Int J Software Informatics, Vol.4, No.1, March 2010, pp. 23–37 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
 Tel: +86-10-62661040

Mapping Relational Databases to the Semantic Web with Original Meaning*

Dmitry V. Levshin

(Dept. of Computational Mathematics and Cybernetics, Moscow State University, Moscow, Russia)

Abstract Most of nowadays Web content is stored in relational databases. It is important to develop ways for representation of this information in the Semantic Web to allow software agents to process it intelligently. The paper presents an approach to translation of data, schema and the most important constraints from relational databases into the Semantic Web without any extensions to the Semantic Web languages.

Key words: semantic Web; relational databases; constraints

Levshin DV. Mapping relational databases to the semantic Web with original meaning. *Int J Software Informatics*, **2010, 4(1): 23–37. http://www.ijsi.org/1673-7288/4/i42.htm**

1 Introduction

The Semantic Web is aimed in allowing computer agents to process intelligently information from different sources. To use semantic technologies, one needs to represent information in standardized machine readable data formats. The World Wide Web Consortium has recommended several formats for representing data in the Semantic Web including RDF, RDF Schema and OWL. The more data is available in the Semantic Web, the more powerful these technologies become.

Wikipedia[36] and databases[2] are good candidates for populating the Semantic Web, because they contain great amounts of information in a structured form. Moreover, today the majority of Web content (so-called Deep Web) is stored in relational databases. Ability to publish it in the Semantic Web is important not only for development of the latter, but also allows search engines to return more relevant Deep Web search results[11]. Mappings databases to OWL ontologies are also applicable in integrating distributed data sources[34].

Many efforts (e.g. Refs.[5, 34]) are concerned on translation of data and schema from databases into OWL and mostly disregard relations implicitly given by primary and foreign key constraints. Several reasons not to loose this information during mappings are presented in Ref.[21]: it can be useful if an exported RDF graph is imported in a relational database at another place, or if updates on a materialized RDF graph have to be performed such that key and foreign key properties have to be checked. It also can be used for semantic query optimizations.

However, semantics of OWL does not allow it to represent integrity constraints including relational keys and it leads to development of different extensions to OWL[8,26].

---

* This paper is extended version of the paper presented at KSEM 2009.
Corresponding author: Dmitry V. Levshin, Email: dm.levshin@gmail.com
Received 2009-07-15; revised 2010-03-21; accepted 2010-03-31.

At the same time, computational and conceptual complexity of the existing formats is relatively high. As consequence, practical applications often use very poor ontologies[15], and there are proposals to simplify OWL for better performance and scalability[35].

Therefore, it seems natural to use for mappings existing languages without any extensions. Two formats are useful in this case: SWRL is suggested as rule language and SPARQL is recommended as query language for the Semantic Web.

The main contribution of the paper can be summarized as follows:

- *an approach to mapping databases to the Semantic Web with the most important constraints is proposed;*
- *it uses the full strength of the existing formats without any extensions.*

Important aspect of the approach is usage of SPARQL. It is also proposed in Ref.[22] to use RDF and SPARQL for representing constraints. In contrast to Ref.[22], the approach presented in the paper does not propose any extensions to SPARQL and performs mappings in more natural way due to using SWRL.

The paper is organized as follows. Section 2 overviews basic notions of relational model and the Semantic Web. Section 3 presents our approach to mapping of relational databases into ontologies and verifying database-style constraints. Section 4 discusses some reasoning aspects involved in the approach. Section 5 overviews related work before Section 6 concludes.

2 Preliminaries

This Section briefly recalls basic notions of relational databases that are considered in the paper and then introduces the Semantic Web languages.

2.1 Relational terminology

A *relation schema* Hr is a set of ordered pairs $\langle A, T\rangle$, where A is a name of an *attribute*, and T is it's domain. A *tuple* t corresponding to the schema Hr is a set of ordered triples $\langle A, T, v\rangle$, one triple for each attribute in Hr, where v is a value of the attribute (written $t.A = v$). Relation body Br is a set of tuples corresponding to Hr. A relation *instance* (*relational table*) is a pair $\langle Hr,\ Br\rangle$.

A *potential key* over a relation instance is a subset K of it's attributes such that the following holds:

$$\forall t_1, t_2 \in Br:\ t_1 \neq t_2 \ \longrightarrow\ \neg(\forall A \in K : t_1.A\ =\ t_2.A)\ , \tag{2.1}$$

and there is no any $K' \subset K$ for which (2.1) holds too. Any attribute from K can have `NULL` value. Although $(a = \texttt{NULL})$ in general is computed to be *unknown* for each value a, `NULL` values are considered to be equal in (2.1). A *primary key* PK is a potential key such that no any attribute from PK can be `NULL`.

