Efficient OLAP Operations For RDF Analytics

Elham Akbari-Azirani, François Goasdoué, Ioana Manolescu, Alexandra Roatis

Efficient OLAP Operations For RDF Analytics

Elham Akbari Azirani [#1], François Goasdoué [*#2], Ioana Manolescu [#*3], Alexandra Roatiş [⋆#4]

[#] *INRIA, France* [*] *U. Rennes 1, France* [⋆] *U. Paris-Sud, France*

[1] elham@lri.fr [2] fg@irisa.fr [3] ioana.manolescu@inria.fr [4] roatis@lri.fr

ABSTRACT

RDF is the leading data model for the Semantic Web, and dedicated query languages such as SPARQL 1.1, featuring in particular aggregation, allow extracting information from RDF graphs. A framework for analytical processing of RDF data was introduced in [1], where analytical schemas and analytical queries (cubes) are fully re-designed for heterogeneous, semantic-rich RDF graphs. In this novel analytical setting, we consider the following optimization problem: how to reuse the materialized result of a given RDF analytical query (cube) in order to compute the answer to another cube. We provide view-based rewriting algorithms for these cube transformations, and demonstrate experimentally their practical interest.

1. MOTIVATION AND OUTLINE

Graph-structured, semantics-rich, heterogeneous RDF data needs dedicated warehousing tools [2]. In [1], we have introduced a novel framework for RDF data analytics. At its core lies the concept of *analytical schema* (AnS, in short), which reflects a view of the data under analysis. Based on an analytical schema, *analytical queries* (AnQ, in short) can be posed over the data; they are the counterpart of the "cube" queries in the relational data warehouse scenario, but specific to our RDF context. Just like in the traditional relational data warehouse (DW) setting, AnQs can be evaluated either on a materialized AnS instance, or by composing them with the definition of the AnS; the latter is more efficient when AnQ answering only needs a small part of the AnS instance.

In this work, we focus on efficient OLAP operations for the RDF data warehousing context introduced in [1]. More specifically, we consider the question of computing the answer to an AnQ based on the (intermediary) answer of another AnQ, which has been materialized previously. While efficient techniques for such "cube-to-cube" transformations have been widely studied in the relational DW setting, new algorithms are needed in our context, where, unlike the traditional relational DW setting, a fact may have (i) multiple values along each dimension and (ii) multiple measure results.

In the sequel, to make this paper self-contained, Sections 2 recalls the core notions introduced in [1], by borrowing some of its examples; we also recall OLAP operation definitions from that work. Then, we provide novel algorithms for efficiently evaluating such operations on AnQs, briefly discuss related work, and conclude. An experimental evaluation of our techniques is described in [3].

2. BACKGROUND: RDF ANALYTICS

RDF analytical schemas and queries. RDF analytical schemas can be seen as *lenses through which data is analyzed*. An AnS is a labeled directed graph, whose nodes are *analysis classes* and

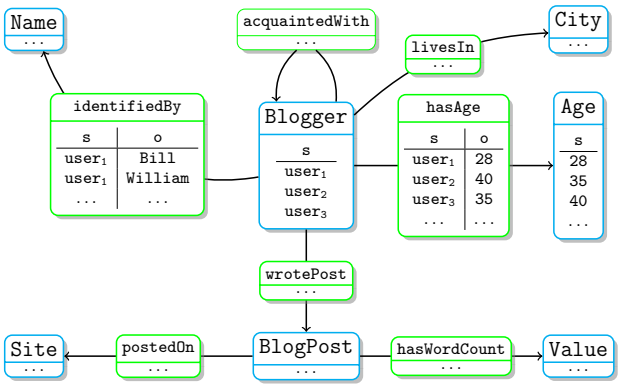


Figure 1: Sample Analytical Schema (AnS).

whose edges are *analysis properties*, deemed interesting by the data analyst for a specific analysis task. The *instance* of an AnS is built from the base data; it is an RDF graph itself, heterogeneous and semantic-rich, restructured for the needs of the analysis.

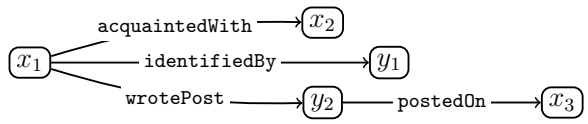
Figure 1 shows a sample AnS for analyzing bloggers and blog posts. An AnS node is defined by an *unary query*, which, evaluated over an RDF graph, returns a set of URIs. For instance, the node `Blogger` is defined by a query which (in this example) returns the URIs $\texttt{user}_1$, $\texttt{user}_2$ and $\texttt{user}_3$. The interpretation is that the AnS *defines the analysis class* `Blogger`, whose *instances* are these three users. An AnS edge is defined by a *binary query*, returning pairs of URIs from the base data. The interpretation is that for each $(\texttt{s}, \texttt{o})$ URI pair returned by the query defining the analysis property `p`, the triple `s p o` holds, i.e., `o` is a value of the property `p` of `s`. Crucial for the ability of AnSs to support analysis of heterogeneous RDF graphs is the fact that AnS *nodes and edges are defined by completely independent queries*. Thus, for instance, a user may be part of the `Blogger` instance whether or not the RDF graph comprises value(s) for the analysis properties `identifiedBy`, `livesIn` etc. of that user. Further, just like in a regular RDF graph, each blogger may have multiple values for a given analysis property. For instance, $\texttt{user}_1$ is identified both as `William` and as `Bill`.

