# Translating SQL Applications to the Semantic Web

Syed Hamid Tirmizi, Juan Sequeda, and Daniel Miranker

Department of Computer Sciences, The University of Texas at Austin, USA
{hamid,jsequeda,miranker}@cs.utexas.edu

**Abstract.** The content of most Web pages is dynamically derived from an underlying relational database. Thus, the success of the Semantic Web hinges on enabling access to relational databases and their content by semantic methods. We define a system for automatic transformation of SQL DDL schemas into OWL DL ontologies. This system goes further than earlier efforts in that the entire system is expressed in first-order logic. We leverage the formal approach to show the system is complete with respect to a space of the possible relations that can be formed among relational tables as a consequence of primary and foreign key combinations. The full set of transformation rules is stratified, thus the system can be executed directly by a Datalog interpreter.

## 1 Introduction

It has been estimated that Internet accessible databases contain up to 500 times more data compared to the static Web and that three-quarters of these databases are managed by relational database management systems [HeP07]. Thus, enabling the integration of relational databases and their content with the Semantic Web is critical to the Semantic Web's success.

The Semantic Web provides an ontology-based framework for integration, search and sharing of data drawn from diverse sources. Broadly stated, there are two architectural approaches to integrating databases with the Semantic Web. The more commonly researched approach is the development of wrapper systems that map a relational database schema to an existing domain ontology [AnB05, Bar04, Che06, Lab05, Lab06, Rod06]. To date there has been little work automating the creation of such wrappers. Thus, wrapper systems appear to be a labor-intensive solution.

The second approach, which is the subject of the work in this paper, concerns the automatic transformation of database content and/or schema to a Semantic Web representation, i.e. RDF and OWL [Biz03, LiD05, Ast07]. In this approach it is assumed that the data model entails a logical model of the application domain, and by syntactically analyzing the model's physical encoding in SQL Data Description Language (DDL) the logical model may be recovered. While many legacy databases were defined using strict relational syntax and semantics, and thus may encode modest application domain semantics, the current SQL standard coupled with modern software design methodology enables rich expression of domain semantics; albeit not in a form readily accessible to automated inference mechanism [Seq07]. In addition to foreign key constraints, SQL DDL supports a variety of constraints on the range of values allowed in a table. Building on related work we define a system for automatic transformation of relational

S.S. Bhowmick, J. Küng, and R. Wagner (Eds.): DEXA 2008, LNCS 5181, pp. 450–464, 2008.

databases into OWL-DL ontologies. Two critical elements distinguish our transformation system from past efforts. *First*, the entire system is defined in first order logic (FOL) eliminating syntactic and semantic ambiguities in our rules. Much of the related work is expository in nature, at times influenced by domain specific examples and/or specifying the resulting rules in English prose. Often the influence of examples from a particular domain can result in incorrect rules. *Second*, we present a notion of completeness of our system in terms of a space of all possible relations describable by SQL DDL considering the interactions of primary and foreign keys in relations. We have partitioned the space of relations and have covered the transformation of each partition with sets of rules applicable to that partition.

Further, we observe that the FOL expression of our transformation system is stratified. Thus, in addition to implementation in Prolog environments, the system may integrate with databases supporting Datalog interpreters.

## 2 Related Work

A number of researchers have made inroads on this problem and serve as a foundation for our work [Sto02, LiD05, Ast07].

Stojanovic et al. [Sto02] provide rules for translation of relational schemas to Frame Logic and RDF Schema. This work formally defines rules for identification of classes and properties in relational schemas. It does not have the capability of capturing richer semantics that cannot be expressed in RDF Schema.

Li et al. [LiD05] propose a set of rules for automatically learning an OWL ontology from a relational schema. They define the rules using a combination of some formal notation and English language. Our analysis shows that some of their rules miss some semantics offered by the relational schema and some rules produce specific results for inheritance and object properties that may not accurately depict concepts across domains or database modeling choices. We believe these shortcomings are due to lack of a formal system and thorough examination of examples capturing a variety of modeling choices in various domains.

Astrova et al. [Ast07] provide expository rules and examples to describe a system for automatic transformation of a relational schema to OWL Full. When it was published this work was the most comprehensive. Since the rules were not formally defined, a number of transformations are ambiguous.

In addition to the lack of correctness due to informal specification of rules, these systems do not provide any notion of completeness of their rules. By completeness we mean consideration of all possible database key structures that may encode an ontological relationship. We present the results of a construction that enumerates such key structures and document that our transformation system is complete and unambiguous. In the rest of the paper, first we present the disparities between relational databases and ontologies. Then we systematically present how relational database schemas can be transformed into OWL ontologies. First, with the help of an example, we show how a domain expert can translate a relational schema in SQL DDL into an OWL ontology. Then, we present our assumptions and transformation rules, and explain them using the same example. We also provide a comparison of human and automatically generated ontologies and relate the differences using our discussion on disparities as a basis.

## 3 Extracting Knowledge from a Relational Schema

Consider a relational database for a university and its definition (see Table 1).

