

# Translation of Ontology Retrieval Problem into Relational Queries

Jaroslav Pokorný<sup>1</sup>, Jana Pribolová<sup>2</sup>, and Peter Vojtáš<sup>1</sup>

<sup>1</sup> Department of Software Engineering,  
Charles University, Prague, Czech Republic  
{Pokorny, Vojtas}@ksi.mff.cuni.cz

<sup>2</sup> Institute of Computer Science,  
Faculty of Science, University of P. J. Šafárik,  
{Jana.Pribolova}@upjs.sk

**Abstract.** Ontology as a knowledge base can provide different reasoning tasks, e.g. to check consistency of the ontology or to check whether a resource is instance of a concept or not. In this paper we want to focus on retrieval problem. There already exist systems resolving this problem, but they are not effective within large datasets. Our idea is to transform ontology into a relational database. We present particular algorithms of this transformation both on the scheme level and SQL level with special handling of functional roles and definitions. This enables to query such database by usual tools of SQL, i.e. to solve the retrieval problem.

**Keywords:** ontology, translation, relational database, SQL

## 1 Introduction

OWL enables the creation of ontologies and provides extensive semantics for Web data. This language is heavily influenced by description logics (DL, see [1]). Research on DL reasoners consists of the solving reasoner problems such as satisfiability, consistency or retrieving instances of the concept. Relational database is an excellent system for storing and querying data, but their inferring capabilities are limited just to querying. In this paper we describe the method to extend relational database to store ontology.

In present some research groups are interesting in topic of ontology storing and maintenance. Some of them are trying to limit expressive power of language to speed up reasoner tasks. But some of them are trying to fuse two or more kind of systems.

One of the prominent directions in this area is blending ontologies and logic databases [5, 3]. They create so called *description logic programs* which are description logic expressions and logic programs mixed together.

Another direction of the research area is combining ontologies and relational databases. First experiments focus on XML document translation into corresponding relational tables [9]. Inspired by the XML storing in relational

databases there are some research projects concerned with ontology storing in the relational databases. One of the first projects is published in [4]. However, many others occurred, e.g., the system HAWK [10] and its ancestor the system DLDB [7], as well as the projects described in [2, 6].

In Section 2 we write about a knowledge base represented through DL and relational data model. Further, in subsection 2.1 we deal with potential and valid domains as well as with ranges. The valid domains are important input parameters of the algorithm to construct relational scheme which we present in Section 2.2. Subsection 2.3 explains creating the relational database. Section 3 concentrates on ontology implementation in an SQL environment. This enables to query such database by usual tools of SQL, i.e. to solve the retrieval problem. In fact, it means creating tables and views (Subsection 3.1) in the SQL language. Then we explain how to insert data in the tables (Subsection 3.2). Section 4 concludes the paper.

## 2 Knowledge Base in Relational Data Model

Basics of DL, ontologies and all used symbols are described in [8], here we refer to unexplained notions. Our language consists of names for atomic concepts  $A, B \in \mathbf{NC}$ , names for roles  $R \in \mathbf{NR}$ . Roles are either functional ( $\mathbf{NFR}$ ) or non-functional ( $\mathbf{NNR}$ ). Concept constructions we usually denote by  $C, D$  and we understand them as nonterminal symbols.

The main idea of the knowledge base representation with relational data model is shown in Figure 1. For every concept or role we create unary or binary relation, respectively, except for functional roles. For every functional role we create only one attribute in a relation or one attribute in more than one relation depending on so called valid domains (see Section 2.1). The ABox assertions are translated by inserting some tuples into relations. TBox assertions of the type  $\sqsubseteq$  are translated into integrity constraints and equalities into special relations called  $T_C^{view}$ . In practice, such a relation is represented by view in SQL.

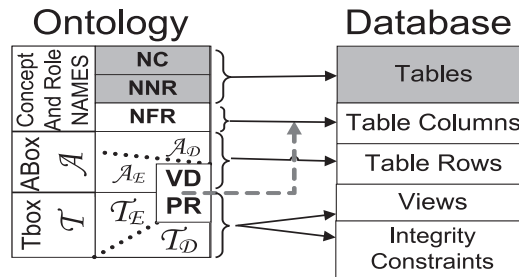


Fig. 1. Translation mapping the ontology elements to database structures.