We consider the conjunctive subset of SPARQL consisting of *basic graph pattern* (BGP) queries, denoted $q(\bar{x})$:- $t_1, \ldots, t_\alpha$, where $\{t_1, \ldots, t_\alpha\}$ are triple patterns. Unless we explicitly specify that a query has *bag* semantics, the default semantics we consider is that of *set*.

The head of q, denoted $\texttt{head}(q)$ is $q(\bar{x})$, while the body $t_1, \ldots, t_\alpha$ is denoted $\texttt{body}(q)$. We use letters in italics (possibly with subscripts) to denote variables. A *rooted BGP query* is a query q where each variable is reachable through triples from a distinguished variable, denoted *root*. For instance, the following query is a rooted BGP, whose root is x_1:

$$q(x_1, x_2, x_3) \text{ :- } x_1 \texttt{ acquaintedWith } x_2,\ x_1 \texttt{ identifiedBy } y_1,\ x_1 \texttt{ wrotePost } y_2,\ y_2 \texttt{ postedOn } x_3$$

The query's graph representation below shows that every node is reachable from the root x_1.



An analytical query consists of *two BGP queries homomorphic to the AnS and rooted in the same AnS node*, and an *aggregation* function. The first query, called a *classifier*, specifies the facts and the aggregation dimensions, while the second query, called the *measure*, returns the values that will be aggregated for each fact. The measure query has bag semantics. Example 1 presents an AnQ over the AnS defined in Figure 1.

EXAMPLE 1. (ANALYTICAL QUERY) *The query below asks for the number of sites where each blogger posts, classified by the blogger's age and city:*

$$Q \text{ :- } \langle c(x, d_{age}, d_{city}), m(x, v_{site}), \texttt{count} \rangle$$

where the classifier and measure queries are defined by:

$$c(x, d_{age}, d_{city}) \text{ :- } x \texttt{ rdf:type Blogger}, x \texttt{ hasAge } d_{age}, x \texttt{ livesIn } d_{city}$$
$$m(x, v_{site}) \text{ :- } x \texttt{ rdf:type Blogger}, x \texttt{ wrotePost } p, p \texttt{ postedOn } v_{site}$$

The semantics of an analytical query is:

DEFINITION 1. (ANSWER SET OF AN ANALYTICAL QUERY) *Let $\mathcal{I}$ be the instance of an analytical schema with respect to some RDF graph. Let Q :- $\langle c(x, d_1, \ldots, d_n), m(x, v), \oplus \rangle$ be an analytical query against $\mathcal{I}$.*

The answer set *of Q against $\mathcal{I}$, denoted $ans(Q, \mathcal{I})$, is:*

$$ans(Q, \mathcal{I}) = \{\langle d_1^j, \ldots, d_n^j, \oplus(q^j(\mathcal{I}))\rangle \mid \langle x^j, d_1^j, \ldots, d_n^j\rangle \in c(\mathcal{I}) \text{ and } q^j(\mathcal{I}) \text{ is the bag of all values } v_k^j \text{ such that } (x^j, v_k^j) \in m(\mathcal{I})\}$$

where $q^j(\mathcal{I})$ is a bag *containing all measure values v corresponding to x^j, and the operator $\oplus$ aggregates all members of this bag. If $q^j(\mathcal{I})$ is empty, the aggregated measure is undefined, and x_j does not contribute to the cube. In the following, for conciseness, we use $ans(Q)$ to denote $ans(Q, \mathcal{I})$, where $\mathcal{I}$ is considered the working instance of the analytical schema.*

In other words, the AnQ returns each tuple of dimension values from the answer of the classifier query, together with the aggregated result of the measure query for those dimension values. 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.

The counterpart of a *fact*, in this framework, is any value to which the first variable in the classifier, x above, is bound, and that has a non-empty answer for the measure query. In RDF, a resource may have zero, one or several values for a given property. Accordingly, in our framework, *a fact may have multiple values for each measure*; in particular, some of these values may be identical, yet they should not be collapsed into one. For instance, if a product is rated by 5 users, one of which rates it $\star\star\star$ while the four others rate it $\star$, the number of times each value was recorded is important. This is why we assign *bag* semantics to $q^j(\mathcal{I})$. In all other contexts, we consider BGPs with *set* semantics; this holds in particular for any classifier query $c(x, d_1, \ldots, d_n)$.

EXAMPLE 2. (ANALYTICAL QUERY ANSWER) *Consider the AnQ in Example 1, over the AnS in Figure 1. Suppose the classifier query's answer set is:*

$$\{\langle \texttt{user}_1, 28, \texttt{Madrid}\rangle, \langle \texttt{user}_3, 35, \texttt{NY}\rangle, \langle \texttt{user}_4, 35, \texttt{NY}\rangle\}$$

the measure query is evaluated for each of the three facts, leading to the intermediary results:

x^j	$\texttt{user}_1$	$\texttt{user}_3$	$\texttt{user}_4$
$q^j(\mathcal{I})$	$\{\!\{\langle \texttt{s}_1\rangle, \langle \texttt{s}_1\rangle, \langle \texttt{s}_2\rangle\}\!\}$	$\{\!\{\langle \texttt{s}_2\rangle\}\!\}$	$\{\!\{\langle \texttt{s}_3\rangle\}\!\}$

where $\{\!\{ \cdot \}\!\}$ denotes the bag constructor. Aggregating the sites among the classification dimensions leads to the AnQ answer:

$$\{\langle 28, \texttt{Madrid}, \mathbf{3}\rangle, \langle 35, \texttt{NY}, \mathbf{2}\rangle\}$$

OLAP for RDF. On-Line Analytical Processing (OLAP) [4] technologies enhance the abilities of data warehouses (so far, mostly relational) to answer multi-dimensional analytical queries. In a relational setting, the so-called "OLAP operations" allow computing a cube (the answer to an analytical query) out of another previously materialized cube.

