Preprint of the paper:
ME Papadaki, Y. Tzitzikas, N. Spyratos, Analytics over RDF Graphs, Post-proceedings of the 13th International Workshop on Information Search, Integration, and Personalization (ISIP'2019), Lecture Notes in Computer Science book series (LNCS, volume 1197), 2020 (to appear)

Analytics over RDF Graphs

Maria-Evangelia Papadaki[1], Yannis Tzitzikas[1], and Nicolas Spyratos[2]

[1] Institute of Computer Science, ICS-FORTH and
Department of Computer Science, University of Crete, Greece,
marpap|tzitzik@ics.forth.gr,
[2] Université de Paris-Sud, Laboratoire de Recherche en Informatique,France,
spyratos@lri.fr

Abstract. The continuous accumulation of multi-dimensional data and the development of Semantic Web and Linked Data published in RDF bring new requirements for data analytics tools. Such tools should take into account the special features of RDF graphs, exploit the semantics of RDF and support flexible aggregate queries. In this paper, we present an approach for applying analytics to RDF data, based on a high-level functional query language called HIFUN. According to that language, each analytical query is considered as a well-formed expression of a functional algebra and its definition is independent of the nature and structure of the data. In this work, we detail the required transformations, as well as the translation of HIFUN queries to SPARQL and we introduce the primary implementation of a tool, developed for these purposes.

Keywords: Analytics, RDF, Linked Data

1 Introduction

The amount of data available on the Web today is increasing rapidly due to successful initiatives, such as the Linked Open Data movement[3]. More and more data sources are being exported or produced using the Resource Description Framework (or RDF, for short) standardized by the W3C[4]. SPARQL[5], which is the standard query language for RDF data, supports complex querying using regular path expressions, grouping, aggregation, etc., but the application of analytics to RDF data and especially to large RDF graphs is not so straightforward. The structure of such graphs tends to be complex, due to several factors: (a) different resources may have different sets of properties, (b) properties can be multi-valued (i.e. there can be triples where the subject and predicate are the same but the objects are different) and (c) resources may or may not have types. In addition, the analytical tools that have been developed, are not capable of supporting RDF graph analytics effectively, as they (i) focus on relational data, (ii) can only work with a single homogeneous data set, (iii) neither support multiple central concepts, nor RDF semantics, (iv) demand deep knowledge of specific query languages, depending

[3] http://lod-cloud.net/
[4] https://www.w3.org/RDF/
[5] https://www.w3.org/TR/rdf-sparql-query/

on data's structure and (v) do not offer flexible choices of dimension, measure, and aggregation.

In view of the above challenges, there is a need for a common formal framework that can be applied to one or more linked data sets and demands no programming skill. Motivated from this need, we are investigating an approach based on a high-level query language, called HIFUN [23], for applying analytics to RDF graphs. We study how that language can be applied to RDF data by clarifying how the concept of analysis context can be defined, what kind of transformations are required and how HIFUN queries can be translated to SPARQL. Moreover, we describe the primary implementation of an analytical tool based on the above.

The remainder of the paper is organized as follows: Section 2 describes the requirements and the related work. Section 3 introduces the related background knowledge. Section 4 focuses on how HIFUN can be applied to RDF data. Section 5 describes how HIFUN queries can be translated to SPARQL. Section 6 refers to application issues and describes the current implementation, and finally Section 7 concludes this work and discusses issues for future research.

2 Requirements and Related Work

2.1 Requirements

Today, there are domain-specific semantic warehouses, such as in the marine domain [26], or the cultural domain [10], as well as general-purpose knowledge bases, such as DBpedia and WikiData[6] (see [18] for a survey). These warehouses store huge volumes of integrated data from two or more disparate sources and their data can be analyzed in various ways. For example, the data warehouse of [26] contains data describing fish species according to several perspectives including water areas, countries, families, etc. Such data can be analyzed in order to find the number of different fish species by country or by water area. General-purpose knowledge bases on the other hand, such as DBpedia, can be analyzed for various purposes, (e.g. for finding the number of French actors, born in 1980). Apart from the above, analytics can be useful also for checking the quality of semantic integration activities, (e.g. for measuring the commonalities between several data sets as in [14, 17]).

So, we need a way for applying analytics to any kind of RDF graph, - not only to multidimensional data expressed in RDF, but also to domain-specific or semantic data of general-purpose; a way, that would be applicable to several RDF data sets, as well as to any data source. We need an analytical tool that allows the user to select the desired data set(s) or desired parts thereof, formulate an analytic query without having any programming knowledge and finally get the results in the form of tables, plots or any other kind of visualization, intuitively.

2.2 Related Work

There are many cases where it would be useful to publish multi-dimensional data (such as statistics) on the web in such a way that it can be linked to related data sets and

[6] https://www.wikidata.org

concepts. The RDF Data Cube vocabulary[7] (QB) provides a means to publish such data on the web using the W3C RDF standard. That vocabulary consists of three main components: i) the *measures*, which are the observed values of primary interest, ii) the *dimensions*, which are the value keys that identify the measure and iii) the *attributes*, which are the metadata. However, even though this vocabulary can be used for structuring and publishing multi-dimensional data, it cannot be used for applying analytics over it. In view of this limitation, several approaches have been proposed.

