# Generating OWL Ontology for Database Integration

Nasser Alalwan, Hussein Zedan, François Siewe
Software Technology Research Laboratory
De Montfort University
Leicester, UK
[nasser, hzedan, fsiewe]@dmu.ac.uk

*Abstract*– **Today, databases provide the best technique for storing and retrieving data, but they suffer from the absence of a semantic perspective, which is needed to reach global goals such as the semantic web and data integration. Using ontologies will solve this problem by enriching databases semantically. Since building an ontology from scratch is a very complicated task, we propose an automatic transformation system to build Web Ontology Language OWL ontologies from a relational model written in Structured Query Language SQL. Our system also uses metadata, which helps to extract some semantic aspects which could not be inferred from the SQL. Our system analyzes database tuples to capture these metadata. Finally, the outcome ontology of the system is validated manually by comparing it with a conceptual model of the database (E/R diagram) in order to obtain the optimal ontology.**

**Keywords - reverse engineering, ontology, conceptual models, ontology learning, information extraction, Deep Web, Semantic Deep Web annotation.**

### Introduction

Today, the semantic web is gaining a significant amount of attention from researchers, because it has the ability to solve complicated problems such as data integration. The core of the semantic web is ontologies. An ontology is an explicit specification of a conceptualization [1]. Also it can be defined as a formally explicit set of terms formed in hierarchically structured way for describing concepts in a domain of discourse which can be used as a skeletal foundation for a knowledge base [2]. Ontologies play an important role in realizing the idea of database interoperability because of significant characteristics [3] such as:

- Adding rich, machine- readable semantics to data.
- Sharing the semantic perspective of the information structure with people or software agents.
- Separating domain knowledge from operational knowledge.
- Making explicit assumptions for a domain.

Although the use of an ontology is not proposed as a substitute for database technology, a database is still more powerful than an ontology for storing large-scale data sets. However, an ontology can be used with a database to provide a conceptual vision of heterogeneous data sources distributed in a number of databases with an interface built on an ontological model.

Thus, we need a system that utilizes both database and ontology techniques. However, while databases are widely available, the corresponding ontologies are not. Furthermore, constructing an ontology from scratch is tedious, time-consuming, error-prone and labour-intensive, while building one by hand presents the same difficulties [4] [5].

The proposed solution therefore starts by transforming a given database to an ontology with some rules as guidelines, which can be used for manual transformation or as the basis for an automatic transformation process [6]. Before going further, we have to clarify the difference between the terms 'transformation' and 'mapping'. The transformation of a database to an ontology means creating an ontology from the rules, either manually or by using a system, whereas in mapping, the database and the ontology both exist [7] [8].

This paper is organized as follows: we explain the motivation for our work in Section II, then offer an overview of relational databases and OWL ontologies in Section III and an outline of our approach in Section IV. Section V describes the overall rules for constructing an ontology (classes, object properties, datatype properties and instances) with a walkthrough example. The implementation is explained in Section VI. The validation and evaluation of instances in Section VII show that the transformation is correct. We present related work in Section VIII, then a summary of our contribution, a conclusion and suggestions for future work in Section IX.

## I. Motivation

Most approaches fail in their rules to differentiate entities having an 'IS-A' relationship from fragmented entities, such as [4] [5], or not mention it, such as [7]. Some use specific examples as rules, which can be misleading in respect of the results and can affect both the generalization of the rules and the accuracy of the system. Furthermore, some approaches avoid some circumstances, by making unrealistic assumptions such as no ternary relationships, no IS-A relationships or no fragmentation of tables. Others fail to cover some issues such as multi-valued attributes and composite attributes from the relational database side or cardinality restrictions, has-value restrictions and enumerated properties on the ontology side.

978-0-7695-3833-4/09 $26.00 © 2009 IEEE
DOI 10.1109/SEMAPRO.2009.21

In our approach, we test different situations with our rules to cover different circumstances and we combine the benefits of the relational model, the conceptual (E/R) model and the analysis of database tuples.

## II. RELATIONAL DATABASES AND OWL ONTOLOGIES: AN OVERVIEW

In the last 30 years, the relational model, which is the core of any relational database, has become more widely accepted than older approaches such as the hierarchical and network models. This applies even to the new models, such as object-relational or object-oriented databases, because relational databases are widely accepted by users, designers and vendors. More than three-quarters of the data on the current Web are stored in relational databases [9]. Both functional and domain expertise are also widely available for relational databases, but not for new models such as object-relational or object-oriented databases.

There are many advantages in using databases. For example, data need storing only once, multiple locations can be used, they are easy to back up, can have multiple layers of security and are scalable; and standards are enforceable. These advantages of databases are opposed by some general limitations, such as meaningless table or attribute names, poor semantics, bad design and poor performance.

However, the most serious limitation lies in database design, which goes through two phases: building the conceptual model in the entity relationship model (E/R diagram), then converting it to the relational model constructs typically found in Structured Query Language-Data Definition Language (SQL-DDL). The problem is the heavy semantic losses during the process of converting an E/R model to SQL [5].

Hence, the solution to database limitations and the shortcomings of the relational model is to move to an ontological model, using one of the ontology languages. Web Ontology Language (OWL) is the latest standard of W3C; it facilitates greater machine interpretability of web content than that supported by XML, RDF or RDF Schema, by providing additional vocabulary along with formal semantics. OWL has three sublanguages:

- OWL Lite is designed for a simple class hierarchy and simple constraints.
- OWL DL is based on description logics and is more expressive than OWL-Lite. Moreover, the reasoning software is able to support complete reasoning for every feature of OWL-DL. This paper focuses on OWL-DL.
- OWL Full is designed for high expressiveness. However, it does not have a computational guarantee [10].

## III. DATABASE INTEGRATION APPROACH

We assume complete cooperation from database owners who choose to participate in integrating their databases with others. The architecture of our approach is depicted in Fig.1.

Our system is divided into four phases: (i) generating the OWL ontology from SQL statements; (ii) validating and refining the ontology produced, by a comparison with the E/R diagram; (iii) mapping the ontology produced (local ontology) to a global ontology and (iv) integrating the global ontology by linking it with the database.