In our data warehouse framework specifically designed for graph-structured, heterogeneous RDF data, a cube corresponds to an AnQ; for instance, the query in Example 1 models a bi-dimensional cube on the warehouse related to our sample AnS in Figure 1. Thus, we model traditional OLAP operations on cubes as AnQ rewritings, or more specifically, rewritings of *extended AnQ*s which we introduce below.

DEFINITION 2. (EXTENDED AnQ) *Let $\mathcal{S}$ be an AnS, and $d_1, \ldots, d_n$ be a set of dimensions, each ranging over a non-empty finite set V_i. Let Σ be a total function over $\{d_1, \ldots, d_n\}$ associating to each d_i, either V_i or a non-empty subset of V_i. An* extended analytical query *Q is defined by a triple:*

$$Q \text{ :- } \langle c_\Sigma(x, d_1, \ldots, d_n), m(x, v), \oplus \rangle$$

where c is a classifier and m a measure query over $\mathcal{S}$, $\oplus$ is an aggregation operator, and moreover:

$$c_\Sigma(x, d_1, \ldots, d_n) = \bigcup_{(\chi_1, \ldots, \chi_n) \in \Sigma(d_1) \times \ldots \times \Sigma(d_n)} c(x, \chi_1, \ldots, \chi_n)$$

In the above, the extended classifier $c_\Sigma(x, d_1, \ldots, d_n)$ is the set of all possible classifiers obtained by *replacing each dimension variable d_i with a value from $\Sigma(d_i)$*. We introduce Σ to constrain some classifier dimensions, i.e., to restrict the classifier result. The *semantics of an extended analytical query* is derived from the semantics of a standard AnQ (Definition 1) by replacing the tuples from $c(\mathcal{I})$ with tuples from $c_\Sigma(\mathcal{I})$. Thus, an extended analytical query can be seen as a union of a set of standard AnQs, one for each combination of values in $\Sigma(d_1), \ldots, \Sigma(d_n)$. Conversely, an analytical query corresponds to an extended analytical query where Σ only contains pairs of the form (d_i, V_i).

We define the following **RDF OLAP operations**:

A SLICE operation binds an aggregation dimension to a single value. Given an extended query Q :- $\langle c_\Sigma(x, d_1, \ldots, d_n), m(x, v), \oplus\rangle$, a SLICE operation over a dimension d_i with value v_i returns the extended query Q_{SLICE} :- $\langle c_{\Sigma'}(x, d_1, \ldots, d_n), m(x, v), \oplus\rangle$, where $\Sigma' = (\Sigma \setminus \{(d_i, \Sigma(d_i))\}) \cup \{(d_i, \{v_i\})\}$.

Similarly, a DICE operation constrains several aggregation dimensions to values from specific sets. A DICE on Q over dimensions $\{d_{i_1}, \ldots, d_{i_k}\}$ and corresponding sets of values $\{S_{i_1}, \ldots, S_{i_k}\}$, returns the query Q_{DICE} :- $\langle c_{\Sigma'}(x, d_1, \ldots, d_n), m(x, v), \oplus\rangle$, where $\Sigma' = (\Sigma \setminus \bigcup_{j=i_1}^{i_k}\{(d_j, \Sigma(d_j))\}) \cup \bigcup_{j=i_1}^{i_k}\{(d_j, S_j)\}$.

A DRILL-OUT operation on Q over dimensions $\{d_{i_1}, \ldots, d_{i_k}\}$ corresponds to removing these dimensions from the classifier. It leads to a new query $Q_{\text{DRILL-OUT}}$ having the classifier $c_{\Sigma'}(x, d_{j_1}, \ldots, d_{j_{n-k}})$, where $d_{j_1}, \ldots, d_{j_{n-k}} \in \{d_1, \ldots, d_n\} \setminus \{d_{i_1}, \ldots, d_{i_k}\}$ and $\Sigma' = (\Sigma \setminus \bigcup_{j=i_1}^{i_k}\{(d_j, \Sigma(d_j))\})$.

Finally, a DRILL-IN operation on Q over dimensions $\{d_{n+1}, \ldots, d_{n+k}\}$ which all appear in the classifier's body and have value sets $\{V_{n+1}, \ldots, V_{n+k}\}$ corresponds to adding these dimensions to the head of the classifier. It produces a new query $Q_{\text{DRILL-IN}}$ having the classifier $c_{\Sigma'}(x, d_1, \ldots, d_n, d_{n+1}, \ldots, d_{n+k})$, where the dimensions $d_{n+1}, \ldots, d_{n+k} \notin \{d_1, \ldots, d_n\}$, $\Sigma' = (\Sigma \cup \bigcup_{j=i_1}^{i_k}\{(d_j, V_j)\})$.

