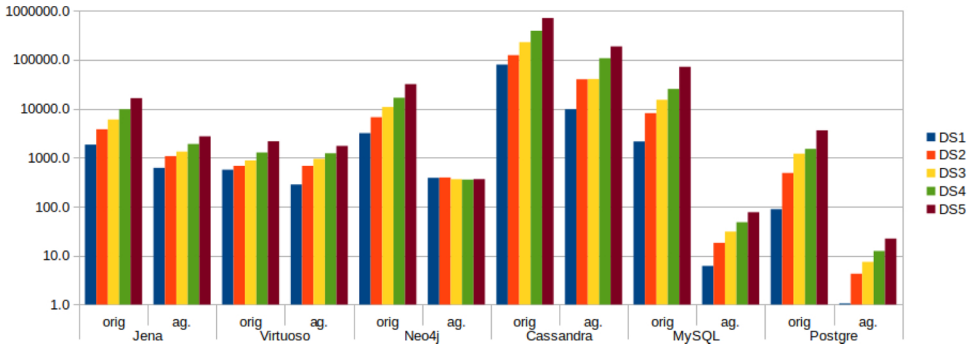


Fig. 13. Average execution time (in ms) of queries Q2–Q12 in datasets of different volumes.

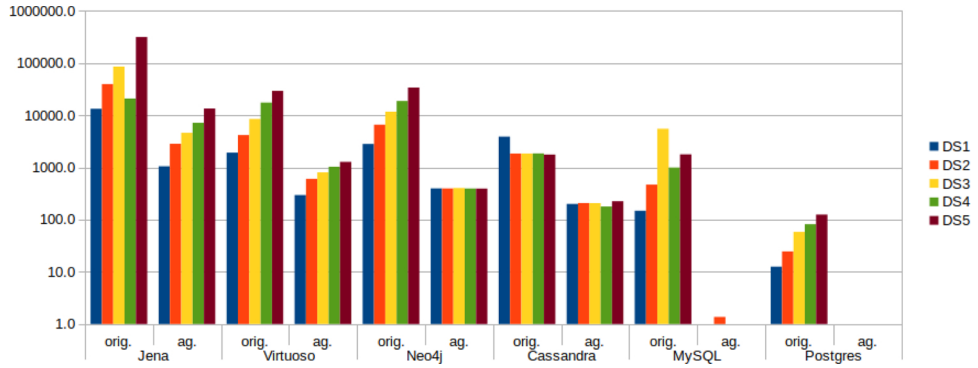


Fig. 14. Average execution time (in ms) of queries Q14–Q24 in datasets of different volumes.

Table 4
Gain after aggregation for DS1 et DS5.

		Jena	Virtuoso	Neo4j	Cassandra	MySQL	Postgres
Without condition	DS1	66.4%	50.2%	87.8%	87.8%	99.7%	98.8%
	DS5	83.4%	19.6%	98.8%	73.7%	99.8%	99.4%
With condition	DS1	92.1%	84.6%	86.0%	94.9%	99.8%	97.0%
	DS5	95.7%	95.6%	98.8%	87.3%	100.0%	99.5%

conceptual model with aggregates allows for representing aggregates independently of its implantation in different data stores. This theoretical model allows for the implantation of aggregates for further exploration in practical scenarios as well as the exploitation of it in further experimentations involving additional data store systems.

8.2. Theoretical contributions

Our theoretical contributions can be seen from different levels.

First, we formalise the concept of aggregates for statistical RDF data. The objective is to represent and materialise pre-computed summarised observation values according to different granularity levels on different analysis axes to avoid on-the-fly computing of detailed data during querying.

Second, based on the proposed concept of aggregate, we apply the multidimensional modelling principles to represent statistical RDF data. Our proposed modelling solution makes a distinction between raw statistical RDF data and pre-computed aggregates. It relies on the concepts of *bipartite* graph composed of measures, attributes and aggregates. It is a generic modelling solution which can be freely implemented in different data stores.

Third, we present a design process to convert statistical RDF data without aggregates at the physical level into the conceptual Multidimensional Graph. During the conversion, the process automatically enriches the Multidimensional Graph by all possible aggregates.

Forth, we introduce a transformation process from a conceptual Multidimensional Graph including both original data and aggregates into different logical and physical implementations (RDF, property-graph, column and relational data stores).

8.3. Implications for practice

From the practical point of view, our study is carried out in a systematic manner by using different types of data stores based on large amounts of real-world data.

It is worth noticing that short runtime (a few milliseconds) is always obtained with pre-computed aggregates. Meanwhile, the execution of corresponding queries based on detailed data without pre-computed aggregates takes several seconds even minutes. In another word, even though the amount of RAM is considerably larger than the dataset volume, on-the-fly computing of aggregate from detailed data is still not recommended for its low efficiency. Correspondingly, with pre-computed aggregates, a query can be significantly simplified (cf. Section 6.) The query becomes quite easy to handle (notably in Neo4j and relational data store): it contains only simple operations (projection) and the volume of data involved in this query become so small that a data store can produce the result immediately.

Pre-computing aggregates is not only an efficient technique to deal with large amounts of statistical RDF data but also a scalable solution. In fact, as we can see from our experiments, even though data volume increases rapidly (by 21 times), the average query runtime in all three data stores with aggregates does not increase exponentially.

We can also notice that the execution of the same query does not take the same time in different data stores. Thus, the choice of data stores impacts the querying efficiency. Among all types of data stores studied during our experiments, we recommend relational databases to manage statistical RDF data enriched with aggregates.

8.4. Limitations and future research direction

A first limitation of our study concerns the data stores. We have not include all potential systems for all data stores types. For instance, among the RDF data stores, we can further study the performance of Strabon, GraphDB, and Stardog. Among the relational databases, we can complete our study with more mature DBMS, such as Oracle, SQL Server and DB2. Among the graph and column data stores, we also intent to implement the Multidimensional Graph in OrientDB, and HBase. Meanwhile, our future study must include data of larger scale (e.g. exceeding memory limits). A long-term research direction could include in our study cloud data management systems (Senyo, Addae, & Boateng, 2018), such as Neptune and Cosmos. By doing so, our objective is to go through a hybrid approach for dynamically dispatching the query according to their type and previous characterised performance of the data store (i.e. taking the best of each data store).

A second limitation of our study is that it does not include any query optimisation technique, which we believe can further improve querying efficiency. Furthermore, while our experimental evaluation in this paper consists of a horizontal comparison of different data store with native tuning during multidimensional analyses, a future direction of our research consists in applying specific optimisation techniques (indexing, query plan, hint,etc.) to each data store and further investigate

the impact of them in the querying process. The objective is to study in a vertical manner the best performance that each data store can reach when dealing with analytic queries on statistical RDF data.

Another research direction would be improving the RDF modelling vocabulary of our Multidimensional Graph model. In the line of the work of Calvanese et al. (2017), the objective would be to support ontology-based data access to enable SPARQL querying over other data stores than RDF triple stores. For that, we will propose R2RML mappings for translating the original schemes to our ontology.

9. Conclusion

The primary aim of this paper was to improve the efficiency of complex queries involving data from different granularity levels. To that extent, we have proposed an approach for querying large volume of statistical RDF data, relying on pre-aggregation strategies. The proposal covers from a high level conceptual modelling solution to an implementation in different data stores with real-world large datasets. From the theoretical point of view, we have contributed to a novel multidimensional modelling solution coupled with a design and an implementation processes. This modelling solution allows for representing both detailed data and pre-computed aggregates. From the practical point of view, we have illustrated through experiments that pre-computed aggregates reduce query execution time in all data stores. The performance of analytic queries on relational databases with aggregates outperforms RDF triple stores, with a good performance of Neo4j NoSQL.

Authors' contribution

Franck Ravat: investigation, methodology, conceptualisation, writing – original draft preparation.

Jiefu Song: investigation, conceptualisation, writing – original draft preparation, software.

Olivier Teste: investigation, conceptualisation, reviewing.

Cassia Trojahn: investigation, methodology, writing – original draft preparation.

Acknowledgements

We warmly thank Tianyuan LIU who loaded the datasets in Cassandra, MySQL and PostGres. The authors have been partially supported by the French CIMI Labex project IBLiD (Integration of Big and Linked Data for On-Line Analytics).

References

Abadi, D. J., Boncz, P. A., & Harizopoulos, S. (2009). Column-oriented database systems. *Proceedings of the VLDB Endowment, 2*(2), 1664–1665. https://doi.org/10.14778/1687553.1687625.

Benedetti, F., Po, L., & Bergamaschi, S. (2014). A visual summary for linked open data sources. *Proceedings of the ISWC 2014 posters & demonstrations track a track within the 13th international semantic web conference, ISWC 2014* (pp. 173–176).

Bizer, C., & Schultz, A. (2009). The berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems, 5*(2), 1–24. https://doi.org/10.4018/jswis.2009040101.

Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., et al. (2013). *Building an efficient RDF store over a relational database. Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD'13*. New York, NY, USA: ACM121–132. https://doi.org/10.1145/2463676.2463718.

Botoeva, E., Calvanese, D., Cogrel, B., Rezk, M., & Xiao, G. (2016). OBDA beyond relational DBS: A study for mongodb. In: *Proc. of the 29th int. workshop on description logics (DL 2016)*.

Bouakkaz, M., Ouinten, Y., Loudcher, S., & Strekalova, Y. (2017). Textual aggregation approaches in OLAP context: A survey. *International Journal of Information Management, 37*(6), 684–692.

Bouhali, R., & Laurent, A. (2015). *Exploiting RDF open data using NoSQL graph databases. AIAI: artificial intelligence applications and innovations*. Bayonne, France: Springer177–190.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., et al. (2017). Ontop: Answering SPARQL queries over relational databases. *Semantic Web, 8*(3), 471–487.

Campinas, S., Perry, T. E., Ceccarelli, D., Delbru, R., & Tummarello, G. (2012). Introducing RDF graph summary with application to assisted sparql formulation. *2012 23rd international workshop on database and expert systems applications,* 261–266. https://doi.org/10.1109/DEXA.2012.38.

Cebiric, S., Goasdoue, F., & Manolescu, I. (2015). Query-oriented summarization of RDF graphs. In S. Maneth (Ed.). *Data science* (pp. 87–91). Springer International Publishing.

Chen, C., Yan, X., Zhu, F., Han, J., & Yu, P. S. (2008). Graph OLAP: Towards online analytical processing on graphs. *2008 eighth IEEE international conference on data mining,* 103–112. https://doi.org/10.1109/ICDM.2008.30.

Cudré-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., et al. (2013). *Nosql databases for RDF: An empirical evaluation. The semantic web – ISWC 2013*. Berlin, Heidelberg: Springer310–325.

de Vasconcelos, J. B., & Rocha, Á. (2019). Business analytics and big data. *International Journal of Information Management, 46*, 320–321. https://doi.org/10.1016/j.ijinfomgt.2018.10.019.

Diao, Y., Manolescu, I., & Shang, S. (2017). Dagger: Digging for interesting aggregates in RDF graphs. *Proceedings of the ISWC 2017 posters & demonstrations and industry tracks co-located with 16th international semantic web conference*.

Etcheverry, L., Vaisman, A., & Zimányi, E. (2014). *Modeling and querying data warehouses on the semantic web using QB4olap. Data warehousing and knowledge discovery (Vol. 8646)*. Springer International Publishing: Cham45–56. https://doi.org/10.1007/978-3-319-10160-6_5.

Etcheverry, L., & Vaisman, A. A. (2017). Efficient analytical queries on semantic web data cubes. *Journal on Data Semantics, 6*(4), 199–219. https://doi.org/10.1007/s13740-017-0082-y.

Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management, 35*(2), 137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007.

Ghrab, A., Romero, O., Skhiri, S., Vaisman, A., & Zimányi, E. (2015). *A framework for building OLAP cubes on graphs. Advances in databases and information systems*. Cham: Springer92–105.

Goasdoué, F., Karanasos, K., Leblay, J., & Manolescu, I. (2011). View selection in semantic web databases. *Proceedings of the VLDB Endowment, 5*(2), 97–108. https://doi.org/10.14778/2078324.2078326.

Gür, N., Nielsen, J., Hose, K., & Pedersen, T. B. (2017). Geosemolap: Geospatial OLAP on the semantic web made easy. *Proceedings of the 26th international conference on world wide web companion,* 213–217.

Hernández, D., Hogan, A., Riveros, C., Rojas, C., & Zerega, E. (2016). Querying wikidata: Comparing SPARQL, relational and graph databases. *Proceedings of the 15th international semantic web conference,* 88–103.

Hewitt, E. (2010). *Cassandra: The definitive guide* (1st ed.). O'Reilly Media, Inc.

Husain, M., McGlothlin, J., Masud, M. M., Khan, L., & Thuraisingham, B. M. (2011). Heuristics-based query processing for large RDF graphs using cloud computing. *IEEE Transactions on Knowledge and Data Engineering, 23*(9), 1312–1327. https://doi.org/10.1109/TKDE.2011.103.

Ingalalli, V., Ienco, D., Poncelet, P., & Villata, S. (2016). Querying RDF data using a multigraph-based approach. *Proceedings of the 19th international conference on extending database technology, EDBT 2016,* 245–256.

Jindal, A., Madden, S., Castellanos, M., & Hsu, M. (2015). *Graph analytics using vertical relational database. Proceedings of the 2015 IEEE international conference on big data (big data), BIG DATA'15*. Washington, DC, USA: IEEE Computer Society1191–1200. https://doi.org/10.1109/BigData.2015.7363873.

Joshi, A. K., Hitzler, P., & Dong, G. (2015). Alignment aware linked data compression. *Semantic technology – 5th joint international conference, JIST 2015* (pp. 73–81).

Kämpgen, B., & Harth, A. (2013). No size fits all – Running the star schema benchmark with SPARQL and RDF aggregate views. *The semantic web: Semantics and big data,* 290–304.

Kaoudi, Z., & Manolescu, I. (2013). Triples in the clouds. *2013 IEEE 29th international conference on data engineering (ICDE),* 1258–1261. https://doi.org/10.1109/ICDE.2013.6544918.

Khatchadourian, S., & Consens, M. P. (2010). *Explod: Summary-based exploration of interlinking and RDF usage in the linked open data cloud. The semantic web: Research and applications*. Berlin, Heidelberg: Springer Berlin Heidelberg272–287.

Konrath, M., Gottron, T., Staab, S., & Scherp, A. (2012). Schemex – efficient construction of a data catalogue by stream-based indexing of linked data. *Web Semantics, 16*, 52–58. https://doi.org/10.1016/j.websem.2012.06.002.

Kotsev, V., Minadakis, N., Papakonstantinou, V., Erling, O., Fundulaki, I., & Kiryakov, A. (2016). Benchmarking RDF query engines: The LDBC semantic publishing benchmark. *Proceedings of the workshop on benchmarking linked data (BLINK 2016) co-located with the 15th international semantic web conference (ISWC)* (pp. 2016). .

Liu, Y., Dighe, A., Safavi, T., & Koutra, D. (2016). *A graph summarization: A survey*. arXiv:1612.04883.

Meimaris, M., & Papastefanatos, G. (2016a]). Double chain-star: An RDF indexing scheme for fast processing of sparql joins. *EDBT*.

Meimaris, M., & Papastefanatos, G. (2016b]). The evogen benchmark suite for evolving RDF data. *Joint proceedings of the 2nd workshop on managing the evolution and preservation of the data web (MEPDaW 2016) and the 3rd workshop on linked data quality (LDQ 2016) co-located with 13th European semantic web conference (ESWC 2016)* (pp. 20–35).

Meimaris, M., Papastefanatos, G., Mamoulis, N., & Anagnostopoulos, I. (2017). Extended characteristic sets: Graph indexing for sparql query optimization. *2017 IEEE 33rd international conference on data engineering (ICDE),* 497–508. https://doi.org/10.1109/ICDE.2017.106.

Meimaris, M., Papastefanatos, G., Vassiliadis, P., & Anagnostopoulos, I. (2018).