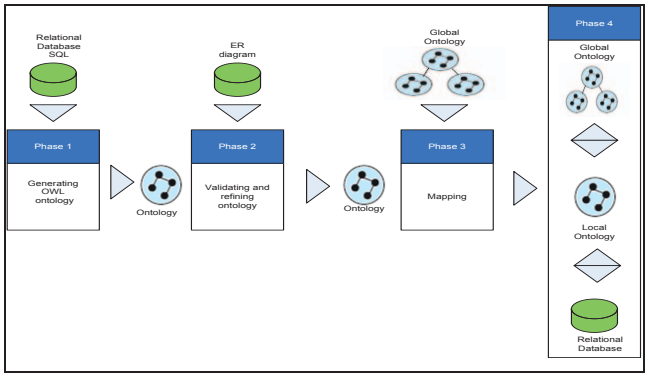


Figure 1. Database Integration Framework

Therefore our system needs to be provided with two inputs. The first input for the first phase is in SQL-DDL and SQL-DML. We can obtain this information from the database dump, which is a way of finding the structure of the table and the data in the tuples. Thereafter, our system applies the rules explained in Section V of this paper to generate a complete OWL ontology without instances, by parsing the SQL/DDL sentences. At this point the first phase is complete.

The second input is the E/R diagram of the database for the second phase, whose purpose is to refine and validate the OWL ontology produced in the first phase. In this second phase the database owner must supply our system with the E/R model. If this is not available, the database administrator is responsible for analyzing the system and producing the E/R model, although there are some programs, such as Power Design, which help database administrators to produce E/R models. Alternatively, they can use the techniques in [11] [12]. After the E/R diagram has been supplied or produced, the rules in Section V are applied. Next, the system will produce a refined and complete OWL ontology for the database. The goal of this second phase is to reach an optimal result without the need for a domain expert.

Finally, in the third phase, an expert will choose an appropriate global ontology designed for the domain.

In this paper, we concentrate on the first two phases, leaving the third and fourth phases to future work.

## IV. RULES FOR TRANSFORMATION OF SQL-DDL TO OWL ONTOLOGY

This section gives formal rules for transforming relational databases to OWL ontologies. In subsection A we define our predicates, then we introduce an example in subsection B. An ontology is built in four stages. The first stage, building classes with their datatype properties, is explained in subsection C, while subsections D and E respectively explain how object properties are created and instances migrated .

We adopt the assumptions of most of the work done in the transformation rules for our methodology:

(i) Relations are in third normal form (3NF), available in SQL-DDL [14].

(ii) New tuples added to the database are consistent with the derived metadata [11].

### A. Notations

We denote by *Dom(x)* the set of all possible values of a single or composite attribute *x*. We adopt the following predicate symbols:

- *PK(x, R)*: *x* is the (single or composite) primary key of the relation *R* and is represented as a set of attributes.
- *OCC(v, x, R)*: there is a tuple in *R* for which the value of *x* is $v \in Dom(x)$.
- *FK(x, R, y, S)*: *x* is a (single or composite) foreign key in relation *R* and references a primary key *y* in relation *S*.
- *IsFK(x, R)* $\triangleq \exists\, y, S: FK(x, R, y, S)$.
  *IsFK (x, R)* means that *x* is a (single or composite) foreign key in the relation *R*.
- *NonFK(x, R):* *x* is an attribute in relation *R* that does not participate in any foreign key.
- *Attr(x, R):* *x* is an attribute in relation *R*.

### B. Running Example

The example in Table I shows a university database of SQL statements. This database covers different cases for entities and relationships, such as relation built from strong entity, relation built from weak entity, relation for a multi-valued attribute, IS-A relationship, binary relationship and ternary relationship.

TABLE I. THE UNIVERSITY SQL-DDL

| |
|---|
| Department table:<br>Create table department(dept_id int, dept_name varchar(70) not null,dept_phone varchar(20),<br>primary key(dept_id)); |
| staff table:<br>Create table staff(staff_id int, staff_name varchar(70) not null,dept_id int, manger_id int,<br>primary key(staff_id),<br>foreign key (dept_id) references dept(dept_id)<br>foreign key (manger_id references staff(staff_id)); |
| Staff-details table:<br>Create table staff-details(staff_id int,staff_firstname varchar(10), staff_midname varchar(5), staff_familyname varchar(20), DOB date,address varchar(50),email varchar(50),memo varchar(120),homepage varchar(20),<br>primary key(staff_id),<br>foreign key (staff_id) references staff(staff_id)); |
| Academic-staff table:<br>Creat table academic-staff(staff_id int,<br>specialty varchar check in ('Demonstrator', 'Instructor','Assistant professor', 'Associate professor', 'Professor')<br>research_area varchar (40),<br>primary key(staff_id),<br>foreign key (staff_id) references staff(staff_id) ); |
| student table:<br>Create table student(st_id int, st_name varchar(70) not null,sex char(1), dept_id int,<br>primary key(st_id),<br>foreign key (dept_id) references dept(dept_id) ); |
| Graduate-student table:<br>Create table graduate-student(st_id int, research_area varchar(70),<br>primary key(st_id),<br>foreign key (st_id) references student(st_id)); |
| Dependent table:<br>Create table dependent(staff_id int, d_id int, depdent_name varchar (20) not null,sex char(1),DOB date, relationship varchar(40),<br>primary key(Staff_id, d_id),<br>foreign key (Staff_id) references staff (staff_id)); |
| course table:<br>Create table course(c_id int, course_name varchar(70) not null, dept_id int,<br>primary key(c_id),<br>foreign key ( dept_id) references dept(dept_id)); |
| registered table:<br>Create table registered(st_id int, c_id int,<br>primary key(st_id,c_id),<br>foreign key (st_id) references student(st_id),<br>foreign key (c_id) references course(c_id)); |
| Teach table:<br>Create table teach (st_id int, c_id int, staff_id int,<br>primary key(st_id,c_id,staff_id),<br>foreign key (st_id) references student(st_id),<br>foreign key (c_id) references course(c_id),<br>foreign key (staff_id)<br>references academic-staff (staff_id)); |

### C. Class and Datatype property Creation Rules

Our transformation rules have the form: IF condition THEN action. In the following rules, relation symbols are assumed to be universally quantified.

