An SQL-based Approach to Semantic Web Reasoning and Query Answering

© Dmitry V. Levshin

NICEVT (Science and Research Center on Computer Technology)
levshin@nicevt.ru

Abstract

The Semantic Web is the extension of WWW whose purpose is to allow software agents to process documents more intelligently. Access to RDF data is one of the key moments for semantic applications and requires implementing both retrieval and reasoning. Reasoning with large ontologies and data sets becomes increasingly important. It is a serious challenge for the most advanced in-memory reasoners, and they begin to exploit databases. Most of the proposals apply a hybrid database/reasoner architecture which has several shortcomings. The paper presents an approach to query Semantic Web data in databases. Both TBox and ABox reasonings are performed using RDBMS features. It can be performed during querying or precomputed; caching of previous queries results is also supported. Integration of data stored in a relational database with Semantic Web data and support of most SPARQL features are obtained using SQL power.

1 Introduction

The Semantic Web is a notion proposed by Tim Berners-Lee [8] in the 90th as an extension of the nowadays Web with a possibility of adding semantics to documents, which computer agents could use to intelligent data processing. Particularly, there are believes that it will significantly improve quality of search in the Internet.

Development of the basic Semantic Web formats follows to the stack architecture proposed by Tim Berners-Lee [7]. It assumes that every new layer (format) extends preceding ones adding new features. Firstly, RDF [19] describing the Semantic Web data model was developed. Then RDFS and OWL [23] were developed on top of it allowing formulation of logical statements. OWL species are based on Description Logics (DLs) [5]; SWRL [15] excels from other proposals for uniting OWL with Datalog rules (e.g. [12, 20]) as W3C member submission. SPARQL language [24] was proposed for querying RDF data. As since the Semantic Web formats have logical foundation, querying in the Semantic Web assumes not only selection of explicit data, but also inference of implicit data on base of statements in DLs and Datalog.

Highly-optimized DL reasoners implementing tableaux algorithm [16] are in general used to support OWL. They perform reasoning in memory. At the same time, as it is marked in [4], there are appearing large-size ontologies and RDF datasets (with thousands of named concepts and millions of triples), which are serious challenge for the most advanced and optimized reasoners. In [14], it is marked that possibility of reasoning with such ontologies will be a requirement for future applications.

Moreover, as [17] notes, tableaux algorithm itself often does not well suit for query answering. The main reason is that tableaux algorithm provides refutation procedure, not query answering algorithm. As a consequence, new ways of reasoning including ones based on translating of DL knowledge bases (KBs) into Datalog (e.g. [12]) are investigated. Further, as it is noted in [29], strict adherence to the completeness properties of reasoning procedure and strong focus on theoretical properties of the most researches does not allow them to obtain required scalability. Potential ways to rectify the situation including *approximate reasoning* (trade of scalability to reasoning completeness) and *incremental reasoning* (leverage caching of prior results) are presented in [29]. The proposal is justified by the fact that practical applications often use very poor ontologies.

Increasing importance of reasoning with large amounts of data for query answering in the Semantic Web leads to use DBMSs for it. Several proposals have appeared based on hybrid reasoner/database architecture. However, they support limited query and reasoning possibilities meanwhile. Also these loosely-coupled approaches have several shortcomings including that DBMS users cannot reference ontology data directly. Only a few approaches [11, 18] for direct support of RDF queries in a DBMS appear. However, [18] leverages XML possibilities of hybrid relational-XML DBMSs, and [11] supports limited reasoning (e.g. TBox reasoning is performed using external reasoners, property restrictions are not supported).

This paper presents an approach of query answering and reasoning for Semantic Web data. It allows both TBox and ABox reasoning to be performed using only database features. For ABox reasoning, a two-stage algorithm flexible enough to any changes in logics used in the Semantic Web is presented. ABox reasoning can

Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems, Saint-Petersburg, Russia, 2009

be performed during querying, or pre-computation or caching prior results can be used for it. It is hard to implement complete and efficient inference engine using only DBMS features. Moreover, approximate reasoning is proposed in [29] as a way for achieving scalability of reasoning. So our approach does not adhere to completeness property. This decision allows to support more expressive ontologies and entail more facts than in approaches which restrict supported constructs to adhere completeness property. Our approach also allows querying data from both database and the Semantic Web simultaneously. The most of the DBMS features (triggers, stored procedures, table functions) used for implementation of reasoning and querying are supported by the majority of RDBMSs. The rule system [28] is the only specific feature of PostgreSQL [1] used for implementations. However, usage of rules is not mandatory, and they can be substituted by triggers. Therefore, our approach can be implemented in other RDBMSs.

The rest of the paper is organized as follows. Section 2 overviews basic notions of DLs and the Semantic Web, and specific PostgreSQL features mentioned in the paper. Section 3 surveys the main results in researches on effective storage of RDF data, and then presents the database schema used in our work. Section 4 presents algorithms for reasoning, and Section 5 demonstrates how RDF data can be queried in a database using these algorithms. Section 6 surveys related work before Section 7 concludes and presents future work.

2 Preliminaries

This Section surveys essentials of the Semantic Web and several database features used for implementations in this work.

2.1 Semantic Web Essentials

The Semantic Web data model defined in RDF represents all statements as oriented marked graphs, in which nodes are marked as resources (URIs) or literals and edges – as properties. Such graph can be also represented in XML notation or as a set of ⟨*subject, property, object*⟩ triples.

OWL [23] allows defining terms and relations between them. OWL species OWL Lite and OWL DL are based on DLs $\mathcal{SHIF}$(**D**) and $\mathcal{SHOIN}$(**D**), resp. A DL knowledge base (KB) usually consists of two components: *terminological* component called TBox and *assertional* one called ABox. TBox consists of subsumption axioms, and ABox contains assertions about individuals named role and concept assertions[1]. DLs differ from each other in that which concept descriptions are supported and how they can be used.

$$Female \sqcap \exists hasChild.Person \sqsubset Mother \quad (1)$$

$$Male \sqcup Female \sqsubset Person \quad (2)$$

$$hasSon \sqsubset hasChild \quad (3)$$

$$hasPar(Peter, Anna) \quad (4)$$

$$Female(Anna) \quad (5)$$

For instance, the subsumption axiom (1) states that anyone who is female and has child is mother, (2) treats both male and female to be persons, (3) describes $hasSon$ as subproperty of $hasChild$; the role assertion (4) states that individual $Peter$ has a parent $Anna$, and the concept assertion (5) also states that $Anna$ is female.

[1]Concept and role assertions are obviously represented as RDF triples

```
PREFIX p:  <http://a.com/ontology#>
SELECT ?n1 ?n2
WHERE {
    {?s p:hasAunt ?m.
     ?s p:name ?n1.
     ?m p:name ?n2}
  UNION
    {?s p:hasUncle ?m.
     ?s p:name ?n1.
     ?m p:name ?n2} }
```

Figure 1: Example of SPARQL query

```
hasSon(?x,?y):-hasPar(?y,?x),Male(?y)   (6)
```

SWRL extends OWL with Horn-like Datalog rules. Concepts (roles) can be used in SWRL rules as unary (resp., binary) atoms. For instance, property $hasSon$ is asserted implicitly in the SWRL rule (6) on base of property $hasPar$ and concept $Male$, and this concept in turn can be defined in some subsumption axioms. Note that uniting DLs with Datalog rules is an actual area for researches. Therefore, changes in SWRL are possible.