Computational methods and optimizations for containment and complementarity in web data cubes. *Information Systems, 75*, 56–74.
Michel, F. (2017). *Integrating heterogeneous data sources in the Web of data (Theses)*. Université Côte d'Azur.
Neumann, T., & Moerkotte, G. (2011). *Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. Proceedings of the 2011 IEEE 27th international conference on data engineering, ICDE'11*. Washington, DC, USA: IEEE Computer Society984–994. https://doi.org/10.1109/ICDE.2011.5767868.
Neumann, T., & Weikum, G. (2010). x-rdf-3x: Fast querying, high update rates, and consistency for RDF databases. *Proceedings of the VLDB Endowment, 3*(1–2), 256–263. https://doi.org/10.14778/1920841.1920877.
Ordonez, C., Cabrera, W., & Gurram, A. (2017). Comparing columnar, row and array DBMSS to process recursive queries on graphs. *Information Systems, 63*, 66–79. https://doi.org/10.1016/j.is.2016.04.006.
Peng, P., Zou, L., Chen, L., & Zhao, D. (2019). Adaptive distributed RDF graph fragmentation and allocation based on query workload. *IEEE Transactions on Knowledge and Data Engineering, 31*(4), 670–685. https://doi.org/10.1109/TKDE.2018.2841389.
Pérez, J., Arenas, M., & Gutierrez, C. (2009). Semantics and complexity of sparql. *ACM Transactions on Database Systems, 34*(3).
Ravat, F., Song, J., & Teste, O. (2016). *Designing multidimensional cubes from warehoused data and linked open data. 2016 IEEE tenth international conference on research challenges in information science (RCIS 2016)*. Grenoble, France: IEEE171–182. https://doi.org/10.1109/RCIS.2016.7549337.
Ravat, F., Teste, O., Tournier, R., & Zurfluh, G. (2008). Algebraic and graphic languages for OLAP manipulations. *International Journal of Data Warehousing and Mining, 4*(1), 17–46.
Ravindra, P., Kim, H., & Anyanwu, K. (2016). Optimization of complex SPARQL analytical queries. *Proceedings of the 19th international conference on extending database technology,* 257–268.
Schmidt, M., Hornung, T., Küchlin, N., Lausen, G., & Pinkel, C. (2008). An experimental comparison of RDF data management approaches in a sparql benchmark scenario. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, & K. Thirunarayan (Eds.). *The semantic web – ISWC 2008* (pp. 82–97). Berlin, Heidelberg: Springer Berlin Heidelberg.
Schulz, M., Winter, P., & Choi, S.-K. T. (2015). On the relevance of reports-integrating an automated archiving component into a business intelligence system. *International Journal of Information Management, 35*(6), 662–671. https://doi.org/10.1016/j.ijinfomgt.2015.07.005.
Senyo, P. K., Addae, E., & Boateng, R. (2018). Cloud computing research: A review of research themes, frameworks, methods and future research directions. *International Journal of Information Management, 38*(1), 128–139. https://doi.org/10.1016/j.ijinfomgt.2017.07.007.
Spahiu, B., Porrini, R., Palmonari, M., Rula, A., & Maurino, A. (2016). ABSTAT: Ontology-driven linked data summaries with pattern minimalization. *The semantic web – ESWC 2016 satellite events* (pp. 381–395).
Sun, W., Fokoue, A., Srinivas, K., Kementsietsidis, A., Hu, G., & Xie, G. (2015). *Sqlgraph: An efficient relational-based property graph store. Proceedings of the 2015 ACM SIGMOD international conference on management of data, SIGMOD'15*. New York, NY, USA: ACM1887–1901. https://doi.org/10.1145/2723372.2723732.
Thakkar, H., Punjani, D., Lehmann, J., & Auer, S. (2018). *killing two birds with one stone – Querying property graphs using SPARQL via GREMLINATOR.* ArXiv e-prints.
Tian, Y., Hankins, R. A., & Patel, J. M. (2008). Efficient aggregation for graph summarization. *Proceedings of the 2008 ACM SIGMOD international conference on management of data, SIGMOD'08,* 567–580.
Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., & Boncz, P. (2012). *Heuristics-based query optimisation for sparql. Proceedings of the 15th international conference on extending database technology*. New York, NY, USA: ACM324–335.
Wang, X., Staab, S., & Tiropanis, T. (2016). ASPG: generating OLAP queries for SPARQL benchmarking. *Semantic technology – 6th joint international conference* (pp. 171–185).
Wu, Y., Zhong, Z., Xiong, W., & Jing, N. (2014). Graph summarization for attributed graphs. *2014 international conference on information science, electronics and electrical engineering (Vol. 1),* 503–507.
Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., et al. (2018a]). Ontology-based data access: A survey. *Proceedings of the twenty-seventh international joint conference on artificial intelligence, IJCAI-18,* 5511–5519. https://doi.org/10.24963/ijcai.2018/777.
Xiao, G., Kontchakov, R., Cogrel, B., Calvanese, D., & Botoeva, E. (2018). Efficient handling of sparql optional for obda. In D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L.-A. Kaffee, & E. Simperl (Eds.). *The semantic web – ISWC* (pp. 354–373). Cham: Springer International Publishing.
Yuan, P., Liu, P., Wu, B., Jin, H., Zhang, W., & Liu, L. (2013). Triplebit: A fast and compact system for large scale RDF data. *Proceedings of the VLDB Endowment, 6*(7), 517–528. https://doi.org/10.14778/2536349.2536352.
Zhao, P., Li, X., Xin, D., & Han, J. (2011). Graph Cube: On warehousing and OLAP multidimensional networks. *Proceedings of the 2011 ACM SIGMOD international conference on management of data,* 853–864.
Zneika, M., Lucchese, C., Vodislav, D., & Kotzinos, D. (2016). Summarizing linked data RDF graphs using approximate graph pattern mining. *In Proceedings of the 19th international conference on extending database technology, EDBT 2016* (pp. 684–685). . https://doi.org/10.1016/j.ijinfomgt.2017.06.005.