These operations are illustrated in the following example.

EXAMPLE 3. (OLAP OPERATIONS) *Let Q be the extended query corresponding to the query-cube defined in Example 1, that is: Q :- $\langle c(x, d_{age}, d_{city}), m(x, v_{site}), \texttt{count}\rangle$, $\Sigma = \{(d_{age}, V_{age}), (d_{city}, V_{city})\}$ (the classifier and measure are as in Example 1).*

A SLICE *operation on the d_{age} dimension with value* `35` *replaces the extended classifier of Q with $c_{\Sigma'}(x, d_{age}, d_{city}) = \{c(x, \texttt{35}, d_{city})\}$ where $\Sigma' = \Sigma \setminus \{(d_{age}, V_{age})\} \cup \{(d_{age}, \{\texttt{35}\})\}$.*

A DICE *operation on both d_{age} and d_{city} dimensions with values $\{\texttt{28}\}$ for d_{age} and $\{\texttt{Madrid}, \texttt{Kyoto}\}$ for d_{city} replaces the extended classifier of Q with $c_{\Sigma'}(x, d_{age}, d_{city}) = \{c(x, \texttt{28}, \texttt{Madrid}), c(x, \texttt{28}, \texttt{Kyoto})\}$ where $\Sigma' = \{(d_{age}, \{\texttt{28}\}), (d_{city}, \{\texttt{Madrid}, \texttt{Kyoto}\})\}$.*

A DRILL-OUT *on the d_{age} dimension produces $Q_{\text{DRILL-OUT}}$:- $\langle c'_{\Sigma'}(x, d_{city}), m(x, v_{site}), \texttt{count}\rangle$ with $\Sigma' = \{(d_{city}, V_{city})\}$ and $\texttt{body}(c') \equiv \texttt{body}(c)$.*

Finally, a DRILL-IN *on the d_{age} dimension applied to the query $Q_{\text{DRILL-OUT}}$ above produces Q, the query of Example 1.*

3. OPTIMIZED OLAP OPERATIONS

The above OLAP operations lead to new queries, whose answers can be computed based on the AnS instance. The focus of the present work is on answering such queries by using the materialized results of the initial AnQ, and (only when that input is insufficient) more data, such as intermediary results generated while computing AnQ results, or (a small part of) the AnS instance. These results are often significantly smaller than the full instance, hence obtaining the answer to the new query based on them is likely faster than computing it from the instance. Figure 2 provides a sketch of the problem.

In the following, all relational algebra operators are assumed to have bag semantics.

Given an analytical query Q whose measure query (with bag semantics) is m, we denote by $\bar{m}$ the *set-semantics* query whose body is the same as the one of m and whose head comprises *all the variables of m's body*. Obviously, there is a bijection between the bag result of m and the set result of $\bar{m}$. Using $\bar{m}$, we define next the intermediary answer of an AnQ.

DEFINITION 3. (INTERMEDIARY QUERY OF AN ANQ) *Let Q :- $\langle c, m, \oplus\rangle$ be an AnQ. The intermediary query of Q, denoted $int(Q)$, is:*

$$int(Q) = c(x, d_1, \ldots, d_n) \bowtie_x \bar{m}(x, v)$$

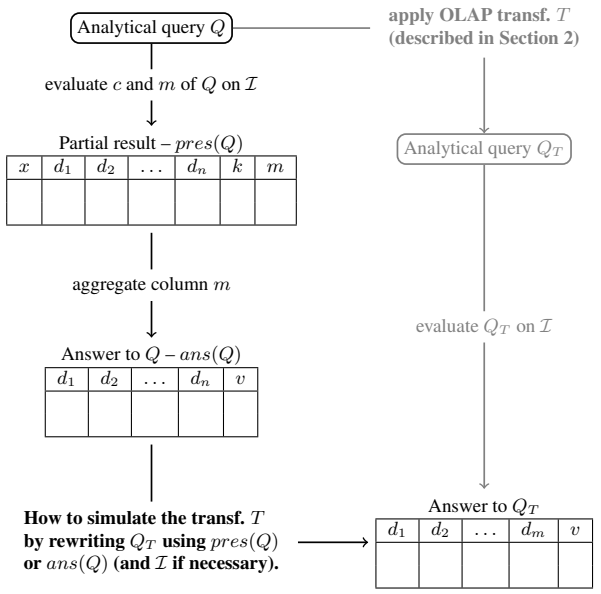


Figure 2: Problem statement.

It is easy to see that $int(Q)$ holds *all the information needed in order to compute both c and m*; it holds *more* information than the results of c and m, given that it preserves all the different embeddings of the (bag-semantics) m query in the data. Clearly, evaluating int is at least as expensive as evaluating Q itself; while int is conceptually useful, we do not need to evaluate it or store its results.

