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 SoftwareProject 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))

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))

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))

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)

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 r1 and r2, there are four types of:

- **Key correlation**: key equality: r1\[K1\]=r2\[K2\], key inclusion: r1\[K1\]⊂r2\[K2\], key intersection: r1\[K1\]∩r2\[K2\]≠∅, r1\[K1\]-r2\[K2\]≠∅, r2\[K2\]-r1\[K1\]≠∅ and key disjointness: r1\[K1\]∩r2\[K2\]=∅
- **Data correlation**: data equality: r1\[K1\]=r2\[K2\], data inclusion: r1\[K1\]⊂r2\[K2\], data intersection: r1\[K1\]∩r2\[K2\]≠∅, r1\[K1\]-r2\[K2\]≠∅, r2\[K2\]-r1\[K1\]≠∅ and data disjointness: r1\[K1\]∩r2\[K2\]=∅
- **Column correlation**: column equality: r1\[K1\]=r2\[K2\], column inclusion: r1\[C1\]⊂r2\[C2\], column intersection: r1\[C1\]∩r2\[C2\]≠∅, r1\[C1\]-r2\[C2\]≠∅, r2\[C2\]-r1\[C1\]≠∅ and column disjointness: r1\[C1\]∩r2\[C2\]=∅.

where K1 and K2 are primary keys of r1 and r2, respectively; C1 and C2 are columns of r1 and r2, respectively; r1\[K1\]=\pi_{K1}(r1) and
$r_2[K_2]=\pi_{K_2}(r_2)$ 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">  
<owl: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 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 REFERENCES Project,
    language VARCHAR)

<owl:Class rdf:ID="Project">  
<owl:DatatypeProperty rdf:ID="projectID">  
<rdfs:domain rdf:resource="#Project"/>  
<rdfs:range rdf:resource="&xsd;integer"/>  
</owl:DatatypeProperty>
</owl:Class>

<owl:Class rdf:ID="HardwareProject">  
<owl:subClassOf rdf:resource="#Project"/>  
<owl:DatatypeProperty rdf:ID="supplier">  
<rdfs:domain rdf:resource="#HardwareProject"/>  
<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,
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.

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.

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.
**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'))
```

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:


