On Direct Mapping for Integrating SQL Databases with the Semantic web

Undergraduate Honors Thesis

Juan F. Sequeda
Department of Computer Sciences
The University of Texas at Austin
jsequeda@cs.utexas.edu

Supervising Professor: Dr. Daniel P. Miranker

Table of Contents

1. INTRODUCTION 5

2. ONTOLOGIES AND DATABASES 6

3. DIRECT MAPPING OF A RELATIONAL DATABASE TO THE SEMANTIC WEB 7

4. INCREASINGLY EMPOWERED SQL DATA MODELS 7

5. SQL AND SEMANTIC WEB LAYER CAKE 8

6.1. Relational Model and RDF 9

6.2. Table Definition and RDFS 9

6.3. Constraints and OWL 9

6.4. Triggers and Rules 10

7. EXTRACTING KNOWLEDGE FROM A RELATIONAL SCHEMA 10

8. DISPARITIES BETWEEN RELATIONAL DATABASES AND ONTOLOGIES 12

8.1. Inheritance Modeling 12

8.2. Characteristics of Relationships 13

8.3. The Effect of Open/Closed World Assumptions 14

9. COMPLETENESS OF TRANSFORMATION 14

10. RESEARCH EFFORTS OF DIRECT MAPPING 16

10.1. Relational Database to RDF 16

10.2. Relational Database to RDFS 16

10.3. Relational Database to OWL 18

11. PROPOSED SYSTEM OF DIRECT MAPPING OF A RELATIONAL DATABASE TO THE SEMANTIC WEB 21

11.1. Translating SQL to the Semantic Web 21

11.1.1. Assumptions 21
11.1.2. Predicates and Functions 21
11.1.3 Transformation Rules and Examples 22
11.1.3.1 Producing Unique Identifiers (URIs) and Label 22
11.1.3.2 Transformation of Data Types 23
11.1.3.3 Identifying Classes 23
11.1.3.4 Identifying Object Properties 24
11.1.3.5 Identifying Data Type Properties 26
11.1.3.6 Identifying Inheritance 27

12. CONCLUSIONS 28

13. REFERENCES 30

14. APPENDIX 32

Abstract

Relational databases are a critical source of Internet content. Therefore, the success of the Semantic Web hinges on enabling access to relational databases and their content by semantic methods. This paper surveys methods for the direct mapping of relational/SQL databases to the Semantic Web. *Direct mapping* means that the methods avoid making connections between a database and an existing domain ontology. Compared to wrapping methods that do integrate domain ontologies, an advantage of the direct mapping methods is that they are intrinsically completely automated. As an organizing principle we first suggest a correspondence between individual syntactic structures in SQL-DDL and the layers in the Semantic Web stack. This approach enables us to characterize each research effort with respect to the expressive hierarchy of the Semantic Web languages themselves. Building on the foundation of the prior work we define a direct mapping system, and show that it is complete with respect to all possible associations formed by combinations of primary-key and foreign-key constructs.

1. Introduction

Tim Berners-Lee envisioned the Semantic Web as follows[1]:

"The Semantic Web is not about the meaning of English documents. It's not about marking up existing HTML documents to let a computer understand what they say. It's not about the artificial intelligence areas of machine learning or natural language understanding -- they use the word semantics with a different meaning. ***It is about the data which currently is in relational databases, XML documents, spreadsheets, and proprietary format data files, and all of which would be useful to have access to as one huge database****."*

Since 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 databases [HeP07], it is necessary for the Semantic Web to support integration between the relational database data and a knowledge layer that can represent the semantics of the relational database.

The current web has been designed for the specific intention of human understandable content where most of the content is dynamically generated. On the other hand, the Semantic Web aims to provide reusable and shared data which is machine understandable and can better help users in their tasks. Due to the fact that most of the data on the Internet is sourced in relational databases, the success of the Semantic Web hinges on effective mappings of relational databases and their content to the Semantic Web. Likewise, the labor required to make a database available as Semantic Web content usually rests with the database administrator. It is rare that integrating a database with the Semantic Web accrues obvious benefit to the owners of a database. Thus, organizationally, integrating a database with the Semantic Web is rarely a priority. Hence, it is imperative for the community to make it easy as possible to bridge relational database content and the Semantic Web.

Broadly stated, there are two architectural approaches to integrating databases with the Semantic Web.

- Relational Database to Ontology Mapping: Given the centrality of ontologies to the Semantic Web, most efforts are concerned with wrapper systems that connect the database and its schema to a domain ontology. The ontology may be either a shared global or local ontology. Depending on the aggressiveness of the individual project, the construction of the wrapper may range from strictly manual wrapper programming to sophisticated mapping languages and environments that may even contribute automatic inferences to help define some of the mapping systems. [AnB05] [AnM06] [BaC04] [BaG06] [Che06] [Duo06] [Kor04] [Svi04] [Lab06] [Lab05] [Lac06] [XuZ06]
- Direct Mapping of a Relational Database to the Semantic Web: In a direct mapping an external ontology is not considered. The database content data and its schema are used to export the database to a Semantic Web representation. [Biz04] [Ast04] [Astr07] [LiD05] [Sto02]

The second approach, which is the subject of the work in this paper, concerns the automatic transformation of database content and schema to a Semantic Web representation, i.e. RDF and OWL. Resource Description Framework (RDF) is a language that represents web resources in a triple format of subject-predicate-object. Web Ontological Language (OWL) is a knowledge representation language to create ontologies. In this paper we first define and compare ontologies and databases. In particular we examine the expressive power of the SQL Data Definition Language (DDL), a language for defining the structure of a relational database, with respect to the Semantic Web stack and suggest, perhaps not by coincidence, a correspondence between individual syntactic structures in SQL-DDL and the layers in the Semantic Web stack (Figure 1). A material result is a hierarchy of SQL-DDL grammars whose expressive power corresponds to the stack of Semantic Web languages. Existing research efforts on direct mapping will be characterized and compared utilizing the suggested correspondence between SQL and the Semantic Web.