Instead, we propose to evaluate (possibly as part of the effort for evaluating Q), *store and reuse* a more compact result, defined as follows. For a given query Q whose measure (with bag semantics) is m, we term *extended measure result* over an instance $\mathcal{I}$, denoted $m^k(\mathcal{I})$, the *set* defined by:

$$\{(newk(), t) \mid t \in m(\mathcal{I})\}$$

where $newk()$ is a key-creating function returning a distinct value at each call. A very simple implementation of $newk()$, which we will use for illustration, returns successively $1, 2, 3$ etc. We assign a key to each tuple in the measure so that multiple identical values of a given measure for a given fact would not be erroneously collapsed into one. For instance, if

$$m(\mathcal{I}) = \{\!\{(x_1, m_1), (x_1, m_1), (x_1, m_2), (x_2, m_3)\}\!\},$$

then

$$m^k(\mathcal{I}) = \{(1, x_1, m_1), (2, x_1, m_1), (3, x_1, m_2), (4, x_2, m_3)\}.$$

DEFINITION 4. (PARTIAL RESULT OF AN ANQ) *Let Q :- $\langle c, m, \oplus\rangle$ be an AnQ. The* partial result *of Q on an instance $\mathcal{I}$, denoted $pres(Q, \mathcal{I})$ is:*

$$pres(Q, \mathcal{I}) = c(\mathcal{I}) \bowtie_x m^k(\mathcal{I})$$

One can see $pres(Q, \mathcal{I})$ as the input to the last aggregation performed in order to answer the AnQ, *augmented with a key*. In the

following, we use $pres(Q)$ to denote $pres(Q, \mathcal{I})$ for the working instance of the AnS.

Problem Statement : (ANSWERING AnQS USING THE MATERIALIZED RESULTS OF OTHER AnQS) Let Q, Q_T be AnQs such that applying the OLAP transformation T on Q leads to Q_T. The problem of answering Q_T using the materialized result of Q consists of finding: (i) an equivalent rewriting of Q_T based on $pres(Q)$ or $ans(Q)$, if one exists; (ii) an equivalent rewriting of Q_T based on $pres(Q)$ and the AnS instance, otherwise.

Importantly, the following holds:

$$\pi_{x,d_1,\ldots,d_n,v}(int(Q)(\mathcal{I})) = \pi_{x,d_1,\ldots,d_n,v}(pres(Q,\mathcal{I})) \quad (1)$$

$$Q \equiv \gamma_{d_1,\ldots,d_n,\oplus(v)}(\pi_{x,d_1,\ldots,d_n,v}(int(Q))) \quad (2)$$

$$ans(Q)(\mathcal{I}) = \gamma_{d_1,\ldots,d_n,\oplus(v)}(\pi_{x,d_1,\ldots,d_n,v}(pres(Q,\mathcal{I}))) \quad (3)$$

Equation (1) directly follows from the definition of $pres$. Equation (2) will be exploited to establish the correctness of some of our techniques. Equation (3) above is the one on which our rewriting-based AnQ answering technique is based.

3.1 Slice and Dice

In the case of SLICE and DICE operations, the data cube transformation is made simply by row selection over the materialized final results of an AnQ.

EXAMPLE 4. (DICE) *The query Q asks for the average number of words in blog posts, for each blogger's age and residential city.*

$$Q :\text{-} \langle c(x, d_{age}, d_{city}), m(x, v_{words}), \texttt{average}\rangle$$

$$\begin{aligned} c(x, d_{age}, d_{city}) :\text{-}\ & x\ \texttt{rdf:type Blogger},\\ & x\ \texttt{hasAge}\ d_{age}, x\ \texttt{livesIn}\ d_{city}\\ m(x, v_{words}) :\text{-}\ & x\ \texttt{rdf:type Blogger},\\ & x\ \texttt{wrotePost}\ p,\\ & p\ \texttt{hasWordCount}\ v_{words} \end{aligned}$$

Suppose the answer of c over $\mathcal{I}$ is

$$\{\langle \texttt{user}_1, 28, \texttt{Madrid}\rangle, \langle \texttt{user}_3, 35, \texttt{NY}\rangle, \langle \texttt{user}_4, 28, \texttt{Madrid}\rangle\}$$

and the answer of m over $\mathcal{I}$ is

$$\{\!\{\langle \texttt{user}_1, 100\rangle, \langle \texttt{user}_1, 120\rangle, \langle \texttt{user}_3, 570\rangle, \langle \texttt{user}_4, 410\rangle\}\!\}$$

Joining the answers of c and m in such a query results in:

$$\{\!\{\langle \texttt{user}_1, 28, \texttt{Madrid}, 100\rangle, \langle \texttt{user}_1, 28, \texttt{Madrid}, 120\rangle, \langle \texttt{user}_3, 35, \texttt{NY}, 570\rangle, \langle \texttt{user}_4, 28, \texttt{Madrid}, 410\rangle\}\!\}$$

The final answer to Q after aggregation is:

$$\{\langle 28, \texttt{Madrid}, 210\rangle, \langle 35, \texttt{NY}, 570\rangle\}$$

The query Q_{DICE} is the result of a DICE *operation on Q, restricting the d_{age} to values between 20 and 30. Q_{DICE} differs from Q only by its classifier which can be written as $c_{\Sigma'}(x, d_{age}, d_{city})$ where $\Sigma' = \Sigma \setminus \{(d_{age}, V_{age})\} \cup \{(d_{age}, \{d_{age}\}_{20 \leq d_{age} \leq 30})\}$.*

Applying DICE *on the answer to Q above yields the result:*

$$\{\langle 28, \texttt{Madrid}, 210\rangle\}$$

Now, we calculate the answer to Q_{DICE}. The result of the classifier query $c_{\Sigma'}$, obtained by applying a selection on the d_{age} dimension is:

$$\{\langle \texttt{user}_1, 28, \texttt{Madrid}\rangle, \langle \texttt{user}_4, 28, \texttt{Madrid}\rangle\}$$