These approaches could be divided into two major groups: i) those that extract Multi-dimensional Data (MD), that is data related to more than two dimensions from the web and load it into traditional data management systems for OLAP analysis [11, 19], and ii) those that perform OLAP analysis directly over the Semantic Web data, representing MD data in RDF [1].

The work in [12] analyzes data expressed in the RDF Data Cube format by constructing OLAP queries, which are then transformed into SPARQL. However, the proposed method requires a ROLAP engine to execute the OLAP queries and analyzes the resulting cubes through classical OLAP operations. On the other hand, the work in [9] presents a framework for analyzing LOD data. It differs from our work since the proposed method is not based on the usage of dedicated OLAP cube vocabularies (e.g. RDF Data Cube Vocabulary). Instead, it stores the RDF data in property tables (PTs) and transforms linked data to relational data so that it can be exploited using typical OLAP systems.

The representation of MD data in RDF can further be organized in two categories: i) those that are based on specialized RDF vocabularies [5, 6] and ii) those that implicitly define a data cube over existing RDF graphs[8]. Our work follows the first approach since we apply analytics over data that has been expressed in the RDF Data Cube format (although the objective is to be applicable to any RDF data set).

The work in [29] defines OLAP operations on analysis cubes over graphs. However, its approach does not support heterogeneous graphs, and thus it cannot handle multi-valued attributes (e.g., a person being both "Greek" and "French"), nor semantics. Additionally, [3] presents a graph model for OLAP directly on RDF graphs and an extension of SPARQL for OLAP querying. However, according to [6], it cannot be guaranteed that the cubes on RDF graphs are multi-dimensional compliant.

The existing methods can also be classified into i) those that require programming knowledge for analyzing the data and ii) those that do not deal with lower-level technicalities. The work in [27] presents a system for analytics over (large) graphs. It achieves efficient query answering, by dividing the graph into partitions. However, in contrast to our work, the user should have some programming knowledge, since it is necessary to write a few lines of code to submit the query. The work in [28] presents a method for applying statistical calculations on numerical linked data. It stores the data in arrays and performs the calculations on the arrays' values. Nevertheless, contrary to our work, it requires deep knowledge of SPARQL for formulating the queries.

In order to overcome one's difficulty in background programming knowledge, high-level languages have been developed for data analysis, too. However, there has not been

[7] https://www.w3.org/TR/vocab-data-cube/
[8] https://team.inria.fr/oak/projects/warg/

much activity in introducing high-level languages, suitable for analytics on RDF data. While general-purpose languages, such as PIG Latin [20] and HiveQL [25] can be used, they are not tailored to address the peculiarities of the RDF data model. Even though, [7, 8] present high-level query languages enabling OLAP querying of an extended format of data cubes [6], they are only applicable to data already represented and published using a corresponding vocabulary. As a consequence, they fall short in addressing a wide variety of analytical possibilities in non-statistical RDF data sources. In addition, [20] proposes a high-level language that supports semantics. However, it is targeted at processing structured relational data, limiting its use for semi-structured data such as RDF. Further, it provides only a finite set of primitives that is inadequate for the efficient expression of complex analytical queries.

Finally, a survey that is worth mentioning is [4], which introduces warehouse-style RDF analytics. There are similarities with our approach, since each analytical schema node corresponds to an RDF class, while each edge corresponds to an RDF property. Nonetheless, since the facts are encoded as unary patterns, they are limited to vertices instead of arbitrary subgraphs (e.g. paths).

In conclusion, in contrast to the aforementioned works, we focus on developing a user-friendly interface, where the user will be able to apply analytics to RDF data without dealing with lower-level technicalities. We envision a system, that will not be based on any specialized vocabulary and will be capable of analyzing one or several linked data sets.

3 Background

3.1 Resource Description Framework (RDF) and Linked Data

The Resource Description Framework (RDF) [2, 16] is a graph-based data model for linked data interchanging on the web. It denotes resources through the use of Uniform Resource Identifiers (URIs), or anonymous resources (blank nodes) and constants (Literals). This framework uses triples, which are statements of the form subject-predicate-object (s, p, o), in order to relate a resource with other resources or constants.

Definition 1 (RDF Triple, RDF Data set, RDF Graph). A *triple* is considered to be any element of $T = (U \cup B) \times (U) \times (U \cup B \cup L)$, where U, B and L denote the sets of URIs, blank nodes and literals, respectively. An *RDF graph* (or *RDF data set*) is any finite subset of T. □

For instance, if "schema" is the URI prefix "https://schema.org/", then schema:FinancialProduct is the URI of the class FinancialProduct. If "myStore" is the URI prefix "https://myStore.com", then myStore:product1 is the URI of a particular product. Consequently, the statement (myStore:product1, rdf:type, schema:FinancialProduct) is a triple, indicating that myStore:product1 is an instance of the class schema:FinancialProduct. In addition, (myStore:product1, schema:purchaseDate, "2019-05-09") is a triple, denoting that the product was purchased by its owner in "2019-05-09". The set of URIs could be classified in three different subsets, (i) entities (e.g. myStore:product1), (ii) properties (e.g. schema:purchaseDate) and (iii) classes (e.g. schema:FinancialProduct). An entity can be a subject or object in a triple, a property is always a predicate, while a class can be

found in the object of a triple and corresponds to the type/category, in which an entity belongs to.