#### 1) Fragmentation rules

##### a) Rule $C_1$: Fragmentation class rule

If the information of one entity is spread across more than one relation, which is known as fragmentation (vertical partitioning of a table), then it should be integrated into one class, as follows:

$$Rule\_C_1 \quad \begin{cases} -\ \text{Class condition} \\ \exists x_1, x_2, \ldots, x_n : \begin{pmatrix} (a)\ \wedge_{i=1}^{n} PK(x_i, R_i)) \ \wedge\ \neg\, isFK(x_1, R_1)\ \wedge \\ (b)\ \wedge_{i=2}^{n} FK(x_i, R_i, x_1, R_1)\ \wedge \\ (c) \forall\, v\ \in Dom(x_1): (OCC(v, x_1, R_1) \Rightarrow \wedge_{i=2}^{n}\ OCC(v, x_i, R_i)) \end{pmatrix} \\ -\ \text{Class action} \\ \quad create\ class\ C\ from\ the\ relation\ \left(\bigcup_{i=1}^{n} R_i\right) \end{cases} \tag{1}$$

To apply this rule, all participating relations must satisfy all the conditions to be merged into one class.

1. There are two or more relations sharing the same primary key.
2. There is a master relation where
   a. its primary key is not a foreign key.
   b. the primary keys of all the remaining relations also act as foreign keys referring to the master relation.
3. All the primary key values present in the master relation must exist in all remaining relations.

If all three of these conditions are satisfied then we can merge these tables into one class in the ontology.

This relationship is of the one-to-one type, with a cardinality ratio (participation constraint) of (1, 1) on each side.

*b) Rule $DP_1$: Fragmentation datatype property rule*

All attributes present in each relation participating in the fragmentation rule and not forming a foreign key will be allocated to the integrated class created from Rule $C_1$, as depicted in equation 2. For example, the table *staff* has the attributes (*staff_id*, *staff_ family_name*) which do not form foreign keys. Similarly, the table *staff-details* has the attributes (*staff_first_name*, *staff_mid_name*, *DOB*, *address*, *email*, *memo*, *homepage*) which also do not form foreign keys, except that any repeated attribute will appear once in the integrated class. A datatype property is then created for each attribute in these two relations and allocated to the class *staff* which corresponds to the tables *staff* and *staff_details*. These datatype properties will consider the class *staff* as their domain and the corresponding datatype in Table II as their range; we refer to them by *dom(x)* in our rules. For example, *staff_first_name* will have the range of *xsd*: *string*.

$$Rule_{DP_1}\begin{cases} - \text{ Datatype condition:} \\ \quad \bigwedge_{i=1}^{n} (Attr(x, R_i) \ \wedge NonFK((x, R_i)) \\ - \text{ Datatype action :} \\ \quad create\ DP(x)\ with\ Domain\ C\ \ and\ Range\ dom(x) \end{cases} \quad (2)$$

TABLE II. DATATYPE BETWEEN DATABASE AND XML

| Type | DB | XML/ OWL |
|---|---|---|
| number | integer/int | *Xsd: integer* |
| | float | *Xsd: float* |
| Char | char /varchar/vchar | *Xsd:string* |
| date/Time | time | *Xsd:time* |
| | date | *Xsd:date* |
| | datetime | *Xsd: datetime* |
| Boolean | boolean | *Xsd:boolean* |

*2) Hierarchy rules*

*a) Rule $C_2$: Hierarchy class rule*

One of the most important stages in building the ontology is constructing the hierarchy. The term 'hierarchy' refers to the specification of the relationships between classes and subclasses. The class-subclass relationship appears as an IS-A relationship between entities in the E/R model. To extract the inheritance relationship, we require the analysis of tuples in each relation.

For example, some approaches [4] [5] fail to distinguish between the fragmentation of one entity into more than one relation and the IS-A relationship, which represents two different entities with the idea of superentity and subentity (generalization /specialization). Other approaches let the user decide [6]. The condition and the action below illustrate the rules for inheritance.

$$Rule\_C_2\begin{cases} - \text{ Class condition:} \\ \quad \exists x, y : \begin{pmatrix} (a) PK(x, R_1)) \ \wedge PK(y, R_2) \ \wedge \\ (b) FK(y, R_2, x, R_1) \wedge \\ (c) \exists\ v\ \in Dom(x) : ((OCC(v, x, R_1) \wedge \neg\ OCC(v, y, R_2)) \end{pmatrix} \\ - \text{ Class Action:} \\ \quad \begin{pmatrix} 1)\ create\ \ class\ C_1\ from\ R_1 \\ 2)\ create\ \ class\ C_2\ from\ R_2 \\ 3)\ make\ \ class\ C_2\ (subOf)\ C_1 \end{pmatrix} \end{cases} \quad (3)$$

To apply this rule, all participating relations must satisfy all the conditions to create two classes (superclass, subclass).

1. There are two relations sharing the same primary key.
2. There is a master relation where
   a. its primary key is not a foreign key, in which case it becomes a master relation. Otherwise there will be multiple inheritances.
   b. the primary key of the other relation also acts as a foreign key (referring to the master relation or to the sub-master).
3. Not all the primary key values present in the master relation must exist in the other relation.

When this rule applies, we can build a superclass from the master table and the other table will be a subclass in the ontology.

We consider this relationship to have one-to-one cardinality. The participation ratio must be (1, 1) in the master relation and (0, 1) in the other relation.

*b) Rule $DP_2$: Hierarchy datatype property rule*

Rule 4 demonstrates the datatype property rule for relations participating in hierarchy rules. From the class hierarchy rule ($C_2$) we create two classes, one for the master relation and the other for the subrelation. Here we have to ensure that all the attributes present in each relation will be moved to the corresponding class. For example, the tables *student* and *graduate_student* form a superclass and subclass respectively. Therefore, all the attributes in the table *student* do not form a foreign key which will be allocated to the class *student*. In our example (Table I) the attributes (*st_id st_name*, *sex*) will have the class *student* as their domain, while the class *graduate_student* will be the domain for the attribute *research_area*. Furthermore, all the attributes present in the class *student* will be inherited automatically by the class *graduate_student*.

$$Rule_{DP_2}\begin{cases} - \text{ Datatype condition:} \\ \quad \begin{pmatrix} Attr(x, R_1) \wedge NonFK\ (x, R_1) \wedge \\ Attr(y, R_2) \wedge NonFK\ (y, R_2) \end{pmatrix} \\ - \text{ Datatype action :} \\ \quad \begin{pmatrix} create\ DP(x)\ with\ Domain\ C_1\ \ and\ Range\ dom(x) \\ create\ DP(y)\ with\ Domain\ C_2\ and\ Range\ dom(y) \end{pmatrix} \end{cases} \quad (4)$$