A *foreign key* over a relation instance R_1 is a subset FK of its attributes such that a relation instance R_2 with the potential key FK exists and the following condition holds:

$$\forall t_1 \in Br_1 :\ \begin{array}{l}(\forall A \in FK :\ t_1.A\ \texttt{IS NULL}) \quad \vee \\ (\exists\, t_2 \in Br_2 :\ \forall A \in FK :\ t_1.A = t_2.A)\end{array}\ . \tag{2.2}$$

Most commercial RDBMS are not pure relational. More precisely, they are based on SQL data model. It has several significant differences from relational model. For instance, there are three kinds of foreign key constraints in SQL:

- `MATCH SIMPLE`. It is satisfied, if the following condition holds for all tuples $t_1 \in Br_1$: (1) some attribute $A \in FK$ contains `NULL` or (2) exactly one tuple $t_2 \in Br_2$ exists such that value of the foreign key of t_1 coincides with value of a potential key of t_2.
- `MATCH PARTIAL`. It is satisfied, if the following condition holds for all tuples $t_1 \in Br_1$: (1) all attributes $A \in FK$ contain `NULL` or (2) at least one tuple $t_2 \in Br_2$ exists such that $\forall A \in FK : (t_1.A \texttt{IS NOT NULL}) \longrightarrow (t_1.A = t_2.A)$.
- `MATCH FULL`. This constraint is the same as constraint (2.2) in relational algebra.

In addition to primary and foreign keys constraints, `CHECK`-constraints can be specified on particular attributes, tables or on the whole database. A database is consistent, if all specified constraints hold.

2.2 Semantic Web language stack

The Semantic Web has stack architecture assuming that upper layers (languages) are based on underlying ones and extend them with new features.

First, *Resource Description Framework* (RDF) uses triples of the form *'subject-property-object'* with URIs to represent data. RDF documents can be presented in graph notation and, thus, are also called RDF graphs.

Next, *RDF Schema* and *Web Ontology Language* extends RDF with the ability to express statements about domain of interest. OWL species, OWL Lite and OWL DL[1], are based on Description Logics (DLs)[3]. DLs allow definition of relations between binary *roles* and unary *concepts*. DLs differ one from each other in that which constructs and in which way are allowed for concept description. Usually these constructs include set operations (intersection, union, complement) and property restrictions.

$$Person \sqcap (\geqslant 1\, hasChild) \sqsubseteq Parent \tag{2.3}$$

For instance, the inclusion axiom (2.3) states that all instances of class $Person$, who has at least one value for property $hasChild$, are also instances of class $Parent$. Although OWL allows usage of restrictions in concept descriptions, it should be clear that OWL restrictions are not the same as database-style constraints. While constraints specify conditions to be satisfied by stored data, restrictions are used to infer some new information from it. For instance, let an ontology consist of (2.3) and the following assertions:

$$Person(Peter) \tag{2.4}$$

$$hasChild(Peter,\, Pablo) \tag{2.5}$$

[1] OWL Full is not considered in the paper, because it is undecidable and not supported by many popular tools.

Treating (2.3) as constraint, one can decide that the ontology is inconsistent due to absence of the following assertion:

$$Parent(Peter) \ . \tag{2.6}$$

On the contrary, OWL reasoners will use (2.3) to infer (2.6). This difference between semantics is studied in Refs.[8, 26].

SWRL[17] extends OWL DL with the ability to specify Horn-like rules, where only unary and binary predicates are permitted. Although SWRL rules are similar to Datalog rules, SWRL applies OWL semantics. Therefore, negation-as-failure and constraints are not supported by SWRL. In the paper, rules are written in SWRL human readable syntax for the better understanding.

These languages are developed for knowledge representation allowing its intelligent processing by software agents. *SPARQL* query language[29] was developed for querying data from the Semantic Web. Its basal notion is a triple pattern in which variables are denoted by the leading symbol '?' or '$'. For instance, the pattern `{?p hasChild ?c}` specifies that variables `p` and `c` should be bound to resources connected by the *hasChild* property in the queried RDF graph. More complicated queries with graph patterns can be postulated using the SPARQL features, some of which have SQL analogues: patterns can be joined with '`.`', query results can be united with `UNION` feature, `OPTIONAL` is analogue of `OUTER JOIN`, and `FILTER` is used to restrict query results. For instance, the following query is used to find persons with their children, who have not consorts:

```
SELECT $x $y WHERE {
       ?x rdf:type Person.  ?x hasChild ?y.
       OPTIONAL{ ?y hasConsort ?z}.  FILTER(!bound(?z))}
```

Note that `bound` built-in predicate is used to check, if the variable was bound in the query, and '!' operator is used to specify that this condition does not hold. Therefore, SPARQL is the only one from the developed Semantic Web languages implements negation-as-failure. Its expressivity was studied: it was shown that SPARQL queries can be viewed as Datalog rules[28], and that SPARQL is as expressive as relational algebra[1]. More detailed description of standardized formats can be found in Ref.[16].

3 Mapping Approach

Presented mapping approach can be logically divided into two parts: *relation mapping* and *constraints mapping*. By relation mapping we understand set of rules describing one how to present database relations, columns, tuples in terms of RDF and OWL. Constraints mapping is the central point of the interest of the paper and considered as a set of rules for representing database-style constraints in the Semantic Web. Particular attention is paid to validation of these constraints represented in OWL and SWRL. The Section first presents mapping of relations into OWL ontology, and then presents mappings for database-style constraints allowing consistency checking.

3.1 Relation mapping

Relations are mapped in a very straightforward way close to value-based approach in Ref.[21], because it simplifies expressing constraints in SWRL.

Classes and functional data-valued properties are created for relations and their attributes, respectively. It is required that different relations (attributes) correspond to different classes (resp., properties). It can be done in the following way: class with URI `host/database/Rel` corresponds to relation *Rel*, and property with URI `host/database/Rel/Att` – to its attribute *Att*. The following notation is used further: created class has the same name as base relation, and name of property is concatenation of relation's and attribute's names (`relAtt`).

A vocabulary (further denoted as `db:`) was created to represent database terminology considered in the paper. All created classes are defined to be sub-classes of `db:Tuple` to distinguish them from any other OWL concepts. Assertions ⟨`relAtt rdfs:domain Rel`⟩ are added to the ontology to mark attributes of a relation. Domain of the attribute is pointed with `rdfs:range` property.

Each tuple t of a relation instance *Rel* is translated into an instance `t` of the class `Rel` (i.e. ⟨`t rdf:type Rel`⟩), where `t` is URI which might be obtained by attaching tuples counter value to `Rel`. If a tuple t has not-`NULL` value v for an attribute *Att*, ⟨`t relAtt v`⟩ statement is added. Because domain of a column is specified as `range` of the corresponding property, assertion of a value of incompatible type leads to inconsistency of the ontology. Therefore, DL reasoners can be used to verify types' compatibility[2].

It is impossible to use `relAtt` properties for `NULL` values of attributes. Absence of any assertions about some attribute's value is treated as inconsistency in the presented approach. Therefore, `NULL` values are specified explicitly in the generated ontology: sub-class of the `Rel` class (denoted `RelAttNo`) is created for any attribute *Att*; assertion ⟨`t rdf:type RelAttNo`⟩ is added to the ontology for all tuples t such that ($t.Att$ `IS NULL`).

3.2 Constraints mapping

Development of a constraint mapping was directed by the following task. It is assumed that generated RDF graph can be updated, and checking its consistency with respect to given constraints must be as simple as possible. Therefore, constraints are represented as SWRL rules, and their verification is performed as answering on a fixed query using SPARQL engine coupled with reasoner. All constraints are satisfied in the RDF graph, if the query has empty answer.

It is worth to note that SPARQL is developed as a query language for RDF data model and disregards OWL semantics. At the same time, several alternative query languages are proposed to access Semantic Web data as Description Logic knowledge bases[7,9,14,20]. SPARQL specification[29] also provides notion of entailment regimes. SPARQL-DL[32] is one notable example of such regime. The proposed mapping approach assumes that at least rules presented in the paper are used while answering queries to verify constraints. This assumption is not challenging for current Semantic Web reasoning systems, since many of them use SPARQL as their end point

[2] It is worth to note that data types support is not strong aspect of existing reasoners.

(e.g. Refs.[6, 23]).

Two kinds of constraints can be outlined which we call *positive* and *negative*. Positive constraints are violated, if incorrect data exists. Their violation condition can be specified in SWRL rule. To do so, `db:` vocabulary contains property `violates` to relate incorrect tuples with violated constraints.

A negative constraint is violated, if some expected data is absent. Verification of such constraints requires negation-as-failure. Since it is supported only by SPARQL (see Sect. 2.2), these constraints are handled in a more complicated way. Property `db:hasNC` is used to specify in the ontology that instances of some class must satisfy a negative constraint. Then SPARQL query is used to find all instances of this class which do not satisfy it, i.e. have no property `db:satisfies`. Domain of these three properties is `db:Tuple`, and range is `db:Constraint`. Mappings for both kinds of constraints are presented below.

3.2.1 Positive constraints

Mapping for positive constraints is demonstrated on example of `NOT NULL`-constraints, potential and primary keys. The following query is used to find all positive constraints `c` violated by tuple `x` from table `t`:

```
Q1:   SELECT $c $t $x WHERE {
            ?t rdfs:subClassOf db:Tuple.
            ?x rdf:type ?t.
            ?x db:violates ?c}
```

Let `NOT NULL`-constraint be specified for an attribute *Att* of a relation instance *Rel*. Then its violation condition is specified using the following rule:

$$\texttt{RelAttNo(?t)} \rightarrow \texttt{db:violates(?t, relAttNN)}\ . \tag{3.7}$$

Note that the mapping for `NULL` values makes this specification very simple. In the rule, `relAttNN` is an instance of `db:Constraint` tailored by *Att*.

Let $K = \{Att_1, \ldots, Att_n\}$ be a potential key on a relation instance *Rel*. The violation condition for a potential key can be obtained from (2.1):

$$\exists t_1, t_2 \in Hr : (t_1 \neq t_2) \wedge (\forall A \in K : t_1.A = t_2.A)\ . \tag{3.8}$$

As it was mentioned in Sect. 2.1, `NULL` values are considered to be equal in this case. Thus, property `relAttEq` is created with `Rel` as its domain and range, and the following rules are added for any *Att* from K:

$$\texttt{RelAttNo(?t1) \& RelAttNo(?t2)} \rightarrow \texttt{relAttEq(?t1,?t2)} \tag{3.9}$$

$$\texttt{relAtt(?t1, ?v) \& relAtt(?t2, ?v)} \rightarrow \texttt{relAttEq(?t1,?t2)} \tag{3.10}$$

When this is done, (3.8) is obviously represented as the SWRL rule:

$$\texttt{relAtt1Eq(?t, ?s) \& ...\&}$$
$$\texttt{relAttnEq(?t, ?s) \& differentFrom(?t,?s)} \rightarrow \texttt{db:violates(?t, K)} \tag{3.11}$$

Although relational model and SQL standard handle `NULL` values in this manner, in some DBMSs (e.g. PostgreSQL) two `NULL` values are not considered equal in (3.8). In the latter case, the following rule can be used instead of (3.11) and (3.9–3.10):

```
relAtt1(?t1,?v1) & relAtt1(?t2,?v1) &
                                 ...&
relAttn(?t1,?vn) & relAttn(?t2,?vn) &
              differentFrom(?t1,?t2)  → db:violates(?t1,K)
```
(3.12)

This rule does not infer constraint violation for a tuple t, if ($t.Att_i$ `IS NULL`) for some Att_i from K, because individual `t` has no property `relAtti`.