[1] http://consortiuminfo.org/bulletins/semanticweb.php

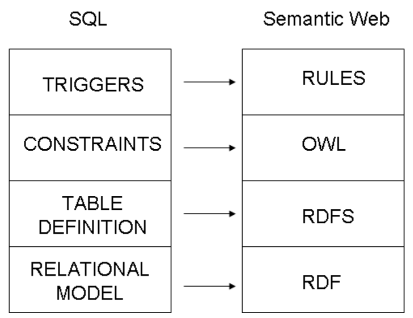

Figure 1. Layer Cake Mapping between SQL and the Semantic Web

Building on the related work described in this paper, we define a system for automatic transformation of relational databases into OWL ontologies. Two critical elements distinguish this new 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, sometimes 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 that are based on enumerating examples instead of focusing on formalizable patterns. *Second*, we have also presented 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.

2. Ontologies and Databases

Studer et al. define ontology as “a formal, explicit specification of a shared conceptualization” [Stu98]. Ontologies contain a vocabulary of the abstract model or concept and the relationships between them in the specific domain. Ontologies capture general knowledge and then therefore can be accepted by a group and be shared.

A relational database, hereafter simple “database”, represents the structure and integrity of the data that is being used in an application. The schema is developed for a specific application and it is not intended to be shared by others.

The main difference is that a database organizes the way data is going to be stored in an efficient manner while an ontology describes the data and its relation to a domain. Many domain experts confuse database schema and ontology because they are not fully aware of the differences between the ontological and relational model. Primary differences are:

- A database is a collection of information that has been organized in a specific manner while an ontology is a “specification of a conceptualization”
- Database contents are typed while everything in the ontology is a concept
- Concepts have either data type property or object (concept) type property while a database table has attributes and relationship with other tables
- Concepts and properties in the ontology can be organized by hierarchies.
- Databases encode application semantics implicitly, ontologies explicitly.
- By design, ontologies are trivially extensible. Database schemas are brittle.

In general, ontologies are more expressive and have richer semantics than relational schemas, but that does not mean that a relational schema do not encode domain semantics. These semantics are implicit in the relational schema. The following approaches demonstrate that the implicit semantics can be extracted to create ontologies with explicit semantics.

3. Direct Mapping of a Relational Database to the Semantic Web

For a relational database to be accessed on the Semantic Web, a mapping between an ontology and a relational database schema has to exist. Direct mapping of a relational database to the Semantic Web automatically generates the ontology from the relational database schema and avoids manually mapping existing domain ontologies to the relational database schema. Through this automated mapping, the database content data and its schema can be exported to a Semantic Web representation (e.g. RDF, OWL).

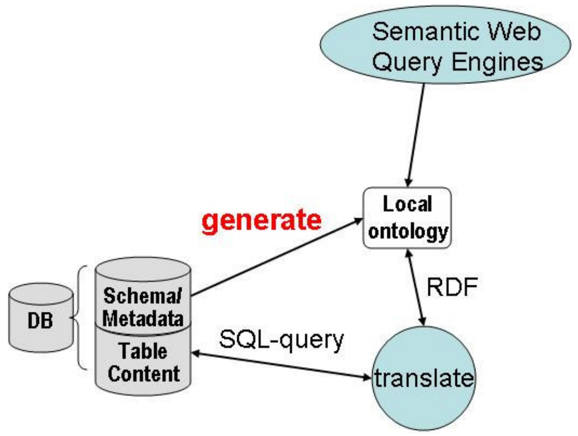

Figure 2. Generating an Ontology from the Relational Database schema

A system for automatic transformation of a relational database to an ontology is best served by the SQL-DDL representation of the relational schema which is based on the following two observations. First, since SQL DDL is not a knowledge representation language, the semantics of most of its constructs are not directly comparable with constructs of an ontology language like OWL. However, SQL DDL is capable of capturing some semantics of the domain modeled in a SQL application. Such semantics can be extracted from the schema by identifying SQL DDL patterns. Secondly, a functioning relational database system is prone to modifications, which usually do not appear in logical models. Hence, the most accurate representation of its schema is usually available in SQL DDL.

4. Increasingly Empowered SQL Data Models

SQL itself is an evolving computer language. Over the last two decades, in an ongoing sequence of expanding standards, (SQL 86-89, 92, 99, 2003) continues to evolve. SQL is organized in parts, the data definition language (SQL-DDL), the query language and the data manipulation language. SQL-DDL is a declarative language for defining both the logical and physical representation of data in a database. Taking some literary license we will provide an overview of the DDL features as they have evolved and show they run parallel with the expressive hierarchy reflected in the Semantic Web layer cake.

Prior to the creation of relational query languages the relational model was largely concerned with the representation of data (Figure 1a) as n-ary relations and the correctness of syntactic transformations relational algebra [Mei83]. Data types for the columns were practically an afterthought. The central concern was the theory of normal forms. A primary goal was to reduce storage overhead.

The first SQL standard, SQL86-89 was limited. For the purpose of this paper, starting from relational algebra following the succinct description suffices, a standard set of data types were defined, abbreviated formal relational notations were replaced full English words (SELECT, CREATE TABLE, etc), and the now familiar accoutrements of human readable computer languages added (Figure 1b). The often heard claim that SQL databases do not provide for the encoding of data semantics is apt for this version of SQL [Pra90].

SQL 92 added data integrity constraints: CHECK, PRIMARY KEY, FOREIGN KEY and UNIQUE [Pra95]. This was the first approach to offer domain semantics by connecting relations and permitting us to know beforehand what values are allowed. In conjunction with the basic table definition of SQL 88, this version of SQL is the current one that is used today. See Figure 1c.

