WaRG: Warehousing RDF Graphs

Dario Colazzo, Tushar I. Ghosh, François Goasdoué, Ioana Manolescu, Alexandra Roatis

WaRG: Warehousing RDF Graphs*

Dario Colazzo‡ Tushar I. Ghosh§,⋆ François Goasdoué†
Ioana Manolescu§,⋆ Alexandra Roatiş⋆,§
§Inria Saclay, France ⋆Université Paris-Sud, France ‡Université Paris Dauphine, France †Université Rennes 1, France
firstname.lastname §@inria.fr ⋆@lri.fr ‡@dauphine.fr †@univ-rennes1.fr

ABSTRACT

We propose to demonstrate *WaRG*, a system for performing warehouse-style analytics on RDF graphs. To our knowledge, our framework is the first to keep the warehousing process purely in the RDF format and take advantage of the heterogeneity and semantics inherent to this model.

1. INTRODUCTION

Databases of *facts*, each characterized by multiple *dimensions*, whose values are recorded in *measures*, are at the core of *multidimensional data warehouses* (DWs in short) [13]. The facts can then be analyzed by means of *aggregating* the measures, e.g., *"what is the average sale price of item I every month in every store?"*

Data warehouses are typically built to *analyze (some aspects of) an enterprise's business processes*. First, analysts *choose* the data sources of interest for a given class business questions. Next, they describe the facts, dimensions, and measures to be analyzed. Then, for each relevant business question, an *analytical query* (commonly known as a *cube*) is formulated, by (*i*) classifying facts along a set of dimensions and (*ii*) reporting the aggregated values of their measures. Data warehousing has attracted enormous interest, both from practitioners, e.g., [14] and from the research community, e.g., [8, 12]; warehousing tools are now part of major relational database servers. Relational data warehousing can be thus considered a pretty mature area.

Recent years witnessed a steady and important interest in Semantic Web data, represented within the W3C's Resource Description Framework (or RDF, in short) [18]. The RDF model allows describing *resources*, by specifying the *values* of their *properties*. The RDF language is used to *export, share, and collaboratively author data*, in many settings. It serves as a *metadata language*, e.g., to describe cultural artifacts in large digital libraries, and to encode protein sequence data (e.g., the Uniprot data set). RDF is a natural target for representing heterogeneous facts contributed by millions of Wikipedia users, gathered within the DBpedia data source, as well as for the *Linked Open Data* effort, aiming at connecting and sharing collectively produced data and knowledge.

The current popularity of RDF raises interest in *models and tools for RDF data analytics*. Consider applications seeking to *harvest, aggregate and analyze user data* from various sources (such as social networks, blog posts, comments on public Web sites). The data is *heterogeneous* (facts about the user age, gender, region, endorsements) and *graph-structured* (describes relationships between users, places, companies). It comes from multiple sources and may have attached *semantics*, based on ontologies for which RDF is an ideal format. Analyzing such data with warehouse-style tools enables understanding user behavior and needs, and proposing services better suited to the users.

Despite the perceived need, there is currently *no satisfactory conceptual and practical solution for large-scale RDF analytics*. Relational DW tools are not suitable, since loading RDF data in a relational analytical schema may lead to facts with unfilled or multiply-defined dimensions or measures; the latter does not comply with the relational multidimensional setting and DW tools. Further, to fully exploit RDF graphs, *the heterogeneity and rich semantics of RDF data should be preserved* through the warehouse processing chain and up to the analytical queries. In particular, RDF analytical queries should be allowed to *jointly query the schema and the data*, e.g., ask for *most frequently specified properties of a CollegeStudent*, or the *top-three most represented groups of Inhabitants*. Changes to the underlying database (e.g., a new subclass of *Inhabitant*) should not cause the warehouse schema to be re-designed; instead, the new resources (and their properties) should propagate smoothly to the analysis schema and cubes.