Although different reasoning services are supported in DLs, Semantic Web data should be accessed via SPARQL query language. In SPARQL, unions of conjunctive queries can be formulated. For instance, query in Fig. 1 is used to find persons who have uncle or aunt. For more information about DLs, the formats of the Semantic Web and SPARQL please refer to [5, 15, 19, 23, 24].

2.2 Database Features

The reader is assumed to be familiar with stored procedures and triggers. Table function is a stored procedure returning sets of rows and allowing specification of columns at the time of its invocation. Rules in PostgreSQL are similar to triggers in that what we expect from them: to perform automatically some specified action, when some event on the specified table occurs. Only update rules are considered in the paper, and the following simplified syntax for rules is used:

```
CREATE [ OR REPLACE ] RULE name AS ON event
    TO table [ WHERE condition ]
    DO [ ALSO | INSTEAD ]
      { NOTHING | command }
```

However, two significant differences should be marked. First, additional qualifications can be specified in rules to restrict firing conditions, and redundant computations can be reduced. Second, in contrast to triggers, the rule system rewrites a query before scheduler optimizes it. Consequently, a query is optimized in this case taking into account rules, and it might be more effective to use rules than triggers. The interested reader may refer to [1, 28] for more in-depth information about the rule system and the other features.

3 Database Schema

Choice of a database schema has a great impact on efficiency of storing and querying data. Much attention was paid to schemas for storing Semantic Web data in relational databases. This Section views results achieved in the area and then presents the schema used in our system.

3.1 Overview of Existing Approaches

As since RDF data can be viewed as a set of triples, it seems trivial to persist it in a single three-column table. Further, for reducing of used disk space, URIs and literals can be stored in this table not directly, but as unique integer values (further, *identifiers*) referring to their values. Mappings between these identifiers and actual values can be stored in one or more special tables (which further is called *dictionary*). This *normalisation* is practical, because URIs and literals can have arbitrary length, and can occur frequently. This approach can be called **single store**. Particularly, it is applied by Oracle [11].

In Jena1 [30], single store approach was used too, but triple store has two object columns to distinguish literals from URIs. Unlike [11], dictionary consists of two tables: one is for literals and one - for URIs. In Jena2 un-normalized schema is used: all literals and URIs, whose text representation length does not overcome some threshold, are stored directly in a triple store. Dictionary is used only for values overcame the threshold. To distinguish actual values from identifiers columns are coded with a special prefix. To reduce used disk space it is proposed to use special table for namespace prefixes, which should be small and lie in memory.

Another important proposal of Jena2 is the usage of **property tables**. This approach assumes determination of properties often accessed together and storing them in a separate table. This property table stores all occurrences of these properties. It means that these properties can't be met in other tables used for this RDF graph. Such tables are declared to be very useful for properties with maximal cardinality equal to 1. In this case, each table row contains property values (or `NULL`s for unknown values) for some subject. As marked in [3], properties also may be clustered into separate tables with regard to certain classes. In this case, one property can occur in different tables. Regardless of a particular clusterization algorithm, triples with properties not clustered in separate tables are stored in a three-column triple store.

As it is mentioned in [3], the property table approach is not widely used. In particular, in Jena2 property tables are used only for reification statements. More widely they are used in Sesame [10]. Main disadvantages of this approach can be summarized as the following: it is difficult to define a property clusterization algorithm, which is very important for effective querying; storage of multivalued properties in separate tables might be ineffective, and property tables might be sparse.

Vertical partitioning approach is presented in [3] as a solution of these problems. It can be viewed as a special case of the property table approach with a simple clusterization algorithm: a separate two-column table is created for every met property. This approach is shown to be comparable with general property table approach and better than the single store approach in terms of scalability and efficiency of querying, when number of properties is restricted. However, [25] shows that vertical partitioning has a weakness when predicates in queries are not fixed: when number of property tables becomes significant, query may require a large number of unions.

3.2 The Schema Used in Our Work

In our work, normalised schema with one `values`[2] table as a dictionary is used. As since both URIs and literals are stored together, field `value_type` is added to distinguish them. Field `literal_type` is used for typed literals as a reference to the table of supported literal types, which stores information how to convert text representation of a literal into corresponding type and conversely. For each typed literal, text value stored in the dictionary is obtained by sequential conversion of it to its datatype and conversely (e.g. integer-valued literals '6' and '+6' will be both converted to 6 and stored as '6' in one row of the dictionary). As consequence, the dictionary does not contain duplicate URIs and literals. Therefore, equality comparison of RDF triples can be performed as comparison of corresponding triples of identifiers. Such comparisons are performed frequently during reasoning, and it avoids look ups into the dictionary during reasoning. Actual values of resources and literals are selected only to return answer on the query.

As the overview shows, property tables (and vertical partitioning) allows better performance comparing to the single store approach, when properties in queries are fixed or number of tables is not large. Therefore, separate tables were created for the properties from RDFS (`domain`, `range`, `subClassOf`, and `subPropertyOf`) and OWL (`inverseOf`, `equivalentClass`, `equivalentProperty`, `differentFrom`, and `sameAs`) vocabularies which has great impact on reasoning. For the symmetric RDFS and OWL properties (e.g. `differentFrom` and `sameAs`), having any of triples $\langle s,p,o\rangle$ or $\langle o,p,s\rangle$, the pair $\langle$`id(s)`,`id(p)`$\rangle$ (where `id(s)` is less than `id(p)`[3]) is stored in the corresponding table; and a view is defined on it to show all the triples. Properties for OWL constructs (unions, intersections and property restrictions), SWRL rules, and `oneOf` and `AllDifferent` entities are also clustered in separate tables. Further all the tables mentioned above in the paragraph are called the *statements tables*.

Two tables are used for the rest properties: property table `typetrip` for `rdf:type` and table `triples` for arbitrary properties. This partition has two reasons: data set is divided on two nearly equal parts for many ontologies; concept assertions very often participate in inference. Thus, usage of `typetrip` and the statements tables allows both limiting the searching space of candidate triples during reasoning and reducing disk space. Partition for arbitrary properties is not applied, since it complicates the reasoning process and may have worse performance for large number of properties. Column `isinf` is used in the property tables and the `triples` table to distinguish explicit triples from implicit ones. The view `tr` uniting all triples using `UNION ALL` operator was created to provide ability to access easily all

[2] All the tables mentioned in the paper are depicted in Fig. 2.

[3] `id(r)` is used to denote identifier asserted for `r` in the dictionary

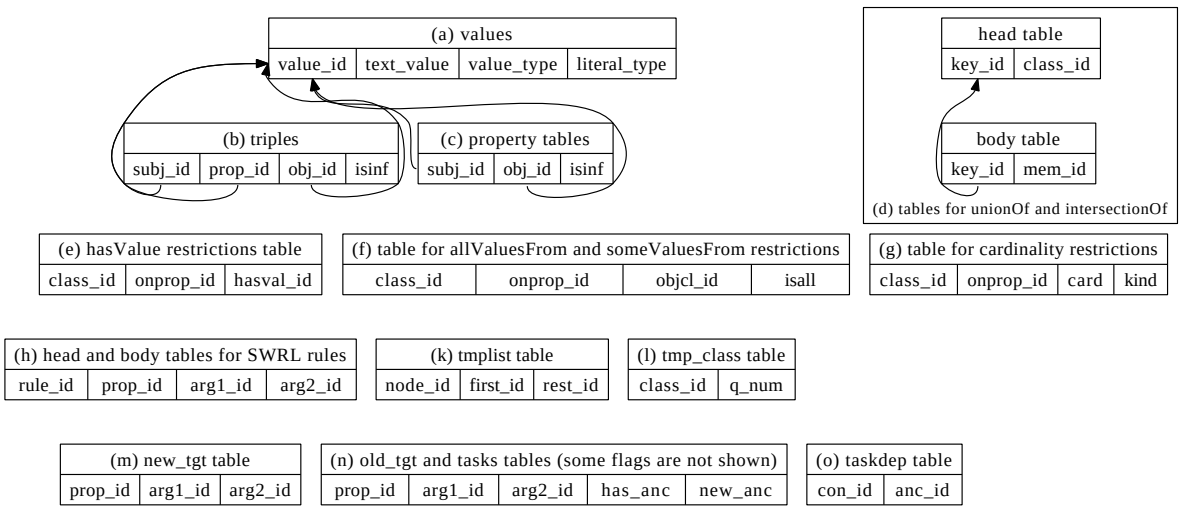

Figure 2: Tables used for storing ontologies and SWRL rules

stored triples in queries or inference rules.

The C program module ($parser$) was tailored to load Semantic Web documents into a database. It parses an input document in XML notation, builds set of triples on base of it, and stores the set into `tmptr` table. Then the tailored stored procedure is invoked in the database to create inference rules[4] and perform bulk load of the triples into the appropriate tables.

Several other tables are created for purposes of reasoning, and they are described in Section 4.2.

4 Reasoning

4.1 Reasoning Schema

According to partitioning of DL KBs into TBoxes and ABoxes, process of reasoning consists of two phases:

- Relations between concepts are determined on base of extensional relations and descriptions. We assume this phase to be performed each time, when the stored ontology is modified (by SQL command or by the parser). Inferred relations are stored in a database steadily. Assuming that size of TBox is significantly less than size of ABox, this permanent storage is not considerable in terms of disk space.

- New facts about individuals are entailed. It can be performed, when the ontology or dataset is modified (static approach) or at the time of query execution (dynamic approach). Choice of the appropriate approach can be viewed as a choice between disk space and query execution time. Note that the static approach can be reduced to the dynamic approach, if it supports caching of prior query results.

This schema of reasoning is determined not only by correspondence to the structure of DL KBs and the possibility to use numerous results in TBox reasoning. It also allows obtaining of more complete results in ABox reasoning. The paper presents the phases in the contrary order to show this influence.

[4]Thus, semantics of all statements are supported only by database features.

4.1.1 Concept Descriptions Representation

The representation of OWL concept descriptions should be introduced to gain a better insight of the algorithms of reasoning. OWL allows nested concept descriptions, and the parser divides them into sets of plain descriptions assigning names (blank nodes in terms of RDF) to auxiliary descriptions. For instance, the subsumption axiom (1) will be represented as follows:

$$D \sqsubset Mother \tag{7}$$

$$D\ is\ Female \sqcap E \tag{8}$$

$$E\ is\ \exists\, hasChild.Person \tag{9}$$

Note that relation is denotes names assigned for descriptions (e.g. E and D) and is not the same as $\equiv$ relation. The representation allows the database schema to store concept descriptions in separate tables (see Sect. 3.2). For instance, the axiom (1) is stored as tuples in the sublasses, intersections and restrictions tables (see Fig. 2(c,d,f)). It also simplifies both TBox and ABox reasoning: inference rules are created only for fixed set of plain descriptions; there is no need in recursive query creation for nested descriptions, because they are supported by a sequence of more simple rules; auxiliary descriptions can be used to avoid duplicate evaluations (e.g., E can be used for description of $Parent$ concept).

However, the representation makes descriptions more sensitive to notation. For instance, two following descriptions

$$D_1\ is\ C_1 \sqcup (C_2 \sqcup C_3) \tag{10}$$

$$D_2\ is\ (C_1 \sqcup C_2) \sqcup C_3 \tag{11}$$

are partitioned into the following 4 descriptions:

$$F\ is\ C_2 \sqcup C_3 \tag{12}$$

$$D_1\ is\ C_1 \sqcup F \tag{13}$$

$$G\ is\ C_1 \sqcup C_2 \tag{14}$$

$$D_2\ is\ G \sqcup C_3 \tag{15}$$

In this case, it may be harder to determine that G is sub-class of D_1 using (13) and (15) than (10) and (11), but thorough determination of TBox inference rules (see Sect. 4.3) can overcome this problem.

4.2 ABox Reasoning

Development of the algorithm for ABox reasoning was directed by the following conditions:

- As TBox reasoning, ABox reasoning is performed by translation of OWL and SWRL statements into SQL commands.
- A set of OWL concept descriptions and relations, and SWRL rules used in reasoning is not static. Moreover, there can be changes in SWRL. Therefore, the algorithm should be flexible enough to support dynamic rule sets or format changes.
- ABox reasoning is performed during query execution. So it is important to compute only consequences relevant to the query. Therefore, the algorithm consists of two stages: firstly, it determines goals relevant to the given query (Sect. 4.2.1) and then executes SQL commands (entailment commands) obtaining these goals (Sect. 4.2.2).

4.2.1 Goals Collecting

The first stage is used to determine triples that should be entailed to answer the given query and to avoid redundant computations. The following notions are introduced for this task:

Definition 1. *We call g a goal, if $g = C(s)$ or $g = R(s, o)$, where C is a concept name, R is a role name, s is either a variable or URI, and o is either a variable, URI or a literal. We denote s as $g.\texttt{subj}$, and o as $g.\texttt{obj}$.*

A goal can be constructed for every concept description, because it has an asserted name in the representation (see Sect. 4.1.1). Variables in different goals may have the same names. However, such variables are not the same. For instance, if we have goals $g_1 = P(?x, ?y)$ and $g_2 = P(?y, ?x)$, they can be rewritten as $g_1 = P(?x_1, ?y_1)$ and $g_2 = P(?x_2, ?y_2)$, respectively.

Definition 2. *We say that a triple t with property `rdf:type` corresponds to a goal $g = C(s)$, if $t.\texttt{obj} = C$, and exists substitution $\theta : t.\texttt{subj} = s\theta$. We say that a triple t with a property R different from `rdf:type` corresponds to a goal $g = R(s, o)$, if exists $\theta : t.\texttt{subj} = s\theta$ and $t.\texttt{obj} = o\theta$. We say that an entailment command (inference rule) corresponds to a goal g, if it entails triples corresponding to g.*

Thus, the task of the first phase is to collect goals corresponding to consequences related to the query. Entailment commands corresponding to the collected goals are executed during the second phase.

SPARQL-like notation (see Fig. 1) is used for RDF queries with the following restrictions: graph pattern is non-empty conjunction of triple patterns; no variable is used in predicate position; if pattern's property is `rdf:type`, constant URI is set in object position. The restrictions are similar to ones used in Pellet [26] and allows goals-collecting. It starts from parsing of the RDF graph pattern in the query. For each triple pattern $\{$`s p o`$\}$, the goal pattern is constructed in the following way: its first element (`prop_id`) is set to `id(p)`;the second element (`subj_id`) is set to `NULL`, if `s` is a variable, and to `id(s)` else; the third element (`obj_id`) is computed by `o` in the same way as `subj_id`. Usage of identifiers (`id(·)`) allows to avoid access to the dictionary during all reasoning stages till the answer is returned to the user. Exactly one goal pattern corresponds to a goal. Thus, goals with the same patterns are called *equal*, and goals are used further in the paper instead of goal patterns for better understanding.