The *Person* table contains data about all the people, some of them may be students and present in *Student* table, and some may be professors and present in *Professor* table. The *Dept* table lists the departments in the university where each department has a unique name, and the *Course* table lists the courses for every department. The *Semester* table contains a list of semesters which have a year and one of the three seasons, Spring, Summer or Fall, associated with them. A course could be offered in a particular semester with a particular professor, and recorded in *Offer* table. Two offered courses could be co-offered, and recorded as a self-relation in the *Offer* table. A student could study an offered course, which is recorded in *Study* table. Also, a student could be registered in a semester with or without taking a course, and this information is recorded in the *Reg* table.

**Table 1.** Schema of a University Database

**University Database Schema**

```
create table PERSON { ID integer primary key, NAME varchar not null }
create table STUDENT { ROLLNO integer primary key, DEGREE varchar,
  ID integer unique not null foreign key references PERSON(ID) }
create table PROFESSOR { ID integer primary key, TITLE varchar,
  constraint PERSON_FK foreign key (ID) references PERSON(ID) }
create table DEPT { CODE varchar primary key,
  NAME varchar unique not null }
create table SEMESTER { SNO integer primary key, YEAR date not null,
  SESSION varchar check in ('SPRING', 'SUMMER', 'FALL') }
create table COURSE { CNO integer primary key, TITLE varchar,
  CODE varchar not null foreign key references DEPT(CODE) }
create table OFFER { ONO integer primary key,
  CNO integer foreign key references COURSE(CNO),
  SNO integer foreign key references SEMESTER(SNO),
  PID integer foreign key references PROFESSOR(ID),
  CONO integer foreign key references OFFER(ONO) }
create table STUDY { ONO integer foreign key references OFFER(ONO),
  RNO integer foreign key references STUDENT(ROLLNO),
  GRADE varchar, constraint STUDY_PK primary key (ONO, RNO) }
create table REG { SID integer foreign key references STUDENT(ID),
  SNO integer foreign key references SEMESTER(SNO),
  constraint REG_PK primary key (SID, SNO) }
```

For a domain expert, it is easy to recognize the concepts in this database structure, and to identify the semantics of their properties and different kinds of relationships that exist between these concepts. Table 2 shows an ontology corresponding to the given schema, developed by a domain expert.

**Table 2.** Parts of an ontology corresponding to the schema in Table 1, developed by a domain expert. The ontology is presented in OWL Abstract Syntax. The highlighted sections in the table are later compared with an automated output.

**Domain Expert's Ontology**

```
Ontology(<urn:sql2owl>
 ObjectProperty(<REG> domain(<STUDENT>) range(<SEMESTER>))
 ObjectProperty(<REG_I> inverseOf(<REG>))
 ObjectProperty(<OFFER.CONO> Transitive Symmetric
  domain(<OFFER>) range(<OFFER>))...
 DatatypeProperty(<COURSE.CNO> Functional
  domain(<COURSE>) range(xsd:integer))
 DatatypeProperty(<SEMESTER.YEAR> Functional
  domain(<SEMESTER>) range(xsd:date))
 DatatypeProperty(<SEMESTER.SESSION> Functional domain(<SEMESTER>)
  range(oneOf("SPRING" "SUMMER" "FALL")) range(xsd:string))...
 Class(<PERSON> partial ...)
 Class(<PROFESSOR> partial <PERSON> ...)
 Class(<STUDENT> partial <PERSON>
  restriction(<STUDY.RNO_I> minCardinality(0)) ...)
 Class(<COURSE> partial restriction(<COURSE.DEPTCODE> cardinality(1))
  restriction(<COURSE.CNO> cardinality(1)) ...)   ...)
```

## 4 Disparities between Relational Databases and Ontologies

While relational databases are capable of efficiently managing large amounts of structured data, ontologies are very useful for knowledge representation. Since these two data models are aimed towards different requirements specified by their domains, it is reasonable to expect some disparities among them in terms of capabilities.

To define a relational database to ontology transformation system, it is important to understand the mismatches between the two data models, and to make educated choices when confronted with such problems. Here we discuss some key issues – inheritance modeling, property characteristics and open/closed world assumptions – that affect a transformation system.

### 4.1 Inheritance Modeling

Relational databases do not express inheritance. However, inheritance hierarchies can be modeled in a variety of ways in relational schemas. Also, some modeling choices are harder to identify automatically. Given a foreign key definition between two entities, we would like to know whether a subclass relationship exists between the entities involved. If such patterns exist, we can map them to subclass relationships in the ontology.

The following list presents possible foreign key patterns to model inheritance:

- Foreign key is also the primary key: The *Professor-Person* relationship in our university schema is an example. This pattern uniquely identifies inheritance. An exception to this would be vertical partitioning of tables, but since we assume 3NF databases for our system, this case can be transformed to inheritance.
- Foreign key and primary key are disjoint: The *Student-Person* relationship in our university schema is an example. However, the *Course-Dept* relationship modeled

in the same schema is a counterexample. In fact, this pattern is the most common one used for expressing one-to-many relationships.