Some approaches mix ontology design and database design. For example, if there are two classes represented in a hierarchy, the subclass will inherit all the datatype properties present in the superclass, so there is no need to define the datatype property domain from the union of the superclass and the subclass. The example below shows this inaccuracy.

```
<owl:DatatypeProperty rdf:ID="staff_id">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="#Staff"/>
              <owl:Class rdf:about="#Acadimac_staff"/>
          </owl:unionOf>
        </owl:Class>
     </rdfs:domain>
     <rdfs:range rdf:resource="&xsd;int"/>
  </owl:DatatypeProperty>
```

*3) Multi-valued rules*

*a) Rule $C_3$: Multi-valued class rule*

Databases cannot deal with a multi-valued attribute efficiently, while ontologies have an efficient way to deal with them. For example, if the *student* relation has the attribute *hobbies* as in Fig. 2, then the database designer has a choice of two ways to represent this attribute. The first is to put all the hobbies into one attribute, separated by commas (see Fig. 2A). Our system rejects this as bad design, adopting instead the second choice, which is to put the multi-valued attribute in separate relations to avoid duplicating tuples. The designer links each multi-valued relation with the master relation by placing the primary key of the master relation as part of the primary key of the multi-valued attribute.

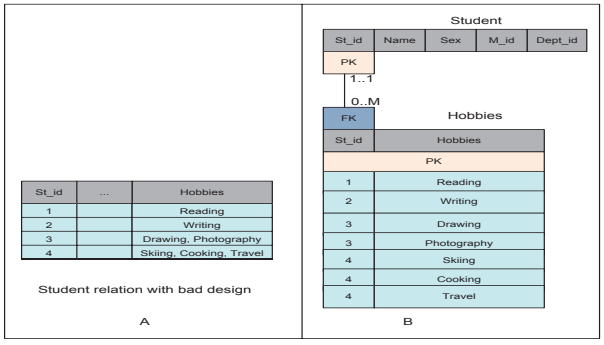


Figure 2. Representation of Multi-Valued Attributes

Thus, in our example, the attributes (*st_id, hobby*) will form the primary key of the *hobbies* relation. Hence, the multi-valued attribute relation appears as a weak entity. However, the difference between the weak entity and the multi-valued attribute relation is simple, because the multi-valued attribute relation has just one attribute beside the primary key of the master relation. Conversely, the weak relation has more than one attribute to describe the relation as well as the primary key of the master relation.

Therefore, in order to represent an entity with a multi-valued attribute, we merge the two relations representing the master relation and the multi-valued relation into one class. This is facilitated by the ability of OWL to combine a single valued attribute with a multi-valued attribute in the same instance. Rule $C_3$ is as follows:

$$Rule\_C_3 \begin{cases} - \text{ Class condition:} \\ \exists x, y : \begin{pmatrix} (a)\ PK(x \cup y, R_2)) \wedge \\ (b)\ \exists z : FK(x, R_2, z, R_1) \wedge \\ (c)\ Attr(y, R_2) \wedge NonFK((y, R_2) \end{pmatrix} \\ - \text{Class Action:} \\ \quad create\ class\ C\ from\ (R_1 \cup R_2) \end{cases} \quad (5)$$

The difference between this case and that of the fragmentation rule is the type of relationship involved. In the fragmentation case, the relationship is of the one-to-one type, while in case of multi-valued attribute, the relationship will be of the one-to-many type.

*b) Rule $DP_3$: Multi-valued datatype property rule*

The multi-valued relation has two attributes, one holding the relationship with the master relation and the other representing the multi-value attribute. Here we integrate the multi-value attribute to the class which corresponds to the master relation. In our example, the relation *hobby* represents the multi-valued attribute which contains the attributes (*st_id*, *hobby_name*), where the relation *hobby* ramifies from the relation *student*. Thus, the attribute *hobby_name* will have the class *student* as its domain.

$$Rule_{DP\,3} \begin{cases} - \text{ Datatype condition:} \\ \quad \bigwedge_{i=1}^{2} (Attr(x, R_i) \ \wedge NonFK((x, R_i)) \\ - \text{ Datatype action:} \\ \quad create\ DP(x)\ with\ Domain\ C\ \ and\ Range\ dom(x) \end{cases} \quad (6)$$

*4) Default rules*

*a) Rule $C_4$: Default class rule*

The condition and the action below demonstrate the default rule. Where rules $C_1$, $C_2$ and $C_3$ are inapplicable, then all remaining relations not representing a binary relationship with many-to-many cardinality will form a class. The default rule will form classes for strong and weak entities.

$$Rule\_c_4 \begin{cases} - \text{ Class condition:} \\ \neg\, \exists x, y : \begin{pmatrix} (a)\ PK(x \cup y, R)) \wedge isFK(x, R) \wedge isFK(y, R) \wedge \\ (b)\ \text{x} \cap \text{y} = \emptyset \wedge \forall z : (isFK(z, R) \Rightarrow z \in \{x, y\}) \wedge \\ (c)\ \forall \text{t} : (Attr(t, R) \Rightarrow t \in x \cup y) \end{pmatrix} \\ - \text{ Class action:} \\ \quad create\ class\ C\ from\ R \end{cases} \quad (7)$$

This rule will include more cases than strong and weak entities, such as the two following cases.

CASE 1: BINARY RELATIONSHIPS WITH ADDITIONAL ATTRIBUTES

Most approaches do not assume the existence of additional attributes describing a relationship, such as the *result* relation

having many-to-many cardinality with the *grade* attribute (see Table III for the schema).

TABLE III. SCHEMA FOR ATTRIBUTES ON RELATIONSHIP

| Relation | Primary Key(s) | Foreign Key(s) |
|---|---|---|
| Result (st_id, c_id, grade ) | St_id, c_id, | st_id (Student)<br>c_id (Course) |

We propose to create a new class for this kind of relationship with two pairs of inverse object properties and to create a datatype property for the additional attribute (see Fig. 3). The creation of object properties is discussed in detail in subsection D.