A primary key PK can be considered as a potential key such that each attribute $Att \in PK$ is defined to be `NOT NULL`. Therefore, $(n+1)$ SWRL rules are used to specify it: n rules (one for each attribute from PK) are similar to (3.7) and the last one is similar to (3.12)[3].

3.2.2 Negative constraints

The following query verifies negative constraints:

```
Q2:   SELECT$c $t $x WHERE {
      1:    ?x rdf:type ?t.  ?t rdfs:subClassOf db:Tuple.
      2:    ?p rdfs:domain ?t.  ?p rdfs:subPropertyOf db:hasNC.
      3:    ?p rdfs:range ?e.  ?c rdf:type ?e.
      4:    OPTIONAL{?ydb:satisfies?c FILTER(sameTerm(?x,?y))}
      5:    FILTER(!bound(?y) && !sameTerm(?p,db:hasNC))}
```

Lines `2-3` allow linking tables `t` with their positive constraints `c`. Lines `4-5` implement negation-as-failure to return only tuples `x` violating these constraints.

Let us consider how a negative constraint is mapped into OWL axioms and SWRL rules to allow one to verify it using the above query. A foreign key constraint is an example of such constraint: it is violated for some tuple without `NULL` values for attributes from FK, if the corresponding tuple in the referenced table does **not exist**. Therefore, the mapping approach is shown on this example.

Let a foreign key $FK = \{Att_1, \ldots, Att_n\}$ be defined for a relation instance Rel_1 and referring to Rel_2. First, a new individual `FK` is created, and a class `CFK` is defined as to be `owl:oneOf(FK)`. Next, a property `pFK` is created with `Rel1` as its domain and `CFK` as range and assigned to be sub-property of `hasNC`[4]. Then the following two rules should be created to specify (2.2):

```
Rel1Att1No(?t)&...& Rel1AttnNo(?t) → db:satisfies(?t,FK)
```
(3.13)

```
rel1Att1(?t,?v1)&rel2Att1(?s,?v1)
                          & ...  &
rel1Attn(?t,?vn) &rel2Attn(?s,?vn)  → db:satisfies(?t,FK)
```
(3.14)

[3] Object in heads of the rules is `PK`.

[4] Three entities are created for one constraint, because OWL Lite and OWL DL require disjoint sets of individuals and classes and, therefore, it is impossible to express that *"c is constraint on t"* in one statement.

Rule (3.13) specifies that foreign-key constraint is satisfied for a tuple, if all its attributes from FK are `NULL`s. Rule (3.14) specifies that it is satisfied for a tuple without `NULL` values for attributes in FK, if corresponding tuple exists in Rel_2.

As it is noted in Sect. 2.1, there are two other kinds of foreign key constraints in SQL. Let us consider their mappings. If FK is a `MATCH SIMPLE`-foreign key, then (3.14) with the following n rules implement it:

$$\begin{array}{l}\texttt{Rel1Att1No(?t)} \rightarrow \texttt{db:satisfiess(?t, FK)}\\ \ldots\\ \texttt{Rel1AttnNo(?t)} \rightarrow \texttt{db:satisfiess(?t, FK)}\end{array} \tag{3.15}$$

`MATCH PARTIAL`-foreign key can be implemented in similar way to potential key. First, the following two rules are added for each attribute $Att \in FK$:

$$\texttt{rel1Att(?t1,?v)\&rel2Att(?t2,?v)} \rightarrow \texttt{FKAttpartialEq(?t1,?t2)} \tag{3.16}$$

$$\texttt{Rel1AttNo(?t1)\& Rel2(?t2)} \rightarrow \texttt{FKAttpartialEq(?t1,?t2)} \tag{3.17}$$

When these auxiliary assertions are determined, foreingn key constraints are specified using (3.13) and the following rule:

$$\begin{array}{l}\texttt{FKAtt1partialEq(?t, ?s) \& ...\&}\\ \texttt{FKAttnpartialEq(?t, ?s)} \rightarrow \texttt{db:satisfies(?t, FK)}\end{array} \tag{3.18}$$

Thus, $2n + 2$ rules are created to implement such foreign key.

If `relAttNp` is sub-property of `db:hasNC` with range `oneOf(relAttSt)` and domain `Rel`, the following rules guarantee attribute Att is set in tuples from Rel:

$$\texttt{RelAttNo(?t)} \rightarrow \texttt{db:satisfies(?t, relAttSt)} \tag{3.19}$$

$$\texttt{relAtt(?t, ?v)} \rightarrow \texttt{db:satisfies(?t, relAttSt)} \tag{3.20}$$

3.2.3 Unknown source of violation

Let consider *anti-key* constraint AK of the form $AntiKey(Rel, [Att_1, \ldots, Att_n])$. This constraint is satisfied, if potential key $PK = \{Att_1, \ldots, Att_n\}$ is violated in Rel (i.e. at least one pair of tuples with identical values for attributes Att_1 through Att_n exists). Note that when anti-key is violated for some relation instance, particular violation source cannot be determined. Although all tuples of this relation instance can be returned as violation sources with `Q2`, it would not be informative. Therefore, constraints like anti-key are considered separately in the paper.

For an anti-key AK, an individual `AK`, a class `CAK` and a property `pAK` are created in the same way as for FK; the only difference is that `pAK` is sub-property of `hasAntiKey`. Rule specifying satisfaction condition for AK is similar to violation condition (3.12) for PK, but has the differenet consequent:

$$\begin{array}{l}\texttt{relAtt1(?t, ?v1) \& relAtt1(?s, ?v1) \&}\\ \texttt{...\&}\\ \texttt{relAttn(?t, ?vn) \& relAttn(?s, ?vn) \&}\\ \texttt{differentFrom(?t,?s)} \rightarrow \texttt{db:satisfies(?t, AK)}\end{array} \tag{3.21}$$

To verify anti-keys, the following query is used, which mainly distinguishes from `Q2` in that it does not return violations sources:

```
Q3: SELECT $c $t WHERE {
          ?t rdfs:subClassOf db:Tuple.
?p rdfs:domain ?t.  ?p rdfs:subPropertyOf db:hasAntiKey
          ?p rdfs:range ?e.    ?c rdf:type ?e.
          OPTIONAL{  ?y rdf:type ?t.  ?y db:satisfies ?c  }.
          FILTER(!bound(?y) && !sameTerm(?p,db:hasAntiKey))}
```

4 Discussion

This Section discusses some aspects which may remain unclear. Constraint verification can be performed with only one query uniting queries `Q1`, `Q2` and `Q3` presented above. The most important constraints (potential, primary and foreign keys and `NOT NULL`) can be mapped to SWRL rules automatically as it is described above. Rules for `CHECK`-constraints are added manually. This task is obvious for interval constraints and some other popular constraints, but is not solvable in general. The reason is that the Semantic Web languages do not support aggregation, and this is a topic of researches now[4]. Therefore, table constraints with aggregates are not expressible in any of these formats with exception to some particular cases, when number of aggregated entities is fixed. This decreases applicability of the mapping only slightly, because such constraints are used not very frequently and not supported by many commercial DBMSs.

4.1 OWL vs SPARQL

It may be preferable to create `violates`-rules for `CHECK`-constraints. The reason is that their implementation is simpler and can be reduced to standard DL reasoning task of ontology consistency checking with the following axiom:

$$\exists violates.\top \sqsubseteq \bot \tag{4.22}$$

However, (4.22) has side effects which can be shown on the following ontology O_1:

$$Aircraft \sqsubseteq \exists\, locate.(Airport \sqcup Repair) \tag{4.23}$$
$$functionalProperty(locate) \tag{4.24}$$
$$RaceAir \sqsubseteq Aircraft \tag{4.25}$$
$$RaceAir(os606) \tag{4.26}$$
$$locate(os606,\, aprt) \tag{4.27}$$

The following rules are defined in O_1 to implement constraints for $RaceAir$:

$$\texttt{RaceAir(?a)\&locate(?a,?l)\&Repair(?l)} \rightarrow \texttt{db:violates(?a,R)} \tag{4.28}$$
$$\texttt{RaceAir(?a)\&locate(?a,?l)\&Airport(?l)} \rightarrow \texttt{db:satisfies(?a,A)} \tag{4.29}$$

Using (4.23)–(4.27), one can entail that *aprt* is an instance of either *Airport* or *Repair*. Rule (4.28) itself does not allow choosing one class among them, and the SPARQL

query returns *os*606 as violating constraint `A` defined in (4.29). However, if (4.22) is used, it can be verified that *Repair*(*aprt*) leads to inconsistency, and *Airport*(*aprt*) does not. Thus, the axiom allows new fact to be entailed. The ontology is not inconsistent, and no constraints are violated in this case.

4.2 Safe rules

Let ontology O_2 contain assertions (4.23)–(4.26) with the following ones:

$$hold(aprt,\ os606) \tag{4.30}$$

$$locate \equiv hold^{-} \tag{4.31}$$

RaceAir is constrained in O_2 to have *locate* property using the negative rule:

$$\texttt{locate(?x,?y)} \rightarrow \texttt{db:satisfies(?x,CL)}\ . \tag{4.32}$$

Although O_2 does not contain (4.27), it can use (4.30) and (4.31) to entail it. It seems natural for constraint `CL` not to be violated in this case. Let now ontology O_3 be obtained from O_2 by removing assertion (4.31). Instead of (4.27), it can be entailed from O_3 using (4.23) that the following holds for an unnamed individual α :

$$locate(os606,\ \alpha)\ . \tag{4.33}$$

In contrary to (4.27), this assertion does not give any explicit information about *os*606, stating only that it has some value for *locate*. Intention of constraint `CL` is to guarantee that we explicitly know this value. Therefore, the constraint must be violated in this case, and (4.32) must not be fired. These reasons lead to applying the following semantics to SWRL rules: *variables in rules can be instantiated only with values explicitly used in underlying ontology.* Note that this semantics is applied in safe rules[27] to solve problems with SWRL decidability[25].