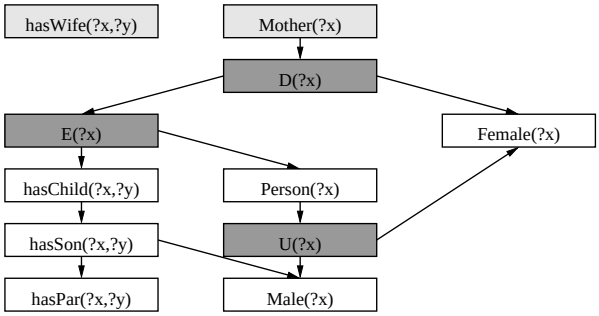

Figure 3: Goals collecting

Let the ontology (1)-(6) extended with the assertions

$$Male(Peter) \tag{16}$$

$$hasWife(Alex, Anna) \tag{17}$$

is loaded into the database, and the following query is given to find anyone whose wife is somebody's mother:

$$\{\texttt{?m hasWife ?w}\}.\{\texttt{?w rdf:type Mother}\} \tag{18}$$

Goals in the rectangles filled with light-grey on Fig. 3 are created by the query. Having these two goals, only entailment rules on base of (7) can be used to find instances of $Mother$, but D is auxiliary concept without explicitly asserted instances. Therefore, new goal is added to the *goals set*. Again, these goals are not enough to answer the query, and (8) allows new goals to be added. Goals-collecting process for the example is depicted on Fig. 3, where dark-grey fill is used to outline goals for auxiliary concepts.

Goals collecting stops, when the goals set, containing some goal g, also contains all goals on which g *depends*, where the notion *depends* is defined as follows:

Definition 3. *We say that a goal g_1 depends on a goal g_2, if there exists some statement which can be used to entail triples corresponding to g_1 on base of triples corresponding to g_2.*

Dependencies are determined on base of SWRL rules and OWL constructs. Therefore, it can be said that we have a set of dependencies built as result of parsing and extended during TBox reasoning (see Sect. 4.3.2). Arrows on Fig. 3 illustrates dependencies determined for the example ontology.

Algorithm 1 describes the goals-collecting process more precisely. Addition of a goal g to the set G in line 4 of the algorithm can be implemented as $G \cup \{g\}$. Note that if goals g and $g\prime$ are equal as defined above (e.g. $g = R(?x1, ?y1)$ and $g\prime = R(?y1, ?x1)$), then $\{g\} \cup \{g\prime\} = \{g\}$. Algorithm 2 of adding goals to a set can be used in line 4 of Algorithm 1 to reduce size of obtained set of goals.

A notion of more general goal is used in Algorithm 2, and we define it in the following way:

Algorithm 1 Collect goals using dependencies

Input: A set G of goals and a set D of dependecies

1: $G' := G$
2: **for all** $g \in G : g = p(s, o)$ **do**
3: **for all** $d \in D : (d =< con, anc > \wedge \exists\theta - substitution : (con)\theta = g)$ **do**
4: **add** $(anc)\theta$ to G'
5: **if** $G \neq G'$ **then**
6: $G := G'$
7: **goto** step 2

Definition 4. *We say that goal pattern p_1 is more general than goal pattern p_2, if p_1 can be obtained from p_2 by substitution of* `subj_id` *or(and)* `obj_id` *with* `NULL` *value(s). We say that goal g_1 is more general than goal g_2, if $g_1(g_2)$ has pattern $p_1(p_2$, resp.), and p_1 is more general than p_2.*

This relation defines partial order on the set of all goals. For instance, goal $P(?x, ?y)$ is more general than $P(s_1, o)$ and $P(s_2, ?x)$, goal $R(s_1, ?x)$ is more general than $R(s_1, o)$, but goal $P(s_1, ?z)$ is not comparable by this relation with $P(?x, o)$ or $P(s_2, ?z)$ or $R(s_1, ?z)$, where $s_1 \neq s_2$. Obviously, if a set of goals contains a goal g' more general than a goal g, then the goal g will be redundant in this set. The goal g does not correspond to any entailment command which does not correspond to g'. Moreover, if both goals, g and g', are in the set, some commands will be executed twice to entail the same triples, and it degrades reasoning performance. Note also that addition of a goal to the set in line 15 of Algorithm 2 means recurrent call of the algorithm, unlike lines 7,9,20 and 22.

Algorithm 2 Add a goal to set of goals

Input: A set G of goals, a goal g, a number $L > 0$

1: **if** not (g $\in G \vee \exists$g′ $\in G$: g′ is more general than g) **then**
2: $G := G \backslash \{g' \mid g \text{ is more general than } g'\}$
3: **if** g.prop = `rdf:type` **then**
4: Dub_g := {g′ | g′.prop = `rdf:type` $\wedge$ g′.obj = g.obj}
5: **if** $|\text{Dub}_g| = L$ **then**
6: g.subj := NULL
7: $G := G \setminus \text{Dub}_g \cup \{g\}$
8: **else**
9: $G := G \cup \{g\}$
10: **else**
11: $\text{Dub}_g^O := \{g' \mid g'.\text{prop} = g.\text{prop} \wedge g'.\text{obj} = g.\text{obj}\}$
12: **if** $|\text{Dub}_g^O| = L$ **then**
13: g.subj := NULL
14: $G := G \setminus \text{Dub}_g^O$
15: **add** g to G
16: **else**
17: $\text{Dub}_g^S := \{g' \mid g'.\text{prop} = g.\text{prop} \wedge g'.\text{subj} = g.\text{subj}\}$
18: **if** $|\text{Dub}_g^S| = L$ **then**
19: g.obj := NULL
20: $G := G \setminus \text{Dub}_g^S \cup \{g\}$
21: **else**
22: $G := G \cup \{g\}$

Algorithm 1 terminates, even if infinite recursion in dependencies occurs. Let N_P is a number of all properties (except `rdf:type`) used in concept descriptions, SWRL rules or `subProperty` axioms, N_C is number of concepts used in some OWL statements (axioms, descriptions) or SWRL rules. Let also I is number of all resources and literals in the dictionary. Then size of a set of goals obtained by the Algorithm 1 is limited by $I^2 \cdot N_P + I \cdot N_C$ if Algorithm 2 is not used, and by $L \cdot (I \cdot N_P + N_C)$ else. In the former case the set of goals enlarges on all iterations (except the last one), and finiteness of Algorithm 1 is obvious. For the latter case finiteness can be shown, taking into account that the set can be reduced on some iterations only in controlled manner.

Let us consider now implementation aspects of this stage. Set of dependencies used for goals collecting can be implemented as additional table. However, it is enough to create the view `depends` defined as union of selects from the statements tables for this aim. Command `UNION ALL` is used for better performance, and consequently the view can be not a set. Such implementation does not violate algorithms requirements, but *goals table* implementing set of goals must have no duplicates.

The algorithms can be implemented as a stored procedure which by the input RDF graph pattern constructs goal patterns and inserts it into a goals table, and then searches in `depends` view and the goals table to construct new goals and insert them into the goals table too. Absence of duplicates in the goals table can be guaranteed by the procedure or a trigger avoiding insertion of duplicates into it.