We shall use the example of Fig. 1 as our running example, throughout the paper. The representation is in RDF and it shows a delivery invoice, that took place at "branch3" in "2019-05-09". The product that was delivered to that branch in the quantity of "400", was the "product4" of "Hermes" brand. The founder of that brand is "Manousos", who is both Greek and French.

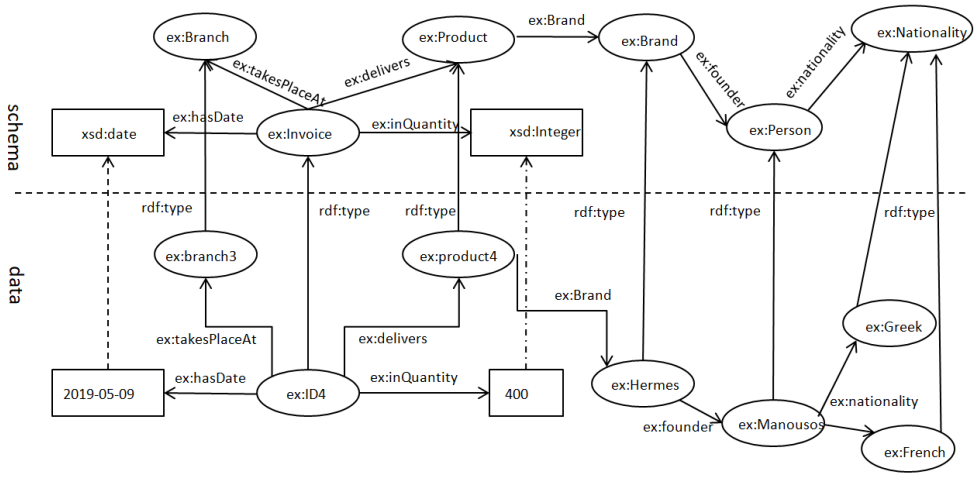

Fig. 1. Running example

3.2 HIFUN - A High Level Functional Query Language for Big Data Analytics

HIFUN [23] is a high-level functional query language for defining analytic queries over big data sets, independently of how these queries are evaluated. It can be applied over a data set that is structured or unstructured, homogeneous or heterogeneous, centrally stored or distributed.

Data set Assumptions. To apply that language over a data set D, two assumptions should hold. The data set should i) consist of *uniquely identified data items*, and ii) have a set of *attributes*, each of which is viewed as a *function* associating each data item of D with a value, in some set of values. For example, if the data set D is a set of all delivery invoices over a year, in a distribution center (e.g. Walmart) which delivers products of various types in several branches, then the attribute "product type" (denoted as pt) is seen as a function $pt : D \rightarrow String$ such that, for each invoice i, $pt(i)$ is the type of product delivered according to the invoice i.

Definition 2 (Analysis Context). Let D be a data set and A be the set of all attributes (a_1, ..., a_k) of D. An *analysis context* over D is any set of attributes from A, and D is considered the origin (or root) of that context. □

Note that an analysis context can consist of more than one roots. While one root means that data analysis concerns a single data set, the existence of two or more roots means that data analysis relates to two or more different data sets, possibly sharing one or more attributes.

A set of attributes could be represented as a directed labeled graph. Fig. 2 shows our running example, expressed as such a context. From a syntactic point of view, the edges of it can be seen as triples of the form (source, label, target).

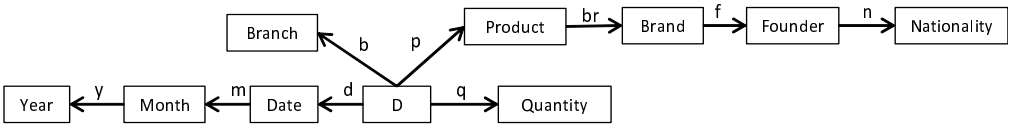

Fig. 2. Running example expressed as a HIFUN context

Direct & Derived Attributes. The attributes of a context are divided into two groups, the *direct* and the *derived*. The first group contains the attributes with origin D: these are the attributes whose values are given. The second group contains the attributes whose origins are different than D and whose values are computed based on the values of the direct attributes. For example, in Fig. 2 the attributes d, b, p and q are direct as their values appear on the delivery invoice, whereas m and y are derived, since their values can be computed from those of the attribute d (e.g. from the date 26/06/2019 one can derive the month 06 and the year 2019).

Definition 3 (HIFUN Analytic Query). A query in HIFUN is defined as an ordered triple $Q = (g, m, op)$ such that g and m are attributes of the data set D, having a common source (that is the *root* D) and op is an aggregate operation (or *reduction operation*) applicable on m-values. The first component of the triple is called *grouping function*, the second *measuring function* (or the *measure*) and the third *aggregate operation* (or *reduction operation*). □

The evaluation of such a query Q is done in a three-step process, as follows: i) items with the same g-value g_i are grouped, ii) in each group of items created, the m-value of each item in the group is extracted from D and iii) the m-values obtained in each group are aggregated to obtain a single value v_i. Actually, the aggregate value v_i is the answer of Q on g_i. This means that a query is a triple of functions and its answer $AnsQ$ is a function, too.