SQL 99 saw the addition of Triggers [SQL3]. Triggers are forward-chaining rules whose predicates are conditioned on changes to table content. Triggers are often used to maintain correctness and to implement heterogeneous data integration [Cer93]. Figure 1d is a trigger that maintains an invariant on the database concerning salary inversion. SQL 2003 introduces XML-related features. We do not yet find that these extensions embody additional semantic properties beyond XML.

a) Relational Model Before SQL	Employee (name, age)
b) Table Definition SQL 86-89	CREATE TABLE employee (name VARCHAR(100), age INTEGER)
c) Constraints SQL 92	CREATE TABLE employee(name VARCHAR(100) PRIMARY KEY, salary INTEGER NOT NULL, type CHAR(8) CHECK(type IN ('TEMP','FULLTIME','CONTRACT')) dept_name FOREIGN KEY (dept) REFERENCES department (name)) CREATE TABLE department(name VARCHAR(100) PRIMARY KEY)
d) Triggers SQL 99	CREATE TRIGGER sal_adjustment AFTER UPDATE OF salary ON employee REFERENCING OLD AS OLD_EMP NEW AS NEW_EMP FOR EACH ROW WHEN (NEW_EMP.SALARY > (OLD_EMP.SALARY *1.20)) BEGIN ATOMIC SIGNAL SQLSTATE '75001'('Invalid Increase: > 20%'); END

Figure 3. Evolution of Relational DDL

5. SQL and Semantic Web Layer Cake

Knowing the historical context, it is easy to decompose SQL-DDL into a stack representing increasing expressive power. It appears that the layers of that stack correspond to the layers of the Semantic Web stack (Figure 1). We observe that systems that translate SQL databases to the Semantic Web have different goals with respect to how much of the expressive power of SQL-DDL targeted and similarly the amount of expressive power of the Semantic Web is exploited. Since both SQL and the Semantic Web are evolving systems, the choices made by individual projects are at least as dependent on the timeframe of the effort as they are on the research itself. The structure illustrated in SQL-DDL can express the domain semantics that the relational database represents by the relationships between the relational database's relations.

As we survey this body of work, we underscore the amount SQL DDL they are consuming. Further, the mappings do not need to agree with the obvious mappings we observe, (in Figure 1), that correspond to the SQL's historic development. For example, when OWL was not a recommendation, the only Semantic Web target was RDFS, which therefore did not exploit the complete expressive power of SQL-DDL. Our evaluation of each system will comprise identifying the portions of SQL-DDL they transform and identifying the actual correspondences.

We begin by demonstrating the implicit semantics that are in SQL-DDL and how they can become explicit. An example of a university relational schema and its corresponding ontology is presented in section 7, thus demonstrating how a relational schema and ontology, of the same domain, are related. We continue to discuss incompatibilities between relational databases and ontologies.

6.1. Relational Model and RDF

The first layer we encounter is the Relational Model, which predates SQL. This layer only expresses the syntactic structure of the data stored in a relational database; therefore we can map it only to the data layer of the Semantic Web: RDF. To export the content in a relational database to the Semantic Web, it is necessary to create an RDF representation of this content.

Likewise, RDF expresses data in a similar way representing it as a labeled graph in the form of a Subject-Predicate-Object statement. The n-tuples are similar to the triple statement of RDF. An example of the RDF representation is in the Appendix. In this example, the column name "age" denotes the edge of the graph.

T1: Employee (Juan, 21)

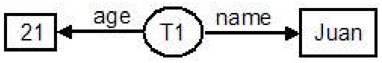

Figure 4: Figure 3a as an RDF Diagram

6.2. Table Definition and RDFS

The relational model offers sufficient semantics to create relations between the data, but unfortunately it does not consist of domain semantics. The first version of SQL (SQL89), introduced the table definition, which creates a relational schema for the data. Each column has a specific data type. Likewise, RDFS is a schema for RDF data which creates classes and properties.

An equally valid representation of a table definition is an n-dimensional graph, where n is the number of column names. This graph representation is similar to RDFS, where it established a schema to represent data in RDF triple form. Therefore, by utilizing this version of SQL, its domain semantics can by mapped only to the RDFS layer. RDFS can represent the table definition (Figure 1b) by having a class Employee with properties name and age. An example of the RDFS format is in the Appendix. In this example, the column name "age" becomes and rdf:Property as shown in Figure 4b.

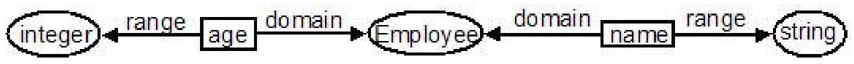

Figure 5a. Figure 3b has an RDFS Diagram

```
<rdf:Property rdf:ID="age">
    <rdfs:comment>Age of Employee</rdfs:comment>
    <rdfs:domain rdf:resource="#employee"/>
    <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Literal"/>
  </rdf:Property>
```

Figure 5b. RDFS format of "age" RDF Property

6.3. Constraints and OWL

Climbing the Semantic Web layer cake, more expressive power is encountered. OWL offers expressiveness that can not be exploited in RDFS. Likewise, SQL99 and in conjunction with elements of SQL92, contains new components that offer more domain semantics. By utilizing PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE and CHECK, domain semantics are more explicit. These semantics are richer than the basic table definition and therefore should be mapped to a higher level in the Semantic Web layer cake. We have encountered that the use of SQL with all the possible constraints will map directly to OWL (hence the use of OWL class instead of RDFS class). Even though RDFS and OWL are both ontology languages, OWL is more expressive due to the fact that it can represent specific type of properties and restrictions. The main similarity between database constraints and OWL is the possibility of using the database's referential constraint to establish relationships between tables as owl:ObjectProperty that connects objects. PRIMARY KEY, UNIQUE and NOT NULL implies owl:FunctionalProperty and the enumerated CHECK constraint implies owl:oneOf (this will be discussed in the next section).

Following the example in Figure 1c, the schema and its constraints can be represented in OWL. In a general assumption, a relation can be considered an OWL class except if the relation is a weak entity, therefore it is an OWL object property; all attributes that are referential constraints are OWL object properties, whose domain is the current relation and range is the referenced relation, and attributes that are not referential constraints are OWL data type properties. In OWL, the "age" column name becomes an owl:DatatypeProperty, as shown in Figure 5. An example of the OWL ontology is in the Appendix.

```
<owl:DatatypeProperty rdf:ID="age">
   <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   <rdfs:domain rdf:resource="#Employee"/>
   <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
```

Figure 6. "Age" attribute of Figure 1c in OWL format

6.4. *Triggers and Rules*

The latest version of SQL (SQL 2003) introduces a new semantic component: triggers. This new component encodes business rules that will guarantee the integrity of data while it is being manipulated in the database. The rules in the Semantic Web have the same objective.

7. Extracting Knowledge from a Relational Schema

Consider a relational database for a university. SQL DDL for this database is given in Figure 6 .