However, we propose to use rules and triggers for the algorithms implementation. Two tables were created: `old_tgt` implements a set of goals and `new_tgt` is used to solve problems with recursion in rules. For all stored statements rules are created in the database; they are fired, when certain goals are inserted into `old_tgt`, and insert new goals into `new_tgt`. For instance, for `subProperty` assertions rule `R1` is created, and for the rule (6) rules `R2` and `R3` will be created.

```
R1: CREATE RULE sp_dep
    AS ON INSERT TO old_tgt
    DO INSERT INTO new_tgt
    sp.subj_id, NEW.arg1_id, NEW.arg2_id
    FROM subprop sp
    WHERE sp.obj_id = NEW.prop_id;

R2: CREATE RULE rule_hs1
    AS ON INSERT TO old_tgt
    WHERE NEW.prop_id = id(hasSon)
    DO INSERT INTO new_tgt
    id(hasPar), NEW.arg2_id, NEW.arg1_id;

R3: CREATE RULE rule_hs2
    AS ON INSERT TO old_tgt
    WHERE NEW.prop_id = id(hasSon)
    DO INSERT INTO new_tgt
    id(rdf:type), NEW.arg2_id, id(Male);
```

Trigger is created on `new_tgt` to implement Algorithm 2 instead of insertion of goals into this table.

Thus, the stored procedure needs only to construct goals by the input RDF graph pattern and insert them into `new_tgt`. All other goals will be collected automat-

ically by fired rules and triggers. This approach makes code of the stored procedure simpler and reduces number of searches in the goals tables. Moreover, it is more flexible: addition of new kinds of dependencies does not require modifying a code of the stored procedure – it is enough to add new rules on `old_tgt`.

4.2.2 Obtaining Answer

When goals are determined, triples corresponding to them should be entailed to answer the query. One possible way to do it is to create a stored procedure which should look at the goals table and for each goal in this table search in the statements tables which statements correspond to the goal and then execute SQL commands built by found statements.

Another way is to create a simpler stored procedure which uses collected goals to invoke tailored rules and triggers. Comparing to the first one, it avoids look ups on the statements tables and it is more flexible: adding or changing of logic constructs demand writing of new triggers and rules[5], not rewriting the whole procedure.

The second way was chosen, and the auxiliary table `tasks` (Fig. 2 (n)) was tailored for creating rules. When some goal is inserted into the table, rules which can entail triples corresponding to the goal are fired. All these rules are created with `INSTEAD` modifier to prevent actual insertion of triples into `tasks` table. There are two ways to create these rules: one rule can be created in advance for all logic statements of some kind (e.g., rule `R4` supports entailment for all `subPropertyOf` statements) or one rule can be created for every particular statement (e.g., rule `R5` is created on base of (6)).

```
R4: CREATE RULE subpropof_rule
    AS ON INSERT TO tasks
      WHERE sp_flag IS TRUE
    DO INSTEAD INSERT INTO tr
    SELECT tr.subj_id, NEW.prop_id,
           tr.obj_id, i.num+1
    FROM tr, subprop sp, iternum i
      WHERE tr.prop_id = sp.subj_id
        AND sp.obj_id = NEW.prop_id
        AND tr.isinf >= i.num;

R5: CREATE RULE hasson_rule
    AS ON INSERT TO tasks
      WHERE prop_id = id(hasSon)
    DO INSTEAD INSERT INTO tr
    SELECT t1.obj_id, NEW.prop_id,
           t1.subj_id, i.num+1
    FROM tr t1, typetrip t2, iternum i
      WHERE t1.prop_id = id(hasPar)
        AND t2.subj_id = t1.subj_id
        AND t2.obj_id = id(Male)
        AND (t1.isinf >= i.num
          OR t2.isinf >= i.num);
```

When there are too much rules created in the database, it is hard to understand, how conclusions were computed, or to find possible errors. On the other hand, usage of rules created for particular statements can be more effective. Firstly, such rules have more serve firing conditions. Secondly, they do not need to look in the statements tables (e.g., `subprop` table is selected in rule `R4`, and the SWRL rules tables are not selected in rule `R5`). Moreover, if trigger or rule is created to support all SWRL rules, it will generate SQL commands for every appropriate SWRL rule every time it is fired.

Therefore, both kinds of rules are used: new rules are created for every stored concept descriptions and SWRL rules, and other rules are created for other kinds of logic statements. Special columns in `tasks` and `old_tgt` tables are used to avoid firing of the latter rules for goals not requiring it. For instance, flag `sp_flag` is set to TRUE for some goal, if its property has subproperty, and only such goals can fire rule `R4`.

Algorithm 3 Entailment of triples driven by collected goals

Input: A set G of goals, a set D of dependencies, a set R of entailment commands, and a set T of triples. Let also two boolean flags `has_anc` and `inf_new` are assotiated with every goal g$\in G$.

Output: The set T enriched with implicit triples, corresponding to goals from the set G

```
1:  for all goal g∈G do
2:    g.has_anc := TRUE
3:    g.inf_new := FALSE
4:  while ∃g∈G:has_anc IS TRUE do
5:    T_new := T
6:    for all g∈G:has_anc IS TRUE do
7:      T_new := T_new ∪ { t | ∃r∈R: r corresponds to g
        and it's evaluation on T_new entails t }
8:    for all g∈G do
9:      if exist t∈ (T_new \ T): t corresponds to g then
10:       g.inf_new := TRUE
11:     else
12:       g.inf_new := FALSE
13:   for all each g∈G do
14:     if exists g' ∈G,d∈D: g'.inf_new IS TRUE ∧ d
        = (g,g') then
15:       g.has_anc := TRUE
16:     else
17:       g.has_anc := FALSE
18:   T := T_new
```

We propose Algorithm 3 for entailment of implicit triples related to the query. Flags `has_anc` and `new_anc` correspond to fields of the same names in `old_tgt` and `tasks` tables. These flags allow execution of commands, which knowingly can not entail new triples, to be avoided. Indeed, `has_anc` flag is set to TRUE for some goal only if at least one triple is entailed on the last iteration of the algorithm, which can be used in entailment of triples corresponding to this goal.

Probability of entailment of duplicate triples is reduced using `isinf` field: it is set to number of iteration on which triple was entailed (and to 0 for asserted ones); and additional conditions on this field are used in rules to prevent entailment on base of only triples already taken into account at the previous iterations. Usage of `isinf` (as `sp_flag`) is outlined with italics in rules `R4` and `R5`.

In the example rules (`R4`-`R5`) insertion is performed into `tr` view for simplicity. Actually, rules can be created to insert into appropriate property tables. Rules corre-

[5] Here an anology with modular programming can be drawn

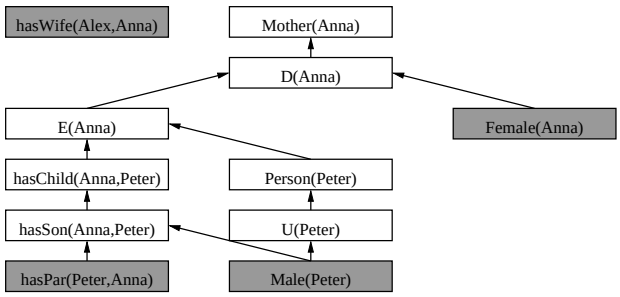

Figure 4: Entailment to answer the query

sponding to `rdf:type` and properties not clustered into separate tables insert implicit triples into intermediate tables. The stored procedure eliminates duplicates from these tables, and only then inserts new triples into appropriate tables.

Entailment process for answering the query (18) is depicted in Fig. 4. Note that 3 rules support (1): `R4` is applied to (7) and two rules like `R5` are tailored for (8) and (9). Number of iterations of Algorithm 3 in this example can vary from 2 to 6 depend on order of rules firing and usage of intermediate tables.

Table `taskdep` was created to realize the D set. It populates after all goals are determined, and before starting of the entailment algorithm in the following way: values of its fields are unique identifiers automatically generated for each goal in `old_tgt` table, and if *tup* is a row in the `taskdep` table, it means that the goal with id *tup*.`con_id` depends on the goal with id *tup*.`anc_id`. This table makes it easier to find dependencies between goals by reducing this task to search in one table by integer field and avoiding search in multiple dependency tables with more complicated conditions.