4.3 How to distinguish individuals

Some constraints like potential or primary keys can be violated only by a pair of different individuals. Two individuals are considered to be different in the corresponding SWRL rules, if they are related via `owl:differentFrom` property. There are several ways to distinguish individuals built by tuples from a database. First, since attributes are mapped to functional properties, any two individuals with different values for some of these properties are also different. Next, the following rule is created for any attribute *Att* of a relation *Rel* to distinguish individuals t_1 and t_2 such that (t_1.*Att* `IS NULL`) and (t_2.*Att* `NOT IS NULL`):

$$\texttt{RelAttNo(?t1) \& relAtt(?t2,?v)} \rightarrow \texttt{differentFrom(?t1,?t2)}\ . \tag{4.34}$$

Finally, individuals not distinguished in these ways must be explicitly related via `owl:differentFrom` property.

Another possible way to distinguish individuals is to apply Unique Name Assumption (UNA), i.e. individuals with different names are considered to be different. Although OWL does not apply UNA[33], it can be done with SWRL, if built-in atom

`URI2literal` converting URI to string is added. In this case, UNA is implemented using a functional property `db:hasName` instantiated with the following rule:

$$\texttt{db:Tuple(?t) \& URI2literal(?t,?y)} \rightarrow \texttt{db:hasName(?t,?y)} \ . \quad (4.35)$$

However, the author claims that UNA is not useful in Web environment. As noted in Ref.[15], one of the most useful features of the Semantic Web is `owl:sameAs` allowing better integration. Let us consider ontology O_{DB} created from a database. One way to achieve benefits from this mapping is to integrate O_{DB} with an expressively rich ontology O_{SEM} available in the Web. To do so, it may be required to state that individuals i_1 from O_{DB} and i_2 from O_{SEM} are the same. These individuals have different URIs, and (4.35) entails that they have different values of `db:hasName`. Because `db:hasName` is functional, the individuals will be entailed to be different. This contradiction leads to inconsistency of $O_{DB} \cup O_{SEM}$. It means that integration with other ontologies becomes impossible, if UNA is used. Therefore, it is not used in the presented approach.

4.4 Extending SPARQL to improve the approach

Although UNA is shown to be not applicable in the Semantic Web in general, there are applications exist which need some forms of UNA. In Ref.[24], it is studied how SPARQL can be used to solve this problem by constructing RDF graph constituted of property `differentFrom` connecting individuals with different names (IriRefs). Results of Ref.[24] may improve the presented approach. However, some additional application is needed in this case. Extensions presented in this Section allows this application to be avoided.

First, as since SPARQL is RDF query language, it does not contain constructions to mark that some triple should be entailed. Entailment might be introduced into SPARQL in a controled way with a new kind of patterns. To do so, rule 21 of SPARQL grammar can be rewritten as follows:

```
[21′]TriplesBlock ::= (TriplesSameSubject | KnownPattern)
                      ('.'  TriplesBlock?  )?
[E1]KnownPattern ::= 'KNOWN''{'VarOrTerm PropertyListNotEmpty'}'
```

Changes in the grammar are outlined with *italics*. Keyword `KNOWN` is introduced to denote new kind of patterns. Ordinary triple patterns are used to access explicit assertions in the RDF graph, while `KNOWN` triple patterns are used to access to all assertions known to a query engine. It means that the query engine is asked to perform entailment to compute solutions (i.e. to evaluate them in entailment regime) only for `KNOWN` patterns. In general, the engine is not required to perform complete reasoning for these patterns, but all solutions to a triple pattern P are required to be solutions of `KNOWN`$\{P\}$.

Also it useful to introduce subqueries into SPARQL. This extension is considered by the SPARQL Working Group[19]. As it will be shown below, subqueries in `FROM` clause can improve the mapping approach. Such extension can be done by changing rule 9 from SPARQL grammar as follows:

```
[9′] DatasetClause  ::=  'FROM' ( DefaultGraphClause | SubQuery
```

```
                                |NamedGraphClause |NamedSubQuery)
[E2] SubQuery      ::= '(' ConstructQuery ')'
[E3] NamedSubQuery ::= 'NAMED' IriRef SubQuery
```

`CONSTRUCT` queries are used in SPARQL to produce RDF data on base of given data source, and only this kind of queries is allowed in subqueries. A graph constructed as a result of an anonymous subquery is merged into default graph of outer query. Result of named subquery is used in the outer query as a named graph. These results are considered as entailed triples and are accessed only in `KNOWN` patterns.

Having these two extensions, the following query can be used to verify `violates` constraints:

```
SELECT  $x $t $c
FROM    (CONSTRUCT { ?x1 owl:differentFrom ?x2 }
         WHERE     { ?t1 rdfs:subClassOf db:Tuple.
                     ?x1 a ?t1.  ?x2 a ?t1
                   OPTIONAL{  KNOWN { owl:sameAs(?x1, ?x3) }
                              FILTER( sameTerm(?x3, ?x2) ) }
                   FILTER( !bound(?x3) ) })
WHERE   { ?t rdfs:subClassOf db:Tuple.
          ?x a ?t.
          KNOWN { ?x db:violates ?c } }
```

Here subquery is used to distinguish triples corresponding to mapped tuples. Introduction of `KNOWN` pattern allows us to apply UNA and `violates` rules only to individuals explicitly asserted to be entities of classes obtained as result of relations mapping. Thus, this query does not face problems discussed in Sect. 4.3. This query also may be performed in a more effective way, because query engine does not need to entail triples for non-`KNOWN` patterns.