We propose to demonstrate WARG, a tool embodying our novel approach for *warehouse-style analytics on Semantic Web data*. WARG supports the first all-RDF formal model for data warehousing. Its main features are:

- a formal model for *analytical schemas*, capturing *the data of interest for a given analysis* of an RDF data set. Importantly, an analytical schema instance is an RDF graph itself, and as such, it preserves the heterogeneity, semantics, and flexible mix of schema and data present in the RDF model;
- *analytical queries*, RDF counter-parts of the relational analytical cubes, and supporting typical analytical operations (slice, dice, drill-down etc.)

In the sequel, Section 2 presents our analytical RDF model, the WARG platform architecture is discussed in Section 3 and we describe use scenarios in Section 4. Finally, we discuss related work in Section 5.

2. DATA MODEL

An **RDF graph** is a set of *triples* of the form `s p o`. A triple states that its *subject* `s` has the *property* `p`, and the value of that property is the *object* `o`. We only consider well-formed triples, $\mathtt{s} \in U \cup B$, $\mathtt{p} \in U$ and $\mathtt{o} \in U \cup B \cup L$, where U is a set of URIs, L of literals (constants), and B are blank nodes (unknown URIs or literals), such that U, B and L are pairwise disjoint.

*This work was performed during the authors' affiliation with Université Paris-Sud and Inria Saclay, France.

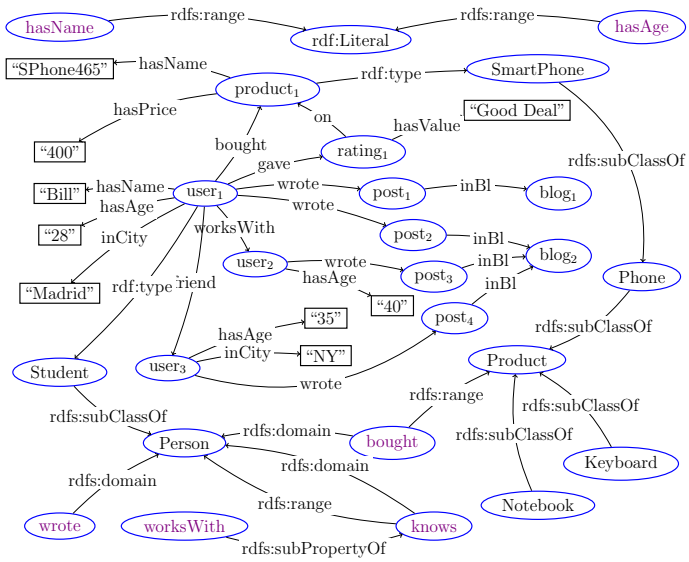

Figure 1: Running example: RDF/S graph.

The RDF standard [18] provides a set of built-in classes and properties, part of the rdf: and rdfs: pre-defined namespaces. We use these namespaces exactly for these classes and properties, e.g., rdf:type which specifies the class(es) to which a resource belongs.

Triples are used to describe resources, i.e., to express class $\mathtt{s}$ rdf:type $\mathtt{o}$ (unary relation – $\mathtt{o}(\mathtt{s})$) and property $\mathtt{s\ p\ o}$ (binary relation – $\mathtt{p}(\mathtt{s},\mathtt{o})$) assertions.

A valuable feature of RDF is **RDF Schema (RDFS)** which enriches the descriptions in RDF graphs, by declaring *semantic constraints* (interpreted under the open-world assumption) between the classes and the properties used in those graphs. Such constraints can define sub-class and sub-property relationship, as well as type the domain and range of properties.

Example 1 (Running Example). *We consider an RDF graph comprising information about users and products (depicted in Figure 1). The graph features a resource* user_1 *whose name is Bill and whose age is 28. Bill works with* user_2 *and is a friend of* user_3*; moreover, Bill is an active contributor to two different blogs, one shared with his co-worker* user_2*. Bill bought a* SPhone465 *and rated it online. The database also contains information about the RDFS schema for this graph.*

We consider the well-known subset of SPARQL consisting of (unions of) ***basic graph pattern* (BGP) queries**, also known as SPARQL conjunctive queries. A BGP is a set of *triple patterns*, or triples in short. Each triple has a subject, property and object. Subjects and properties can be URIs, blank nodes or variables; objects can also be literals.

A boolean BGP query is of the form $\mathtt{ASK\ WHERE}\ \{t_1,\ldots,t_\alpha\}$, while a non-boolean BGP query is of the form $\mathtt{SELECT}\ \bar{x}\ \mathtt{WHERE}\ \{t_1,\ldots,t_\alpha\}$, where $\{t_1,\ldots,t_\alpha\}$ is a BGP; the variables $\bar{x}$ are a subset of the variables occurring in $t_1,\ldots,t_\alpha$. We use the conjunctive query notation $q(\bar{x})$:- $t_1,\ldots,t_\alpha$ for both ASK and SELECT queries (for boolean queries, $\bar{x}$ is empty).

2.1 RDF graph analysis

We model a schema for RDF graph analysis, called *analytical schema*, as a labeled directed graph. Each node represents a set of facts that may be analyzed, while the nodes reachable from its outgoing edges represent the possible dimensions and measures according to which it may be analyzed. Nodes (respectively edges) correspond to class (respectively property) assertions, defined using BGP queries.

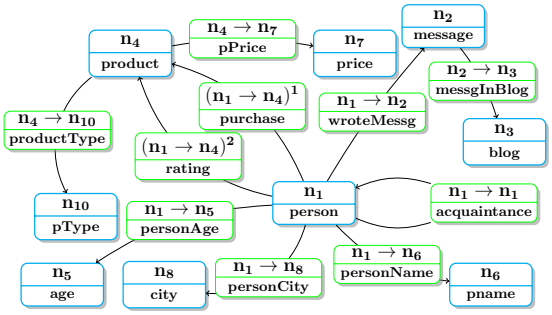

Figure 2: Sample Analytical Schema (*AnS*).

Definition 1 (Analytical Schema). *An* analytical schema *(AnS) is a labeled directed graph* $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$ *in which:*

- $\mathcal{N}$ *is the set of nodes;*
- $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ *is the set of directed edges;*
- $\lambda : \mathcal{N} \cup \mathcal{E} \rightarrow URI$ *is an injective labeling function, mapping nodes and edges to URIs;*
- $\delta : \mathcal{N} \cup \mathcal{E} \rightarrow \mathcal{Q}$ *is a function assigning to each node* $n \in \mathcal{N}$ *a unary BGP query* $\delta(n) = q(x)$*, and to every edge* $n \rightarrow n' \in \mathcal{E}$ *a binary BGP query* $\delta(n \rightarrow n') = q(x, y)$.

We assume that through the λ function, each node in the *AnS* defines a new class, while each edge defines a new property.Since λ is injective, the URIs assigned to the classes and properties defined by an *AnS*' nodes and edges are pairwise disjoint. Just as an analytical schema defines the data available to the analyst in a typical relational data warehouse scenario, in our framework, *the classes and properties created by AnS (and labeled by* λ*) are the only ones visible to further RDF analytics.* Example 2 introduces an *AnS* for our running example.

Example 2 (Analytical Schema). *Figure 2 depicts an AnS* $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$*, for analyzing people and products. For readability, the node and edge labels appear in the figure, while (some of) the BGP queries definind nodes and edges are provided in Table 1; the others are very similar. In Figure 2, a person* (n_1) *may have written messages* $(n_1 \rightarrow n_2)$ *which appear on some blog* $(n_2 \rightarrow n_3)$*. A person may also have bought products* $(n_1 \rightarrow n_4)^1$ *or may have commented on them* $(n_1 \rightarrow n_4)^2$*. The semantics for the rest of the schema can be easily derived from the Figure.*

The nodes and edges in Figure 2 are those considered of interest for the particular sample data analysis scenario. In other words, the AnS offer a perspective (or lens) through which the interesting subset of RDF data will be analyzed. This is formalized as follows:

Definition 2 (Instance of an AnS). *Let* $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$ *be an analytical schema and* $\mathtt{G}$ *an RDF graph. The* instance of $\mathcal{S}$ w.r.t. $\mathtt{G}$ *is the RDF graph* $\mathcal{I}(\mathcal{S}, \mathtt{G})$ *(or simply* $\mathcal{I}$*) defined as:*

$$\bigcup_{n \in \mathcal{N}} \{s \text{ rdf:type } \lambda(n) \mid s \in q(\mathtt{G}) \wedge q = \delta(n)\}$$
$$\cup$$
$$\bigcup_{n_1 \rightarrow n_2 \in \mathcal{E}} \{s\ \lambda(n_1 \rightarrow n_2)\ o \mid s, o \in q(\mathtt{G}) \wedge q = \delta(n_1 \rightarrow n_2)\}.$$

node n	$\lambda(n)$	$\delta(n)$
n_1	*person*	$q(x)$:- x rdf:type Person
n_2	*message*	$q(x)$:- y wrote x, x inBl b , b rdf:type Blog
n_4	*product*	$q(x)$:- x rdf:type Product
n_5	*age*	$q(x)$:- y hasAge x
n_6	*pname*	$q(x)$:- y hasName x
n_{10}	*pType*	$q(x)$:- x rdfs:subClassOf Product

edge $n \to n'$	$\lambda(n \to n')$	$\delta(n \to n')$
$n_1 \to n_1$	*acquaintance*	$q(x,y)$:- x knows y
$n_1 \to n_4$	*purchase*	$q(x,y)$:- x bought y
$n_1 \to n_5$	*personAge*	$q(x,y)$:- x rdf:type Person, x hasAge y
$n_2 \to n_3$	*messgInBlog*	$q(x,y)$:- x rdf:type Message, x inBl y
$n_4 \to n_{10}$	*productType*	$q(x,y)$:- x rdf:type Product, x rdf:type y

Table 1: Labels and queries of some nodes and edges of the analytical schema (AnS) shown in Figure 2.

Example 3 (Analytical Schema Instance). *Table 2 shows part of the instance of the analytical schema introduced in Example 2. For each triple, we indicate at right the node (or edge) of the AnS which has produced it.*

$$\mathcal{I}(\mathcal{S}, \mathtt{G}') = \begin{array}{ll} \{\text{user}_1 \text{ rdf:type person} & n_1 \\ \text{user}_2 \text{ rdf:type person} & n_1 \\ \text{user}_3 \text{ rdf:type person} & n_1 \\ \text{user}_1 \text{ acquaintance user}_2 & n_1 \to n_1 \\ \text{user}_1 \text{ acquaintance user}_3 & n_1 \to n_1 \\ \text{post}_1 \text{ messgInBlog blog}_1 & n_2 \to n_3 \\ \text{product}_1 \text{ rdf:type product} & n_4 \\ \text{user}_1 \text{ personAge “28”} & n_1 \to n_5 \\ \text{user}_1 \text{ personName “Bill”} & n_1 \to n_6 \\ \text{Notebook rdf:type pType} & n_{10} \\ \text{SmartPhone rdf:type pType} & n_{10} \\ \text{product}_1 \text{ pPrice “\$400”} & n_4 \to n_7 \quad \ldots\} \end{array}$$

Table 2: Partial instance of the AnS in Figure 2.

Crucial to our ability to handle RDF heterogeneity is *the disjunctive semantics of an AnS*, materialized by the two levels of $\cup$ operators in Definition 2. *Each node and each edge of an AnS populates $\mathcal{I}$ through an independent query*, and the resulting triples are simply combined through unions. This has two benefits: (i) significant flexibility when designing the AnS, and (ii) the ability to build a (heterogeneous data) warehouse on top of a heterogeneous RDF graph. Consider for instance the three users in the original graph $\mathtt{G}$ (Figure 1) and their properties: user_1, user_2 and user_3 are part of the person class in our AnS instance $\mathcal{I}$ (through n_1's query), although user_2 and user_3 lacks a name. However, those user properties present in the original graph, are reflected by the AnS edges $n_1 \to n_2$, $n_1 \to n_8$ etc. Thus, the inherent heterogeneity of RDF graphs is accepted in the input and present in the output of an AnS.

2.2 Analytical queries

Data warehouse analysis summarizes facts according to relevant criteria into *cubes*, which analyze facts characterized by some *dimensions*, using a *measure*. We consider a set of dimensions $d_1, d_2, \ldots, d_n$, such that each dimension d_i may range over the value set $\{d_i^1, \ldots, d_i^{n_i}\}$; the Cartesian product of all dimensions $d_1 \times \cdots \times d_n$ defines a multidimensional space $\mathcal{M}$. To each tuple t in this multidimensional space $\mathcal{M}$ corresponds a *subset* $\mathcal{F}_t$ of the analyzed facts, having for each dimension d_i, $1 \leq i \leq n$, the value of t along d_i.

A *measure* is a set of values[1] characterizing each analyzed fact f. The facts in $\mathcal{F}_t$ are summarized by the *cube cell* $\mathcal{M}[t]$ by the result of an *aggregation* function $\oplus$ (e.g., count, sum, average, etc.) applied to the union of the measures of the $\mathcal{F}_t$ facts: $\mathcal{M}[t] = \oplus(\bigcup_{f \in \mathcal{F}_t} v_f)$.

An *analytical query* consists of two (rooted) queries and an aggregation function. The first query, known as a *classifier* in traditional data warehouse settings, defines the *dimensions* $d_1, d_2, \ldots, d_n$ according to which the facts matching the query root will be analyzed. The second query defines the *measure* according to which these facts will be summarized. Finally, the aggregation function is used for *summarizing* the analyzed facts.

Definition 3 (Analytical Query). *Given an analytical schema $\mathcal{S} = \langle \mathcal{N}, \mathcal{E}, \lambda, \delta \rangle$, an* analytical query *(AnQ) rooted in the node $r \in \mathcal{N}$ is a triple*

$$Q = \langle c(x, d_1, \ldots, d_n), m(x, v), \oplus \rangle$$

where:

- *$c(x, d_1, \ldots, d_n)$ is a query rooted in the node r_c of its graph G_c. This query is called the* classifier *of x w.r.t. the n dimensions $d_1, \ldots, d_n$.*
- *$m(x, v)$ is a query rooted in the node r_m of its graph G_m. This query is called the* measure *of x.*
- *$\oplus$ is a* function *computing a value (a literal) from an input set of values. This function is called the* aggregator *for the measure of x w.r.t. its classifier.*
- *For every homomorphism h_c from the classifier to $\mathcal{S}$ and every homomorphism h_m from the measure to $\mathcal{S}$, $h_c(r_c) = h_m(r_m) = r$ holds.*

The last item above guarantees the “well-formedness” of the analytical query, that is: the facts for which we aggregate the measure, are indeed those classified along the desired dimensions. It is worth noticing that, from a practical viewpoint, this condition can be easily and naturally guaranteed by giving explicitly in the classifier and the measure either the type of the facts to analyze, using x rdf:type $\lambda(r)$, or a property describing those facts, using x $\lambda(r \to n)$ $\mathtt{o}$ with $r \to n \in \mathcal{E}$. As a result, since the labels are unique in an AnS (its labeling function is injective), every homomorphism from the classifier (respectively the measure) to the AnS does map the query's root node labeled with x to the AnS's node r.

Definition 3 can be easily extended to more than one measure, by defining a set of measure queries and an associate set of aggregation functions.

Example 4 (Analytical Query). *The next query asks for the number of blogs where the user posts, classified by the user age and city:*

$$\langle c(x, a, c), m(x, y), count \rangle$$

where the classifier and measure queries are defined by:

$$c(x, a, c) \text{:- } x \; personAge \; a, x \; personCity \; c$$

$$m(x, b) \text{:- } x \; wroteMessg \; o, o \; messgInBlog \; b$$

[1]It is a set and not a value, due to the structural heterogeneity of the AnS instance, which is also an RDF graph: each fact may have zero or more values for a given measure.

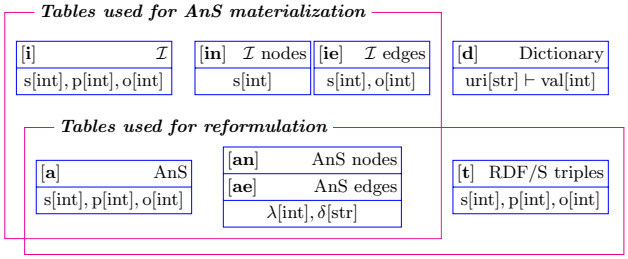

Figure 3: Data layout of the RDF warehouse.