Evaluating m and joining its result with the above set yields:

$$\{\!\{\langle \texttt{user}_1, 28, \texttt{Madrid}, 100\rangle, \langle \texttt{user}_1, 28, \texttt{Madrid}, 120\rangle, \langle \texttt{user}_4, 28, \texttt{Madrid}, 410\rangle\}\!\}$$

The final answer to Q_{DICE} after aggregation is:

$$\{\langle 28, \texttt{Madrid}, 210\rangle\}$$

DICE *applied over the answer of Q yields the answer of Q_{DICE}.*

DEFINITION 5. (SELECTION) *Let* `dice` *be a dice operation on analytical queries. Let Σ' be the function introduced in Definition 2. We define a* selection $\sigma_{\texttt{dice}}$ *as a function on the space of analytical query answers $ans(Q)$ where:*

$$\sigma_{\texttt{dice}}(ans(Q)) = \{\langle d_1, \ldots, d_n, v\rangle | \langle d_1, \ldots, d_n, v\rangle \in ans(Q) \wedge \forall i \in \{1, \ldots, n\}\ d_i \in \Sigma'(d_i)\}$$

PROPOSITION 1. *Let $Q :\text{-} \langle c_\Sigma(x, d_1, \ldots, d_n), m(x, v), \oplus\rangle$ and $Q_{\text{DICE}} :\text{-} \langle c'_{\Sigma'}(x, d_1, \ldots, d_n), m'(x, v), \oplus\rangle$ be two analytical queries such that query $Q_{\text{DICE}} = \texttt{dice}(Q)$. Then $\sigma_{\texttt{dice}}(ans(Q)) = ans(Q_{\text{DICE}})$.*

The proofs for all our results can be found in [3].

3.2 Drill-Out

Unlike the relational DW setting, in our RDF warehousing framework the result of a drill-out operation (that is, the answer to $Q_{\text{DRILL-OUT}}$) cannot be correctly computed directly from the answer to the original query Q, and here is why. Each tuple in $ans(Q)$ binds a set of dimension values to an aggregated measure. In fact, each such tuple represents a set of facts having the same dimension values. Projecting a dimension out will make some of these sets merge into one another, requiring a new aggregation of the measure values. Computing this new aggregated measure from the ones in $ans(Q)$ will require considering whether the aggregation function has the *distributive* property, i.e., whether $\oplus(a, \oplus(b, c)) = \oplus(\oplus(a, b), c)$.

1. *Distributive aggregation function, e.g. sum.* In this case, the new aggregated measure value could be computed from $ans(Q)$ *if the sets of facts aggregated in each tuple of $ans(Q)$ were mutually exclusive*. This is not the case in our setting where each fact can have several values along the same dimension. Thus, aggregating the already aggregated measure values will lead to erroneously consider some facts more than once; avoiding this requires being able to trace the measure results back to the facts they correspond to.

2. *Non-distributive aggregation function, e.g., avg.* For such functions, the new aggregated measure must be computed from scratch.

Based on the above discussion, we propose Algorithm 1 to compute the answer to $Q_{\text{DRILL-OUT}}$, using the partial result of Q, denoted $pres(Q)$ above, which we assume has been materialized and stored as part of the evaluation of the original query Q. This deduplication (δ) step is needed, since some facts may have been repeated in T for being multivalued along d_i. The aggregation function $\oplus$ is applied to the measure column of the resulting relation T, using γ, grouping the tuples along the dimensions $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_n$.

Proposition 2 states the correctness of Algorithm 1.

Algorithm 1 DRILL-OUT cube transformation

1: **Input:** $pres(Q), d_i$
2: $T \leftarrow \Pi_{root,d_1,\ldots,d_{i-1},d_{i+1},\ldots,d_n,k,v}(pres(Q))$
3: $T \leftarrow \delta(T)$
4: $T \leftarrow \gamma_{d_1,\ldots,d_{i-1},d_{i+1},\ldots,d_n,\oplus(v)}(T)$
5: **return** T

PROPOSITION 2. *Let Q be an AnQ and $Q_{\text{DRILL-OUT}}$ be the AnQ obtained from Q by drilling out along the dimension d_i. Algorithm 1 applied on $pres(Q)$ and d_i computes $ans(Q_{\text{DRILL-OUT}})$.*

Example 5 illustrates Algorithm 1, and also shows how relying only on $ans(Q)$ may introduce errors.