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)
}
```

Figure 6. Schema of a University Database

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.

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. Figure 7 shows an ontology corresponding to the given schema, developed by a domain expert.

Domain Expert's Ontology

```
Ontology(<urn:sql2owl>

 ObjectProperty(<REG> domain(<STUDENT>) range(<SEMESTER>))
 ObjectProperty(<REG_I> inverseOf(<REG>))
 ObjectProperty(<COURSE.DEPTCODE> Functional domain(<COURSE>) range(<DEPT>))
 ObjectProperty(<COURSE.DEPTCODE_I> InverseFunctional inverseOf(<COURSE.DEPTCODE>))
 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)) ...)
 ...
)
```

Figure 7. Parts of an ontology corresponding to the schema in Figure 6, developed by a domain expert. The ontology is presented in OWL Abstract Syntax. The highlighted sections in the table are later compared with an automatically produced ontology.

8. 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 basic assumptions and 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. In the following sections, we discuss some key issues that affect a transformation system. First, we discuss why it is hard to identify inheritance and property characteristics in relational schemas. Then we discuss the effect of open world assumption in ontologies, when a relational database with a closed world assumption is translated into an ontology.

8.1. Inheritance Modeling

Relational databases do not provide a mechanism to express inheritance. However, inheritance hierarchies can be modeled in a variety of ways in relational schemas. Our university schema example shows two different modeling choices for inheritance. *Student* and *Professor* entities are subclasses of *Person*, even though the relationships have been modeled differently. In this section, we present some inheritance modeling possibilities and discuss why some modeling choices are harder to identify automatically.

Given that a relational schema contains relationships between entities that can be expressed using only foreign keys, we find it necessary to identify patterns of foreign key definition that can express only the inheritance relationships. In other words, given a foreign key definition between two entities, is it possible to say that 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 that could be used to express inheritance:

- Foreign key is also the primary key: An example of this case is the *Professor-Person* relationship in our university schema. This pattern uniquely identifies inheritance. An exception to this would be vertical partitioning of tables for performance reasons, as in some data warehousing applications. However, with our requirement of 3NF, such a scenario would not occur. Therefore, we are able to automatically identify inheritance modeled in this way, using our rules given earlier.

- Foreign key and primary key are disjoint: The *Student-Person* relationship in our university schema is an example of this pattern. This pattern does not uniquely identify inheritance, and therefore cannot be automatically translated into an inheritance hierarchy in an ontology. A counterexample is the *Course-Dept* relationship modeled in the same schema. 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 another option for modeling inheritance in a relational database. However, other relationships can also be modeled this way, and therefore it is not a good candidate for automatic translation to an inheritance hierarchy in the ontology. A counterexample for this pattern is:
  Order(ONo) – ONo is the primary key
  OrderItem(ONo,INo) – (ONo,INo) is the primary key, ONo is a foreign key to Order
  In a business domain, the relationship most likely means that an order item is a part of an order, instead of representing an inheritance relationship between the two entities.

8.2. 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.

The self-relation on the *Offer* entity, that represents co-location of an offered course with another offered course, has interesting characteristics. First, it is symmetric, because if an offered course A is co-located with an offered course B, it means B is co-located with A as well. And second, it is transitive, which means that if A is co-located with B, and B is co-located with C, then A is co-located with C.

While these characteristics are obvious to a domain expert, the relationship is expressed like any other self-relationship, 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 that captures the manager's ID.

- Clearly, this relationship is not symmetric, because if John is the manager of Peter, that rules out the possibility of the same Peter being the manager of the same John.
- Depending upon the domain semantics, the relationship may or may not be transitive. If an employee's manager means any other employee higher in the organization, then being a manager is a transitive relationship. If it means only the immediate supervisor, then it is not a transitive relationship.

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.

Another confusing example is the use of owl:someValuesFrom and owl:allValuesFrom. In Xu et al.'s approach [20], the owl:allValuesFrom restriction is applied to every class and their object properties. Even though the reason was not explained, it can be understood that because the source is an Entity-Relationship model, the semantics are more explicit than SQL and therefore it can be assumed that all values from one class are the properties of another class. In SQL it is different. If we consider for example:

A(id, x)   B(id, y)   C(A_id, B_id)

We can ask and answer the following questions: Can we say that A and B are classes? Can we say that C represents an object property? An answer, to both, is yes. Can we say that B can only be connected to A using object property? Can we say anything about the cardinality of this relation? We still can not answer the previous two questions. Therefore, our rules do not implement these two restrictions.

Another component that deserves discussion is CHECK. This constraint embodies semantics and rules at the same time. OWL is not expressive enough to represent all the possibilities that the semantics of CHECK offers. The only CHECK constraint that can be applied to OWL is the enumerated constraint which can be represented using owl:oneOf and rdf:List.

8.3. 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. CW is essential for important database concepts like integrity constraints and data validation [Dru06].