Let us emphasize again that the presented approach does not use any extensions to the Semantic Web languages. The SPARQL extensions introduced in this Section are only one possible way to improve it.

5 Related Work

There are a plenty of approaches to mapping of relational databases to RDF, and some of them are surveyed by W3C RDB2RDF Incubator Group in Ref.[31]. The survey also presents different criterions for classifying mappings.

- Data mapping can be either static (RDF dump) or dynamic (query-driven). Since the approach presented in the paper is focused on constraints, it implements RDF dump. Although it can be enhanced to support dynamic instantiation, the current implementation, as Ref.[31] notes, has advantages if entailment rules are applied to the RDF repository to infer new facts.

- Approaches are based either on automatic creation of new ontology from database (e.g. Refs.[34, 12]) or on manual mapping to an existing ontology[2].

- Also these approaches use different languages like Ref.[5].

All the approaches have their advantages and drawbacks, but they mostly disregard semantics of local databases (i.e. primary and foreign keys, constraints).

The recent works[8,26] consider these constraints and show inability of OWL to express them because of difference between the semantics discussed in Sect. 2. As consequence, they propose to extend OWL with features for support of relational constraints. OWL 2[13] supports keys in OWL semantics. Different ways to extend OWL with integrity constraints support are reviewed in Ref.[33].

Although[17] mentions that SWRL can be straightforwardly extended to enable negation-as-failure and database-style constraints, Ref.[18] points that it is not possible because SWRL applies the same logic foundations as in OWL and proposes multi-stack architecture to overcome this drawback. Several proposals to extending DLs with rules have applied this architecture, but it does not meet common acceptance, because it leads to establishment of two Semantic Webs.

In Ref.[10], SWRL is used to represent primary keys for more accurate translation in Automapper. However, constraint verification is not considered in this work.

In Ref.[22], it is proposed to use RDF and SPARQL query language to check consistency of ontologies generated from relational databases. It supposes dynamic generation of SPARQL queries to implement semantics of database-style constraints. However, our approach uses the Semantic Web languages in more natural way: OWL and SWRL are used to implement the constraints' semantics, and static SPARQL query is used to check consistency. In this case SWRL rules can be used for semantic query optimization in the same way as it is shown in Ref.[22]. Our approach supports mapping and verification of so-called negative constraints like anti-keys, for which extension of SPARQL is proposed in Ref.[22].

In contrast to the above-mentioned approaches, the mapping presented in the paper does not extend any Semantic Web language to support constraints, what is important to establish and not to complicate the Semantic Web.

6 Conclusion

The most of nowadays information is stored in relational databases. Therefore, populating the Semantic Web with it is an important step for making future Web more intelligent. It leads to development of a plenty of proposals for mappings databases to ontologies. However, the most of them disregards constraints which can be useful in many ways including semantic query optimizations, database dumps, and supporting materialized RDF graphs. In the distant future, it may simplify migration from relational databases to RDF stores like Ref.[30].

As since the data representation formats separately do not allow constraints representation due to the differences in logical foundations, the paper presents the approach to database mapping which uses the full strength of the Semantic Web stack in a natural way: data is presented in RDF; OWL and SWRL are used to express constraints' satisfaction or validation conditions; and the static SPARQL query is used to verify these constraints. The approach allows one to handle the most important constraints (potential, primary, and foreign keys) as like as many `CHECK`-constraints (including `NOT NULL`) without any extensions to the Semantic Web languages. As proof of concept, it was implemented on Java using JDBC for automatic mapping from PostgreSQL databases.

It is shown that SPARQL better suits for constraints verification than OWL. Safe rules are shown to be applicable not only in solving the decidability problem, but in constraints representation too. Another problem connected with database modeling is support of UNA. The author claims that it is not really useful in the Web. However, SPARQL extensions might be used to implement UNA for particular situations like database mapping presented in the paper. These extensions may improve the presented approach, but they are not mandatory for it.

The future work may be directed on extending the presented approach to express integrity constraints for arbitrary OWL concepts. Also it is worth to investigate ways for introducing aggregation into some Semantic Web language for more complete support of database-style constraints.

References

[1] Angles R, Gutierrez C. The expressive power of SPARQL. In Sheth, AP, Staab S, Dean M, Paolucci M, Maynard D, Finin TW, Thirunarayan K, eds. ISWC 2008. LNCS 5318, Springer, 2008, 114–129.

[2] Auer S, Dietzold S, Lehmann J, et al. Triplify: light-weight linked data publication from relational databases. In: Proc. of the 18th International Conference on World Wide Web, Madrid, Spain. 2009. 621–630.

[3] Baader F, Calvanese D, McGuinness D, et al. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003.

[4] Barbieri DF, Braga D, Ceri S, et al. C-SPARQL: SPARQL for continuous querying. In: Proc. of the 18th International Conference on World Wide Web, Madrid, Spain. 2009. 1061–1062.

[5] Bizer C, Seaborne A. D2RQ - treating non-RDF databases as virtual RDF graphs. In: 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, 2004. Poster.

[6] Broekstra J, Kampman A, van Harmelen F. Sesame: A generic architecture for storing and querying RDF and RDF schema. In: Horrocks I, Hendler, J A, eds. ISWC 2002. LNCS., Springer, 2002, 2342: 54–68.