EXAMPLE 5. (DRILL-OUT) *Consider an analytical query Q such that its classifier c_1, measure m and intermediary answer $pres(Q)$ have the results shown below.*

c_1

$root$	d_1	$\ldots$	d_{n-1}	d_n
x	a_1	$\ldots$	a_{n-1}	a_n
x	a_1	$\ldots$	a_{n-1}	b_n
y	a_1	$\ldots$	a_{n-1}	b_n

m

k	$root$	v
1	x	m_1
2	y	m_2

$pres(Q)$

$root$	d_1	$\ldots$	d_{n-1}	d_n	k	v
x	a_1	$\ldots$	a_{n-1}	a_n	1	m_1
x	a_1	$\ldots$	a_{n-1}	b_n	1	m_1
y	a_1	$\ldots$	a_{n-1}	b_n	2	m_2

(i)

Let $Q_{\text{DRILL-OUT}}$ be the result of a DRILL-OUT *operation on Q eliminating dimension d_n. The measure of $Q_{\text{DRILL-OUT}}$ is still m, while its classifier c_2 has the answer shown next:*

c_2

$root$	d_1	$\ldots$	d_{n-1}
x	a_1	$\ldots$	a_{n-1}
y	a_1	$\ldots$	a_{n-1}

Note that c_2 returns only one row for x, because it has one value for the dimension vector $\langle d_1, \ldots, d_{n-1}\rangle$. $pres(Q_{\text{DRILL-OUT}})$ yields:

$root$	d_1	$\ldots$	d_{n-1}	k	v
x	a_1	$\ldots$	a_{n-1}	1	m_1
y	a_1	$\ldots$	a_{n-1}	2	m_2

Applying aggregation over the above table leads to:

d_1	$\ldots$	d_{n-1}	v
a_1	$\ldots$	a_{n-1}	$\oplus(\{m_1, m_2\})$

(ii)

Algorithm 1 on the input (i), first projects out d_n. Table T after projecting out d_n from $pres(Q)$ is:

$root$	d_1	$\ldots$	d_{n-1}	k	v
x	a_1	$\ldots$	a_{n-1}	1	m_1
x	a_1	$\ldots$	a_{n-1}	1	m_1
y	a_1	$\ldots$	a_{n-1}	2	m_2

After eliminating duplicates ($\delta(T)$), grouping and aggregation, we obtain:

d_1	$\ldots$	d_{n-1}	v
a_1	$\ldots$	a_{n-1}	$\oplus(\{m_1, m_2\})$

(iii)

The output of Algorithm 1 above, denoted (iii), is the same as (ii), showing that our algorithm answers $Q_{\text{DRILL-OUT}}$ correctly using the intermediary answer of Q.

Next we examine what would happen if an algorithm took the answer of Q as input. First, we compute $ans(Q)$, by aggregating the measures along the dimensions from the intermediary result depicted in (i):

$ans(Q)$

d_1	$\ldots$	d_{n-1}	d_n	v
a_1	$\ldots$	a_{n-1}	a_n	$\oplus(\{m_1\})$
a_1	$\ldots$	a_{n-1}	b_n	$\oplus(\{m_1, m_2\})$

Next we project out d_n from $ans(Q)$:

d_1	$\ldots$	d_{n-1}	v
a_1	$\ldots$	a_{n-1}	$\oplus(\{m_1\})$
a_1	$\ldots$	a_{n-1}	$\oplus(\{m_1, m_2\})$

and aggregate, assuming that $\oplus$ is distributive.

d_1	$\ldots$	d_{n-1}	v
a_1	$\ldots$	a_{n-1}	$\oplus(\{m_1, m_1, m_2\})$

(iv)

Observe that (iv) is different from (ii). More specifically, the measure value corresponding to the multi-valued entity x has been considered twice *in the aggregated measure value of (iv).*

3.3 Drill-In

The drill-in operation increases the level of detail in a cube by adding a new dimension. In general, this additional information is not present in the answer of the original query. Hence, answering the new query requires extra information.

Algorithm 2 DRILL-IN cube transformation

1: **Input:** $pres(Q, \mathcal{I}), c, d_{n+1}$
2: build $q^Q_{aux}(dvars, d_{n+1})$:- *body* ▷ as per Definition 6
3: $T \leftarrow pres(Q, \mathcal{I}) \bowtie_{dvars} q^Q_{aux}(\mathcal{I})$
4: $T \leftarrow \gamma_{d_1,\ldots,d_n,d_{n+1},\oplus(v)}(T)$
5: **return** T

Algorithm 2 uses the partial result of Q, denoted $pres(Q)$, and consults the materialized AnS instance to obtain the missing information necessary to answer $Q_{\text{DRILL-IN}}$. We retrieve this information through an auxiliary query defined as follows.

DEFINITION 6. (AUXILIARY DRILL-IN QUERY) *Let Q :- $\langle c(x, d_1, \ldots, d_n), m(x, v), \oplus\rangle$ be an AnQ and d_{n+1} a non-distinguished variable in c. The auxiliary* DRILL-IN *query of Q over d_{n+1} is a conjunctive query $q^Q_{aux}(dvars, d_{n+1})$:- $body_{aux}$, where*

- *each triple $t \in body(c)$ containing the variable d_{n+1} is also in $body_{aux}$;*

$\mathcal{I}$

s	p	o
website1	hasUrl	URL1
website1	supportsBrowser	firefox
website2	hasUrl	URL2
website2	supportsBrowser	chrome
video1	postedOn	website1
video1	postedOn	website2
video1	type	Video
video1	viewNum	n

$pres(Q)$

x	d_2	k	v
video1	URL1	1	n
video1	URL2	2	n

$ans(Q)$

d_2	v
URL1	n
URL2	n

$pres(Q_{\text{DRILL-IN}})$

x	d_2	d_3	k	v
video1	URL1	firefox	1	n
video1	URL2	chrome	2	n

$ans(Q_{\text{DRILL-IN}})$

d_2	d_3	v
URL1	firefox	n
URL2	chrome	n

x	d_2	d_3	k	v
video1	URL1	firefox	1	n
video1	URL2	chrome	1	n

Figure 3: DRILL-IN example.