- Foreign key is a subset of the primary key: This is also not a good candidate for automatic translation to an inheritance hierarchy in the ontology. Many counterexamples for this pattern can be presented, particularly the ones for modeling 'part-of' relationships between entities.

### 4.2 Characteristics of Relationships

While relational schemas can capture some cardinality constraints on relationships between entities by defining constraints on foreign keys, they lack the expressive power to define relationships with interesting logical characteristics, like symmetry and transitivity etc. On the other hand, expressing such characteristics of relationships is natural to ontology languages like OWL, which are based on some form of logic.

For example, the self-relation on the *Offer* entity, that represents co-location of an offered course with another offered course, is symmetric and transitive. While these characteristics are obvious to a domain expert, the relationship is expressed like any other self-relationship in the relational schema, which may not have the same characteristics. Consider the example: `Employee(ID,Name,MgrID)`, where `ID` is the primary key, and `MgrID` is a foreign key to the Employee table itself referencing manager's ID. This relationship is clearly not symmetric, and may or may not be transitive.

The example clearly shows that it is hard to identify logical characteristics of relationships in a relational schema without using the domain knowledge. Therefore, our rules do not capture these characteristics automatically.

### 4.3 The Effect of Open/Closed World Assumptions

Relational databases usually operate under the closed world (CW) assumption. This means that whatever is not in the database is considered false. On the other hand, knowledge bases operate in open world (OW) where whatever is not in the knowledge base is considered unknown. This assumption is natural for knowledge bases that often contain incomplete knowledge, and grow incrementally.

Therefore, the concept of a constraint has very different meanings in the two worlds [Mot07]. In a database setting, a constraint is mainly used for validation. In contrast, in an ontology, a constraint expresses some characteristics of classes or relationships but does not prevent assertion of any facts. In addition, some assertions may even result in unintuitive inferences.

When developing an ontology based on a relational schema, it is very important to keep these differences in mind. The question whether the open world should be closed or not depends upon the domain and application requirements. In our system, we produce an ontology with open world assumption. If needed, one way to close the world will be to assert that all inferred classes are pair-wise disjoint.

# 5 Translating SQL to Semantic Web

In this section, we explain the transformation of a relational schema to an ontology. First we present our assumptions and explain the rationale behind them. Then, we list the predicates and functions we have defined to express transformation rules in first order logic. In the next section, we explain the transformations for data types, classes, properties and inheritance, and provide mapping tables or first order logic rules to formally define the transformations.

## 5.1 Assumptions

In order to translate a relational schema into an ontology, we make the following assumptions:

- *The relational schema, in its most accurate form, is available in SQL DDL.* Databases evolve due to changing application requirements. Such modifications are often reflected solely in the physical model, usually expressed in SQL DDL, making it the most accurate source for the structure of the database.
- *The relational schema is normalized, at least up to third normal form.* While all databases might not be well normalized, it is possible to automate the process of finding functional dependencies within data and to algorithmically transform a relational schema to third normal form [DuW99, Wan00].

## 5.2 Predicates and Functions

We have defined a number of predicates and functions to aid the process of defining transformation rules in first order logic.

There are two sets of predicates in our system. *RDB predicates* test whether an argument (or a set of arguments) matches a construct in the domain of relational databases. Such predicates are listed below:

| | |
|---|---|
| *Rel(r)* | *r* is a relation; e.g. *Rel(PERSON)* holds, *Rel(ID)* does not |
| *Attr(x,r)* | *x* is an attribute in relation *r*; e.g. *Attr(ID,PERSON)* holds |
| *NN(x,r)* | *x* is an attribute (or a set of attributes) in relation *r* with NOT NULL constraint(s); e.g. *NN(NAME,PERSON)* holds |
| *Unq(x,r)* | *x* is an attribute (or a set of attributes) in relation *r* with UNIQUE constraint; e.g. *Unq({NAME},DEPT)* holds |
| *Chk(x,r)* | *x* is an attribute in relation *r* with enumerated list (CHECK IN) constraint; e.g. *Chk(SESSION,SEMESTER)* holds |
| *PK(x,r)* | *x* is the (single or composite) primary key of relation *r*; e.g. *PK({ONO,RNO},STUDY)* holds; Also: $PK(x,r) \rightarrow Unq(x,r) \wedge NN(x,r)$ |
| *FK(x,r,y,s)* | *x* is a (single or composite) foreign key in relation *r* and references *y* in relation *s*; e.g. *FK({ID},STUDENT,{ID},PERSON)* holds |
| *NonFK(x,r)* | *x* is an attribute in relation *r* that does not participate in any foreign key; e.g. *NonFK(NAME,DEPT)* holds |

On the other hand, *ontology predicates* test whether an argument (or a set of arguments) matches a construct that can be represented in an OWL ontology. These predicates are:

| | |
|---|---|
| *Class(m)* | *m* is a class |
| *ObjP(p,d,r)* | *p* is an object property with domain *d* and range *r* |
| *DTP(p,d,r)* | *p* is an data type property with domain *d* and range *r* |
| *Inv(p,q)* | when *p* and *q* are object properties, *p* is an inverse of *q* |
| *FP(p)* | *p* is a functional property |
| *IFP(p)* | *p* is an inverse functional property |
| *Crd(p,m,v)* | the (max and min) cardinality of property *p* for class *m* is *v* |
| *MinC(p,m,v)* | the min cardinality of property *p* for class *m* is *v* |
| *MaxC(p,m,v)* | the max cardinality of property *p* for class *m* is *v* |
| *Subclass(m,n)* | *m* is a subclass of class *n* |

The constructs represented by ontology predicates are described as they appear in the rules mentioned in the upcoming sections of this paper.

