AUTOMATIC TRANSFORMATION OF SQL RELATIONAL DATABASES TO OWL ONTOLOGIES

Irina Astrova and Ahto Kalja
Tallinn University of Technology, Tallinn, Estonia

Keywords: Relational databases, ontologies, SQL and OWL.

Abstract: This paper proposes a novel approach to automatic transformation of relational databases (written in SQL) to ontologies (written in OWL), where domains and constraints CHECK are also considered. The proposed approach can identify (inverse) functional, symmetric and transitive properties, cardinality and value restrictions, and enumerated classes and data types.

1 INTRODUCTION

Today it is common to get data from relational databases over the Web. These databases are generally separate and not easily used as merged data sources. The W3C vision sees ways to unify the description and retrieval of the data using ontologies, thus allowing much of the Web to be part of a large interoperable database.

Thus, there is a need to transform relational databases to ontologies. However, manual transformation is hard to do and often takes a lot of time. Thus, there is also a need to automate the transformation.

2 RELATED WORK

While there are several approaches to automatic transformation relational databases to ontologies - e.g. (Buccella et al., 2004; Li et al., 2005; Shen et al., 2006; Astrova and Kalja, 2006; Sequeda et al., 2007), many situations are too complex or require more flexibility than the existing approaches enable.

E.g. a company may wish to trace the skills of its employees in order to assign the employees to the projects. Since an employee may have many skills, `skill` becomes multivalued. It is possible to represent `skill` as a data type property in the ontology. But it is not possible to represent `skill` as a column in the relational database, because the column may have at most one value for each row in the table (atomicity). One solution to this problem is to create a separate table (McFadden et al., 1999). This table could map to a data type property during the transformation. However, the existing approaches cannot recognize such a situation. Rather, they map the table to a class. Moreover, the existing approaches cannot identify inverse functional, symmetric and transitive properties, value restrictions, and enumerated classes.

As an attempt to resolve these problems, this paper proposes a novel approach to automatic transformation of relational databases to ontologies. The main objective of the proposed approach is to preserve as many semantics as possible during the transformation. The proposed approach assumes that a relational database is written in SQL (SQL, 2002) and that an ontology is written in OWL (OWL, 2004).

3 APPROACH

The proposed approach maps constructs of a relational database (i.e. tables, domains, columns, constraints, and rows) to an ontology using the names of constructs in the relational database as the names of constructs in the ontology. Next this mapping will be illustrated by example. An example is the relational database of a company.

3.1 Mapping Tables

A table can be mapped to three different constructs in the ontology: a class, a data type property, and an object property and its inverse.

Astrova I. and Kalja A. (2008).
AUTOMATIC TRANSFORMATION OF SQL RELATIONAL DATABASES TO OWL ONTOLOGIES.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 131-136
DOI: 10.5220/0001515101310136

A table Project in Figure 1 has its own primary key. Therefore, this table maps to a class Project.

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

Figure 1: Table maps to class.

The primary key of a table Software-Project in Figure 2 is a foreign key to another table Project. Therefore, this table maps to a class SoftwareProject.

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

Figure 2: Table maps to class (contd.).

The primary key of a table Involvement in Figure 3 is composed of foreign keys to two other tables Project and Employee, indicating a binary (many-to-many) relationship. Since there are no other columns in the table Involvement, it maps to two object properties: EmployeeID (that uses classes Project and Employee as its domain and range, respectively) and ProjectID. The latter is an inverse of the former, meaning that the relationship is bidirectional (i.e. a project involves employees and an employee is involved in projects). If the table Involvement had an additional column say hours, it would be mapped to a class Involvement.

```
CREATE TABLE Involvement(
 EmployeeID INTEGER REFERENCES
Employee,
 ProjectID INTEGER REFERENCES Project,
 PRIMARY KEY (EmployeeID, ProjectID))
                    ↓
<owl:ObjectProperty
rdf:ID="EmployeeID">
 <rdfs:domain rdf:resource="#Project"/>
 <rdfs:range rdf:resource="#Employee"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ProjectID">
 <owl:inverseOf
rdf:resource="#EmployeeID"/>
</owl:ObjectProperty>
```

Figure 3: Table maps to object property and its inverse.

The primary key of a table SkillValue in Figure 4 is composed of a column skill and a foreign key to another table Employee, meaning that the column skill is multivalued (i.e. an employee may have zero or more skills). Since there are no other columns in the table SkillValue, it maps to a data type property skill that uses a class Employee as its domain. If the table SkillValue had an additional column say level, it would be mapped to a class SkillValue.

```
CREATE TABLE SkillValue(
 skill VARCHAR,
 EmployeeID INTEGER REFERENCES
Employee,
 PRIMARY KEY (skill, EmployeeID))
                    ↓
<owl:DatatypeProperty rdf:ID="skill">
 <rdfs:domain
rdf:resource="#Employee"/>
 <rdfs:range
rdf:resource="&xsd;sting"/>
</owl:DatatypeProperty>
```

Figure 4: Table maps to data type property.

The primary key of a table Involvement in Figure 5 is composed of foreign keys to three other tables Employee, Project and Skill, indicating a ternary relationship. Since only binary relationships can be represented through object properties, this table maps to a class Involvement.

```
CREATE TABLE Involvement(
 EmployeeID INTEGER REFERENCES
Employee,
 ProjectID INTEGER REFERENCES Project,
 SkillID INTEGER REFERENCES Skill,
 PRIMARY KEY (EmployeeID, ProjectID,
SkillID))
                    ↓
  <owl:Class rdf:ID="Involvement"/>
```

Figure 5: Table maps to class (contd.).

3.2 Mapping Domains

A domain maps to a class unless there is a constraint CHECK with enumeration on it. Then it maps to an enumerated class.

A domain ProjectType in Figure 6 is defined as the data type of all strings. Therefore, this domain

maps to a class ProjectType, with a data type property say type that uses string as its range.

```
CREATE DOMAIN ProjectType AS VARCHAR
                        ↓
<owl:Class rdf:ID="ProjectType">
 <owl:DatatypeProperty rdf:ID="type">
  <rdfs:range
rdf:resource="&xsd;string"/>
 </owl:DatatypeProperty>
</owl:Class>
```

Figure 6: Domain maps to class.

A domain ProjectType in Figure 7 is defined as the data type of all strings, again. However, there is now a constraint CHECK on it. This constraint specifies the domain ProjectType through a list of values Software and Hardware (also known as enumeration). Therefore, it maps to an enumerated class ProjectType, with individuals for each value in the list.

```
CREATE DOMAIN ProjectType AS VARCHAR
 CONSTRAINT  ProjectType_Constraint
 CHECK IN ('Software', 'Hardware')
                        ↓
<owl:Class rdf:ID="ProjectType">
 <owl:oneOf rdf:parseType="Collection">
  <owl:Thing rdf:about="#Software"/>
  <owl:Thing rdf:about="#Hardware"/>
 </owl:oneOf>
</owl:Class>
```

Figure 7: Domain maps to enumerated class.

3.3 Mapping Columns

A column that is not (part of) a foreign key maps to a data type property unless it uses a domain as its data type. Then it maps to an object property. This is because the domain maps itself to either a class or an enumerated class (see Section 3.2).

A column ssn in a table Employee in Figure 8 is not a foreign key. Therefore, this column maps to a data type property ssn that uses a class Employee as its domain. This property has a maximum cardinality of 1, because the column ssn may have at most one value for each row in the table Employee (atomicity). Alternatively, the property ssn could be defined as functional, which is the same as saying that the maximum cardinality is 1. It should be noted that if the column ssn were a surrogate key, it would be ignored. A surrogate key is internally generated by the relational database management system using an automatic sequence number generator or its equivalence; e.g. an IDENTITY in SQL Server and Sybase, a SEQUENCE in Oracle and an AUTO_INCREMENT in MySQL.

```
CREATE TABLE Employee(
 ssn INTEGER CHECK (ssn > 0))
                        ↓
<owl:DatatypeProperty rdf:ID="ssn">
 <rdfs:domain
rdf:resource="#Employee"/>
 <rdfs:range
rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
<owl:Class rdf:ID="Employee">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="#ssn"/>
   <owl:maxCardinality rdf:datatype=
"&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 8: Column maps to data type property.

Most of the mapping of columns has to do with the mapping of data types from SQL to XSD. Unlike SQL, OWL does not have any built-in data types. Instead, it uses XSD (XML Schema Data types).

A column ssn in Figure 8 uses INTEGER as its data type. Therefore, a data type property ssn could use integer as its range. However, there is a constraint CHECK on the column ssn. This constraint further restricts the range of values for the column ssn to all integers greater than 0 (i.e. all positive integers). Therefore, the data type property ssn uses positiveInteger as its range.

3.4 Mapping Constraints

SQL supports constraints UNIQUE, NOT NULL, REFERENCES, FOREIGN KEY, PRIMARY KEY, CHECK, and DEFAULT. However, not all the constraints can be mapped to OWL. E.g. a constraint DEFAULT (that defines a default value for a given column) has no correspondence in OWL. Therefore, it is ignored.

3.4.1 Mapping Constraints UNIQUE

UNIQUE is a column constraint. It maps to an inverse functional property.

A constraint UNIQUE in Figure 9 specifies that a column ssn in a table Employee is unique,

meaning that no two rows in the table Employee have the same value for the column ssn (i.e. social security numbers uniquely identify employees). Therefore, this constraint maps to an inverse functional property.