- *for each triple t_{aux} in $body_{aux}$ and t in the body of c such that t and t_{aux} share a non-distinguished variable of c, the triple t also belongs to $body_{aux}$;*
- *each variable in $body_{aux}$ that is distinguished in c is a distinguished variable q^Q_{aux}.*

The auxiliary query q^Q_{aux} comprises all triples from Q's classifier having the dimension d_{n+1}; to these, it adds all the classifier triples sharing a *non-distinguished (existential)* variable with the former; then all the classifier triples sharing an existential variable with a triple previously added to $body_{aux}$ etc. This process stops when there are no more existential variables to consider. The variables distinguished in c, together with the new dimension d_{n+1}, are distinguished in q^Q_{aux}.

PROPOSITION 3. *Let Q :- $\langle c, m, \oplus\rangle$ be an analytical query and d_{n+1} be a non-distinguished variable in c. Let $Q_{\text{DRILL-IN}}$ be the analytical query obtained from Q by drilling in along the dimension d_{n+1}, and $\mathcal{I}$ be an instance. Algorithm 2 applied on $pres(Q, \mathcal{I})$, c and d_{n+1} computes $ans(Q_{\text{DRILL-IN}})(\mathcal{I})$.*

EXAMPLE 6. (DRILL-IN REWRITING) *Let Q :- $\langle c, m, \texttt{sum}\rangle$ be the following AnQ:*

$c(x, d_2)$:- x `rdf:type` `Video`, x `uploadedOn` d_1,
d_1 `hasUrl` d_2, d_1 `supportsBrowser` d_3
$m(x, v)$:- x `rdf:type` `Video`, x `viewNum` v

$Q_{\text{DRILL-IN}}$ is the result of a DRILL-IN *that adds the dimension d_3, having the classifier query $c'(x, d_2, d_3)$.*
Figure 3 shows the materialized analytical schema instance, the partial and final answer to Q, and the partial and final answer to $Q_{\text{DRILL-IN}}$. Now, let us see how to answer $Q_{\text{DRILL-IN}}$ using Algorithm 2. We have:

$q^Q_{aux}(x, d_2, d_3)$:- x `postedOn` d_1, d_1 `hasUrl` d_2,
d_1 `supportsBrowser` d_3

Based on $\mathcal{I}$, the answer to q^Q_{aux} is:

x	d_2	d_3
video1	*URL1*	*firefox*
video1	*URL2*	*chrome*

Joining the above with $pres(Q)$ yields the last table in Figure 3, which after aggregation yields the result of $Q_{\text{DRILL-IN}}$.

4. RELATED WORK

Previous RDF data management research focused on efficient stores, query processing, view selection etc. BGP query answering techniques have been studied intensively, e.g., [5, 6], and some are deployed in commercial systems such as Oracle 11g's "Semantic Graph" extension. Our optimizations can be deployed on top of any RDF data management platform, to extend it with optimized analytic capabilities.

The techniques we presented can be seen as a particular case of view-based rewriting [7], where partial AnQ results are used as a materialized view. Novel algorithms were required due to the novel AnQ language we introduced in [1].

SPARQL 1.1 [8] features SQL-style grouping and aggregation, less expressive than our AnQs, as our measure queries allow more flexibility than SPARQL. Thus, the OLAP operation optimizations we presented can also apply to the more restricted SPARQL analytical context.

OLAP has been thoroughly studied in a relational setting, where it is at the basis of a successful industry; in particular, OLAP operation evaluation by reusing previous cube results is well-known. The heterogeneity of RDF, which in turn justified our novel RDF analytics framework [1], leads to the need for the novel algorithms we described here, which are specific to this setting.

5. CONCLUSION

Our work focused on optimizing the OLAP transformations in the RDF data warehousing framework we introduced in [1], by using view-based rewriting techniques. To this end, for each OLAP operation, we introduced an algorithm that answers a transformed query based on the final or on an intermediary result of the original analytical query. We formally prove the correctness of our techniques, and describe experiments performed with our algorithms, in our technical report [3].

6. REFERENCES

[1] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatis, "RDF analytics: lenses over semantic graphs," in *WWW*, 2014.

[2] S. Spaccapietra, E. Zimányi, and I. Song, Eds., *Journal on Data Semantics XIII*, ser. LNCS, vol. 5530. Springer, 2009.

[3] E. Akbari, F. Goasdoué, I. Manolescu, and A. Roatiş, "Efficient OLAP operations for RDF analytics," INRIA Research Rep. RR-8668, Jan. 2015.

[4] "OLAP Council White Paper," http://www.olapcouncil.org/research/resrchly.htm.

[5] F. Goasdoué, I. Manolescu, and A. Roatiş, "Efficient query answering against dynamic RDF databases," in *EDBT*, 2013.

[6] J. Pérez, M. Arenas, and C. Gutierrez, "nSPARQL: A navigational language for RDF," *J. Web Sem.*, vol. 8, no. 4, 2010.

[7] A. Y. Halevy, "Answering queries using views: A survey," *VLDB J.*, vol. 10, no. 4, 2001.

[8] W3C, “SPARQL 1.1 query language,”
http://www.w3.org/TR/sparql11-query/, March 2013.