The semantics of an analytical query is:

DEFINITION 4 (ANSWER SET OF AN ANQ). *Let $\mathcal{I}$ be the instance of an AnS with respect to some RDF graph. Let $Q = \langle c(x, d_1, \ldots, d_n), m(x, v), \oplus \rangle$ be an AnQ against $\mathcal{I}$. The* answer set *of Q against $\mathcal{I}$, denoted $ans(Q, \mathcal{I})$, is:*

$$ans(Q,\mathcal{I}) = \{\langle d_1^j, \ldots, d_n^j, \oplus(q^j(\mathcal{I}))\rangle \mid \langle x^j, d_1^j, \ldots, d_n^j\rangle \in c(\mathcal{I}) \text{ and } q^j \text{ is defined as } q^j(v) \text{:- } m(x^j, v)\}$$

assuming that the type of each value returned by $q^j(\mathcal{I})$ belongs (or can be converted by the SPARQL rules [19]) to the input type of the aggregator $\oplus$. Otherwise, the answer set is undefined.

The analytical query returns each tuple of dimension values from the answer of the classifier, together with the aggregated result of the measure query. The answer set of an *AnQ* can thus be represented as a cube of n dimensions, holding in each cube cell the corresponding aggregate measure.

EXAMPLE 5 (ANALYTICAL QUERY ANSWER). *Consider the query in Example 4, over the analytical schema in Figure 2. Some triples from the instance of this analytical schema were shown in Table 2. The classifier query returns:*

$$\langle \text{user}_1, 28, \text{“}Madrid\text{”}\rangle, \langle \text{user}_3, 35, \text{“}NY\text{”}\rangle$$

while the measure query returns:

$$\langle \text{user}_1, \text{blog}_1\rangle, \langle \text{user}_1, \text{blog}_2\rangle, \langle \text{user}_3, \text{blog}_2\rangle$$

Aggregating the blogs among the classification dimensions leads to the AnQ answer:

$$\langle 28, \text{“}Madrid\text{”}, 2\rangle, \langle 35, \text{”}NY\text{”}, 1\rangle$$

3. ARCHITECTURE

We deploy our approach using kdb+ v3.0 (64 bits) [1], an in-memory column database used in decision support analytics applications. kdb+ provides arrays (tables), which can be manipulated through the q interpreted programming language. We store in kdb+ the RDF graph G, the AnS definitions and AnS instance, when we choose to materialize it. We also translate BGP queries into q queries that kdb+ interprets.

Figure 3 outlines our data layout. As customary in RDF databases, we *dictionary-encode* all URIs into integers; these mappings are stored within the [**d**] table. The graph saturation is stored in [**t**]. Analytical schema *definitions* are stored as follows. The [**a**] table stores the λ function encoded as triples $\lambda(n_1)$ $\lambda(n_1 \rightarrow n_2)$ $\lambda(n_2)$ for all the nodes n_1, n_2 and edges $n_1 \rightarrow n_2 \in$ AnS. The [**an**] table stores the δ function for AnS nodes, and similarly [**ae**] stores the δ function for AnS edges. Finally, we use either the [**i**] table to store the AnS instance $\mathcal{I}$, or several [**in**] and [**ie**] tables when considering partitioned tables.

kdb+ stores each column in a table independently; search by value inside a column is fast, supported by an in-memory index, and *self-joins* are also fast. Joins on different columns (e.g., s=o) are not advantaged.

The user-facing GUI is implemented in Java 1.6, using Prefuse [9], a Java-based framework for visualizing and interacting with data. The interface retrieves data from kdb+ and stores it into an internal data structure provided by Prefuse, who then renders the graphs.

4. SCENARIO

The demo will rely on a subset of DBPedia.

Visualizing analytical schemas Figure 4 shows an example of a small analytical schema shown in our GUI. Our visualization tool allows selecting a node, which results in highlighting its connected nodes within the schema. This feature is useful when building an analytical query, since such a query must be a connected component from the schema.

