Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications, Elounda, Greece, August 18-20, 2006 (pp375-380)

Mapping of SQL Relational Schemata to OWL Ontologies

IRINA ASTROVA, AHTO KALJA
Institute of Cybernetics
Tallinn University of Technology
Akadeemia tee 21, 12618 Tallinn
ESTONIA

Abstract: A novel approach is proposed. This approach maps a relational database defined by a relational schema to an ontology. The ontology has a hierarchical structure, and it is no longer "impaired" by optimization and bad database design of the relational schema. The approach can be used for migrating HTML pages (especially those that are dynamically generated from a relational database) to the ontology-based Semantic Web.

Key-Words: Relational databases, ontologies, SQL, OWL, Web, Semantic Web

1 Introduction

One of the main driving forces for the Semantic Web has always been the expression, on the Web, of the vast amount of relational database information in a way that can be processed by machines [1]. Indeed, most information on the Web is not machine-processable, because it is often represented in HTML. This language describes how the information looks like and not what it is. In order for machines to process the information, it must be represented in an ontology language (e.g. OWL) and linked to ontologies. An ontology can be used for annotating HTML pages with semantics.

Manual or semi-automatic semantic annotation [2] is time-consuming, subjective and error-prone. It is even impossible on scale of the Web that contains billions of pages. Most pages even do not exist until they are dynamically generated from relational databases at the time of submitting HTML forms.

An alternative to the semantic annotation is automatic or semi-automatic mapping of relational databases (defined by relational schemata) to ontologies [3]. However, this mapping is difficult because a relational schema often captures few explicit semantics. It is often optimized for performance reasons. And it is often bad designed, as it may be done by novice and untrained database designers who are not familiar with database theory and database methodology [4].

2 Related Work

A majority of the work has been done on extracting entity-relationship (ER) models from relational databases. There are few approaches that consider OWL ontologies as the target; e.g.:

Colomb et al. [5] propose an approach to automatic mapping of ER models to OWL ontologies (and back) via DL. The drawback of this approach is that it **does not discover semantics**; it just changes syntactic form.

Upadhyaya and Kumar [6] propose an approach to automatic mapping of extended ER models to OWL ontologies. The drawback of this approach is that it **does not suit legacy systems** that often come with no or out-of-date extended ER models.

Buccella et al. [7] propose an approach to semi-automatic mapping of SQL relational schemata to OWL ontologies. The drawback of this approach is that it **does not address optimization and bad database design** that often occur in practice. And it **ignores inheritance**, thus extracting an ontology that looks rather "relational"; i.e. the ontology has the same flat structure as the original relational schema.

As an attempt to rectify the drawbacks of existing approaches, we propose a novel approach.

3 Our Approach

Our approach maps a relational schema to an ontology. The relational schema is represented in SQL [8], the standard relational database language. This language includes syntax for specifying tables, columns with data types, constraints on columns, and other semantics. The ontology is represented in OWL [9], the standard ontology language. This language includes syntax for specifying classes, properties with domains and ranges, inheritance of

classes and properties, constraints on classes and properties, and other semantics.

There are five steps of our approach: (1) classification of tables, (2) mapping tables, (3) mapping columns, (4) mapping relationships, and (5) mapping constraints. Next these steps will be illustrated by example.

3.1 Classification of Tables

The first step of our approach is classification of tables. Each table is classified into one of the three categories: base, dependent and composite tables.

If a table is independent of any other table in the relational schema, it is a *base table*. If a table depends on another table, it is a dependent table. That is, a *dependent table* is a table whose primary key includes the primary key of another table. All other tables fall into the category of composite tables. That is, a *composite table* is a table that is neither base nor dependent.

Example 1. A table `Employee` in Fig. 1 is a base table, because it has no foreign key.

```
CREATE TABLE Employee(
 employeeID INTEGER PRIMARY KEY)
```

Figure 1. Base tables

Example 2. A table `Project` in Fig. 2 is also a base table, because its foreign key is not (part of) its primary key.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 managerID INTEGER REFERENCES Employee)
```

Figure 2. Base tables (contd.)

Example 3. A table `SoftwareProject` in Fig. 3 is a dependent table, because its foreign key is its primary key.