Note that ontology as a relational database offers the users some kind of inferencing. The main feature is to retrieve all instances of the concept. Moreover such a database supports query construction. In DL, an equivalent of the query is a concept defined as equality. However, in practice ontology-based systems are applied for more complicated queries than concepts can be. The W3C standard for such a set of queries is the query language called SPARQL [13]. A lot of such queries are supported by our system.

Section 2.2 presents the solution to find out all instances of the concept, not only instances that are explicitly known, i.e. those expressed by  $C(a)$ .

## 2.1 Domains and Ranges of Roles

In DL the knowledge base comprises TBox and ABox. We understand both of them, ABox  $\mathcal{A}$  about and TBox  $\mathcal{T}$ , as sets of assertions (about individuals or concepts, respectively). Each set of assertions can be divided into two categories. One, denoted by subscript  $\mathcal{E}$ , includes *extensional assertions*, the second one comprehends as set of additionally *deduced assertions* and it is denoted by subscript  $\mathcal{D}$ . The set  $\mathcal{T}_{\mathcal{E}}$  consists of acyclic definitions  $A := C$  and axioms  $C \equiv D$  and  $C \sqsubseteq D$ . The set  $\mathcal{T}_{\mathcal{D}}$  is derived with respect to (symmetric, transitive) properties of the assertions  $\equiv$  and  $\sqsubseteq$ . For all TBox sets the following holds:

- $\mathcal{T} = \mathcal{T}_{\mathcal{E}} \cup \mathcal{T}_{\mathcal{D}}$ ,
- $\mathcal{T}_{\mathcal{E}} \cap \mathcal{T}_{\mathcal{D}} = \emptyset$ .

ABox  $\mathcal{A}_{\mathcal{E}}$  consists of statements of the form  $B(a)$  and  $R(a, b)$ . Let us also mention that  $\mathcal{A} = \mathcal{A}_{\mathcal{E}} \cup \mathcal{A}_{\mathcal{D}}^{\mathcal{T}}$ . The set  $\mathcal{A}_{\mathcal{D}}^{\mathcal{T}}$  depends on the TBox  $\mathcal{T}$ , because we derive assertions on basis of  $\mathcal{A}_{\mathcal{E}}$  and TBox assertions. If it is evident which  $\mathcal{T}$  inducted  $\mathcal{A}_{\mathcal{D}}^{\mathcal{T}}$ , we omit superscript  $\mathcal{T}$ .

*Example 1.* Let us have a knowledge base  $\mathcal{O}$  with concepts shown in Figure 2. The concept *Student* is defined as follows:

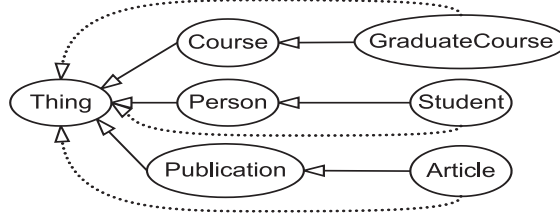
$$Student := Person \sqcap \exists takesCourse.Course$$

The set of assertions  $\mathcal{T}$  includes previous definition of concept *Student* and also subsumption assertions shown in Figure 2 with solid line. Also there are some roles in  $\mathcal{O}$ , i.e.  $\mathbf{NFR} = \{hasName, hasAddress\}$  and  $\mathbf{NNR} = \{takesCourse\}$ .

Note that the name of the concept *GraduateCourse* is abridged to *GCourse*. The deduced assertions are shown in Figure 2 with dotted arrows.

Also ABox consists of the extensional assertions:

$$\mathcal{A}_{\mathcal{E}} = \left\{ \begin{array}{lll} Person(S_2), & GCourse(C_2), & hasName(P_1, Name_b), \\ Student(S_1), & takesCourse(S_1, C_1), & hasName(C_1, Name_c), \\ Publication(P_1), & takesCourse(S_2, C_1), & hasName(C_2, Name_d), \\ Article(P_1), & hasName(S_1, Name_a) & hasAddress(S_1, Address_a), \\ Course(C_1), & & \end{array} \right\}$$



**Fig. 2.** IS-A hierarchy of the knowledge base  $\mathcal{O}$ .

and also of the deduced ones:

$$\mathcal{A}_D^T = \{Person(S_1), Course(C_2), Student(S_2)\}$$

As we can see the assertion  $Publication(P_1)$  belongs to the set of extensional assertions. Though it can be deduced also, because the assertion  $Article \sqsubseteq Publication$  is in the TBox  $\mathcal{T}$  and  $Article(P_1)$  is an extensional assertion.