[7] Calvanese D, Giacomo GD, Lembo D, *et al.* EQL-Lite: Effective first-order query processing in description logics. In: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6–12, 2007. 274–279.

[8] Calvanese D, De Giacomo G, Lembo D, *et al.* Can OWL model football leagues? In: OWLED 2007. Volume 258 of CEUR Workshop Proceedings, 2007.

[9] Fikes R, Hayes PJ, Horrocks I. OWL-QL - a language for deductive query answering on the semantic web. Journal of Web Semantics, 2004, 2(1): 19–29.

[10] Fisher M, Dean M, Joiner G. Use of owl and swrl for semantic relational database translation. In: Fourth International Workshop OWL: Experiences and Directions (OWLED 2008 DC), Washington, DC, 2008.

[11] Geller J, Chun SA, An YJ. Towards the semantic deep web. IEEE Computer, 2008, 41(9): 95–97.

[12] Ghawi R, Cullot N. Database-to-ontology mapping generation for semantic interoperability. In: Third International Workshop on Database Interoperability (InterDB 2007), Vienna, Austria, 2007.

[13] Grau CB, Horrocks I, Motik B, et al. OWL 2: The next step for OWL. Journal of Web Semantics, 2008, 6(4): 309–322. [doi: 10.1016/j.websem.2008.05.001]

[14] Haarslev V, Möller R, Wessel M. Querying the semantic web with racer + nRQL. In: Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04), Ulm, September 20-21, 2004. Volume 115 of CEUR Workshop Proceedings, 2004.

[15] Hendler J. The dark side of the semantic web. IEEE Intelligent Systems, 2007, 22(1): 2–4.

[16] Herman I. W3C semantic web activity, 2001. http://www.w3.org/2001/sw/

[17] Horrocks I, Patel-Schneider PF, Boley H, *et al.* 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

[18] Kifer M, de Bruijn J, Boley H, *et al.* A realistic architecture for the semantic web. In: RuleML 2005. Volume 3791 of LNCS., Springer, 2005, 17–29.

[19] Kjernsmo K, Passant, A. SPARQL new features and rationale. W3C Working Draft (July 2 2009) http://www.w3.org/TR/2009/WD-sparql-features-20090702

[20] Kubias A, Schenk S, Staab S, *et al.* OWL SAIQL - an OWL DL query language for ontology extraction. In: OWLED 2007. Volume 258 of CEUR Workshop Proceedings. 2007.

[21] Lausen G. Relational databases in rdf: Keys and foreign keys. In: SWDB-ODBIS 2007. LNCS 5005, Springer, 2007, 43–56.

[22] Lausen G, Meier M, Schmidt M. SPARQLing constraints for RDF. In: EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, ACM, 2008, 499–509.

[23] Levshin DV. An SQL-based approach to semantic web reasoning and query answering. In: Proc. of the Spring Young Researcher's Colloquium on Database and Information Systems (SYRCoDIS 2009), Saint-Petersburg, Russia, May 28-29, 2009. Volume 454 of CEUR Workshop Proceedings, 2009.

[24] Levshin DV. Introducing unique names into the semantic web. In: SISY 2009, 7th International Symposium on Intelligent Systems and Informatics, Subotica, Serbia, 25-26 September 2009. 329–333.

[25] Levy AY, Rousset MC. The limits on combining recursive horn rules with description logics. In: AAAI/IAAI, 1996,(1): 577–584.

[26] Motik B, Horrocks I, Sattler U. Bridging the gap between OWL and relational databases. In: Proc. of the 16th International Conference on World Wide Web, Banff, Alberta, Canada. 2007. 807–816. [doi: 10.1016/j.websem.2009.02.001]

[27] Patel-Schneider P. Safe rules for owl 1.1. In: Fourth International Workshop OWL: Experiences and Directions (OWLED 2008 DC), Washington, DC, 2008.

[28] Polleres A. From SPARQL to rules (and back). In: Proc. of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007, ACM, New York, 2007. 787–796.

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

[30] Ramanujam S, Gupta A, Khan L, *et al.* Relationalizing RDF stores for tools reusability. In: Proc. of the 18th International Conference on World Wide Web, Madrid, Spain. 2009. 1059–1060.

[31] Sahoo SS, Halb W, Hellmann S, *et al.* A survey of current approaches for mapping of relational databases to rdf (8 January 2009) http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf

[32] Sirin E, Parsia B. SPARQL-DL: SPARQL query for OWL-DL. In: OWLED 2007. Volume 258 of CEUR Workshop Proceedings, 2007.

[33] Sirin E, Smith M, Wallace E. Opening, closing worlds - on integrity constraints. In: OWLED 2008. Volume 432 of CEUR Workshop Proceedings, 2008.

[34] Suwanmanee S, Benslimane D, Champin PA, *et al.* Wrapping and integrating heterogeneous relational data with OWL. In: Proc. of the 7th International Conference on Enterprise Information Systems (ICEIS 2005), 2005. 11–18.

[35] Volz R. Change paths in reasoning! In: New forms of reasoning for the Semantic Web: scalable, tolerant and dynamic. Proc. of the First International Workshop. co-located with ISWC 2007 and ASWC 2007.

[36] Wu F, Weld DS. Automatically refining the wikipedia infobox ontology. In: Proc. of the 17th International Conference on World Wide Web, Beijing, China. 2008. 635–644.