4 Using HIFUN as an Interface to RDF Dataset

4.1 Motivation

There are several ways in which HIFUN can be used, such as for studying rewriting of analytic queries in the abstract [23] or for defining an approach to data exploration [24]. In this paper, we use HIFUN as a user-friendly interface for defining analytic queries over RDF data sets. To understand the proposed approach, consider a data source S with query language L (e.g. S could be a relational data set and L the SQL language). In order to use HIFUN as a user interface for S, we need to (a) define an analysis context, that is a subset D of S to be analyzed and some attributes of D that are relevant for the analysis and (b) define a mapping of HIFUN queries to queries in L.

Defining a subset D of S can be done using a query of L and defining D to be its answer (i.e. D is defined as a view of S); and similarly, the attributes that are relevant to the analysis can be defined based on attributes of D already present in S. However, defining a mapping of HIFUN queries to queries in L might be a tedious task. In [24] such mappings have been defined from HIFUN queries to SQL queries and from HIFUN queries to MapReduce jobs.

The main objective of this paper is to define a user-friendly interface allowing users to perform analysis of RDF data sets. To this end, we use the HIFUN language as the interface. In other words, we consider the case, where the data set S mentioned above is a set of RDF triples and its language L is the SPARQL language. Our main contributions are: (a) the proposal of tools for defining a HIFUN context from the RDF data set S and (b) defining a mapping from HIFUN queries to SPARQL queries. With these tools at hand, a user of the HIFUN interface can define an analysis context of interest over S and issue analytic queries using the HIFUN language. Each such query is then translated by the interface to a SPARQL query, which in turn is evaluated over the RDF triples of D and the answer is returned to the user.

4.2 Applicability of HIFUN

Recall that two assumptions must be satisfied in order to apply HIFUN (see §3.2). The first assumption is satisfied by the RDF Data since each resource in RDF is identified by a distinct URI. Therefore, the data set D in HIFUN can be any subset of the set of all the available URIs. The second assumption, the functionality of attributes, is satisfied by the RDF properties, which are defined as functional (i.e. owl:FunctionalProperty) or are effectively functional (i.e. even if they are not declared as functional, they are single-valued for the resources in the data set D). Consequently, the cases that require special handling, include the following: (a) properties with no value, since HIFUN assumes that there are no empty values in the data, in contrast to RDF, where properties with no value may exist, (b) properties, that are multi-valued, and (c) definition of analysis contexts that correspond to a transformation of the original data.

These issues are discussed in the following sections.

4.3 How to Specify the Context of Analysis

In order to specify an analysis context, the classes and the properties of interest of an RDF graph should be selected. The user can select as the *root* of the analysis context any class of the RDF graph and as attribute any property of it (whose domain is that class). For example, any of the classes "ex:Invoice", "ex:Branch", "ex:Product", "ex:Brand", "ex:Person", "ex:Nationality" of Fig. 1 can be selected as the root of a context, while any of the properties "ex:hasDate", "ex:takesPlaceAt", "ex:delivers", "ex:inQuantity", "ex:Brand", "ex:founder", "ex:nationality" as its attributes, since they do have a common root. An analysis context can also be defined by transforming the original data, and such issues are discussed in Section 6.

5 Translation of HIFUN Queries to SPARQL

In this section, we show how a HIFUN query can be translated to a SPARQL query. Recall, that a query in HIFUN is defined as an ordered triple $Q = (g, m, op)$, where g is the grouping function, m the measuring function and op the aggregate operation. On the other hand, an aggregate query in SPARQL is defined as:

```
SELECT ?group (function(?var) AS ?result)
WHERE {
....
}
GROUP BY ?group
```

Based on the above definitions, a HIFUN query Q can be encoded as a SPARQL group-by query, as shown in row 1 of Table 1. The grouping function corresponds to the projections in the SELECT clause as well as to the aggregate variable(s) in the GROUP BY clause, the reduction (or aggregate operation) to the aggregate SPARQL function, and the measuring function to the argument of that function. Note that, we define as target the *codomain* (or *target set*) of the HIFUN attributes. The answer to this query is a binary table with two variables, $target(g)$ and Res (Res is a user-defined variable that holds the aggregate result).

Definition 4 (Translation of a HIFUN Query to a SPARQL query). A HIFUN query Q over a context C is translated to SPARQL by grouping the items with the same projection value g_i, extracting the aggregate function's argument value m_i of each item of the created groups and aggregating the m_i values obtained in each group, in order to obtain a single value v_i. □

Example 1. Suppose, that we would like to find the total quantities of products, delivered to each branch during the year. This query would be expressed in HIFUN and SPARQL respectively, as it is shown in row 2 of Table 1.

Complex grouping/measuring function. A grouping (as well as a measuring) function in HIFUN can be *more complex* using the following four operations on functions,

Table 1. HIFUN to SPARQL

id	name	HIFUN query Q	SPARQL query

1	Plain	$(grouping, measuring, reduction)$	SELECT ?target(grouping) reduction(?target(measuring)) As ?Res
			WHERE {
			....
			}
			GROUP BY ?target(grouping)

2	Plain	(b, q, SUM)	SELECT ?branch SUM(?quantity) AS ?TOTALS
			WHERE {
			?ID ex:takesPlaceAt ?branch .
			?ID ex:inQuantity ?quantity .
			}
			GROUP BY ?branch

3	Pairing	$(b \wedge p, q, SUM)$	SELECT ?branch ?product
			SUM(?quantity) AS ?TOTALS
			WHERE {
			?ID ex:takesPlaceAt ?branch .
			?ID ex:delivers ?product .
			?ID ex:inQuantity ?quantity .
			}
			GROUP BY ?branch ?product

4	Complex grouping function	$((d \circ m) \wedge b, q, SUM)$	SELECT ?branch ?mon SUM(?quantity) AS ?TOTALS
			WHERE {
			?ID ex:hasDate ?date .
			?date rdfs:label ?label .
			?ID ex:inQuantity ?quantity .
			?ID ex:delivers ?product .
			?product ex:hasBrand ?branch .
			}
			GROUP BY ?branch (month(?label) AS ?mon)

5	Attribute-restricted	$(e/E, e', op)$	SELECT ?target(e), op(?target(e')) As ?Res
			WHERE {
			FILTER(op(?target(e')) op')
			}
			GROUP BY ?target(e)

6	Attribute-restricted	$(b/\text{“}branch1\text{”}, q, SUM)$	SELECT ?product SUM(?quantity) AS ?TOTALS
			WHERE {
			?ID ex:takesPlaceAt ?branch.
			?ID ex:delivers ?product.
			?ID ex:inQuantity ?quantity.
			
			FILTER regex((?branch), "branch1", "i")
			}
			GROUP BY ?product

7	Result-restricted	$(e, e', op)/F$	SELECT ?target(e), op(?target(e')) As ?Res WHERE { ... } GROUP BY ?target(e) HAVING(op(?target(e'))op')

8	Result-restricted	$(b, q, SUM)/F$, where $F = \{b_i \in Branch/ans\,(b_i) > 300\}$	SELECT ?branch SUM(?quantity) AS ?TOTALS
			WHERE {
			?ID ex:takesPlaceAt ?branch.
			?ID ex:inQuantity ?quantity.
			}
			GROUP BY ?branch
			HAVING (SUM(?quantity) > 300)

that are defined in [23]: *pairing ($\wedge$), composition ($\circ$), Cartesian product projection ($\times$)* and *restriction ($/$)*.

These operations form the so called functional algebra [22] and they are well known, elementary operations except probably for pairing, which works as a tuple constructor and is defined as follows:

Pairing: Let $f : X \rightarrow Y$ and $g : X \rightarrow Z$ be two functions with common domain X. The pairing of f and g, denoted $f \wedge g$ is a function from X to $Y \times Z$ defined by: $f \wedge g(x) = (\mathrm{f(x)}, \mathrm{g(x)})$, for all x in X [23].

Example 2. Suppose, that we ask for the total quantities delivered by branch and product. The answer to this query Q is a function, associating each pair $(branch, product)$ with a total quantity. In other words, Q asks for the total quantities delivered by branch and product and it would be formulated as it is shown in row 3 of Table 1.

Example 3. Suppose we want the total quantities of products delivered, grouped by branch and month. Since the attribute of *month* is a derived one, it would have to be expressed using the operator of *composition*. That operator would be used to combine the attribute of *date* with that of *month*. On the other hand, the corresponding query in SPARQL would be expressed by "extracting" the value of month from the date by applying the built-in SPARQL function of *month*, which operates on date values. These queries would be defined as it is shown in row 4 of Table 1.

Restricted Query. A query Q in HIFUN can further be enriched by introducing functional restrictions either at the level of attributes or at the level of query answers.

Regarding the case of attribute-restricted queries, the restrictions are applied at the level of the attributes, filtering the results internally and the queries are defined as shown in row 5 of Table 1. The HIFUN query is evaluated by computing the restriction e/E (where E is any subset of the Data set D) and then, the query $(e/E, e', op)$ over E. On the other hand, in SPARQL the query is evaluated by specifying the subset of the data set D the user is interested in (using the $FILTER$ operator, triples patterns, etc.)

Example 4. Suppose, that we would like to find the total quantities of products, that received by a specific branch e.g. "branch1", group by product. Then, the queries would be expressed as shown in row 6 of Table 1. Alternatively, the restriction could be performed using a triple pattern, by specifying the particular branch that a delivery invoice took place. In that case, each branch would have been represented with a URI, e.g. $ex : branch_i$ and the constraint would be defined using triples of the form $?ex : ID$ $ex : takesPlaceAt$ $ex : branch_i$.

The decision between a triple pattern and filter for expressing a restriction in the inner results of a SPARQL query depends on the way our data has been represented. The first case concerns data that has been represented with $URIs$ and the use of triple patterns is preferred. The second one relates to data that has been produced using $literals$ and in that case, the operator of $filter$ is applied. This operator is also used in boolean conditions where any unwanted results should have to be filtered out.

Regarding the case of result-restricted queries, the final result can be filtered by setting restrictions on them. The queries, in this case, are defined as it is shown in row 7 of Table 1. The query in HIFUN is evaluated by first evaluating the query $Q = (e, e', op)$ over D and then computing the restriction $ansQ/F$. The corresponding SPARQL is evaluated and its result is filtered using the $HAVING$ operator.

Example 5. Suppose that we would like to find the number of products received by branch, but only for those branches that received more than 300 products. The corresponding queries would be expressed as shown in row 8 of Table 1.

6 Application and Implementation

6.1 Defining an Analysis Context over RDF Data

As discussed in Section 4.1, in order to define an analysis context D, the query language of the data source can be exploited. This is required mainly in general-purpose knowledge bases (as discussed in §2). Below, we describe some methods for defining such a context over RDF data. Here, we consider an analysis context as a pair (E, F), where E is a set of resources (i.e. a set of URIs), and F is a set of attributes for the objects in E. Such a pair can be defined, as follows:

• M_{plain}: If the data set D consists only of a single class (say C) and all the properties have as domain or range that class, the analyst has just to select the desired subset of these properties. In this case, $E = \{u \in U \mid (u, \text{rdf:type}, C)\}$, and F is any non empty subset of $Props(C) = \{u \mid (\text{p,rdfs:domain,C})\} \cup \{u \mid (\text{p,rdfs:range,C})\}$. Note that, this case captures data expressed in the RDF Data Cube format.

• M_{QLview}: The analyst can use a SPARQL SELECT query for defining a view, having all the attributes required for the analysis. For instance, if $v_1, \ldots v_k$ is the set of variables in the SELECT part of the query, then E can be considered to be the set of bindings of v_1, while F the bindings of the set of variables $v_2, \ldots, v_k$.

• $M_{QLobjects}$: The analyst can write a SPARQL query for defining only the objects of interest, i.e. the set E, not their attributes. Then, a tool could be used to suggest (or let the analyst select) the applicable properties F based on the schema and/or data. Also, note that the objects of interest E can be defined explicitly, i.e. by just providing the list of the desired URIs.

Special cases. Regarding the $M_{QLobjects}$ case, note that if all the properties in F have a value for each E and they are single-valued too, then the context has already been defined. However, there are cases that may require special handling: (i) there are properties with no value (such cases can occur in M_{QLview} if the OPTIONAL keyword is used), (ii) the analyst is interested in a path of properties, not a single property, and (iii) there are properties, which are multi-valued. To tackle such cases, some transformations may be required. A few feature operators that could be used in such cases, are indicated in Table 2. That table lists the nine most frequent *Linked Data-based Feature Creation Operators* (for short *FCO*s), as defined in [15] and they have been re-grouped according to our requirements. $\mathcal{T}$ denotes a set of triples, P a set of properties and p, p_1, p_2 denote properties. In detail,

• fco_1 suits to the normal case, i.e. to properties that are functional, e.g. the date that

each product was delivered, the branch where each invoice took place, and its value can be numerical or categorical.
• fco_2 and fco_3 are related to issues that concern missing and multi-valued properties.
• fco_4 can be used for transforming a multi-valued property to a set of single-valued features, e.g. one boolean feature for each nationality, that a founder may have.
• fco_5 and $fco6$ relate to the degree of an entity.
• fco_7 to $fco9$ investigate paths in an RDF graph, e.g. whether at least one founder of a brand is "French".

Consequently, the aforementioned cases could be handled by transforming our data set properly, using the feature operators already described. Specifically, regarding case (i), i.e. properties with empty values, the transformations 2 and 3 of Table 2 could be applied for turning such properties into integers. Concerning case (ii), the transformations 7 to 9 could be used for specifying a path (a sequence of properties $p_1, p_2, ..., p_n$ etc.) and handle it as an individual property p. Finally, relating the case (iii), the transformations 2 to 4 could be used for inspecting the existence of any multi-valued properties and the conversion of them to single-valued.

Application of HIFUN in Special Cases. Some of the denoted special cases can be handled in HIFUN without having previously transformed our data.

For example, regarding the case (ii), where the user may be interested in a path P of an RDF graph, the operator of the composition ($\circ$) can be used for expressing such a path in HIFUN i.e. by combining all the attributes of P.

Example 6. Suppose that we would like to find the quantity of sold products per month, that belong to brands of French founders. The query in HIFUN would be defined as, $((m \circ d)/E, q, SUM)$, where $E = \{x | x \in D \wedge (n \circ f \circ br \circ p)(x)) = French\}$.

Example 7. Suppose now that, we would like to find the sum of total sales of the month "September" grouped by the nationality of the founders. Such a query would be expressed in HIFUN as, $((n \circ f \circ br \circ p)/E, q, SUM)$ where $E = \{x | x \in D \wedge (m \circ d)(x)) = September\}$.

As regards case (iii), we could apply HIFUN to multi-valued attributes, if we used "$\in$" instead of "$=$" in the restriction. Note that, a multi-valued attribute should always correspond to a terminal node of a HIFUN context.