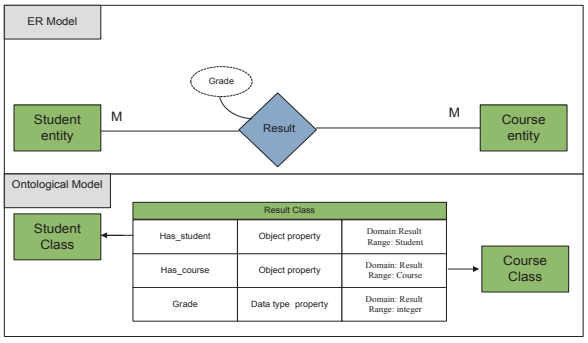


Figure 3. Many-to-many relationships with additional attributes

Note: We have chosen to deal with the number of referenced relations instead of the number of attributes in the foreign key. This will remove any confusion with a foreign key having composite attributes.

[4] and [6] fail to provide a general rule to decide whether a relationship is of the type (N-M) or not, because their rules suppose that each relation has just one attribute for the primary key. Thus, they do not consider the many-to-many relationship between a strong entity whose primary key has one attribute and a weak entity with a primary key having composite attributes.

CASE 2: N-ARY RELATIONSHIPS

OWL does not deal with n-ary relationships when n>2, so some approaches such as [4] [5] suggest decomposing any ternary or n-ary relationship to a binary relationship. Our approach first creates a class to deal with ternary or higher relationships, then decomposes these to binary relationships. This is the only way to ensure that such a relationship exists as a whole.

An example will explain the necessity to create a class for an n-ary relationship. Suppose we have the ternary relationship *teach*. When we decompose this to binary relationships, we will get three relations (see Fig. 4). The figure illustrates the repetition of data in all three tables, which violates the characteristic of the relational model by duplicating tuples, thus affecting the design of the ontology.

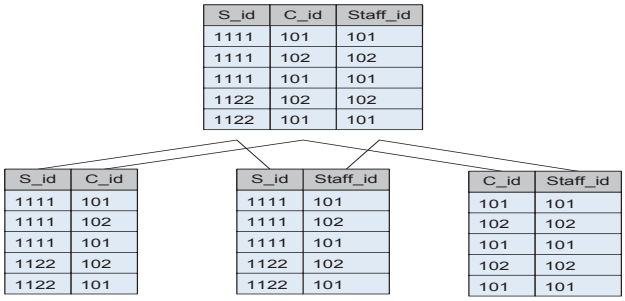


Figure 4. Ternary relation after decomposing

*b) Rule $DP_4$: Default datatype property rule*

For all relations not participating in any of the rules $DP_1$, $DP_2$ or $DP_3$, datatype properties can be created from their attributes which do not form foreign keys. Then each class will be allocated to its corresponding datatype properties.

$$Rule\_DP_4 \begin{cases} - \text{ Datatype condition:} \\ \quad Attr(x,R) \wedge (NonFK((x,R)) \\ - \text{ Datatype action :} \\ \quad create\ DP(x)\ with\ Domain\ C\ \ and\ Range\ dom(x) \end{cases} \quad (8)$$

We do not use a general rule for all cases such as in 8. Instead, we make specific datatype property rules for each class rule for more accuracy.

*c) Applying other SQL aspects to datatype property rules*

Table IV shows the rules for creating some SQL aspects.

TABLE IV. RULES FOR SOME SQL ASPECTS

| Sql aspect | OWL |
|---|---|
| Primary Key | *Functional, Cardinality is 1* |
| Not Null | *minCardinality is1* |
| Unique | *maxCardinality is 1* |
| Check in | *owl : oneOf* |

*5) Applying classes and datatype rules in our example*

The *Department* and *Course* relations are strong entities and do not satisfy rules $C_1$, $C_2$ or $C_3$. Even a weak entity can form a class unless it represents a relationship with a many-to-many cardinality. For example, the *dependent* relation is a weak entity, although it can form a class.

However, the *Registered* relation violates rule $C_4$, because it is a relation used to express the (N-M) relationship.

Finally, those classes are created from Table I by class creation rules:

- Fragmentation rule→ (*Staff* + *staff-details*)= *Staff*
- Hierarchy rule→ (*Academic- Staff*, *Staff*), (*Graduate-*student, *Student*)
- Default rule→ *Department*, *Course*, *dependent*, *teach*

The corresponding code below represents the end of the first stage.

*<owl:Class rdf:ID=" Department "/>*
*<owl:Class rdf:ID=" Course "/>*
*<owl:Class rdf:ID=" dependent "/>*

```
<owl:Class rdf:ID=" Staff "/>
<owl:Class rdf:ID="Academic- Staff "/>
    <rdfs:subClassOf rdf:resource="# Staff " />
<owl:Class rdf:ID=" Student"/>
<owl:Class rdf:ID="Graduate-student"/>
    <rdfs:subClassOf rdf:resource="# Student " />
```

The following example defines a datatype property in a class.

```
<owl:DatatypeProperty rdf:ID="staff_name">
      <rdfs:domain rdf:resource="#Staff"/>
      <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
```

*D. Rules for the creation of object properties*

Properties in OWL are of two types: object properties and datatype properties. This subsection presents the rules applying to object properties. The rules for creating datatype properties have been dealt with above.

Object properties in OWL are similar to relationships in the database. Before going further, we have to distinguish between the degree of the relationships and their cardinality. For example, a relationship of degree one exists in one entity, and is called a unary relationship; a relationship of degree two is a binary relationship between two entities and so on. In terms of cardinality, relationships can be of three types: one-to-one (1-1), one-to-many (1-M), or many-to-many (N-M). We deal with one-to-one relationships by either the fragmentation rule or the IS-A relationship. Rule $OP_1$ below concerns many-to-many relationships, while rules $OP_2$ and $OP_3$ apply to different cases representing one-to-many relationships, including n-ary relationships.

In fact, the general rule is that each relationship can be represented by a pair of inverse object properties or by two pairs of inverse object properties with a class.