On the other hand, a knowledge-base community like the Semantic Web has an open world (OW) approach 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 with the manual discovery of new domain knowledge and automatic inferences.

Due to this difference, 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 and prevents incorrect data from entering the database. In contrast, in an ontology, a constraint expresses some characteristics of classes or relationships but does not prevent assertion of any facts. Due to these constraints, some assertions may even result in unintuitive inferences.

Consider this example: In our university database, the relationship between a course and a department is expressed by a foreign key constraint. The foreign key constraint will allow the Course relation to have the record (CS386,CS,Databases) where CS is a department. However, the record (CS386,John,Databases) – John is a student – will not be allowed because it violates the integrity constraint. In the ontology, the same constraint appears as object property *CODE*, with range restriction of *Dept* on the relationship. Unlike database constraints, the ontology constraint will not only allow the assertion of the triple *CODE(CS386,John)*, it will also infer that *John* is an instance of *Dept*, because of the range restriction on *CODE* property.

While there are obvious differences between closed and open worlds, open worlds can be closed by explicitly stating required negations. OWL provides a way to state such facts by providing constructs to express disjoints.

When developing an ontology based on a relational schema, it is very important to keep these differences in mind. The question of whether the open world should be closed or not, depends upon the domain and application requirements. If the ontology is for use within a particular application, it might make sense to close the world, whereas in a data integration setting, it might make more sense to have an open world.

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.

9. 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. While it is trivial to translate relations into ontology classes, the existence of foreign keys represents relationships between corresponding classes, and poses a challenge for any transformation system. Multiple foreign keys may be present in a table, and each of them may be in a different form, representing different kinds of relationships, e.g. one-to-one, one-to-many etc., between the entities. The interaction of the foreign keys with primary keys provides clues to the properties of these relationships.

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.*

The proof involves a syntactic enumeration of the cases and a closure operation over the space of relations. Figure 1 provides a useful summary of the theorem and its proof.

Proof: Briefly, we first partition the space by examining the number of foreign keys contained in relations. 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. The cases for one or two foreign keys are more interesting and give rise to more possibilities including

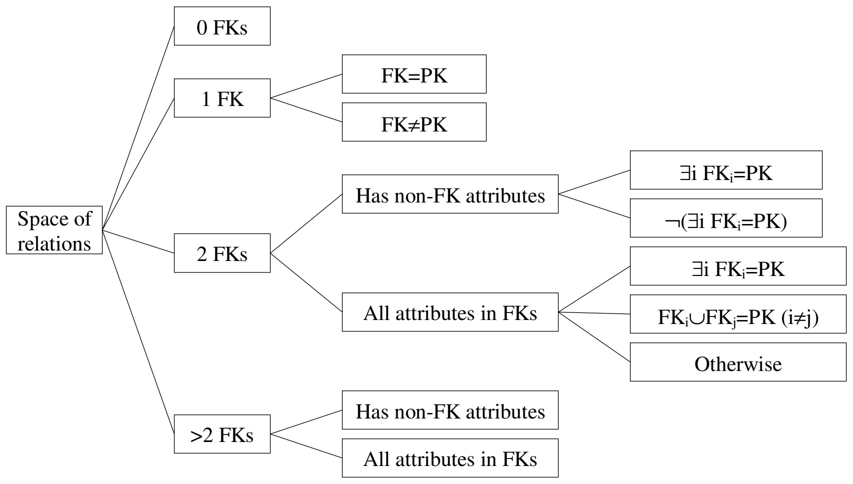

Figure 8. The tree describes the complete space of relations when all possible combinations of primary and foreign keys are considered. For each branch, applicable rules are listed.

binary relations, inheritance and new classes. However, for each possible branch, we have carefully defined sets of rules for producing ontology classes and properties.

A table in a relational database can have either a Primary Key and/or Foreign Key. Both keys can include x amount of attributes where x = 0…n. There can only be one Primary Key in a table, whereas there can be one or more Foreign Keys. Therefore, our initial starting point is:

1. PK: a relation only has a Primary Key
2. C-PK: a relation only has a composite Primary Key
3. S-FK: a relation only has one Foreign Key
4. N-FK: a relation has at least two or more Foreign Keys

This can be represented in the following grammar:

E' → E
E → PK + T | C-PK +T
E → S-FK
E → N-FK
T → S-FK | N-FK

By applying the closure operation of the LR(0) item set, the following elements are obtained:

5. PK + S-FK: a relation has a Primary Key and only one Foreign Key
   a) PK = S-FK: the Foreign Key is the Primary Key
   b) PK ∩ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes
6. PK + N-FK: a relation has a Primary Key and at least two or more Foreign Keys
   a) PK ∩ N-FK = 0: the Foreign Key and the Primary Key do not share any attributes
   b) PK $\subset$ N-FK: one of the Foreign Keys is also the Primary Key
7. C-PK + S-FK: a relation has a Composite Primary Key and only one Foreign Key.
   a) C-PK ∩ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes
   b) S-FK $\subset$ C-PK: the Foreign Key is part of the Primary Key
8. C-PK + N-FK: a relation has a Composite Primary Key and at least two or more Foreign Keys
   a) C-PK ∩ N-FK = 0: all the Foreign Keys and the Primary Key do not share any attributes
   b) N-FK $\subseteq$ C-PK: all the Foreign Keys are part of the Primary Key
   c) C-PK ∩ N-FK $\neq$ 0, C-PK – N-FK $\neq$ 0, N-FK – C-PK $\neq$ 0: The Foreign Keys and Primary Key share common attributes

Therefore, for a transformation system to be complete, it should take into consideration all the possible combinations.

10. Research Efforts of Direct Mapping

The proposed SQL Layer cake can be applied to the any existing approaches of instantiating a relational database to the Semantic Web.

10.1. Relational Database to RDF

The relational model of a relational database is the basis to obtain RDF content (Figure 4). Data is to RDF as schema is to ontology (RDFS or OWL). For there to be data, it has to be stored in a relational database that is made with a schema. Likewise, RDF data is an instance of specific triple schema that is part of an ontology. Therefore to create RDF content from a relational database, it is necessary an ontology. This means that this layer (Relational Database to RDF) depends on the creation of RDFS or OWL ontologies from the database schema.

After creating an ontology from the database schema, the data in the relational database can be extracted and transformed to ontological instances. The problem is when the relational database is updated, and then the extracted data as ontological instances will not represent the current status of the relational database. Another option involves translating SPARQL (RDF query language) to SQL, where one would not have to have to separate sets of data: the relational data and the ontological instances, instead, the SPARQL query can be translated into SQL, extract the data in the relational database, and the translate it, using the ontology that was automatically created, and return the data in RDF format.