```
CREATE TABLE Employee(
 ssn INTEGER UNIQUE)
                    ↓
<owl:InverseFunctionalProperty
rdf:ID="ssn"/>
```

Figure 9: Constraint UNIQUE maps to inverse functional property.

3.4.2 Mapping Constraints NOT NULL

NOT NULL is a column constraint. It maps to a minimum cardinality of 1.

A constraint NOT NULL in Figure 10 specifies that a column ssn in a table Employee is not null, meaning that all rows in the table Employee have values for the column ssn (i.e. all employees are assigned social security numbers). Therefore, this constraint maps to a minimum cardinality of 1.

```
CREATE TABLE Employee(
 ssn INTEGER NOT NULL)
                    ↓
<owl:Class rdf:ID="Employee">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="#ssn"/>
   <owl:minCardinality rdf:datatype=
"&xsd;nonNegativeInteger"1/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 10: Constraint NOT NULL maps to minimum cardinality of 1.

3.4.3 Mapping Constraints REFERENCES and FOREIGN KEY

REFERENCES is a column constraint (to refer to a single column), whereas FOREIGN KEY is a table constraint (to refer to multiple columns). Both constraints are used for specifying foreign keys. A foreign key can be mapped to four different constructs in the ontology: an object property, class inheritance, a symmetric property, and a transitive property.

A constraint REFERENCES in Figure 11 specifies that a column ProjectID in a table Task is a foreign key to another table Project, indicating a binary (one-to-zero-or-one, one-to-one or many-to-one) relationship. Since the foreign key is not the primary key, it maps to an object property ProjectID that uses classes Task and Project as its domain and range, respectively. This property has a maximum cardinality of 1 (atomicity). In addition, the property ProjectID is restricted to all values from the class Project, because the foreign key implies that for each (non-null) value of the column ProjectID there is the same value in the table Project.

```
CREATE TABLE TASK(
 TaskID INTEGER PRIMARY KEY,
 ProjectID INTEGER REFERENCES Project)
                    ↓
<owl:ObjectProperty rdf:ID="ProjectID">
 <rdfs:domain rdf:resource="#Task"/>
 <rdfs:range rdf:resource="#Project"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="Task">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="#ProjectID"/>
    <owl:maxCardinality rdf:datatype=
"&xsd;nonNegativeInteger"1/>
    <owl:allValuesFrom rdf:resource=
"#Project"/>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 11: Foreign key maps to object property.

A constraint FOREIGN KEY in Figure 12 specifies that a column ProjectID in a table SoftwareProject is a foreign key to another table Project, indicating a binary relationship, again. However, since the foreign key is now the primary key, it maps to class inheritance: SoftwareProject is a subclass of Project (i.e. a software project is a project).

```
CREATE TABLE SoftwareProject(
 ProjectID INTEGER PRIMARY KEY,
 FOREIGN KEY (ProjectID) REFERENCES
Project)
                    ↓
<owl:Class rdf:ID="SoftwareProject">
 <rdfs:subClassOf
rdf:resource="#Project"/>
</owl:Class>
```

Figure 12: Foreign key maps to class inheritance.

A constraint REFERENCES in Figure 13 specifies that a column spouse in a table

Employee is a foreign key to the same table, indicating a unary relationship. Therefore, the foreign key maps to a symmetric property spouse that uses a class Employee as both its domain and range (i.e. if one employee is a spouse of another employee, then the second employee is a spouse of the first employee).

```
CREATE TABLE Employee(
 EmployeeID INTEGER PRIMARY KEY,
 spouse INTEGER REFERENCES Employee)
                    ↓
<owl:SymmetricProperty rdf:ID="spouse">
 <rdfs:domain
rdf:resource="#Employee"/>
 <rdfs:range rdf:resource="#Employee"/>
</owl:SymmetricProperty >
```

Figure 13: Foreign key maps to symmetric property.

A constraint REFERENCES in Figure 14 specifies that a column subtask in a table Task is a foreign key to the same table, indicating a unary relationship, again. However, since the foreign key is now accompanied by a trigger ON DELETE CASCADE, this relationship consists of a whole and a part, where the part cannot exist without the whole (i.e. if a task is deleted, then all its subtasks must also be deleted). Therefore, the foreign key maps to a transitive property subtask that uses a class Task as both its domain and range (i.e. if one task is a subtask of another task and the other task is a subtask of yet another task, then the first task is a subtask of the third task).