In DL there are two kinds of the equality meanings. The equality whose left-hand side is atomic concept means the definition. In this paper we denote definition equality as the symbol  $:=$  instead of  $\equiv$  to distinguish definition equalities from the rest. We assume, there are no cycles in definitions.

In OWL language there is a chance to define domain and range of a role but it is not a necessary condition. Emphasize that our approach tries to reduce the number of join operations within the query construction. The main idea is to encode each functional property as an attribute of the relation, that represents domain of the role, not as standalone relation. In case without defined domain (range) of the functional roles we need to find concepts of instances that are related to another through the role. Sometimes domain or range are defined as union of concepts. This case is similar to the previous one. Therefore we define so called potential and valid domains as follows.

**Definition 1.** A concept  $B \in \mathbf{NC}$  is said to be a potential domain for role  $R \in \mathbf{NR}$  with respect to the set assertions  $\mathcal{A}$  if there is  $R(a, b) \in \mathcal{A}$  such that  $B(a) \in \mathcal{A}$ . The set of potential domains for role  $R$  with respect to  $\mathcal{A}$  is denoted  $\mathbf{PD}_R^{\mathcal{A}}$ .

*Example 2.* Let us illustrate the Definition 1 on Example 1. The role  $hasName$  has the following potential domains with respect to  $\mathcal{A}_{\mathcal{E}}$ :

$$\mathbf{PD}_{hasName}^{\mathcal{A}_{\mathcal{E}}} = \{Publication, Article, Course, GCourse, Student\}$$

The role  $hasAddress$  has the following potential domain with respect to  $\mathcal{A}_{\mathcal{E}}$ :

$$\mathbf{PD}_{hasAddress}^{\mathcal{A}_{\mathcal{E}}} = \{Student\}$$

In OOP (Object-Oriented Programming) if the ancestor class has defined a function, the descendant class can use it. We map the concept into meaning of OOP class and functional role into OOP function. That means that we do not need translate the same functional role for ancestor concept and for descendant concept separately. Therefore let us define valid domains – concepts for which we consider the functional role to translate into relational scheme.

**Definition 2.** A valid domain for role  $R \in \mathbf{NFR}$  with respect to  $\mathcal{A}$  is a potential domain  $A \in \mathbf{PD}_R^{\mathcal{A}}$  with property that does not exists  $B$ ,  $B \in \mathbf{PD}_R^{\mathcal{A}}$  so that  $A \sqsubseteq B \in \mathcal{T}$  and  $A \equiv B \notin \mathcal{T}$ , as well as  $B \sqsubseteq A \notin \mathcal{T}$ . The set of valid domains for role  $R$  is denoted as  $\mathbf{VD}_R^{\mathcal{A}, \mathcal{T}}$ . If we are interested in valid domains with respect to whole TBox assertions, e.g.  $\mathcal{T}$ , we can omit the superscript  $\mathcal{T}$ .

Note that if ABox is changed it is necessary to revise the potential and valid domains again. This process prevents ontology deformation of the original modeling intent.

*Example 3.* With assistance of the previous examples we can present the valid domains of the role *hasName* with respect to  $\mathcal{A}_{\mathcal{E}}$ :

$$\mathbf{VD}_{hasName}^{\mathcal{A}_{\mathcal{E}}} = \{Publication, Course, Student\}.$$

The concept *GCourse* does not belong to the set  $\mathbf{VD}_R^{\mathcal{A}_{\mathcal{E}}}$  because there exists *Course*  $\in \mathbf{VD}_R^{\mathcal{A}_{\mathcal{E}}}$  so that *GCourse*  $\sqsubseteq$  *Course* and neither *Course*  $\sqsubseteq$  *GCourse* nor *Course*  $\equiv$  *GCourse* is in  $\mathcal{T}$ .

It is useful to define "inverse" function that can find for any concept  $A$  all roles for which the concept is valid domain.

**Definition 3.** The role  $R \in \mathbf{NFR}$  is said to be a role defined on the concept  $B$  with respect to the set of assertions  $\mathcal{A}$  if  $B \in \mathbf{VD}_R^{\mathcal{A}, \mathcal{T}}$ . We denote the set of all roles defined on the concept  $B$  as  $isIn\mathbf{VD}_B^{\mathcal{A}, \mathcal{T}}$ . Similarly as in Definition 2 if we use all assertions of  $\mathcal{T}$ , we can omit the superscript and denote the set of all roles defined on the concept  $B$  as  $isIn\mathbf{VD}_B^{\mathcal{A}}$ .

*Example 4.* In the running example of this paper an interesting point is to compute  $isIn\mathbf{VD}_{Student}^{\mathcal{A}_{\mathcal{E}}}$ :

$$isIn\mathbf{VD}_{Student}^{\mathcal{A}_{\mathcal{E}}} = \{hasName, hasAddress\}$$

Our solution requires valid domains to encode functional properties. To keep some integrity constraints it is useful to define also range of the roles.

**Definition 4.** A potential range for role  $R \in \mathbf{NR}$  (with respect to  $\mathcal{A}$ ) is concept  $B$  for which there exists  $R(a, b) \in \mathcal{A}$  and  $B(b) \in \mathcal{A}$ . We denote the set of potential ranges of role  $R$  as  $\mathbf{PR}_R$ .

*Example 5.* We compute potential ranges for all roles as follows:

$$\begin{aligned} \mathbf{PR}_{hasName}^{\mathcal{A}} &= \{String\}, \\ \mathbf{PR}_{takesCourse}^{\mathcal{A}} &= \{Course\}. \end{aligned}$$

Computation of valid ranges is unnecessary because we use ranges only to keep integrity.

## 2.2 Construction of a Relational Scheme

The first part of ontology translation into relational database consists of creating a relational scheme.

**Algorithm 1** Let  $\mathcal{O}$  be a knowledge base with TBox  $\mathcal{T}$  and ABox  $\mathcal{A}$ .  $\mathcal{T}$ ,  $\mathcal{A}$ , concept's names, and role's name are translated into relational database  $\mathcal{D} = (\mathbf{R}, \mathbf{I})$ . Here  $\mathbf{R}$  denotes a relational database scheme consisting of basic relational schemes and view definitions using relational algebra expressions (RA expressions). Second,  $\mathbf{I}$  denotes a set of integrity constraints. The translation is done by induction as described below.

First part of translation depends only on the language, the second part depends also on ABox and the last depends on the TBox too.

Note that names of attributes are motivated by RDF (*subject*, *predicate*, *object*) and *resource* terminology.

The construction is based on the following steps:

First translation steps are based solely on the description logic language

1. For all  $A \in \mathbf{NC}$  we add to  $\mathbf{R}$  new relation  $T_A$  with scheme  $T_A(\underline{resource})$ .
2. For all  $R \in \mathbf{NNR}$  we add to  $\mathbf{R}$  a relation scheme  $T_R(\underline{subject}, \underline{object})$ .

Following translation steps depend on the ABox (and deduced valid domains)

3. For all  $A \in \mathbf{NC}$  for which  $isInVD_A^{A^\varepsilon} = \{R_1, R_2, \dots, R_n\} \subseteq \mathbf{NFR}$ ,  $n \geq 1$  we modify  $T_A(\underline{resource}) \in \mathbf{R}$  to relation  $T_A^{mod} \in \mathbf{R}$  with scheme
 
$$T_A^{mod}(\underline{resource}, R_1.\underline{object}, \dots, R_n.\underline{object})$$
 If  $R \in \mathbf{NFR}$  and  $VD_R^{A^\varepsilon} = \emptyset$ , then we add to  $\mathbf{R}$  a new relational scheme  $T_R(\underline{subject}, \underline{object})$ .

The following translations depend on the TBox. First we deal with definitions:

4. For all  $A \in \mathbf{NC}$  such that there is a concept construction  $C$  with  $A := C \in \mathcal{T}$  we add to  $\mathbf{R}$  a new relation  $T_A^{view}$  with scheme  $T_A^{view}(\underline{resource})$  and view definitions so that ( $S_D$  and  $S_E$  are defined in step 5):
  - If  $C := D \sqcap E$  then
 
$$T_A^{view} = T_A \cup (S_D \cap S_E)$$
  - If  $C := \exists R.D$  and  $R \in \mathbf{NNR}$  or  $VD_R^{A^\varepsilon} = \emptyset$  then
 