Algorithm 3 terminates, because the set T expands on all iterations, and its size is limited at worst by N^3, where N is number of entities in the dictionary. Although SWRL is known to be undecidable, our estimate is correct, because anonymous individuals are not generated during ABox reasoning. Only resources from the dictionary are considered, when SWRL rules are used for entailment. Therefore, even Datalog safeness is not mandatory for SWRL rules, because variables violating it can have only values from the dictionary. The same is done in [20] by addition of O predicate to the body of rules. However, true support of SWRL built-ins will require some safeness conditions.

4.3 TBox Reasoning

The following ways can be used to build a concept hierarchy (i.e. to determine relations between concepts):

- Construct an OWL document on base of stored information about concept descriptions and relations between them, use some complete reasoner (e.g. Pellet [26]) for TBox reasoning on it, and then store implicit relations in the database. The database schema used in our work allows constructing such OWL document easily. This approach can be practical in cases, when size of stored ontologies is not very large and completeness of TBox reasoning is important.
- Perform reasoning using DBMS features. In this case reasoning will be sound, but may be incomplete. However, as it was mentioned above, ontologies with large number of named concepts are appearing, which are serious challenges for complete DL reasoners, and usage of such reasoners can be unpractical in certain applications. So we implement TBox reasoning in database as it is presented below in this Section.

4.3.1 Strategy of TBox Reasoning

Result of TBox reasoning is determination of implicit relations (`subClassOf`, `equivalentClassOf` and `disjointWith`) between concepts (both named and anonymous) and role inclusion axioms. To solve this task, a set of inference rules easily translated into SQL queries over the statements tables is determined on base of OWL constructs. It includes the following ones:

$$C_1 \sqsubset C_2 \wedge C_2 \sqsubset C_3 \implies C_1 \sqsubset C_3 \quad (19)$$

$$D \; is \; C_1 \sqcup \ldots \sqcup C_i \sqcup \ldots \sqcup C_n \implies C_i \sqsubset D \quad (20)$$

$$\left.\begin{array}{r} D_1 \; is \; C_1 \sqcup \ldots \sqcup C_n \\ \forall i = 1, \ldots, n : C_i \sqsubset D_2 \end{array}\right\} \implies D_1 \sqsubset D_2 \quad (21)$$

$$\left.\begin{array}{r} D_1 \; is \; \exists P.C_1 \\ D_2 \; is \; \forall P.C_2 \\ C_1 \sqcap C_2 \sqsubset \bot \end{array}\right\} \implies D_1 \sqcap D_2 \sqsubset \bot \quad (22)$$

The simplest algorithm for TBox reasoning is to execute determined rules iteratively until no more relations can be entailed. However, as since the set of inference rules is static, it is possible to analyze dependencies between rules and to determine order of rules execution which allows obtaining the same set of relations more effectively. For instance, rules (19)-(20) produce and depend on `subClassOf`-axioms and they are executed together in one loop. Rule (22) produces and depends on `disjointWith`-axioms and can be executed once before or after the loop. Thus, process of TBox reasoning consists of a sequence of commands and iterations of commands. It terminates, because number of relations increases on all iterations and is limited by $O(N_C^2 + N_P^2)$, where N_C is a number of stored concepts and N_P is a number of stored properties.

4.3.2 Influence on ABox reasoning

TBox reasoning produces information about concepts from the ontology which itself can be interesting for the user. In addition, it can improve ABox reasoning. Goals collecting is driven by the existing set of dependencies which can be extended during TBox reasoning.

Let us consider axioms (10) and (11) represented as (12)-(15). Let it is also known that some individual r is instance of either C_1 or C_2 (i.e. $G(r)$). Although axioms (10) and (11) imply $D_1(r)$, the Abox reasoning algorithm doesn't allow one to find any instance of D_1 using only (12)-(15). The situation is changing, if TBox reasoning was performed previously. Application of (20) to (12)-(13) entails that C_1 and F are subclasses of D_1, and F itself has C_2 as subclass. This allows (19) to entail that C_2 is also subclass of D_1. The entailed axioms and (14) are used by (21) to entail that G is subclass of D_1 what allows to find that r is instance for D_1.

Therefore, performing TBox reasoning before ABox reasoning allows obtaining more complete answers.

5 RDF Querying

5.1 Implementation

Table function RDF_QUERY was implemented to provide users ability to access RDF data. Its input is SPARQL-like RDF graph pattern with restrictions described in Sect. 4.2.1 and result is table with columns corresponding to the variables in the given pattern. The function parses the pattern to build goal patterns and SQL command to retrieve results. Then it inserts the generated goal patterns into new_tgt to start goals collecting. When goals are collected, it invokes the entailment procedure implementing Algorithm 3. Finally, it executes built SQL command to return answer on the query. Thus, the user does not need to know actual database and reasoning schemas to access RDF data.

As [29] notes, caching previous query results or full pre-computation can speedup or avoid reasoning at the query time. Although TBox reasoning is pre-computed, the presented approach also allows pre-computation and caching previous results for ABox reasoning. Since all entailed triples are stored in the same tables as asserted ones, it is enough to not remove them after query answering to apply caching. Stored procedure was tailored for full pre-computation. It creates goals from consequences of all stored dependencies, inserts them into old_tgt and invokes the entailment procedure. Table function RDF_QUERYWOI was created to access RDF data, when reasoning is pre-computed. It has the same interface with RDF_QUERY, and the difference is that it does not construct goals and perform reasoning.

5.2 Query examples

This Section shows some examples of queries to RDF data with reasoning on base of OWL statements and SWRL rules. The examples are queries to the Family ontology from [2]. In this ontology most family relations (including that used below) are asserted implicitly using SWRL rules and OWL statements (including property restrictions, intersections, etc.). Query Q1 returns names of all persons who are known to be sisters. Note that prefix p is specified for conciseness of the query.

```
Q1: SELECT n1,n2 FROM RDF_QUERY(
    '{!PREFIX p http://a.com/ontology#}
    {?s p:hasSister ?m}
    {?s p:name ?n1}{?m p:name ?n2}')
    AS res(s text,m text,n1 text,n2 text);
```

5.2.1 Restriction Definitions

Our extension of SPARQL graph pattern is ability to define restrictions. Query Q2 demonstrates how restrictions can be defined and used. The query returns names of all persons who are known to have at least 2 brothers.

```
Q2: SELECT n FROM RDF_QUERY(
    '{!PREFIX p http://a.com/ontology#}
    {!RESTR r p:hasBrother [minCard 2]}
    {$s rdf:type $r}{$s p:name $n}')
    AS res(s text,n text);
```

If this restriction is not stored in the database, the query procedure will store it into the corresponding table and create inference rule for it. RDF_QUERYWOI also entails instances of it. It can be not desired to store all restrictions defined in queries. Therefore, table tmp_cls stores for each such restriction number of queries utilizing it, and this information can be used to decide either it should be stored or deleted from the database.