We have also defined the following functions:

| | |
|---|---|
| *fkey(x,r,s)* | takes a set of attributes *x*, relations *r* and *s*, and returns the foreign key defined on attributes *x* in *r* referencing *s* |
| *type(x)* | maps an attribute *x* to its suitable OWL recommended data type (we discuss data types in more detail in a later section) |
| *list(x)* | maps an attribute *x* to a list of allowed values; applicable only to attributes with a CHECK IN constraint, i.e. *Chk(x)* is true |

In addition to the predicates and functions listed above, we describe the concept of a *binary relation*, written *BinRel*, as a relation that only contains two (single or composite) foreign keys that reference other relations. Such tables are used to resolve many-to-many relationships between entities. Using RDB predicates, we formally define *BinRel* as follows:

**Rule Set 1:**

$$BinRel(r,s,t) \leftarrow \begin{array}{l} Rel(r) \wedge FK(q,r,\_,t) \wedge FK(p,r,\_,s) \wedge p \neq q \wedge Attr(y,r) \wedge \neg NonFK(y,r) \wedge \\ FK(z,r,\_,u) \wedge fkey(z,r,u) \in \{fkey(p,r,s), fkey(q,r,t)\} \end{array}$$

This rule states that a binary relation *r* between two relations *s* and *t* exists if *r* is a relation that has foreign keys to *s* and *t*, and *r* has no other foreign keys or attributes (each attribute in the relation belongs to one of the two foreign keys). Note that there is no condition that requires *s* and *t* to be different, allowing binary relations that have their domain equal to their range.

### 5.3 Transformation Rules and Examples

In this section we present rules and examples for transformation of a relational database to OWL ontology.

*Producing Unique Identifiers (URIs) and Labels*

Before we discuss the transformation rules, it is important to understand how we can produce identifiers and names for classes and properties that form the ontology.

The concept of globally unique identifiers is fundamental to OWL ontologies. Each class or property in the ontology must have a unique identifier, or URI. While it is possible to use the names from the relational schema to label the concepts in the ontology, it is necessary to resolve any duplications, either by producing URIs based on fully qualified names of schema elements, or by producing them randomly. In addition, for human readability, RDFS labels should be produced for each ontology element containing names of corresponding relational schema elements. Due to lack of space, we have not used fully qualified names in our examples. When needed, we append a name with an integer to make it unique, e.g. ID1, ID2 etc.

*Transformation of Data Types*

Transformations from relational schemas to ontologies require preserving data type information along with the other semantic information. OWL (and RDF) specifications recommend the use of a subset of XML Schema types [XMLSch] in Semantic Web ontologies [OWLRef, RDFSem].

In Table 3 we present a list of commonly used SQL data types along with their corresponding XML Schema types. During transformation of data type properties, the SQL data types are transformed into the corresponding XML Schema types.

**Table 3.** Common SQL types and corresponding XML Schema types recommended for OWL

| SQL Data Type | XML Schema Type | SQL Data Type | XML Schema Type |
|---|---|---|---|
| INTEGER | xsd:integer | VARCHAR | xsd:string |
| FLOAT | xsd:float | DATE | xsd:date |
| BOOLEAN | xsd:boolean | TIMESTAMP | xsd:dateTime |

*Identifying Classes*

According to OWL Language Guide [OWLGde], "the most basic concepts in a domain should correspond to classes …". Therefore we would expect basic entities in the data model to translate into OWL classes.

Given the definition of a binary relation, it is quite straightforward to identify OWL classes from a relational schema. Any relation that is not a binary relation can be mapped to a class in an OWL ontology, as stated in the rule below.

**Rule Set 2:**

$$Class(r) \leftarrow Rel(r) \land \neg BinRel(r,\_,\_)$$

Remember that a binary relation has exactly two foreign keys and no other attributes (see Rule Set 1). Keeping that in mind, we can see that this very simple rule covers a number of cases for identifying classes:

- All tables that do not have foreign keys should be transformed to classes. Therefore, we conclude *Class(PERSON)*, i.e. *Person* should be mapped to a class since it has no foreign key. The same reasoning holds for the *Dept* and *Semester* tables.
- All tables with one foreign key can be mapped to classes since they cannot be binary relations. Hence *Student*, *Professor* and *Course* should be mapped to classes.
- Tables with more than two foreign keys should be transformed to classes as well. Such tables may represent an entity or an N-ary relationship between entities. Fortunately, in OWL, both the cases can be modeled the same way, i.e. by translating the entity or the N-ary relationship into a class [Noy06]. From our example, *Offer* represents an N-ary relationship, and modeled as a class using the given rule.