```
CREATE TABLE SoftwareProject(
 projectID INTEGER PRIMARY KEY REFERENCES
Project)
```

Figure 3. Dependent tables

Example 4. A table `Task` in Fig. 4 is also a dependent table, because its foreign key is part of its primary key.

```
CREATE TABLE Task(
 taskID INTEGER,
 projectID INTEGER PRIMARY KEY REFERENCES
Project,
 CONSTRAINT Task_PK PRIMARY KEY(taskID,
projectID))
```

Figure 4. Dependent tables (contd.)

Example 5. A table `Involvement` in Fig. 5 is a composite table, because its primary key is composed of the primary keys of two other tables: `Employee` and `Project`.

```
CREATE TABLE Involvement(
 employeeID INTEGER REFERENCES Employee,
 projectID INTEGER REFERENCES Project,
 CONSTRAINT Involvement_PK PRIMARY
KEY(employeeID, projectID))
```

Figure 5. Composite tables

3.2 Mapping Tables

The second step of our approach is mapping tables. Each table maps to a class, with the exception of composite tables. Composite tables represent relationships. Therefore, they map to classes or object properties[1], depending on their structures.

Example 1. Consider a table `Project` in Fig. 6. This table maps to a class `Project`, because it is a base table.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 managerID INTEGER REFERENCES Employee)
                    ↓
<owl:Class rdf:ID="Project"/>
```

Figure 6. Mapping base tables

Example 2. Consider a table `Software-Project` in Fig. 7. This table maps to a class `SoftwareProject`, because it is a dependent table.

```
CREATE TABLE SoftwareProject(
 projectID INTEGER PRIMARY KEY REFERENCES
Project)
                    ↓
<owl:Class rdf:ID="SoftwareProject"/>
```

Figure 7. Mapping dependent tables

Example 3. Consider a composite table `Involvement` in Fig. 8. This table represents a binary relationship between two tables: `Employee` and `Project`. Since the table entirely consists of

[1] A composite table can also map to multiple inheritance. But as there is no really good way to represent multiple inheritance in a relational schema, the table will map to a class. Users can then replace that class with multiple inheritance.

the primary keys of the two tables, it maps to a pair of object properties: `involves` and `involvedBy`. One object property is the inverse of another, meaning that the relationship is bidirectional; i.e. a project involves an employee. And an employee is involved in a project.

```
CREATE TABLE Involvement(
 employeeID INTEGER REFERENCES Employee,
 projectID INTEGER REFERENCES Project,
 CONSTRAINT Involvement_PK PRIMARY
KEY(employeeID, projectID))
                    ↓
<owl:ObjectProperty rdf:ID="involves">
 <rdfs:domain rdf:resource="#Project"/>
 <rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="involvedIn">
 <owl:inverseOf
rdf:resource="#involves"/>
</owl:ObjectProperty>
```

Figure 8. Mapping composite tables

Example 4. Consider a composite table `Involvement` in Fig. 9. Not only does that table consist of the primary keys of the two tables, but it also contains an additional column `hours`. Therefore, it maps to a class `Involvement` and a pair of object properties for each table participating in the relationship. (The object properties are not shown in Fig. 9 because of a lack of space.)

```
CREATE TABLE Involvement(
 employeeID INTEGER REFERENCES Employee,
 projectID INTEGER REFERENCES Project,
 hours INTEGER,
 CONSTRAINT Involvement_PK PRIMARY
KEY(employeeID, projectID))
                    ↓
<owl:Class rdf:ID="Involvement"/>
```

Figure 9. Mapping composite tables (contd.)

Example 5. Consider a composite table `Involvement` in Fig. 10. This table represents a ternary relationship between three tables: `Employee`, `Project` and `Skill`. Since only a binary relationship can be represented through a pair of object properties, the table maps to a class `Involvement` and a pair of object properties for each table participating in the relationship. (The object properties are not shown in Fig. 10 because of a lack of space.)

```
CREATE TABLE Involvement(
 employeeID INTEGER REFERENCES Employee,
 projectID INTEGER REFERENCES Project,
 skillID INTEGER REFERENCES Skill,
 CONSTRAINT Involvement_PK PRIMARY
KEY(employeeID, projectID, skillID))
                    ↓
<owl:Class rdf:ID="Involvement"/>
```

Figure 10. Mapping composite tables (contd.)