Persons with at least 2 brothers also can be found with query Q3 which does not use restriction definitions. Though, usage of restrictions has several advantages: First, a user does not need to care how RDF data is stored, and the database schema changes do not cause rewriting of queries. In contrast, explicit selections from triples and the dictionary (denoted by get_id) are used in query Q3. Second, it is easier to write and understand query Q2 than Q3. Note that although restriction to have at least 2 brothers is implemented as outer condition in Q3, the pattern with property hasBrother is added to infer all persons who have a brother to obtain expected answer[6]. Third, additional condition in RDF_QUERY may reduce number of rows returned through table function interface. Finally, when query with definition of restriction is executed, information about individuals satisfying it is stored in the database, and will be used for following queries. So if it is queried frequently, query Q2 is preferable. And query Q3 may be preferable else.

```
Q3: SELECT n FROM RDF_QUERY(
    '{!PREFIX p http://a.com/ontology#}
    {$s p:name $n}{$ p:hasBrother $b}')
    AS res(s text,n text,b text) WHERE
    (SELECT count(*) FROM triples WHERE
     subj_id = get_id(res.s) AND prop_id =
     id('http://a.com/ontology#hasBrother')
    ) >= 2;
```

5.2.2 Data Integration

Query Q3 shows that data returned by RDF_QUERY can be interpreted as an ordinary table: arbitrary conditions can be used for selection from it, some statements, aggregates can be computed on returned data, there can be joins with other tables in the database, even not only used for storing of RDF data. Thus, we have a way to integrate data stored in relational databases with Semantic Web data. For instance, let we have a table emp(emp_name,dep_id) which is used to store names of employees and numbers of their departments. Then we can find all persons who work with their brothers in the same department using query Q4:

```
Q4: SELECT e1.* FROM RDF_QUERY(
    '{!PREFIX p http://a.com/ontology#}
    {?s p:hasBrother ?m}
    {?s p:name ?n1}{?m p:name ?n2}')
    AS res(s text,m text,n1 text,n2 text),
    emp e1, emp e2
    WHERE e1.emp_name = n1
     AND e2.emp_name = n2
     AND e1.dep_id = e2.dep_id;
```

[6] it is not required for RDF_QUERYWOI

5.2.3 SPARQL Features

Although only core feature of SPARQL is supported, SQL power can be used in outer query to implement more comprehensive queries to RDF data with such SPARQL features as FILTER, OPTIONAL, UNION. Query `Q5` implements the SPARQL query on Fig. 1 which uses UNION[7]:

```
Q5: SELECT n1,n2 FROM (
    SELECT * FROM RDF_QUERYWOI(
    '{!PREFIX p http://a.com/ontology#}
    {?s p:hasAunt ?m}
    {?s p:name ?n1}{?m p:name ?n2}'
    ) AS res (s text,m text,n1 text,n2 text)
    UNION
    SELECT * FROM RDF_QUERYWOI(
    '{!PREFIX p http://a.com/ontology#}
    {?s p:hasUncle ?m}
    {?s p:name ?n1}{?m p:name ?n2}'
    ) AS res (s text,m text,n1 text,n2 text)
    ) AS foo;
```

5.3 Completeness and Soundness

In researches on translation of DL knowledge bases into Datalog rules usage of DL constructs is often restricted to adhere to completeness of reasoning. For instance, in DLP [12] it is forbidden to use `allValuesFrom` restrictions on the left-hand side of `subClassOf` axioms, and `someValuesFrom` restrictions and `unionOf` construct on the right-hand side. Our approach does not restrict OWL DL somehow. Although reasoning may be incomplete in this case (e.g. constructs with non-determinism are not supported fully), query answering for more expressive ontologies is allowed.

Let consider the following toy ontology:

$$MovieLover \sqsubset \exists interestedIn.Movies \quad (23)$$

$$SportFan \sqsubset \forall interestedIn.Sports \quad (24)$$

$$Sports \sqcap Movies \sqsubset \bot \quad (25)$$

$$(\geqslant 2\, hasFriend) \sqsubset Sociable \quad (26)$$

$$SportFan(Peter) \quad (27)$$

$$MovieLover(Anna) \quad (28)$$

$$hasFriend(Alex,\, Peter) \quad (29)$$

$$hasFriend(Alex,\, Anna) \quad (30)$$

This ontology is not in DLP, and our approach allows someone to find individuals of the concept $Sociable$ described in it. First, it applies (22) to (23), (24) and (25) during TBox reasoning to entail

$$MovieLover \sqcap SportFan \sqsubset \bot \quad . \quad (31)$$

Then it uses (31) and the assertions (27)-(28) to entail

$$\texttt{differentFrom}\,(Anna,\, Peter) \quad . \quad (32)$$

Finally, it uses the axiom (26), the assertions (29), (30) and (32) to entail the answer

$$Sociable(Alex) \quad . \quad (33)$$

[7] If `RDF_QUERY` were used in `Q5`, ABox reasoning would be performed twice.

Table 1: LUBM(1) – LUBM(10) experimental results

Explicit	Inferred	Reason (m:s)	Query (ms)
100808	49725	00:22	18.54-732.46
230345	112802	00:59	51.16-1044.4
337427	165228	01:31	59.66-1208
478105	233710	02:12	53.74-1496.8
624875	305373	03:02	79.57-1760.1
723311	353275	03:35	96.77-1924.5
884525	432072	04:34	137.4-2513.4
1001817	489582	04:43	155.6-2715.2
1125031	549965	05:17	161.8-3156.6
1273014	622223	06:06	176.8-3875.6

Although answers for some queries may be incomplete, all answers are sound. Inference rules were implemented to apply open-world semantics of DLs, when closed-world (and closed-domain) semantics are more natural for databases. For instance, the presented approach does not entail that any individual from the ontology (23)-(30) is instance of concept described as $(\leqslant 2\, hasFriend)$. Indeed, although all the individuals have no more than 2 objects for $hasFriend$, the ontology does not say that they can not have more objects for it. For in-depth description of differences between the semantics please refer to [22].

Further, Unique Name Assumption (UNA) is often used in researches for DLs. It assumes that individuals with different names are different. Therefore the answer (33) can be returned without entailment of (31) and (32). Although UNA simplifies reasoning, it is not applicable for OWL, because of its distributed nature. Therefore, it is not applied in the implementation of inference rules.

Moreover, although answers on RDF queries are incomplete in the general case, it might be enough to handle practical ontologies. Popular benchmark LUBM [13] was used for experiments with the implementation. The testing system is Intel Pentium IV 2.6 GHz machine with 2 Gb of main memory running Linux, PostgreSQL 8.2. ABox reasoning was pre-computed. The results for the first ten universities (number of explicit and inferred triples, reasoning time, minimal and maximal querying time) are shown in Table 1. Good performance and scalability of reasoning and query answering were obtained in the experiments. However, experiments with greater number of universities are planned to examine scalability further. Answers for all the queries from the benchmark are sound and complete.

Reasoning for the Family ontology was performed during querying. Number of inferred triples is about 1400, and query time is less than 3 seconds. All the family relations were determined fully and correct, and the most of concepts were instantiated, except $Sibling$ which uses restriction $(\forall\, hasSibling.Sibling)$ in its description. It does not cause infinite recursion during reasoning, but can't be instantiated in our approach. Its elimination from the description allows $Sibling$ to be instantiated correctly.

Because the approach allows only sound answers for queries, it can be used for partial performing of standard DL reasoning tasks in the following sense. If answer set for the query {`?x rdf:type owl:Nothing`} is not empty, it can be said that the stored ontology is

inconsistent. However, if the set is empty, it can be said nothing about consistency of the ontology. Therefore, the approach can be used also for ontology debugging.

6 Related Work

Researches on usage of relational DBMS in DL reasoning with large-scale datasets had begun before the notion of the Semantic Web was proposed. For example, in [9] a problem of effective reasoning in knowledge base management system CLASSIC with large-scale dataset stored in relational databases is regarded.