*1) Rule $OP_1$: Object property rules for binary relationships (many-to-many)*

This rule is applied to relations built on top of a binary relationship with many-to-many cardinality. In an informal way, we check the primary key of a relation to see if it also plays the role of foreign key for this relation and then whether the foreign key attributes refer to the primary keys of two different relations. We ensure that there is a many-to-many relation where their attributes are a concatenation of the primary keys of two different relations.

$$Rule\_op_1 \begin{cases} Object\ condition \\ \exists x, y, x', y' : \begin{pmatrix} (a)\ FK(x, R, x', R_1)) \wedge FK(y, R, y', R_2) \wedge PK(x \cup y, R) \wedge \\ (b)\ x \cap y = \emptyset \wedge \forall z : (isFK(z, R) \Rightarrow z \in \{x, y\}) \wedge \\ (c)\ \forall t : (Attr(t, R) \Rightarrow t \in x \cup y) \end{pmatrix} \\ Action \\ 1 - Create\ an\ object\ OP_1\ with\ domain\ class\ C_1\ corresponding\ to\ (R_1) \\ \quad and\ rang\ class\ C_2\ corresponding\ (R_2) \\ 2 - Create\ an\ object\ OP_2\ with\ domain\ class\ C_2\ corresponding\ to\ (R_2) \\ \quad and\ rang\ class\ C_1\ corresponding\ (R_1) \\ 3 - OP_1\ and\ OP_2\ are\ inverse\ properties. \end{cases} \quad (10)$$

*2) Object property rules for one-to-many relationships*

There are two cases in this type of relationship.

*a) Rule $OP_2$: Object property rules for a relation with unary relationship*

For a relation R having a foreign key referencing the primary key of R itself (Fig. 5), we create two inverse object properties, as shown in the condition and action below. This is because this case has the one-to-many cardinality, regardless of the degree of the relationship.

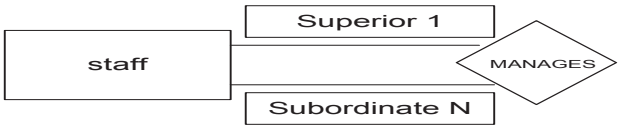


Figure 5. Relationships in the same relation

$$Rule\_op_2 \begin{cases} \text{Object condition}: \\ \exists x, y : \begin{pmatrix} (a)\ PK(x, R) \wedge \\ (b) FK(y, R, x, R) \wedge\ x \bigcap y = \emptyset \end{pmatrix} \\ \text{Action}: \\ 1 - Create\ an\ object\ OP_1\ with\ domain\ class\ C\ corresponding\ to\ (R) \\ \quad and\ rang\ class\ C\ corresponding\ (R)\ and\ cardinality\ 1 \\ 2 - Create\ an\ object\ OP_2\ with\ domain\ class\ C\ corresponding\ to\ (R) \\ \quad and\ rang\ class\ C\ corresponding\ (R) \\ 3 - OP_1\ and\ OP_2\ are\ inverse\ properties \end{cases} \quad (11)$$

The corresponding OWL descriptions are as follows.

```
<owl:ObjectProperty rdf:ID="has_subordinate">
      <rdfs:domain rdf:resource="#Staff"/>
      <owl:inverseOf rdf:resource="#has_superior"/>
     <rdfs:range rdf:resource="# Staff "/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_superior">
     <rdfs:domain rdf:resource="# Staff "/>
           <owl:inverseOf
     rdf:resource="#has_subordinate"/>
      <rdfs:range rdf:resource="# Staff "/>
  </owl:ObjectProperty>
  <owl:Restriction>
  <owl:onProperty rdf:resource="#has_ superior "/>
  <owl:cardinality rdf:datatype="&xsd;int">1
</owl:cardinality>
  </owl:Restriction>
```

*b) Rule $OP_3$: Object default rule*

For relations $R_1$ and $R_2$, if there is an attribute A part of $R_1$ referencing $R_2$, and regardless of whether A is part or not part of the primary key of $R_1$, then an object property ($OP_1$) and the inverse of the object property ($OP_2$) can be created between the classes $C_1$ and $C_2$ corresponding to $R_1$ and $R_2$ respectively. The $OP_1$ domain is $C_1$ and the range is $C_2$, and vice versa for $OP_2$.

$$Rule\_op_3 \begin{cases} \text{Object condition:} \\ \quad \exists x, y : FK(x, R_1, y, R_2) \qquad (12) \\ \text{Action:} \\ 1 - Create\ an\ object\ OP_1\ with\ domain\ class\ C_1 corresponding\ to\ (R_1) \\ \quad and\ rang\ class\ C_2\ corresponding\ (R_2) \\ 2 - Create\ an\ object\ OP_2\ with\ domain\ class\ C_2 corresponding\ to\ (R_2) \\ \quad and\ rang\ class\ C_1\ corresponding\ (R_1) \\ 3 - OP_1\ and\ OP_2\ are\ inverse\ properties \end{cases}$$

Some approaches such as [4] differentiate between a relationship existing between two relations $R_1$ and $R_2$, when an attribute A, a subset of the primary key of $R_1$, refers to the primary key of $R_2$, or if A is not part of the $R_1$ primary key. For the former case they create two object properties, $OP_1$ and $OP_2$. For the second case they create just one object property, OP, with domain $C_1$ and range $C_2$. We choose to represent each relationship by a pair of inverse object properties because this allows more semantic detail to be held.

This is the description in OWL of the objects *Staff work in Department* and *Department has worker Staff*:

```
<owl:ObjectProperty rdf:ID=" work_in">
      <rdfs:domain rdf:resource="# Staff "/>
      <owl:inverseOf rdf:resource="# has _worker "/>
      <rdfs:range rdf:resource="# department "/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID=" has _worker">
      <rdfs:domain rdf:resource="# department "/>
      <owl:inverseOf rdf:resource="# work_in"/>
      <rdfs:range rdf:resource="# Staff "/>
</owl:ObjectProperty>
```

This rule will also include the cases of:

- Binary relationships of cardinality (many-to-many) with additional attributes.
- Ternary and higher relationships.

In the case of the binary relationship (many-to-many) with additional attributes, we have already created a class to represent the binary relationship relation rule ($C_4$). Then the relationship will be changed to two one-to-many relationships between the three classes, as shown in Fig. 6. Thus, in this case, the relationship shows two pairs of inverse object properties with a class.

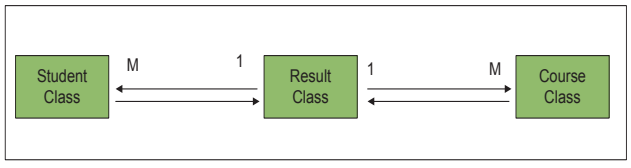


Figure 6. Binary Relationship with additional attribute

This rule will also treat a ternary or higher relationship in the same manner. For example, when we have a ternary relationship, we first create a class for it, using rule $C_4$, then we decompose the ternary relationship to a binary one, before creating two inverse object properties between each of the corresponding classes. The number of object properties in the n-ary relationship will be:

*N*(N-1).*

### *E. Rules for instances*

This step is optional in our system because we prefer that the data stay in the database, which is more powerful for storing large-scale data sets than the ontology. The system also considers moving queries from the global ontology website to the database tuples. However, if we need to create a knowledge base for the ontology produced by our system, we can use a two-step algorithm to migrate database tuples to ontological instances:

1- Each tuple is given a unique label name and migrated with all its data attributes except for the foreign keys.
2- A natural join link between the label tables and the foreign keys to instantiate each object property with its corresponding individual.