$$T_A^{view} = T_A \cup (T_R(\underline{subject}, \underline{object})$$

$$[T_R.\underline{object} = S_D.\underline{resource}]$$

$$S_D(\underline{resource}))[T_R.\underline{subject}].$$
  - If  $C = \exists R.D$  and  $R \in \mathbf{NFR}$  and  $VD_R^{A^\varepsilon} \neq \emptyset$ ,  $n > 0$  then
 
$$T_A^{view} = T_A \cup (S_R^{rec}(\underline{subject}, \underline{object})$$

$$[S_R^{rec}.\underline{object} = S_D.\underline{resource}]$$

$$S_D(\underline{resource}))[S_R^{rec}.\underline{subject}]$$

where  $S_R^{rec}$  is the RA expression for reconstruction of the role  $R$  from appropriate columns of  $T_B^{mod}$  tables

$$S_R^{rec}(subject, object) = \bigcup_{B \in \mathbf{VD}_R^{A_\varepsilon}} (T_B^{mod}[resource, R.object])$$

Here we assume that this is a lossless encoding of all ABox information about  $R$ .

5. For a non-atomic concept construction  $C$  such that there is in  $\mathcal{T}$  no definition with right hand side  $C$  and  $C$  is a sub construction of a concept definition in  $\mathcal{T}$ , then we create a new RA expression  $S_C$  with the only attribute resource so that:
  - If  $C = D \sqcap E$  then
 
$$S_C = (S_D \cap S_E)$$
  - If  $C = \exists R.D$  and  $R \in \mathbf{NNR}$  or  $\mathbf{VD}_R^{A_\varepsilon} = \emptyset$  then
 
$$S_C = (T_R(subject, object)[T_R.object = S_D.resource]S_D(resource))$$

$$[T_R.subject]$$
  - If  $C = \exists R.D$  and  $R \in \mathbf{NFR}$  and  $\mathbf{VD}_R^{A_\varepsilon} \neq \emptyset$  then
 
$$S_C = (S_R^{rec}(subject, object)[S_R^{rec}.object = S_D.resource]S_D(resource))$$

$$[S_R^{rec}.subject]$$
6. To transform axioms in  $\mathcal{T}$ , we add the following integrity constraint to  $\mathbf{I}$ :
  - if  $C \equiv D \in \mathcal{T}$  and  $C, D$  are non atomic concept constructions then
 
$$S_C = S_D \in \mathbf{I},$$
  - if  $C \sqsubseteq D \in \mathcal{T}$  then  $S_C \subseteq S_D \in \mathbf{I}$ .

The following interesting observations result from the previous algorithm.

1. For all  $R \in \mathbf{NNR}$  and for all  $R \in \mathbf{NFR}$  for which  $\mathbf{VD}_R^{A_\varepsilon} = \emptyset$ :
  - $T_R[subject] \subseteq \bigcup_{B \in \mathbf{PD}_R^{A_\varepsilon}} T_B,$
  - $T_R[object] \subseteq \bigcup_{B \in \mathbf{PR}_R} T_B.$
2. For all  $B \in \mathbf{NC}$  for which  $isIn\mathbf{VD}_B^{A_\varepsilon} \neq \emptyset$  and for all  $R \in isIn\mathbf{VD}_B^{A_\varepsilon}$ :

$$T_B^{mod}[R.object] \subseteq \bigcup_{A \in \mathbf{PR}_R} T_A.$$

These assertions check "integrity" of the translation of role assertions. In more detail, the mentioned assertions take care to preserve the subject in area of role domains and the object in area of the role ranges.

The previous algorithm is illustrated on the following example.

*Example 6.* According to previous examples and applying Algorithm 1 we receive the following database scheme:

$$\begin{aligned} & T_{Publication}^{mod}(\underline{resource}, \underline{hasName.object}), \quad T_{Article}(\underline{resource}), \\ & T_{Course}^{mod}(\underline{resource}, \underline{hasName.object}), \quad T_{GCourse}(\underline{resource}), \\ & T_{Student}^{mod}(\underline{resource}, \underline{hasName.object}), \quad T_{Person}(\underline{resource}), \\ & T_{takesCourse}(\underline{subject}, \underline{object}). \end{aligned}$$

and

$$\mathbf{I} = \{T_{Article}[resource] \subseteq T_{Publication}[resource], \\ T_{GCourse}[resource] \subseteq T_{Course}[resource], \\ T_{Student}[resource] \subseteq T_{Person}[resource]\}.$$