Early systems supporting OWL use in-memory reasoners based on the tableaux algorithms. Some of them utilize databases to persist large RDF data sets and often do not even leverage their query optimization power. Thus, Sesame [10] uses a special module which decomposes conjunctive queries to a set of SQL queries to reduce influence of a particular DBMS. As since much attention was paid to effective retrieval of RDF triples from databases, it results in development of various database schemas for storing RDF data (e.g. [3, 10, 11, 25, 30]).

The OWL instance Store [6] is the system which uses a hybrid database/reasoner architecture to perform reasoning over large volumes of instance data. However, the system supports answering instance retrieval queries, not arbitrary SPARQL queries. It is also mentioned that role assertions are not supported.

In [21], a proposal for scalable query answering is presented. It is based on translating DL KB (OWL ontology) into DL2DB KB, which contains a fixed set of inference rules. These Datalog rules are used for ABox reasoning. As in our work, bi-directional strategy is proposed for Abox reasoning. However, there are several distinctions between these algorithms. Firstly, we suggest to implement ABox reasoning in DBMS, not in some middle-ware layer. Secondly, in [21] the set of inference rules is fixed and supports logic $\mathcal{SHI}$, less expressive than OWL. Finally, in contrast to simple iterative execution of all instantiated rules in BottomUp algorithm, we change a set of goals on all iterations of Algorithm 3 and propose some techniques to avoid redundant computations. Reasoning in [21] is performed during querying as in our work. However, in contrast to our work, neither pre-computation nor caching of previous queries results are used in [21]. Finally, TBox reasoning is performed using external reasoners.

Owlgres [27] is the OWL reasoner using PostgreSQL connected via JDBC to optimize query answering. Supported OWL dialect does not accept features like transitive properties and `someValuesFrom` restrictions in the l.h.s. of axioms. Thus, even LUBM ontology used in our experiments is not compatible with Owlgres.

These loosely-coupled approaches have several shortcomings: (1) DBMS users cannot reference ontology data directly, (2) the query processing and optimization power of a DBMS is not used fully. Inefficiency often is incurred by transformation of data from SQL to other formats. However, for the best of our knowledge, there are only a few proposals to use DBMS for ontology management without external reasoners.

Oracle implements inference engine for ABox reasoning, suggesting TBox reasoning to be performed using external reasoners. Initially, the Oracle's approach [11] supported reasoning based only on RDFS and user-defined rules with limited recursion. Inference possibilities were extended in Oracle 11g to support a subset of OWL not including several constructs (`unionOf`, `intersectionOf`, and cardinality restrictions). Unlike our approach, the inference engine in Oracle 11g does not obtain complete answers for several queries in LUBM benchmark [13] without external DL reasoner. As since the majority of nowadays data is stored in relational databases, ability to access RDF and relational data in one SQL query presented by both Oracle's and our approaches is very important.

Another proposal [18] is based on leveraging of XML support in hybrid relational-XML DBMSs to persist taxonomies contained in ontologies, and using SQL/XML and XQuery to query RDF data. However, only reasoning on base of transitive relations is considered in [18].

7 Conclusion and Future Work

The paper presents an approach to query answering and reasoning for Semantic Web data. Both ABox and TBox reasoning are performed using only RDBMS features. Most of them are common for the majority of RDBMSs. Use of the only exception (the rule system) is not mandatory, and it can be replaced by triggers. Therefore, the approach can be implemented in any RDBMS supporting triggers, stored procedures and table functions.

Following to proposals in [29], the approach does not adhere to completeness property of reasoning. This allows our approach to support sound reasoning for more expressive ontologies. If completeness is important for a certain application, our database schema allows usage of external state of the art reasoners to perform TBox reasoning. As since entailed triples are stored with extensional ones, it is possible to use caching of prior results for reasoning or even pre-compute it.

Although only core features of SPARQL language are implemented now, it is shown in the paper that more expressive features are supported using the implemented RDF query functions and SQL power. The approach also allows querying both data stored in a relational database and Semantic Web data simultaneously.

Future work may be directed on aspects of the Semantic Web which are not in the focus of researches' attention now, but could be very useful in practice: full support of data-valued properties and the Proof layer of the architecture [7].

8 Acknowledgments

The author thanks Alexander S. Markov for his contribution to the paper.

References

[1] PostgreSQL home page. www.postgresql.org.

[2] Protégé ontologies library. http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary.

[3] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. Scalable semantic web data management

using vertical partitioning. In *VLDB*, pages 411–422, 2007.

[4] M. Aslani and V. Haarslev. Towards parallel classification of TBoxes. In *Proc. of the 2008 International Workshop on Description Logics (DL 2008)*, volume 353 of *CEUR*, 2008.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[6] S. Bechhofer, I. Horrocks, and D. Turi. The OWL instance store: System description. In *CADE*, volume 3632 of *LNCS*, pages 177–181. Springer, 2005.

[7] T. Berners-Lee. Standards, semantics and survival. *SIIA Upgrade*, pages 6–10, June/July 2003.

[8] T. Berners-Lee, J. Handler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[9] A. Borgida and R. J. Brachman. Loading data into description reasoners. *ACM SIGMOD Record*, 22(2):217–226, 1993.

[10] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *Proc. of International Semantic Web Conference (ISWC)*, pages 54–68, 2002.

[11] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient SQL-based RDF querying scheme. In *VLDB*, pages 1216–1227, 2005.

[12] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *WWW*, pages 48–57, 2003.

[13] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[14] I. Horrocks. Semantic Web: The story so far. In *Proc. of the 2007 international cross-disciplinary conference on Web accessibility (W4A)*, pages 120–125, 2007.

[15] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[16] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *LPAR*, volume 1705 of *LNCS*, pages 161–180. Springer, 1999.

[17] U. Hustadt and B. Motik. Description logics and disjunctive datalog — the story so far. In *Proc. of the 2005 International Workshop on Description Logics (DL 2005)*, volume 147 of *CEUR*, 2005.

[18] L. Lim, H. Wang, and M. Wang. Semantic data management: Towards querying data with their meaning. In *ICDE*, pages 1438–1442. IEEE, 2007.

[19] F. Manola and E. Miller. RDF primer. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[20] J. Mei, H. Boley, J. Li, V. C. Bhavsar, and Z. Lin. The Datalog^{DL} combination of deduction rules and description logics. *Computational Intelligence Journal*, 23(3):356–372, 2007.

[21] J. Mei, L. Ma, and Y. Pan. Ontology query answering on databases. In *Proc. of International Semantic Web Conference (ISWC)*, pages 445–458, 2006.

[22] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and logic programming live together happily ever after? In *Proc. of International Semantic Web Conference (ISWC)*, pages 501–514, 2006.

[23] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, 10 February 2004. http://www.w3.org/TR/owl-semantics/.

[24] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/.

[25] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store support for RDF data management: not all swans are white. In *VLDB*, pages 1553–1563, 2008.

[26] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[27] M. Stocker and M. Smith. Owlgres: A scalable OWL reasoner. In *Proc. of the 5th OWLED Workshop on OWL: Experiences and Directions, co-located with ISWC 2008*, volume 432 of *CEUR*, 2008.

[28] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in database systems. In *Proc. ACM-SIGMOD Conference on Management of Data*, pages 281–290, 1990.

[29] R. Volz. Change paths in reasoning! In *Proc. of the 1st International Workshop "New forms of reasoning for the Semantic Web: scalable, tolerant and dynamic", co-located with ISWC 2007 and ASWC 2007*, volume 291 of *CEUR*, 2007.

[30] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In *Proc. of SWDB'03, First International Workshop on Semantic Web and Databases*, pages 131–150, 2003.