Example 8. Suppose that, the attribute of "nationality" is a multi-valued property i.e. a person can have more than one nationalities, as shown in the example of Figure 1 and we would like to find the quantity of sold products per month, that belong to brands of French founders. Then, this query would be defined in HIFUN as,

$((m \circ d)/E, q, SUM)$, where $E = \{x | x \in D \wedge (n \circ f \circ br \circ p)(x)) \in French\}$.

Table 2. Feature Creation Operators

id	Operator defining f_i	Type	$f_i(e)$
Plain selection of one property			
1	p.value	num/categ	$f_i(e) = \{ v \mid (e,p,v) \in \mathcal{T}\}$
For missing values and multi-valued properties			
2	p.exists	boolean	$f_i(e) = 1$ if (e,p,o) or $(o,p,e) \in \mathcal{T}$, otherwise $f_i(e) = 0$
3	p.count	int	$f_i(e) = \lvert\{ v \mid (e,p,v) \in \mathcal{T}\}\rvert$
For multi-valued properties			
4	p.values.AsFeatures	boolean	for each $v \in \{ v \mid (e,p,v) \in \mathcal{T}\}$ we get the feature $f_{iv}(e) = 1$ if (e,p,v) or $(v,p,e) \in \mathcal{T}$, otherwise $f_{iv}(e) = 0$
General ones			
5	degree	double	$f_i(e) = \lvert\{(s,p,o) \in \mathcal{T} \mid s = e \text{ or } o = e\}\rvert$
6	average degree	double	$f_i(e) = \frac{\lvert triples(C)\rvert}{\lvert C\rvert}$ s.t. $C = \{ c \mid (e,p,c) \in \mathcal{T}\}$ and $triples(C) = \{(s,p,o) \in \mathcal{T} \mid s \in C \text{ or } o \in C\}$
Indicative extensions for paths			
7	p1.p2.exists	boolean	$f_i(e) = 1$ if $\exists\ o2$ s.t. $\{(e,p1,o1),(o1,p2,o2)\} \subseteq \mathcal{T}$
8	p1.p2.count	int	$f_i(e) = \lvert\{ o2 \mid (e,p1,o1),(o1,p2,o2) \in \mathcal{T}\}\rvert$
9	p1.p2.value.maxFreq	num/categ	$f_i(e) =$ most frequent $o2$ in $\{ o2 \mid (e,p1,o1),(o1,p2,o2) \in \mathcal{T}\}$

6.2 Implementation Issues

As regards the specification of the analysis context, the cases of M_{plain} and M_{QLview} (as well as the case of CSV files) can be supported by tools like Facetize [13], that also offer cleaning functionality, as well as the ability to organize the values of some dimensions, hierarchically. The case of $M_{QLobjects}$ can be supported by tools like LODSyndesisML [15], that reads a list of URIs and enrich them with attributes, by exploiting several data sets published as Linked Data. The output of the above tools can be straightforwardly converted to the RDF Data Cube format.

In our work, we have currently developed a tool, called HIFUN$_{RDF}$ which applies the HIFUN query language to RDF data. For the time being, only data in the RDF Data Cube format is supported. In order to execute a query, the user should define: 1) the grouping function, 2) the measuring function, 3) the aggregate operation, and optionally, 4) set restrictions to the grouping, the measuring functions or to the final results. The inserted values are used to construct the corresponding HIFUN query, which is subsequently converted to the respective SPARQL. The latter is executed on the triple store OpenLink Virtuoso[9] (where the input data has been uploaded) and the returned results are saved in a .csv file, in the form of - $var_1, var_2, ..., var_i$, TOTALS. Indicative flows between HIFUN$_{RDF}$ and other tools are illustrated in Figure 3.

Now, we are in the process of extending our application, to support analytics over RDF data in general (and not only to data expressed in the RDF Data Cube format). We are designing an interface, that will let the user specify the analysis context (as well as the required transformations) and formulate the HIFUN query, interactively. After

[9] https://virtuoso.openlinksw.com/

that step, we will focus on the visualization of the results, probably by extending the visualization method presented in [21][10].

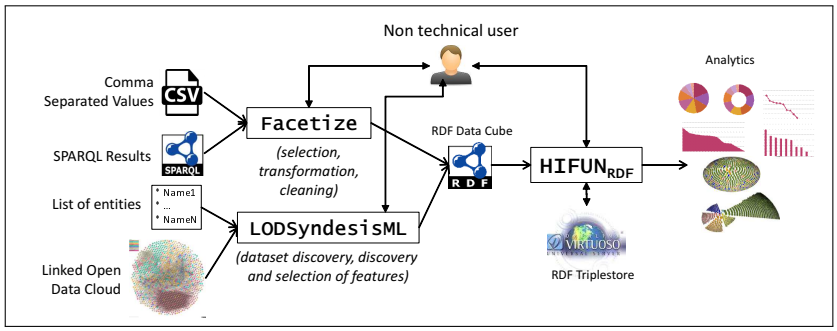

Fig. 3. A few indicative workflows involving HIFUN$_{RDF}$.