## V. IMPLEMENTATION

ER diagram and ontology have a similar structure. However, starting with E/R is difficult for many reasons for instances, the ER diagram is not available, the attributes do not have domain and there is no instances. So we generate our OWL ontology from SQL-DDL. We have designed and built a prototype system for our approach. This applies all the rules for creating classes, object properties and datatype properties. We have added an option for migrating data in tuples to the ontology instances. Our prototype is also flexible; For example, if a user prefers to use the SQL structure without any analysis of database tuples, our system gives him the choice of omitting the optional fragmentation rule. Therefore, there is the option of following a user-defined order of class creation rules (see Fig. 7). For example, if the user specifies the order of 2-3-4 for the creation of a class, this means applying the hierarchy rule for both first rule cases (fragmentation relations and multi-valued attribute relations).

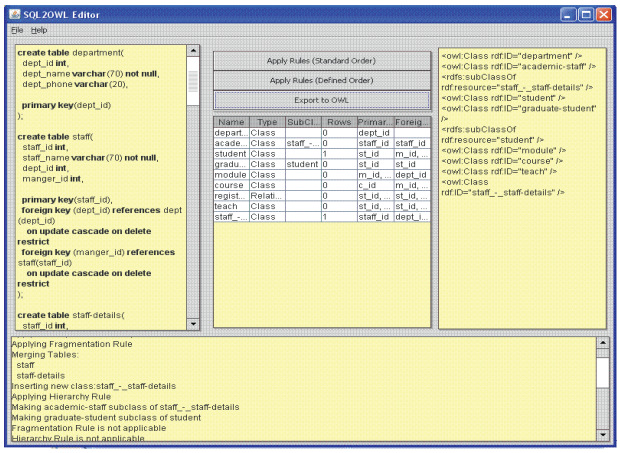


Figure 7. Screenshot of SQL2OWL Prototype

One of the characteristics of our system is that it shows the user the status of each table after applying the rules. For example, as each relation is converted to a class its type also changes to a class. Conversely, for a relation which has not

been converted to a class, the type stays as a relation in the *Registered* relation. The system also shows the user that classes have been integrated together by the combination of their names. Furthermore, we add to each object property the class names which relate to it.

## VI. VALIDATION

Because the validation process is still not automated in our system, we suggest involving domain experts in validation to help refine the ontology produced. The process uses a comparison between the E/R model of the database and the OWL ontology with these characteristics:

- The number of entities with relationships of the n-ary type where (n>2) in the E/R must be equal to the number of classes in the ontological model.
- A binary relationship with additional attributes must be counted as an entity.
- The IS-A relationship between two entities must be demonstrated between the classes derived from those entities by *subClassOf*.
- For each one-to-many relationship between two entities, two inverse object properties must exist.
- In each many-to-many relationship, two inverse object properties exist without a class.
- If any n-ary relationships exist where n>2, then there are two inverse object properties for each foreign key. The same applies to any binary relationship with an additional attribute.

In the example given in Table I, counting the entities in the E/R model and those in the ontological model will show them to be equal (see Table V). The reason is that both models draw on the conceptual model shown in Fig. 8.

TABLE V. E/R AND ONTOLOGY ELEMENTS

| E R model | no | Ontology model | no |
|---|---|---|---|
| Entity + ternary relationships | 6+1 | Classes | 7 |
| Binary relationships (foreign key) | 2 | Two inverse object properties | 2 |
| Ternary relationships (foreign keys) | 3 | Two inverse object properties for each foreign key in the n-ary relationship | 3 |
| (1-M) strong-weak entity | 1 | Two inverse object properties | 1 |
| (1-M) foreign keys | 4 | Two inverse object properties | 4 |
| IS-A relationships | 2 | subClassOf | 2 |

The remaining semantic features which should be transferred manually from the E/R to the ontology are as follows:

- If an entity contains a composite attribute, the user must build a class for it, then add an object property to link the sub-attribute to the main class. For example, the attribute *Name* has sub-attributes (*first name*, *middle name*, *family name*) in the *Staff* class. We create a class for the name, then create the datatype properties for the first name, middle name and family name. Next, we create a functional object property relating the *Name* class to the *Staff* class, such as *has_name*, and inverse functional object properties, to ensure that the relation is of the one-to-one type. All attributes in the *Name* class will also have a cardinality of one. Thus we ensure that each appears once [12].
- We then add the cardinality participation for each object property, because the cardinality restrictions cannot be obtained from SQL [13].

We add the quantifier restrictions "*allValuesFrom*" or "*someValuesFrom*", depending on the semantics, because these could not be inferred from the SQL statements.

Work related to our approach can be found in the community of the semantic web known as semantic annotation. However, the database community considers it to be reverse engineering of a database [6].

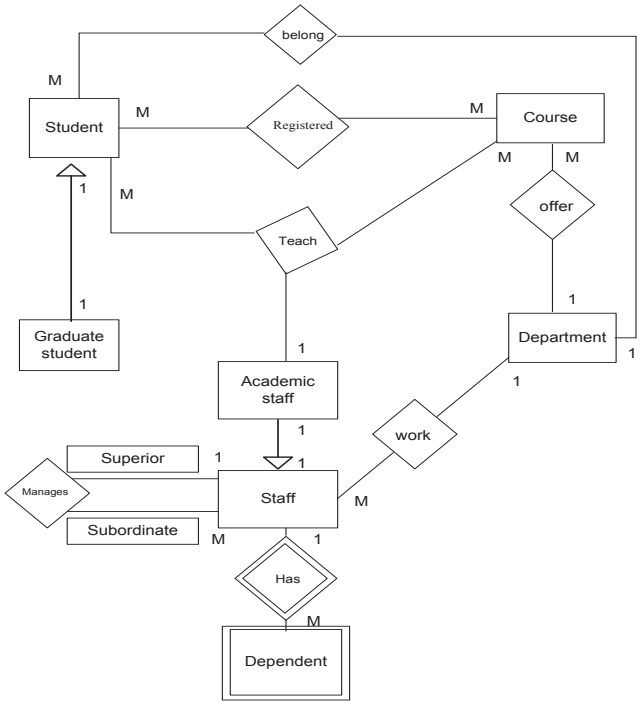


Figure 8. E/R Diagram for university (the attributes and keys are omitted for simplicity)

## VII. RELATED WORK

Many approaches rely on a variety of sources, such as E/R diagrams, extended E/R, relational schemata, SQL-DDL, database tuples, analysis of user queries and analysis of HTML, to construct the rule for the transformation of a relational database into an ontology. Most of these approaches use a combination of sources.