- For tables containing exactly two foreign keys, presence of independent attributes qualifies them to be translated to classes. The table *Study*, with an independent attribute *Grade*, is an example, and is translated to an OWL class.

Thus Rule Set 2 identifies the OWL classes from the database schema. For example:
*Class(PERSON), Class(STUDENT), Class(DEPT), Class(STUDY), Class(OFFER)*

*Identifying Object Properties*

An object property is a relation between instances of two classes in a particular direction. In practice, it is often useful to define object properties in both directions, creating a pair of object properties that are inverses of each other. OWL provides us the means to mark properties as inverses of each other. In our work, when we translate something to an object property, say *ObjP(r,s,t)*, it implicitly means we have created an inverse of that property, which we write as *ObjP(r',t,s)*.

There are two ways of extracting OWL object properties from a relational schema. One of the ways is through identification of binary relations, which represent many-to-many relationships. The following rule identifies an object property using a binary relation.

**Rule Set 3:**

$$ObjP(r,s,t) \leftarrow BinRel(r,s,t) \wedge Rel(s) \wedge Rel(t) \wedge \neg BinRel(s,\_,\_) \wedge \neg BinRel(t,\_,\_)$$

This rule states that a binary relation *r* between two relations *s* and *t*, neither being a binary relation, can be translated into an OWL object property with domain *s* and range *t*. Notice that the rule implies *Class(s)* and *Class(t)* hold true, so the domain and range of the object property can be expressed in terms of corresponding OWL classes.

From our university database schema, the *Reg* table fits the condition. *Reg* is a binary relation between *Student* and *Semester* entities, which are not binary relations. Therefore, *ObjP(REG,STUDENT,SEMESTER)* holds, and since we can create inverses, *ObjP(REG',SEMESTER,STUDENT)* and *Inv(REG,REG')* also hold true.

Foreign key references between tables that are not binary relations represent one-to-one and one-to-many relationships between entities. A pair of object properties that are inverses of each other and have a maximum cardinality of 1 can represent one-to-one relationships. Also, one-to-many relationships can be mapped to an object property with maximum cardinality of 1, and an inverse of that object property with no maximum cardinality restrictions.

In OWL, a property with min cardinality of 0 and max cardinality of 1 is called *functional* which we represent by the functor *FP*. If an object property is functional, then its inverse is *inverse functional*, represented by the functor *IFP*. In addition to specifying cardinality restrictions on properties in general, we can also specify such restrictions when a property is applied over a particular domain. In our rules, we use ontology predicates *Crd*, *MinC* and *MaxC* to specify these restrictions. The examples following the rules explain the use of these predicates.

The following rule set identifies object properties and their characteristics using foreign key references (not involving binary relations, covered in Rule Set 3) with various combinations of uniqueness and null restrictions. To simplify the rules, we first define a predicate *NonBinFK* that represents foreign keys not in or referencing binary relations and then express the rules in terms of this predicate.

**Rule Set 4:**

$NonBinFK(x,s,y,t) \equiv FK(x,s,y,t) \wedge Rel(s) \wedge Rel(t) \wedge \neg BinRel(s,\_,\_) \wedge \neg BinRel(t,\_,\_)$

| | | | |
|---|---|---|---|
| *a.* | $ObjP(x,s,t), FP(x), MinC(x',t,0)$ | $\leftarrow$ | $NonBinFK(x,s,y,t) \wedge \neg NN(x) \wedge \neg Unq(x)$ |
| *b.* | $ObP(x,s,t), FP(x), Crd(x,s,1), MinC(x',t,0)$ | $\leftarrow$ | $NonBinFK(x,s,y,t) \wedge NN(x) \wedge \neg Unq(x)$ |
| *c.* | $ObjP(x,s,t), FP(x), FP(x')$ | $\leftarrow$ | $NonBinFK(x,s,y,t) \wedge \neg NN(x) \wedge Unq(x)$ |
| *d.* | $ObjP(x,s,t), FP(x), Crd(x,s,1), FP(x')$ | $\leftarrow$ | $NonBinFK(x,s,t) \wedge NN(x) \wedge Unq(x) \wedge \neg PK(x,s)$ |

Each rule in Rule Set 4 states that a foreign key represents an object property from the entity containing the foreign key (domain) to the referenced entity (range). Since a foreign key references at most one record (instance) of the range, the object property is functional. This entails that the inverse of that object property is inverse functional. An example is the foreign key from *Study* to *Student* which gives us: *ObjP(RNO,STUDY,STUDENT)*, *FP(RNO)*, *Inv(RNO',RNO)*, *IFP(RNO')*.

Rules 4a and 4b represent variations of one-to-many relationships.

- We can apply a stronger restriction on cardinality of the object property if the foreign key is constrained as NOT NULL. Without this constraint (rule 4a), the minimum cardinality is 0, which is covered by functional property predicate. With this constraint (rule 4b), we can set the maximum and minimum cardinality to 1.
- According to these rules, we can infer only the minimum cardinality restriction of 0 on the inverse property. Since an instance in the range could be referenced by any number of instances in the domain, we cannot apply a maximum cardinality restriction on the inverse property.

The other two rules, 4c and 4d, represent one-to-one relationships, modeled by applying a uniqueness constraint on the foreign key. It means that an instance in the range can relate to at most one object in the domain, making the inverse property functional too. This also means that the original object property is inverse functional as well.

The difference between rules 4c and 4d is that of a NOT NULL constraint that, like one-to-many relationships mentioned above, if present, gives us a stronger cardinality restriction on the object property represented by the foreign key.

Notice that none of the rules allow the foreign key to be the same as the primary key of the domain relation. Rule 4d restricts this by providing an extra condition, whereas the negation of uniqueness or NOT NULL constraints in rules 4a-c, by definition, implies this condition.

Examples of object properties and their characteristics obtained from the relational schema by applying Rule Sets 3 and 4 are:

*ObjP(REG,STUDENT,SEMESTER), ObjP(REG',SEMESTER,STUDENT), Inv(REG,REG')*
*ObjP(RNO,STUDY,STUDENT), FP(RNO), IFP(RNO'), MinC(RNO',STUDENT,0)*
*ObjP(ID1,STUDENT,PERSON), FP(ID1), FP(ID1'), Crd(ID1,STUDENT,1)*

*Identifying Data Type Properties*

Data type properties are relations between instances of classes with RDF literals and XML Schema data types. Like object properties, data type properties can also be functional, and can be specified with cardinality restrictions. However, unlike object properties, OWL DL does not allow them or their inverses to be inverse functional.

Attributes of relations in a database schema can be mapped to data type properties in the corresponding OWL ontology. Rule Set 5 identifies data type properties.

**Rule Set 5:**

*a.* $DTP(x,r,type(x)), FP(x) \leftarrow NonFK(x,r)$

*b.* $DTP(x,r,type(x)), FP(x), Crd(x,r,1) \leftarrow NonFK(x,r) \wedge NN(x,r)$

*c.* $DTP(x,r,type(x) \cap list(x)), FP(x) \leftarrow NonFK(x,r) \wedge Chk(x,r)$

Rule Set 5 says that attributes that do not contribute towards foreign keys can be mapped to data type properties with range equal to their mapped OWL type. Since each record can have at most one value per attribute, each data type property can be marked as a functional property. When an attribute has a NOT NULL constraint, rule 5b allows us to put an additional cardinality restriction on the property. Rule 5c allows us to infer stronger range restrictions on attributes with enumerated list (CHECK IN) constraints.

**Table 4.** Parts of an ontology corresponding to the University Database, produced automatically using our transformation rules. The output format is OWL Abstract Syntax. The underlined parts highlight the differences compared to the human-developed ontology shown in Table 2.

**Automatically Produced Ontology**

```
Ontology(<urn:sql2owl>
 ObjectProperty(<REG> domain(<STUDENT>) range(<SEMESTER>))
 ObjectProperty(<REG_I> inverseOf(<REG>))
 ObjectProperty(<OFFER.CONO> Functional
  domain(<OFFER>) range(<OFFER>))
 ObjectProperty(<OFFER.CONO_I> InverseFunctional
  inverseOf(<OFFER.CONO>))
 ObjectProperty(<STUDENT.ID> Functional InverseFunctional
  domain(<STUDENT>) range(<PERSON>))
 DatatypeProperty(<COURSE.CNO> Functional
  domain(<COURSE>) range(xsd:integer))
 DatatypeProperty(<SEMESTER.YEAR> Functional
  domain(<SEMESTER>) range(xsd:date))
 DatatypeProperty(<SEMESTER.SESSION> Functional domain(<SEMESTER>)
  range(oneOf("SPRING" "SUMMER" "FALL")) range(xsd:string)) ...
 Class(<PERSON> partial ...)
 Class(<PROFESSOR> partial <PERSON> ...)
 Class(<STUDENT> partial restriction(<STUDENT.ID> cardinality(1))
  restriction(<STUDY.RNO_I> minCardinality(0)) ...)
 Class(<COURSE> partial restriction(<COURSE.DEPTCODE> cardinality(1))
  restriction(<COURSE.CNO> cardinality(1)) ...)   ...)
```

In some cases, it may be possible to apply more than one rule to an attribute. In such cases, all possible rules should be applied to extract more semantics out of the relational schema. Some data type properties extracted from our sample university database schema are:

$DTP(ID1,PERSON,xsd{:}integer), FP(ID1), Crd(ID1,PERSON,1)$
$DTP(SESSION,SEMESTER,xsd{:}string \cap \{SPRING,SUMMER,FALL\}), FP(SESSION)$
$DTP(NAME1,PERSON,xsd{:}string), FP(NAME1), Crd(NAME1,PERSON,1)$

*Identifying Inheritance*
Inheritance allows us to form new classes using already defined classes. It relates a more specific class to a more general one using subclass relationships [OWLGde].

Inheritance relationships between entities in a relational schema can be modeled in a variety of ways. Since most of these models are not limited to expressing inheritance alone, it is hard to identify subclass relationships.

The following rule describes a special case that can be used only for inheritance modeling in a normalized database design.

**Rule Set 6:**
$Subclass(r,s) \leftarrow Rel(r) \wedge Rel(s) \wedge PK(x,r) \wedge FK(x,r,\_,s)$
This rule states that an entity represented by a relation *r* is a subclass of an entity represented by relation *s*, if the primary key of *r* is a foreign key to *s*. In our sample university schema, we can clearly identify that *Subclass(PROFESSOR,PERSON)* holds.

As a result of applying our rules on the given relational schema, we get the ontology shown in Table 4

A comparison of the ontologies produced by the domain expert (Table 2) with the one produced automatically using our rules (Table 4) shows a number of differences. For example, our rules are unable to capture the subclass relationship of *Student* with *Person*, or the symmetric and transitive characteristics of the co-location relationship among *Offer* instances. These examples clearly show that automatic translation of a relational schema to an ontology has some limitations, and that these limitations are inline with the disparities we have identified earlier.

### 5.4 Implementation

The FOL expression of our transformation system is stratified enabling direct integration of the transformation system with databases supporting Datalog interpreters.

**Theorem:** *The transformation system defined by the union of rules in rule sets 1 through 6 is stratified.*

The proof is left to the reader. *Hint:* The predicates *BinRel* and *NonBinFK* are the only predicates that appear in both the head and body of a rule.

## 6 Completeness of Transformation

A notion of completeness of a SQL DDL to ontology transformation is that the rules of the transformation system cover the entire range of possible relations that can be described in a SQL schema. The interaction of the foreign keys with primary keys provides clues about the kinds of relationships that exist between the entities, e.g. one-to-one, one-to-many etc.

**Theorem:** *The space of relations describable in SQL DDL using various combinations of primary key and foreign key references between the relations can be partitioned into 10 disjoint cases of key combinations. Our transformation system covers the entire space of relations.*

The formal proof is beyond the space limits of this paper. The proof involves a syntactic enumeration of the cases and a closure operation over the space of relations. Fig. 1 provides a useful summary of the theorem and its proof.

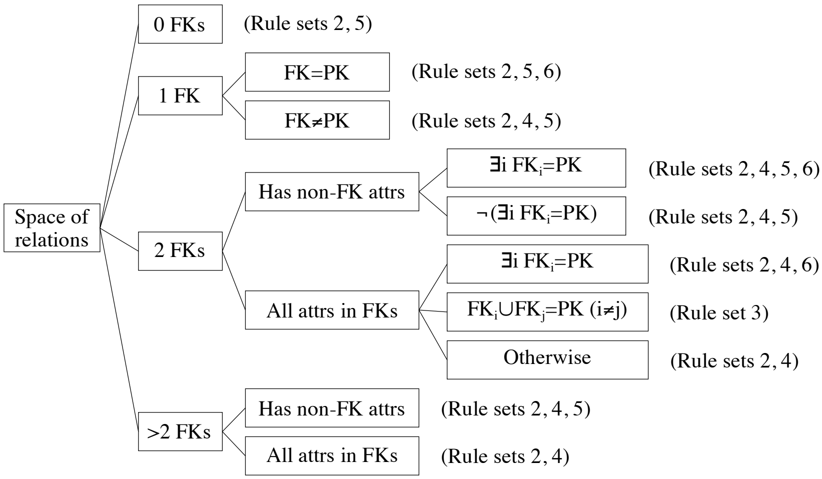


**Fig. 1.** The tree describes the complete space of relations when all possible combinations of primary and foreign keys are considered. For each branch, applicable rules are listed.

Briefly, we first partition the space by examining the number of foreign keys that a relation contains. All relations without any foreign keys can be easily translated into classes in an ontology. Similarly, relations with more than two foreign keys usually represent N-ary relationships, and the rules for N-ary relationships are applicable to them. The cases for one or two foreign keys are more interesting and give rise to more possibilities like binary relations, inheritance or new classes. However, for each possible branch, we have carefully defined sets of rules for producing ontology classes and properties.

## 7 Discussion

SQL DDL is a standard for representing the physical schema of applications that use relational databases. Although SQL DDL it is not a knowledge representation language, it is capable of capturing some semantics of the application domain. We have defined a system for automatic transformation of normalized SQL DDL schemas into OWL DL ontologies. We have defined our entire set of transformation rules in first order logic eliminating syntactic and semantic ambiguities and allowing for easy implementation of the system in languages like Datalog.

Once an ontology is defined for a domain represented by a relational schema, the actual database content can be easily translated into a corresponding RDF representation. We have also ensured compatibility with description logics based OWL DL, which is essential to assuring decidability for reasoning represented by the relational model.

We have demonstrated that an automatic transformation system has its deficiencies when it comes to identifying inheritance and other rich semantic elements. Although it is easy to generate specific examples of relational encodings of inheritance, there is neither a unique encoding, nor an encoding whose syntax, without further qualification, can be strictly interpreted as inheritance. Thus, transformation systems that create inheritance relationships will incorrectly produce too many, or too few. Thus, there may always be an opportunity for human judgment to fill in gap between the expressive power of SQL DDL and OWL.

Independent of the issues that arise from the differences in expressive power, a fair criticism of the automated transformation approach, in general, is that the scope of success may be highly dependent on the amount of domain semantics captured in SQL DDL, which in turn correlates to the age of the database application and the sophistication of its developers. However, if the success of an application of an automated transformation is limited, it is still possible to add missing semantics using the techniques being developed in wrapper-based approaches. Such semi-automated systems have been explored in the context of strict relational data integration [BaM07, Mil00]. Further, functioning relational database applications are prone to schema modification. One can envision a system where an automated transformation bootstraps a more powerful wrapper system. In the advent of database schema evolution a combined system may be able to reason about and propagate the changes.

## References

[AnB05] An, Y., Borgida, A., Mylopoulos, J.: Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. In: Proceedings of On The Move to Meaningful Internet Systems (2005)

[Ast07] Astrova, I., Korda, N., Kalja, A.: Rule-Based Transformation of SQL Relational Databases to OWL Ontologies. In: Proceedings of the 2nd International Conference on Metadata & Semantics Research (October 2007)

[Bar04] Barrasa, J., Corcho, O.: R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. In: Bussler, C.J., Tannen, V., Fundulaki, I. (eds.) SWDB 2004. LNCS, vol. 3372, Springer, Heidelberg (2005)

[BaM07] Barbançon, F., Miranker, D.P.: SPHINX: Schema integration by example. Journal of Intelligent Information Systems (in press, available on-line SpringerLink)

[Biz03] Bizer, C.: D2R MAP - A Database to RDF Mapping Language. In: Proceedings of the Twelfth International World Wide Web Conference (WWW) (2003)

[Che06] Chen, H., Wang, Y., Wang, H., Mao, Y., Tang, J., Zhou, C., et al.: Towards a Semantic Web of Relational Databases: a Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine. In: Proc. of the 5th International Semantic Web Conference (2006)

[DuW99] Du, H., Wery, L.: Micro: A normalization tool for relational database engineers. Journal of Network and Computer Applications 22(4), 215–232 (1999)

[HeP07] He, B., Patel, M., Zhang, Z., Chang, K.C.: Accessing the deep web. Communications of the ACM 50(5), 94–101 (2007)

[Hor03] Horrocks, I., Patel-Schneider, P.F.: Reducing OWL entailment to description logic satisfiability. In: Proceedings of the 2nd International Semantic Web Conference (2003)

[Lab05] de Laborda, C.P., Conrad, S.: Relational. OWL: a data and schema representation format based on OWL. In: Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modeling, vol. 43, pp. 89–96 (2005)

[Lab06] de Laborda, C.P., Conrad, S.: Database to Semantic Web Mapping using RDF Query Languages. In: 25th International Conference on Conceptual Modeling (November 2006)

[LiD05] Li, M., Du, X., Wang, S.: Learning ontology from relational database. In: Proceedings of the Fourth International Conference on Machine Learning and Cybernetics (2005)

[Mil00] Miller, R., Haas, L.L., Hernández, M.: Schema mapping as query discovery. In: Proceedings of the VLDB Conference (2000)

[Mot07] Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. In: Proceedings of the 16th International Conference on World Wide Web (2007)

[Noy06] Noy, N., Rector, A. (eds.): Defining N-ary Relations on the Semantic Web. W3C Working Group Note (11/14/2007), http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/

[OWLGde] Smith, M.K., Welty, C., McGuinness, D.L. (eds.): OWL Web Ontology Language Guide. W3C Recommendation /REC-owl-guide-20040210/> (11/15/2007) (2004), http://www.w3.org/TR/

[OWLRef] Dean, M., Schreiber, G. (eds.): OWL Web Ontology Language Reference. W3C Recommendation (11/14/2007), http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[RDFSem] Hayes, P. (ed.): RDF Semantics. W3C Recommendation (11/26/2007), http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[Rod06] Rodriguez, J.B., Gomez-Perez, A.: Upgrading relational legacy data to the semantic web. In: Proceedings of the 15th international Conference on World Wide Web (2006)

[Seq07] Sequeda, J.F., Tirmizi, S.H., Miranker, D.P.: SQL Databases are a Moving Target. In: Position Paper for W3C Workshop on RDF Access to Relational Databases (October 2007)

[Sto02] Stojanovic, L., Stojanovic, N., Volz, R.: Migrating data-intensive web sites into the semantic web. In: Proceedings of the ACM Symposium on Applied Computing (2002)

[Wan00] Wang, S., Shen, J., Hong, T.: Mining fuzzy functional dependencies from quantitative data. In: IEEE International Conference on Systems, Man and Cybernetics (October 2000)

[XMLSch] Biron, P.V., Permanente, K., Malhotra, A. (eds.): XML Schema Part 2: Datatypes Second Edition. W3C Recommendation (11/26/2007), http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/