7 Concluding Remarks

In this paper, we examined the basics for applying HIFUN over RDF data. We described methods for defining the context of analysis over an RDF graph and we showed how HIFUN queries can be translated to SPARQL. Moreover, we described, in brief, a first implementation of the approach that, for the time being, can be applied to data expressed in the RDF Data Cube format. In the future, we plan to investigate, more complex queries and OLAP-style operations over RDF graphs including the interplay with hierarchies and inference. Besides, we plan to design a graphical user interface appropriate for applying HIFUN to RDF Data, and to further work on the visualization part of the analytical results.

References

1. A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J.-N. Mazón, F. Naumann, T. Pedersen, S. B. Rizzi, J. Trujillo, P. Vassiliadis, et al. Fusion cubes: towards self-service business intelligence. *International Journal of Data Warehousing and Mining (IJDWM)*, 9, 2013.
2. G. Antoniou and F. Van Harmelen. *A semantic web primer*. MIT press, 2004.
3. B. Benatallah, H. R. Motahari-Nezhad, et al. Scalable graph-based olap analytics over process execution data. *Distributed and Parallel Databases*, 34, 2016.
4. D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş. RDF analytics: lenses over semantic graphs. In *Proceedings of the 23rd international conference on World wide web*, 2014.
5. L. Etcheverry and A. A. Vaisman. Enhancing OLAP analysis with web cubes. In *Extended Semantic Web Conference*, 2012.
6. L. Etcheverry and A. A. Vaisman. QB4OLAP: a new vocabulary for OLAP cubes on the semantic web. In *Proceedings of the Third International Conference on Consuming Linked Data*, 2012.
7. L. Etcheverry and A. A. Vaisman. Querying semantic web data cubes. In *AMW*, 2016.
8. L. Etcheverry and A. A. Vaisman. Efficient analytical queries on semantic web data cubes. *Journal on Data Semantics*, 6, 2017.

[10] http://www.ics.forth.gr/isl/3DLod/

9. H. Inoue, T. Amagasa, and H. Kitagawa. An ETL framework for online analytical processing of linked open data. In *International Conference on Web-Age Information Management*, 2013.
10. A. Isaac and B. Haslhofer. Europeana linked open data–data. europeana. eu. *Semantic Web*, 4, 2013.
11. B. Kämpgen and A. Harth. Transforming statistical linked data for use in OLAP systems. In *Proceedings of the 7th international conference on Semantic systems*, 2011.
12. B. Kämpgen, S. O'Riain, and A. Harth. Interacting with statistical linked data via OLAP operations. In *Extended Semantic Web Conference*, 2012.
13. A. Kokolaki and Y. Tzitzikas. Facetize: An interactive tool for cleaning and transforming datasets for facilitating exploratory search. *arXiv preprint arXiv:1812.10734*, 2018.
14. M. Mountantonakis and Y. Tzitzikas. On measuring the lattice of commonalities among several linked datasets. *Proceedings of the VLDB Endowment*, 9, 2016.
15. M. Mountantonakis and Y. Tzitzikas. How linked data can aid machine learning-based tasks. In *International Conference on Theory and Practice of Digital Libraries*, 2017.
16. M. Mountantonakis and Y. Tzitzikas. LODsyndesis: Global scale knowledge services. *Heritage*, 1, 2018.
17. M. Mountantonakis and Y. Tzitzikas. Scalable methods for measuring the connectivity and quality of large numbers of linked datasets. *Journal of Data and Information Quality (JDIQ)*, 9, 2018.
18. M. Mountantonakis and Y. Tzitzikas. Large scale semantic integration of linked data: A survey. *ACM Computing Surveys (CSUR)*, 52, 2019.
19. V. Nebot and R. Berlanga. Building data warehouses with semantic web data. *Decision Support Systems*, 52, 2012.
20. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008.
21. M.-E. Papadaki, P. Papadakos, M. Mountantonakis, and Y. Tzitzikas. An interactive 3D visualization for the LOD cloud. In *EDBT/ICDT Workshops*, 2018.
22. N. Spyratos. A functional model for data analysis. In *International Conference on Flexible Query Answering Systems*, 2006.
23. N. Spyratos and T. Sugibuchi. HIFUN-a high level functional query language for big data analytics. *Journal of Intelligent Information Systems*, 51, 2018.
24. N. Spyratos and T. Sugibuchi. Data exploration in the hifun language. In *International Conference on Flexible Query Answering Systems*, 2019.
25. A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive-a petabyte scale data warehouse using hadoop. In *2010 IEEE 26th international conference on data engineering (ICDE 2010)*, 2010.
26. Y. Tzitzikas, C. Allocca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating heterogeneous and distributed information about marine species through a top level ontology. In *Research conference on metadata and semantic research*, 2013.
27. K. Wang, G. Xu, Z. Su, and Y. D. Liu. GraphQ: Graph query processing with abstraction refinement—scalable and programmable analytics over very large graphs on a single {PC}. In *2015 Annual Technical Conference 15)*, 2015.
28. B. Zapilko and B. Mathiak. Performing statistical methods on linked data. In *International conference on dublin core and metadata applications*, 2011.
29. P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and olap multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.