3.3 Mapping Columns

The third step of our approach is mapping columns. Each column in a table maps to a data type property in a class, with the exception of foreign keys. Foreign keys are ignored for a while, as they represent relationships.

Example 1. Consider a column `projectID` in a table `Project` in Fig. 11. This column maps to a data type property `projectID` in a class `Project`.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY)
                    ↓
<owl:DatatypeProperty rdf:ID="projectID">
 <rdfs:domain rdf:resource="#Project"/>
 <rdfs:range
rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
```

Figure 11. Mapping columns

3.4 Mapping Relationships

The fourth step of our approach is mapping relationships. Relationships map to classes, object properties or inheritance, depending on the types of key, data and column correlations.

Given two tables r_1 and r_2, there are four types of:

- *Key correlation*: key equality: $K_1=K_2$, key inclusion: $K_1 \subset K_2$, key intersection: $K_1 \cap K_2 \neq \emptyset$, $K_1 - K_2 \neq \emptyset$, $K_2 - K_1 \neq \emptyset$ and key disjointness: $K_1 \cap K_2 = \emptyset$
- *Data correlation*: data equality: $r_1[K_1]=r_2[K_2]$, data inclusion: $r_1[K_1] \subset r_2[K_2]$, data intersection: $r_1[K_1] \cap r_2[K_2] \neq \emptyset$, $r_1[K_1] - r_2[K_2] \neq \emptyset$, $r_2[K_2] - r_1[K_1] \neq \emptyset$ and data disjointness: $r_1[K_1] \cap r_2[K_2] = \emptyset$
- *Column correlation*: column equality: $C_1=C_2$, column inclusion: $C_1 \subset C_2$, column intersection: $C_1 \cap C_2 \neq \emptyset$, $C_1 - C_2 \neq \emptyset$, $C_2 - C_1 \neq \emptyset$ and column disjointness: $C_1 \cap C_2 = \emptyset$,

where K_1 and K_2 are primary keys of r_1 and r_2, respectively; C_1 and C_2 are columns of r_1 and r_2, respectively; $r_1[K_1]=\pi_{K1}(r_1)$ and

$r_2[K_2]=\pi_{K2}(r_2)$ are projections of r_1 and r_2 on K_1 and K_2, respectively.

Example 1. Consider a relationship between Employee and Project in Fig. 12, when **key disjointness** holds on it. This relationship relates two tables, which are independent of each other. Since the relationship is binary, it maps to a pair of object properties: manages and managedBy. One object property is the inverse of another, meaning that the relationship is bidirectional; i.e. an employee manages a project. And a project is managed by an employee.

```
CREATE TABLE Employee(
 employeeID INTEGER PRIMARY KEY)
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 managerID INTEGER REFERENCES Employee)
                        ↓
<owl:ObjectProperty rdf:ID="manages">
 <rdfs:domain rdf:resource="#Employee"/>
 <rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="managedBy">
 <owl:inverseOf rdf:resource="#manages"/>
</owl:ObjectProperty>
```

Figure 12. Key disjointness

Example 2. Consider a relationship between Project and SoftwareProject in Fig. 13, when **key equality** and **data inclusion** hold on it. This relationship maps to single inheritance, as all data of SoftwareProject are also included in Project; i.e. a software project is a project. But the converse is not true; e.g. some projects can be hardware projects.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT,
 dueDate DATE)
CREATE TABLE SoftwareProject(
 projectID INTEGER PRIMARY KEY REFERENCES
Project,
 language VARCHAR)
                        ↓
<owl:Class rdf:ID="SoftwareProject">
 <rdfs:subClassOf
rdf:resource="#Project"/>
</owl:Class>
```