Also relation defined as follows belongs to the database scheme:

$$T_{Student}^{view} = T_{Student}[resource] \cup T_{Person} \cap \\ (T_{takesCourse}[T_{takesCourse}.object = T_{Course}.resource] \\ ) [T_{takesCourse}.subject]$$

and the following assertions hold for given  $\mathcal{D}$ :

- $T_{takesCourse}[subject] \subseteq T_{Person} \cup T_{Student}$
- $T_{takesCourse}[object] \subseteq T_{Course}$

Note that, the attributes of the relations representing non-functional roles called *subject* and *object* have the same domain. Also note that all instances of a concept  $B \in \mathbf{NC}$ , so that  $B := D$ , are stored in  $T_B^{view}$ .

### 2.3 Construction of a Relational Database

In previous section we have created a relational scheme. Now we present the algorithm to insert the data in the database relations.

**Algorithm 2** Suppose that  $\mathcal{T}$ ,  $\mathbf{NC}$  and  $\mathbf{NR}$  are translated into database  $\mathcal{D}$ . ABox  $\mathcal{A}$  is transferred into  $\mathcal{D}$  by induction as follows:

1. If  $B(a) \in \mathcal{A}$  and also  $B \in \mathbf{NC}$ , then  $\langle a \rangle \in T_B$ .
2. If  $R(a, b) \in \mathcal{A}$  and  $R \in \mathbf{NNR}$ , then  $\langle a, b \rangle \in T_R$ .
3. If  $R(a, b) \in \mathcal{A}$  and  $R \in \mathbf{NFR}$ , then one of the following items:
  - (a) if  $\mathbf{VD}_R^{A_\varepsilon} = \emptyset$  then
 
$$\langle a, b \rangle \in T_R,$$
  - (b) if there exists  $A \in \mathbf{VD}_R^{A_\varepsilon}$  so that  $A(a) \in \mathcal{A}$ , then
 
$$\langle a, b \rangle \in T_A^{mod}[T_A.resource, R.object],$$
  - (c) if there exists  $A \in \mathbf{PD}_R \setminus \mathbf{VD}_R^{A_\varepsilon}$  so that  $A(a) \in \mathcal{A}$ , then there exists a maximal sequence  $A = B_1, B_2, \dots, B_n \in \mathbf{NC}$  so that  $B_n \in \mathbf{VD}_R^{A_\varepsilon}$  and  $B_i \sqsubseteq B_{i+1} \in \mathcal{T}_\mathcal{D}$  for  $i = 1, \dots, n - 1$  Then
 
$$\langle a, b \rangle \in T_{B_n}^{mod}[B_n.resource, R.object].$$

Note that the last step of algorithm the information  $A(a)$  is not lost and moreover information  $B_n(a)$  is a consequence of TBox axioms.

It is important to remember that the valid domains are built with respect to the set of extensional assertions  $\mathcal{A}_\varepsilon$  (Algorithm 2, Steps 3b and 3c). On the other hand, we consider all assertions from ABox  $\mathcal{A}$  to insert them in the proper tables (Step 1).

*Example 7.* After applying Algorithm 2 we obtain:

$$T_{Publication}^{mod} = \{\langle P_1, Name_b \rangle\}, \quad T_{Person} = \{\langle S_1 \rangle, \langle S_2 \rangle\}, \\ T_{Course}^{mod} = \{\langle C_1, Name_c \rangle, \langle C_2, Name_d \rangle\}, \quad T_{Article} = \{\langle P_1 \rangle\}, \\ T_{Student}^{mod} = \{\langle S_1, Name_a \rangle\}, \quad T_{Student}^{view} = \{\langle S_2 \rangle\}.$$



### 3 Ontology Implementation in an SQL Environment

We designed and implemented translation of a description logic knowledge base into an SQL environment, which works in accordance with Algorithms 1, 2. In fact, we create the database scheme which consists of `CREATE TABLE` and `CREATE VIEW` statements of the SQL language.