The idea of transformation was first applied to that from a relational model to an object-oriented model, then from a relational to (RDF) model which is an ontological model.

Stojanovic and others [6] established the rules for transforming databases to Frame Logics. However, their rules are manual and do not produce OWL ontologies. Also

they do not cover in their rules the issues of composite attribute or multi valued attribute. In [13], Bucella and others present global rules which cannot be implemented.
In [7], Astrova and others ignore the hierarchy rule or the fragmentation rule in their table mapping. Furthermore, they concentrate mostly on mapping constraints.

In [4] [12] [14] [15] the rules for transformation to OWL are specified. However, some rules in these approaches are not global, as they fail to cover certain types of entity or relationship, such as unary or binary relationships with additional attributes, or ternary and higher relationships.

Many approaches [4] [5] [13] [14] [16] have misleading rules in case of fragmentations and IS-A relationships (class hierarchies). There are many mistakes in their rules, such that they cannot be applied to all cases which may be presented in different databases.

The method in [11] [12] extracts a conceptual model (E/R diagram) from the source, then builds the transformation rule. The drawback of such approaches is that they build the ontology in the absence of necessary metadata from relations. In [5], the approach is based on the idea that the semantics extracted by analyzing HTML forms will be used to restructure and enrich the relational schema. The limitation of this work is that it involves a great deal of human participation. Also, ontologies can break down after any modification to the structure of the HTMLs on which they are based. Our approach, by contrast, deals successfully with complicated problems, such as distinguishing between fragmentations and IS-A relationships. We also deal with unary and binary relationships with additional attributes, ternary relationships, higher relationships and multi-valued attributes in our approach, whereas most previous approaches have failed to do so.

We utilize three techniques in our approach: the E/R model, the relational model and the analysis of database tuples.

## VIII. CONCLUSION AND FUTURE WORK

Many approaches are currently used to investigate the transformation of a relational model into an ontological one; these use either a relational schema or an E/R diagram. We have succeeded in combining the benefits of the relational model and the conceptual (E/R) model, by applying rules to the SQL statements and the inference of some metadata from the database tuples. We have then validated the resulting ontology with a conceptual E/R model of the database. It should also be mentioned that we have covered situations such as the relation with itself, and additional attributes of binary relationships ignored in other approaches. The dynamic aspects of SQL, such as triggers, assertions and referential actions, will be treated in future work.

## REFERENCES


[1] T. Gruber, A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, June 1993, pp.199-220.

[2] B. Swartout, R. Patil, K. Knight and T. Russ, "Toward Distributed Use of Large-Scale Ontologies", *Ontological Engineering AAAI-97 Spring Symposium Series*, 1997, pp. 138-148.

[3] N. Noy and D. McGuinness, "Ontology Development 101: A guide to creating your first ontology", Report, Stanford University, Stanford, CA, 94305.

[4] M. Li, X. Du and S. Wang, "Learning ontology from Relational Database", in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Guangzhou, August 2005,18-21.

[5] S. Benslimane, D. Benslimane and M. Malki, "Acquiring OWL Ontologies from Data-Intensive Web Sites", USA.ACM, Palo Alto, California, July 11-14, 2006.

[6] L. Stojanovic , N. Stojanovic and R. Volz, "Migrating data-intensive Web Sites into the Semantic Web". In *Proceedings of the 17th ACM symposium on applied computing ACM press*, 2002, pp. 1100-1107.

[7] I. Astrova, N. Korda and A. Kalja, "Rule-Based Transformation of SQL Relational Databases to OWL Ontologies". In *Proceedings of the 2nd International Conference on Metadata & Semantics Research*, October 2007.

[8] I. Astrova, N. Korda and A. Kalja, Storing OWL Ontologies in SQL Relational Databases, *International Journal of Electrical, Computer and System Engineering* , 2007.

[9] W. Hu and Y. Qu, *Discovering Simple Mappings Between Relational Database Schemas and Ontologies*, Springer-Verlag Berlin Heidelberg, 2007, pp. 225–238.

[10] OWL, Web Ontology Language (OWL), http://www.w3.org/2004/OWL,20/07/2009

[11] K. Sonia and S. Khan, "R2O Transformation System: Relation to Ontology Transformation for Scalable Data Integration", in *Proceedings of IDEAS08*, Coimbra, Portugal, September 2008.

[12] S. Upadhyaya and P.Kumar, "ERONTO: A Tool for Extracting Ontologies from Extended E/R Diagrams", *SAC'05 ACM Symposium on Applied Computing*, Santa Fe, New Mexico, USA, March 2005.

[13] A. Bucella, M.R. Penabad, F.J. Rodriguez, A. Farina, A. Cechich "From relational databases to OWL ontologies", Digital Libraries: Advanced Methods and Technologies. Digital Collection, Puschchino, Russia S.

[14] J. Barrasa, O. Corcho and A. G. Perez, "Fund Finder: A Case study of database to ontology Mapping". In *International Semantic Web Conference*, Number 2870 in Lecture Notes in Computer Science, pages 17-22, Sanibel Island, Florida, USA, October 2003. Springer-Verlag.

[15] Z. Xu, S. Zhang, Y. Dong, "Mapping between Relational Database Schema and OWL Ontology for Deep Annotation". In *IEEE/WIC/ACM International Conference on Web Intelligence*, WI 2006.

[16] S.H. Tirmizi, J. Sequeda and D. Miranker, "Translating SQL Applications to the Semantic Web" In S.S. Bhowmick, J. Küng, and R. Wagner (Eds.) *DEXA 2008, LNCS 5181*, pp. 450 – 464, Springer-Verlag, Berlin, 2008.