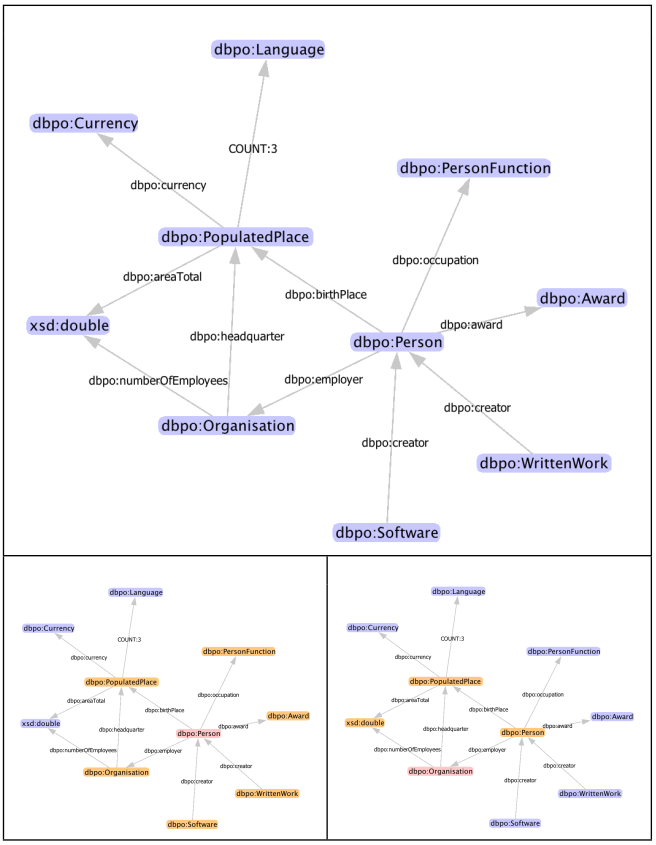

Figure 4: Analytical schema view.

Upon selecting a node additional information regarding the node itself will be displayed. Such information consists of the short (*namespace:term*), the full URI, and the query defining the node. Also if the node has underlying data, this information is displayed too. For instance when selecting the node dbpo:Person the user will see the information illustrated in Table 3.

The software is intended to handle large analytical schemas, possibly containing multiple edges between two nodes. In order to increase visibility, our visualization tool groups multiple edges connecting two nodes: a single edge labeled with a value counting all the distinct edges is displayed (see for

Short URI	**dbpo:Person**
Full URI	http://dbpedia.org/ontology/Person
Query	q(?x):- ?x rdf:type dbpo:Person
Is empty	FALSE
Data:	dbpr : Aristotle dbpr : Neil_Armstrong dbpr : Tim_Berners−Lee ...
Available edges	**dbpo:language** dbpo:officialLanguage dbpo:regionalLanguage
Short URI	**dbpo:language**
Full URI	http://dbpedia.org/ontology/language
Query	q(?x, ?y):- ?x dbpo:language ?y
Is empty	FALSE
Data:	dbpr : France–dbpr : French_language dbpr : Germany–dbpr : German_language dbpr : United_Kingdom–dbpr : English_language ...

Table 3: Node (top) and edge (bottom) information.

example the edge between dbpo:PopulatedPlace and dbpo: Language in Figure 4). Selecting an edge in a graph leads to obtaining information about the edge itself, and the same holds for edges. When a multiple edge is selected, all the edges are displayed (Table 3).

Evaluating queries Users may build and evaluate analytical queries over a defined schema. Building an analytical query consists of defining each of its three components: classifier query, measure query and aggregation function. The classifier query is built starting from a selected node. The system then suggests (highlights) the nodes connected to it. As more nodes are selected, their neighbors are highlighted as potential options. Only nodes that have underlying data are considered at any time, because the intention is to build queries that can have answers. A selected node is interpreted as a variable in the query. The user may also add constants by selecting values from the node's underlying data. The process goes on until the classifier is complete and the dimensions (distinguished variables) are selected. The measure query is built similarly. Finally, an aggregation function is chosen from a drop-down list of functions applicable to the chosen measure.

5. RELATED WORKS

Relational DWs have been thoroughly studied [13], and many efficient tools exist. In the context of the Web, "Web Data Warehouses" have been proposed [3, 16], however, their focus is on distribution, data exchange or enrichment, rather than RDF and analytics.

Some works [6, 20] propose new vocabularies for describing relational cubes in RDF; in contrast, WARG is a generic approach for defining and using pure RDF cubes. Data management techniques for RDF have been a hot topic, with recent works tackling storage [2, 17], query processing and updates [15], materialized views [7], and Map-Reduce based RDF processing [10, 11]; Oracle 11g provides a "Semantic Graph" extension etc. Ours is the first work to formalize RDF analytics and propose analytic schema and queries with well-defined semantics.

Recent works [5, 21] have focused on graph warehousing. While we share certain general principles, these approachs are not meant for *heterogeneous* graphs and thus cannot handle multi-valued attributes. More importantly, they do not consider the semantically rich RDF model.

The separation between grouping and aggregation present in our analytical queries is similar to the MD-join operator introduced in [4] for relational DWs.

Finally, the recent SPARQL 1.1 language [19] includes group-by and aggregation constructs closely inspired from SQL. While this bears similarities to our AnQs, our proposal is the first to outline a full-RDF data analytics chain, including a general framework for specifying analytical schemas which are RDF graphs themselves, and therefore carry all the rich features of the data model, most importantly structure heterogeneity and rich semantics.

6. REFERENCES

[1] [kx] white paper. kx.com/papers/KdbPLUS_Whitepaper-2012-1205.pdf.

[2] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, 2007.

[3] S. Abiteboul, E. Antoine, and J. Stoyanovich. Viewing the web as a distributed knowledge base. In *ICDE*, 2012.

[4] M. Akinde, D. Chatziantoniou, T. Johnson, and S. Kim. The MD-join: An operator for complex OLAP. In *ICDE*, 2001.

[5] D. Bleco and Y. Kotidis. Business intelligence on complex graph data. In *EDBT Workshops*, 2012.

[6] L. Etcheverry and A. A. Vaisman. Enhancing OLAP analysis with web cubes. In *ESWC*, 2012.

[7] F. Goasdoue, K. Karanasos, J. Leblay, and I. Manolescu. View selection in Semantic Web databases. *PVLDB*, 5(1), 2012.

[8] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[9] J. Heer, S. K. Card, and J. Landay. Prefuse: A toolkit for interactive information visualization. In *ACM Human Factors in Computing Systems (CHI)*, 2005.

[10] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL Querying of Large RDF Graphs. *PVLDB*, 4(11), 2011.

[11] M. Husain, J. McGlothlin, M. M. Masud, L. Khan, and B. M. Thuraisingham. Heuristics-Based Query Processing for Large RDF Graphs Using Cloud Computing. *IEEE Trans. on Knowl. and Data Eng.*, 2011.

[12] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Fundamentals of Data Warehouses*. Springer, 2001.

[13] C. S. Jensen, T. B. Pedersen, and C. Thomsen. *Multidimensional Databases and Data Warehousing*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[14] R. Kimball and M. Ross. *The Data Warehouse Toolkit*. Wiley, 2002. 2nd Edition.

[15] T. Neumann and G. Weikum. x-RDF-3X: Fast querying, high update rates, and consistency for RDF databases. *PVLDB*, 3(1), 2010.

[16] N. Preda, G. Kasneci, F. M. Suchanek, T. Neumann, W. Yuan, and G. Weikum. Active knowledge: dynamically enriching RDF knowledge bases by web services. In *SIGMOD*, 2010.

[17] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store support for RDF data management: not all swans are white. *PVLDB*, 1(2), 2008.

[18] W3C. Resource description framework. http://www.w3.org/RDF/.

[19] W3C. SPARQL 1.1 query language. http://www.w3.org/TR/sparql11-query/.

[20] W3C. The RDF data cube vocabulary. http://www.w3.org/TR/vocab-data-cube/, 2012.

[21] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In *SIGMOD*, 2011.