10.2. Relational Database to RDFS

There exist two systems for translation of the table definitions and constraints of a relational database to RDFS (Figure 9). Stojanovic et al. [Sto02] was the first effort to extract domain semantics from SQL DDL. There primary contribution is an "*approach for an (automated) migration of data-intensive websites into the Semantic Web*". It provides rules for translation of relational schemas to Frame Logic and RDF Schema. Their rules consist of creating only classes, subclasses and properties. The only constraints that Stojanovic utilizes are Foreign Keys which are used to create ontological relationships. Astrova et al's [Ast04] initial approach is very close to Stojanovic et al, in which the motivation for their approach "*is to migrate from data-intensive Web pages into the Semantic Web*". It is also based on a reverse engineering approach using SQL DDL as the relational database model and transforming it into an RDFS ontology. They differentiate from Stojanovic et al. stating that they "*assume data equality and data inclusion, thus being able to extract only a subset of semantics embedded within a relational database*". Astrova et al. acknowledges that "hidden" semantics can be discovered by analyzing data overlap (intersection) and data disjointedness (no intersection). Due to the fact that these approaches were present when OWL was not a recommendation, they had no other choice to have RDFS has their target ontology language.

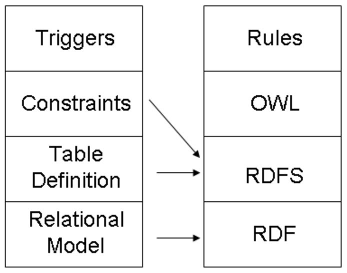

Figure 9. SQL and Semantic Web Layer for Stojanovic et al. and Astrova et al.

While Stojanovic et al.'s work formally defines rules for identification of classes and properties in relational schemas, it does not have the capability of capturing richer semantics of these concepts since they cannot be expressed in their target language, RDF Schema. Its mapping process consists of four steps.

Firstly, it captures information from a relational schema through reverse engineering (relations, attributes, attributes types, primary keys, and foreign keys/inclusion dependencies). It continues by mapping database entities into ontological entities by a set of mapping rules by alignment of top level terms (which relation name corresponds to which concept name), relations to concepts and then relation attributes to concept attributes. The mapping rules consist of identifying Concepts, Inheritance and Relationships. A relationship is a concept except if the relation expresses a many-to-many relationship or if information is spread across several relations; in this case all the relations can be integrated into one concept. Inheritance is an inclusion dependency between two relations exists and both relations are concepts. This rule conflicts when information is spread across several relations. In this case, the user must decide which rule to apply, thus making this effort semi-automatic. Figure A and C represent the translation of SQL-DDL to RDFS. In this example, two tables that share the same primary key could be mapped either to one same concept or to subclass relationship. The user has to decide. In this case, it is mapped to a subclass relationship.

For relationships, a many-to-many relationship, one-to-many relationship (use of foreign keys), one-to-one relationship, and in the last case, a relational attribute is converted into a relationship. If n-ary relationships exist, these must be transformed into further concepts and a set of binary relations. After the mappings are manually evaluated, validated and refined, the data can be migrated by the creating ontological instances based on tuples of the relational database. The process consists of two steps: create the instances with a unique identifier and translate all the attributes, except for foreign keys and then establish relations between instances using the information contained in the foreign keys in the database tuples

Astrova et al.'s transformation systems analyzes the key, data and attribute correlations to extract conceptual schema, which expresses the semantics of a relational database. This process uses a relational database in third normal form (3NF) and consists of the following classifications and mappings. Firstly, relations can be classified as either *Base* (independent of other relations), *Dependent* (the primary key of a relation depends on another relation, meaning that it is also a foreign key) or *Composite* (not Base or Dependent). Secondly, all relations are concepts except if they are a Composite relation. In this case, it can be mapped to a concept, property or an inheritance. N-ary relations become concepts. Third, all attributes are properties except if they are foreign keys or primary keys and foreign keys. Then they are mapped to relationships.

Relationships can be mapped by a combination of key, data and attribute correlation. Attributes that are foreign keys or foreign keys/primary keys are used to map to concepts, properties or relationships. *Key and Data equality and Attribute disjointedness* expresses vertical partitioning and all relations become one concept. *Key and Attribute equality and Data disjointedness* expresses horizontal partitioning and all relations become one concept. *Key equality and Data inclusion* expresses single inheritance. Figure B is an example of SQL-DDL needed to create a single inheritance shown in Figure C. *Key equality, Data overlap and Data inclusion* expresses multiple inheritances by "discovering a new concept". Figure A is an example of the SQL-DDL needed to create a multiple inheritance and discovering the new concept, as shown in Figure D. *Key equality, data disjointedness and Attribute overlap* expresses single inheritance by "discovering a new concept". *Key equality and Data and Attribute overlap* expresses a diamond-shape inheritance. Finally, mapping of constraints are mappings between SQL constraints to F-Logic axioms. Data can be migrated the tuples in the relational database into ontological instances. This process consists of two steps. First, create ontological instances by assigning a name that uniquely identifies it and then establish relationships between the ontological instances.