#### 3.1 Create Statements

First we create tables representing concepts as it is stated in Step 1 of Algorithm 1. A special case is the top concept. For technical reasons, we assign a numerical identifier to every URI which represents the associated instance.

```
for all  $A \in \mathbf{NC}$  do:
  if ( $A = \top$ ) then
    CREATE TABLE  $T_{\top}$ 
      ( resource INT NOT NULL PRIMARY KEY,
        uri VARCHAR NOT NULL );
  else
    CREATE TABLE  $T_A$  ( resource INT NOT NULL PRIMARY KEY );
```

Next step is the Step 2 of Algorithm 1. Therefore we create tables for non-functional roles in this way:

```
for all  $R \in \mathbf{NNR}$  do:
  CREATE TABLE  $T_R$  (
    subject INT NOT NULL,
    object INT NOT NULL,
    PRIMARY KEY(subject, objects) );
```

After that we deal with functional properties – Step 3 of the Algorithm 1:

```
for all  $R \in \mathbf{NFR}$  do:
  for all  $A \in \mathbf{VD}_R^{A^e}$  do: ALTER TABLE  $T_A$  ADD COLUMN  $R\_object$  INT;
```

Let us mention that in DL we do not distinguish different kinds of roles. However, in OWL there are two kinds of roles. Practically a role represents a relationship type (in OWL so called object property) or an attribute type (in OWL data type property). In case of relationship roles there is relationship between two instances represented by URIs. On the other hand, attribute roles represents relationship between instance represented by URI and literal value. It may appear that the problem can become if in the column representing the object functional role there is a literal value, or URI in the case of the data type role. But practically the role can be either object type or data type, not both types simultaneously. So it could not happen the mentioned collision.

For all atomic concepts that are equivalent to other concept, we also create view in the database as it is defined in Algorithm 1 in Step 4. Note, that in [8] we have proved that to any concept defined via other atomic and non-atomic concepts we can construct an SQL view whose definition contains only `INTERSECTION` operations, `SELECTS` and `TABLE R` expressions. Each `SELECT` uses

only join conditions. To achieve this it is necessary to normalize concept definition on the relational algebra level. We omit details of this procedure here. After this comment Step 4 looks like:

```
for all  $A \in \mathbf{NC}$ :  $C \equiv D$  do:
  String[] elements = getExpressionsOf(A)
  if (elements.length > 1) then
    for i=2 to elements.length do:
      elements[1] += " INTERSECT " + elements[i]
  CREATE VIEW ViewA AS elements[1];
```

where the function `getExpressionsOf` is defined as follows:

```
String[] getExpressionsOf(Concept A){
  String[] result;
  for all  $D_i : A := \prod_{i \geq 1} D_i$  do
    if ( $D_i \in \mathbf{NC}$ ) then
      result[i] = "SELECT resource FROM TDi;"
    else if ( $D_i = \exists R.E$  and  $E \in \mathbf{NC}$ ) then
      result[i] = "SELECT TR.subject FROM returnRole(R)
        JOIN TE ON TR.object = TE.resource;"
    else if ( $D_i = \exists R.E$ ) then
      result[i] = "SELECT TR.subject
        FROM " + returnRole(R) + " AS TR
        JOIN " + getExpressionsOf(E) + " AS TE
        ON TR.object = TE.resource;"
  return result;
}
```

and the function `returnRole` is:

```
String returnRole(Role R){
  String result;
  if ( $R \in \mathbf{NNR}$ ) then result = "TR"
  else
    int i = 0;
    for all  $A \in \mathbf{VD}_R^{A\epsilon}$  do:
      i++;
      if (i = 1) then result = "SELECT resource, R_object FROM TA"
      else
        result += "UNION SELECT resource, R_object FROM TA"
  return result;
}
```

At the end of the Algorithm 1 we have to do one more thing – to add some integrity constraints generated in Step 6.

```
for all  $t: t = A \sqsubseteq B$  do:
  ALTER TABLE TA ADD FOREIGN KEY (resource) REFERENCES TB;
```

The equality of two tables implementing  $\equiv$  leads to cyclic referential integrity in relational database schema. Therefore, we omit it from further consideration.

### 3.2 Insert the Data

Let us to suppose that the database scheme in SQL is created. Now we will deal with data – instances as it is described in Algorithm 2. We will show how to insert them into created tables.

```

for all  $A \in \mathbf{NC} \setminus \{\top\}$  do:
  for all  $A(a) \in \mathcal{A}$  do:
    if ( $a \in T_\top$ ) then
      id = (SELECT resource FROM  $T_\top$  WHERE uri = 'a')
    else
      INSERT INTO  $T_\top$ (resource, uri) VALUES ( generateId(a), 'a');
      INSERT INTO  $T_A$ (resource) VALUES ( generateId(a));

```

This code fragment implements Step 1. It uses the function `int generateId(String URI)` which generates a numerical identifier to use it within condition in join operation instead of string identifier.

Step 2 of the Algorithm 2 is implemented in this way:

```

for all  $R \in \mathbf{NNR}$  do:
  for all  $R(a, b) \in \mathcal{A}$  do:
    INSERT INTO  $T_R$  VALUES (getId(a), getId(b));

```

Finally, Step 3 of the Algorithm is interpreted as follows:

```

for all  $R \in \mathbf{NFR}$  do:
  for all  $R(a, b) \in \mathcal{A}$  do:
    if ( $\mathbf{VD}_R^{A^\varepsilon} = \emptyset$ ) then
      INSERT INTO  $T_R$  VALUES (getId(a), getId(b))
    else
      if ( $\exists A \in \mathbf{VD}_R^{A^\varepsilon}$  and  $A(a) \in \mathcal{A}$ ) then
        UPDATE  $T_A$  SET  $R\_object = getId(b)$  WHERE resource = getId(a)
      else
        find  $B \in \mathbf{VD}_R^{A^\varepsilon} : A \sqsubseteq B$ 
        UPDATE  $T_B$  SET  $R\_object = getId(b)$  WHERE resource = getId(a)

```

The function `int getId(String URI)` returns the numerical identifier assigned to a given URI.

## 4 Conclusion

The paper describes an approach to mapping ontology into relational database. The process of mapping is autonomous that means that there is no need of human interaction. We present algorithms of translation ontology into relational scheme and relational data model. In this paper we focused on implementations of the mentioned algorithms – transformation of algorithm's steps in SQL statements.

We work with so called  $\mathcal{EL}$  description logic which contains top, intersect and full existential quantification constructor. We would like to extend the logic with additional concept constructors inspired by relational database operators.

The important area of our research relates to valid domains. We want to do a research about valid domains and the assumption that valid domains preserve ISA-hierarchy.

## Acknowledgment

This paper was supported by Slovak project VEGA 1/0131/09, Slovak project VVGS/UPJ Š/45/09-10, Czech project GAČR 201/09/0990 and Czech project IS 1ET100300517.

## References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook. Theory, implementation, and application*. Cambridge University Press, 2003 United Kingdom.
2. J. Dokulil, J. Tykal, J. Yaghob and F. Zavoral. Semantic Web Repository and Interfaces. *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*. In Proc. of UBICOMM 2007, IEEE Computer Society, 2007, pp. 223–228.
3. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. *Combining Answer Set Programming with Description Logics for the Semantic Web*. Artificial Intelligence Vol. 172, Issues 12–13, 2008, pp. 1495–1539.
4. A. Gali, C. X. Chen, K. T. Claypool, and R. Uceda-Sosa. *From Ontology to Relational Databases*. Springer Verlag, LNCS 3289, 2004, pp. 278–289.
5. B. N. Groszof, I. Horrocks, R. Volz, and S. Decker. *Description Logic Programs: Combining Logic Programs with Description Logic*. In Proc. of WW2003, Hungary, 2003, pp. 48–57.
6. N. Kottmann and T. Studer: *Improving semantic query answering*. DEXA 2007, LNSC 4653, Springer, 2007, pp. 671 – 679.
7. Z. Pan and J. Heflin: *DLDB: Extending relational databases to support semantic web queries*. In Workshop on Practical and Scaleable Semantic Web Systems, ISWC 2003, 2003, pp. 109–113.
8. J. Pokorný, J. Pribolová, and P. Vojtáš. *Ontology Engineering Relationally*. Technical Report 2009-2, Dep. of Software Engineering, Faculty of Mathematics and Physics, Charles University, 2009, 20 p.
9. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In Proceedings of 25 VLDB Conference, 1999, pp. 302–314.
10. HAWK. <http://swat.cse.lehigh.edu/projects/index.html#hawk>
11. MySQL. <http://www.mysql.com/>.
12. Sesame. <http://www.openrdf.org/>.
13. SPARQL. <http://www.w3.org/TR/rdf-sparql-query/>.
14. SWAT Projects - the Lehigh University Benchmark (LUBM). <http://swat.cse.lehigh.edu/projects/lubm/>.