```
CREATE TABLE Task(
 TaskID INTEGER PRIMARY KEY,
 subtask INTEGER REFERENCES Task ON
DELETE CASCADE)
                    ↓
<owl:TransitiveProperty
rdf:ID="subtask">
 <rdfs:domain rdf:resource="#Task"/>
 <rdfs:range rdf:resource="#Task"/>
</owl:TransitiveProperty >
```

Figure 14: Foreign key maps to transitive property.

3.4.4 Mapping Constraints PRIMARY KEY

There are two forms of constraint PRIMARY KEY: using it as a column constraint (to refer to a single column) and using it as a table constraint (to refer to multiple columns). Both constraints are used for specifying primary keys.

To this end, each column in a primary key maps to either a data type property or an object property with a maximum cardinality of 1 (see Sections 3.3 and 3.4.3). This property will be defined as an inverse functional property with a minimum cardinality of 1 if the primary has a single column. Otherwise, it will just have a minimum cardinality of 1.

A constraint PRIMARY KEY in Figure 15 specifies that a column ssn in a table Employee is a primary key, which is the same as saying that the column ssn is both unique and not null. Therefore, this constraint maps to both an inverse functional property and a minimum cardinality of 1.

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

Figure 15: Constraint PRIMARY KEY maps to both inverse functional property and minimum cardinality of 1.

3.4.5 Mapping Constraints CHECK

There are two forms of constraint CHECK: using it as a column constraint (to refer to a single column) and using it as a table constraint (to refer to multiple columns). A constraint CHECK maps to a value restriction unless it has enumeration. Then it maps to an enumerated data type. It should be noted that OWL is not powerful enough to express all the value restrictions that can be imposed by a constraint CHECK (e.g. an employee's age as an integer between 18 and 65).

A constraint CHECK in Figure 16 specifies that a column type in a table Project may have only a value Software. Therefore, a data type property type is restricted to have the same value for all instances in a class Project.

```
CREATE TABLE Project(
 type VARCHAR CHECK (type='Software'))
                    ↓
<owl:Class rdf:ID="Project">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty
rdf:resource="#type"/>
    <owl:hasValue
rdf:datatype="&xsd;string">Software
    </owl:hasValue>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 16: Constraint CHECK maps to value restriction.

A constraint `CHECK` in Figure 17 specifies the range for a column `type` in a table `Project` through a list of values `Software` and `Hardware`. Therefore, this constraint maps to an enumerated data type `Project`, with one element for each value in the list.

```
CREATE TABLE Project (
 type VARCHAR CHECK (type IN
('Software', 'Hardware'))
                    ↓
<owl:DatatypeProperty rdf:ID="type">
 <rdfs:domain rdf:resource="#Project"/>
 <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>
```

Figure 17: Constraint CHECK maps to enumerated data type.

3.5 Mapping Rows

A row maps to an instance.

A row in a table `Project` in Figure 18 has a value `Software` for a column `type`. Therefore, this row maps to an (anonymous) instance of a class `Project` that has the same value for a data type property `type`.

```
INSERT INTO Project (type) VALUE
('Software')
                    ↓
<Project>
 <type
rdf:datatype="&xsd:string">Software
 </type>
</Project>
```

Figure 18: Row in table maps to instance of class.

4 CONCLUSIONS

This paper has proposed a novel approach to automatic transformation of relational databases to ontologies, where domains and constraints `CHECK` are also considered. The proposed approach can map all constructs of a relational database to an ontology, with the exception of those constructs that have no correspondences in the ontology (e.g. constraints `DEFAULT`).

REFERENCES

Astrova, I., Kalja, A., 2006. Towards the Semantic Web: Extracting OWL ontologies from SQL relational schemata. In *ICWI'06, IADIS International Conference WWW/Internet*.

Buccella, A., Penabad, M., Rodriguez, F., Farina, A., Cechich, A., 2004. From relational databases to OWL ontologies. In *RCDL'04, 6th National Russian Research Conference*.

Li, M., Du, X., Wang, S., 2005. Learning ontology from relational database. In *ICMLC'05, 4th International Conference on Machine Learning and Cybernetics*.

McFadden, F., Hoffer, J., Prescott, M., 1999. *Modern Database Management*. 5th edition, Addison-Wesley.

OWL, 2004. *OWL Web Ontology Language Reference*. http://www.w3.org/TR/owl-ref

Sequeda, J., Tirmizi, S., Miranker, D., 2007. SQL Databases are a Moving Target. In *W3C Workshop on RDF Access to Relational Databases*.

Shen, G., Huang, Z., Zhu, X., Zhao, X., 2006. Research on the rules of mapping from relational model to OWL. In *OWLED'06, OWL: Experiences and Directions*.

SQL, 2002. *Database language SQL*. ANSI X3.135. www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

Xu, Z., Zhang, S., Dong, Y., 2006. Mapping between relational database schema and OWL ontology for deep annotation. In *WI'06, IEEE/WIC/ACM International Conference on Web Intelligence*.