Figure 13. Key equality and data inclusion

Example 3. Consider a relationship between HardwareProject and SoftwareProject in Fig. 14, when **key equality, data disjointness** and **column intersection** hold on it. This example also illustrates single inheritance. Because some columns are common to both (disjoint) tables, HardwareProject and SoftwareProject are part of the inheritance hierarchy; but there is no table corresponding to their superclass. Therefore, we create a class Project whose subclasses are HardwareProject and SoftwareProject.

```
CREATE TABLE HardwareProject(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT,
 dueDate DATE,
 supplier VARCHAR)
CREATE TABLE SoftwareProject(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT,
 dueDate DATE,
 language VARCHAR)
                        ↓
<owl:Class rdf:ID="Project">
 <owl:DatatypeProperty
rdf:ID="projectID">
 <rdfs:range
rdf:resource="&xsd;integer"/>
 </owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="budget">
 <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="dueDate">
 <rdfs:range rdf:resource="&xsd;date"/>
 </owl:DatatypeProperty>
</owl:Class>
<owl:Class rdf:ID="HardwareProject">
 <rdfs:subClassOf
rdf:resource="#Project"/>
<owl:DatatypeProperty rdf:ID="supplier">
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
</owl:Class>
<owl:Class rdf:ID="SoftwareProject">
<rdfs:subClassOf
rdf:resource="#Project"/>
<owl:disjointWith
rdf:resource="#HardwareProject"/>
<owl:DatatypeProperty rdf:ID="language">
<rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
</owl:Class>
```

Figure 14. Key equality, data disjointness and column intersection

Example 4. Consider a relationship between HardwareProject and SoftwareProject in Fig. 15, when **key equality, data disjointness** and **column equality** hold on it. This is an example of optimization: horizontal partitioning, where data of a single table have been split into two (disjoint) tables, having the same columns. Therefore, we create a class Project whose individuals are the union of the data of the two tables.

```
CREATE TABLE HardwareProject(
 projectID INTEGER PRIMARY KEY,
```

```
 budget FLOAT,
 dueDate DATE)
CREATE TABLE SoftwareProject(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT,
 dueDate DATE)
                    ↓
<owl:Class rdf:ID="SoftwareProject">
 <owl:disjointWith
rdf:resource="#HardwareProject"/>
 </owl:Class>
<owl:Class rdf:ID="Project">
 <owl:unionOf rdf:parseType="Collection">
  <owl:Class
rdf:about="#SoftwareProject"/>
  <owl:Class
rdf:about="#HardwareProject"/>
 </owl:unionOf>
</owl:Class>
```

Figure 15. Key equality, data disjointness and column equality

Example 5. Consider a relationship between `HardwareProject` and `SoftwareProject` in Fig. 16, when **key equality, data intersection** and **column equality** hold on it. Because some data are common to both tables, we create a class `Project` whose individuals are the intersection of the data of the two tables.

```
CREATE TABLE HardwareProject(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT,
 dueDate DATE)
CREATE TABLE SoftwareProject(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT,
 dueDate DATE)
                    ↓
<owl:Class rdf:ID="Project">
 <owl:intersectionOf
rdf:parseType="Collection">
  <owl:Class
rdf:about="#SoftwareProject"/>
  <owl:Class
rdf:about="#HardwareProject"/>
 </owl:intersectionOf>
</owl:Class>
```

Figure 16. Key equality, data intersection and column equality

3.5 Mapping Constraints

The fifth step of our approach is mapping constraints. Constraints specify if a column in a table is unique or not null, or if the column is a primary key or a foreign key. Constraints also specify a data range for the column.

Example 1. Consider a unique constraint in Fig. 17. This constraint specifies that a column `budget` in a table `Project` is unique, meaning that no two data in the table have the same value for the column. Therefore, the constraint maps to a functional property.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT UNIQUE)
                    ↓
<owl:FunctionalProperty rdf:ID="budget"/>
```

Figure 17. Mapping unique constraints

Example 2. Consider a not null constraint in Fig. 18. This constraint specifies that a column `budget` in a table `Project` is not null, meaning that all data in the table contains values for the column. Therefore, the constraint maps to a minimum cardinality of 1.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 budget FLOAT NOT NULL)
                    ↓
<owl:Class rdf:ID="Project">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="#budget"/>
    <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 18. Mapping not null constraints

Example 3. Consider a primary key constraint in Fig. 19. This constraint specifies that a column `projectID` in a table `Project` is a primary key. Since the primary key implies that the column is both unique and not null, the constraint maps to both a functional property and a minimum cardinality of 1.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY)
                    ↓
<owl:Class rdf:ID="Project">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="#projectID"/>
   <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1/>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:FunctionalProperty
rdf:ID="projectID"/>
```

Figure 19. Mapping primary key constraints

Example 4. Consider a check constraint in Fig. 20. This constraint specifies a data range for a column `type` in a table `Project` through a list of possible values. Therefore, the constraint maps to an enumerated data type.

```
CREATE TABLE Project(
 projectID INTEGER PRIMARY KEY,
 type VARCHAR CHECK IN ("Software",
"Hardware"))
                    ↓
<owl:Class rdf:ID="Project">
 <owl:DatatypeProperty rdf:ID="type">
  <rdfs:range>
   <owl:DataRange>
    <owl:oneOf>
     <rdf:List>
      <rdf:first
rdf:datatype="&xsd;string">"#Software"
      </rdf:first>
      <rdf:rest>
       <rdf:List>
        <rdf:first
rdf:datatype="&xsd;string">"#Hardware"
        </rdf:first>
        <rdf:rest rdf:resource="&rdf;nil"
/>
       </rdf:List>
      </rdf:rest>
     </rdf:List>
    </owl:oneOf>
   </owl:DataRange>
  </rdfs:range>
 </owl:DatatypeProperty>
</owl:Class>
```

Figure 20. Mapping check constraints

4 Conclusion

We have proposed a novel approach to mapping relational schemata to ontologies. Our approach is based on an analysis of key, data and column correlations as well as their combinations. This analysis helps us: (1) discover "hidden" (implicit) semantics; and (2) address optimization and bad database design.

In the future, our approach can be used for migrating HTML pages (especially those that are dynamically generated from relational databases) to the ontology-based Semantic Web. The main reason for this migration is to make the relational database information on the Web machine-processable.

Acknowledgement

This research is partly sponsored by ESF (Estonian Science Foundation) under the grant nr. 5766.

References:

[1] T. Berners-Lee, Relational Databases on the Semantic Web, 2002, http://www.w3.org/DesignIssues/RDB-RDF.html

[2] M. Erdmann, A. Maedche, H. Schnurr and S. Staab, From Manual to Semi-automatic Semantic Annotation: About Ontology-based Text Annotation Tools, *Linköping Electronic Articles in Computer and Information Science Journal (ETAI)*, Vol. 6, No. 2, 2001

[3] L. Stojanovic, N. Stojanovic and R. Volz, Migrating Data-intensive Web Sites into the Semantic Web, *Proceedings of the 17th ACM Symposium on Applied Computing (SAC)*, 2002, pp. 1100-1107

[4] W. Premerlani and Blaha, M.: An Approach for Reverse Engineering of Relational Databases, *Communications of the ACM*, Vol. 37, No. 5, 1994, pp. 42-49

[5] R. Colomb, A. Gerber and M. Lawley, Issues in Mapping Metamodels in the Ontology Development Metamodel, *Proceedings of the 1st International Workshop on the Model-Driven Semantic Web (MSDW)*, 2004, pp. 20-24

[6] S. Upadhyaya and P. Kumar, ERONTO: A Tool for Extracting Ontologies from Extended ER Diagrams, *Proceedings of the 20th ACM Symposium on Applied Computing (SAC)*, 2005, pp. 667-670

[7] A. Buccella, M. Penabad, F. Rodriguez, A. Farina and A. Cechich, From Relational Databases to OWL Ontologies, *Proceeding of the 6th National Russian Research Conference (RCDL)*, 2004

[8] J. Melton and A. Simon, *Understanding the New SQL: A Complete Guide*, San Mateo, CA: Morgan Kaufmann, 1993

[9] OWL Web Ontology Language Guide, 2004, http://www.w3.org/TR/owl-guide