```
create table STUDENT {
  STUDID integer primary key,NAME varchar}

create table PHDSTUDENT {
 STUDID integer primary key,
 YEAR date not null }
```

Figure A. SQL-DDL example

```
create table STUDENT {
  STUDID integer primary key,NAME varchar}

create table PHDSTUDENT {
 STUDID integer primary key references STUDENT,
 YEAR date not null }
```

Figure B. Another SQL-DDL example

```
<rdfs:Class rdfs:ID=”STUDENT”/>
<rdfs:Class rdfs:ID=”PHDSTUDENT”>
  <rdfs:subClassOf rdf:resource=”http://example.org#Student”/>
</rdfs:Class>
```

Figure C. RDFS result of translating the SQL-DDL in Figure A with Stojanovic et al.'s rules and the SQL-DDL in Figure B with Astrova et al.'s rules

```
<rdfs:Class rdfs:ID=”STUDENTPHDSTUDENT”/>
<rdfs:Class rdfs:ID=”STUDENT”>
  <rdfs:subClassOf rdf:resource=”http://example.org#StudentPhDStudent”/>
</rdfs:Class>
<rdfs:Class rdfs:ID=”PHDSTUDENT”>
  <rdfs:subClassOf rdf:resource=”http://example.org#StudentPhDStudent”/>
</rdfs:Class>
```

Figure D. RDFS result of translating the SQL-DDL in Figure B with Astrova et al.'s rules

As a result of these examples, different SQL-DDL can translate to the same RDFS constructs and the same SQL-DDL can translate to different RDFS constructs. Therefore, these systems are prone to different interpretations and ambiguities.

10.3. Relational Database to OWL

After OWL became a recommendation, several research efforts initiated by taking the table definition and constraints to extract OWL ontologies from SQL-DDL (Figure 10). Li et al. [LiD05]presents a semi-formal approach with a set of rules for automatically learning an OWL ontology from a relational schema similar to Stojanovic et al. Astrova et al [Ast07] presents a new informal approach that identifies more semantics than the previous efforts.

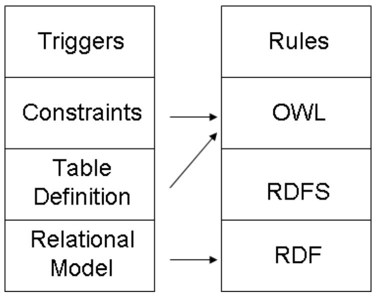

Figure 10. SQL and Semantic Web Layer for Li et al. and Astrova et al.

The five group of learning rules proposed by Li et al. maps the relational database to the OWL ontological language. Classes can be learnt if several relations are used to describe one entity by sharing the same primary key, then they are all integrated into one ontological class or if a relation is used to describe an entity and not a relationship between relations, then it is also mapped into one ontological class.

Object Property can be learnt if an attribute of a relation is also an attribute of another relation (foreign key), then the attribute becomes a property. Additionally, "has-part" and "is-part-of" properties can be created if a relation has a foreign key as a primary key and therefore it refers to another relation. Binary relations that indicate the relationship between two relations also become object properties. Figure E represents the translation of the table REG from Table 1 to the OWL object property. If the relation represents a n-ary relationship, then there is an object property that connects every single relation with the other relations in the n-ary relationship. If an attribute can not be converted into an object property, it is converted into a data type property.

```
<owl:ObjectProperty rdf:ID=" REG ">
  <rdfs:domain rdf:resource="# Student "/>
  <rdfs:range rdf:resource="# Semester "/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID=" INVERSE-REG ">
  <rdfs:domain rdf:resource="# Semester "/>
  <rdfs:range rdf:resource="# Student "/>
  <owl:inverseOf rdf:resource="# REG " />
</owl:ObjectProperty>
```

Figure E. OWL result of translating the SQL-DDL in Table X with Li et al.'s rules

If two relations share the same primary key, then it expresses an inheritance. This is the same rule for learning an ontological class. It has to be decided whether the two relations represent a hierarchy or one ontological class. Thus, a user has to decide which rule to apply, making this system semi-automatic. Cardinalities are learned from the constraints of attributes in the relations. If an attribute is a primary key or foreign key then the minimum and maximum cardinality is 1. If an attribute is NOT NULL, the minimum cardinality is 1. If an attribute is UNIQUE, the maximum cardinality is 1. Finally, the instances of an ontological class consist of tuples in relations corresponding to a class and relations between instances are established using the foreign keys.

Li et al. defines the rules using a combination of some formal notation and English language. Our analysis of their work shows that: 1) some of their rules miss some semantics offered by the relational schema (e.g. Rule 2 for identifying classes, Rule 6 for N-ary relations); 2) some rules produce specific results for inheritance and object properties which may not accurately depict concepts across domains or database modeling choices (e.g. Rules 4, 8); 3) some rules are completely irrelevant (e.g. Rule 11 for uniqueness constraint). We believe that their 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. provides expository rules and examples to describe a system for automatic transformation of a relational schema to an OWL ontology which discovers more semantics than the previous effort. However, the output from this system does not conform to OWL DL restrictions. Astrova et al's approach is to map the relational model and the ontological model which represent the relational database and the ontology respectively. Firstly, it maps tables to a class unless a table exists with only two foreign keys that map to two other tables, then it is mapped to two object properties (one is an inverse of another). Figure F represents the translation of the table REG from Table 1 to the OWL object property. Second, columns become data type properties with a maximum cardinality of 1 unless it is a foreign key. Third, a SQL data type is mapped to XML Schema Data types (XSD). If a CHECK constraint that verifies that an integer is greater than 0 is used, then the XSD used is positiveInteger.

```
<owl:ObjectProperty rdf:ID="SID">
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Semester"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="SNO">
  <owl:inverseOf rdf:resource="#SID"/>
```

```
</owl:ObjectProperty>
```

Figure F. OWL result of translating the SQL-DDL in Table X with Astrova et al.'s rules

It continues by mapping constraints. The UNIQUE column constraint is mapped to an inverse functional property. This rule can not be applied for an OWL ontology because Inverse Functional Properties can only be applied to Object Properties and not Data Type Properties [OWLGuide]. NOT NULL constraints are mapped to a minimum cardinality of 1. PRIMARY KEY constraints are taken has a UNIQUE and NOT NULL constraints. This rule can not be applied, for the same reason as UNIQUE. When mapping references and foreign keys, several cases may apply. If the foreign key is not part of the primary key, then it is mapped as an object property. If the foreign key is part of the primary key, it is also mapped to an object property accompanied with its cardinality of 1. If the foreign key is the primary key, then it is mapped to an inheritance. If a foreign key is a reference to its same table, then it is considered a Symmetric Property. CHECK constraints can be mapped to hasValue property or an enumerated one of. Finally, the INSERT statements are used to create instances. The instances presented are in an XML format instead of the Semantic Web RDF format.

Li et al. and Astrova et al. share very similar rules, for example the inheritances rule. Applying their rules to the Professor and Person relations in Table X, would produce the OWL output in Figure G.

```
<owl:Class rdf:ID=" Professor ">
 <rdfs:subClassOf rdf:resource="# Person " />
</owl:Class>
```

Figure G. OWL result of translating the SQL-DDL in Table X with Li et al. and Astrova et al.'s rules

Astrova et al. identifies more semantics compared to Li et al. However, the output from this system does not conform to OWL DL restrictions. Since the rules have not been formally defined, the system is susceptible to ambiguities. This work provides a several rules that are inaccurate. Firstly, the work gives an example of how to create a symmetric property from the following SQL-DDL:

```
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 >
```

This domain example is accurate for creating a symmetric property, but if the example is changed to "manager" instead of "spouse" then the property would not hold (i.e. if one employee is the manager of another employee, the second employee can not be a manager of the first employee). Another inaccurate rule is when the UNIQUE column constraint is mapped to an inverse functional property. This rule can not be applied in OWL. Inverse Functional Properties can only be applied to Object Properties and not Data Type Properties. UNIQUE describes a key constraint, and would be very likely to lead to computational intractability even in realistic ontologies, and are not supported by any implemented OWL reasoners. [LAHS04a].

Due to the fact the Li et al. and Astrova et al.'s rules are not formally defined, determining if their transformation systems are complete may lead to ambiguity. Li et al. does use a combination of formal notation and English language and has enumerated the rules, thus it is easier to identify which rules apply to specific cases. On the other hand, Astrova et al. only presents examples of transformation without any formalism at all. Therefore, it is ambiguous and there is no formal way to guarantee completeness.

11. Proposed system of Direct Mapping of a Relational Database to the Semantic Web

11.1. Translating SQL to the 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.

11.1.1. 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.* As a good software engineering practice, it is quite common to develop logical models for the relational schema. However, even after deployment, a database undergoes modifications due to changing application requirements. Such modifications are often not reflected on the logical models. Therefore, the physical model, easily expressed in SQL DDL, becomes 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].

11.1.2. 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 (or table) identified by CREATE TABLE statement; for example: *Rel(PERSON)* holds, *Rel(ID)* does not hold
Attr(x,r)	-	*x* is an attribute in relation *r*; for example: *Attr(ID,PERSON)* holds, *Attr(STUDY)* does not hold
NN(x,r)	-	*x* is an attribute (or a set of attributes) in relation *r* with NOT NULL constraint(s); for example: *NN(NAME,PERSON)* holds
Unq(x,r)	-	*x* is an attribute (or a set of attributes) in relation *r* with UNIQUE constraint; for example *Unq({NAME},DEPT)* holds
Chk(x,r)	-	*x* is an attribute in relation *r* with enumerated list (CHECK IN) constraint; for example *Chk(SESSION,SEMESTER)* holds
PK(x,r)	-	*x* is the (single or composite) primary key of relation *r* identified using the PRIMARY KEY constraint; for example: *PK({OFFER,SID},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*; for example: *FK({ID},STUDENT,{ID},PERSON)* holds
NonFK(x,r)	-	*x* is an attribute in relation *r* that does not participate in any foreign key; for example: *NonFK(NAME,DEPT)* holds, *NonFK(SID,REG)* does not hold

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, i.e. inverse of a functional property
Crd(p,m,v)	-	the (maximum and minimum) cardinality of property *p* for class *m* is *v*
MinC(p,m,v)	-	the minimum cardinality of property *p* for class *m* is *v*
MaxC(p,m,v)	-	the maximum 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 *x*; undefined if there is no such foreign key
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 the list of allowed values for it; this function is applicable only to attributes that have a CHECK IN constraint defined on them, 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:

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

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.

11.1.3 Transformation Rules and Examples

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

11.1.3.1 Producing Unique Identifiers (URIs) and Label

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. Therefore, 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.

For the purposes of this paper and 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.

11.1.3.2 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 1 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.

SQL Data Type	**XML Schema Type**	**SQL Data Type**	**XML Schema Type**
INTEGER	xsd:integer	VARCHAR	xsd:string
DECIMAL	xsd:decimal	CHAR	xsd:string
FLOAT	xsd:float	DATE	xsd:date
REAL	xsd:double	TIME	xsd:time
BOOLEAN	xsd:boolean	TIMESTAMP	xsd:dateTime

Table 1. Some common SQL data types and corresponding XML Schema types recommended for OWL

11.1.3.3 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 classes in an OWL ontology.

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 (as defined in 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. In our example schema, the *Person* table does not have a foreign key, so *Rel(PERSON)* is true and *BinRel(PERSON,_,_)* is false. Therefore, we conclude *Class(PERSON)*, i.e. *Person* should be mapped to a class. The same reasoning holds for the *Dept* and *Semester* tables.
- All tables that have one foreign key should also be transformed to classes. No such tables can satisfy the *BinRel* predicate. Using the same rule we conclude that *Student*, *Professor* and *Course* should be mapped to classes.
- The tables with more than two foreign keys should be transformed to classes as well. Such tables may represent an entity (when they have attributes not appearing in a foreign key) or an N-ary relationship between entities (where all attributes appear in foreign keys). Fortunately, in OWL, both these cases can be modeled the same way, i.e. by translating the entity or the N-ary relationship into a class [Noy06]. From our running example, *Offer* represents an N-ary relationship, and can be modeled as a class using the given rule.

- For tables containing exactly two foreign keys, presence of independent attributes qualifies them to be treated as entities instead of binary relations, and translated to classes in OWL ontologies. The table *Study*, with an independent attribute *Grade*, is an example of this case, and is translated to an OWL class.

So, as a result of applying Rule Set 2, we have successfully identified the classes (see Table 2) from our relational schema.

Classes			
Class(PERSON)	*Class(STUDENT)*	*Class(PROFESSOR)*	*Class(DEPT)*
Class(SEMESTER)	*Class(COURSE)*	*Class(STUDY)*	*Class(OFFER)*

Table 2. Classes identified from the relational schema by applying Rule Set 2

11.1.3.4 Identifying Object Properties

A property is a binary relation that lets us assert general facts about the members of classes. There are two major types of properties, object properties and data type properties [OWLGde]. Properties have directions, from domain (which defines the subject) to range (which defines the object) [OWLRef]. We will describe data type properties in the next section.

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, written *r′* in our notation, such that, *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, only 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 (data type or object) property with minimum cardinality of 0 and maximum cardinality of 1 is called a *functional property*, represented as *FP* in our rules. If an object property is functional, then its inverse is an *inverse functional property*, represented as *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* – representing foreign keys not in or referencing binary relations – and then express the rules in terms of this predicate.

Rule Set 4:

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

a. $\mathit{ObjP}(x,s,t), \mathit{FP}(x), \mathit{MinC}(x',t,0) \leftarrow \mathit{NonBinFK}(x,s,y,t) \wedge \neg\mathit{NN}(x) \wedge \neg\mathit{Unq}(x)$

b. $\mathit{ObP}(x,s,t), \mathit{FP}(x), \mathit{Crd}(x,s,1), \mathit{MinC}(x',t,0) \leftarrow \mathit{NonBinFK}(x,s,y,t) \wedge \mathit{NN}(x) \wedge \neg\mathit{Unq}(x)$

c. $\mathit{ObjP}(x,s,t), \mathit{FP}(x), \mathit{FP}(x') \leftarrow \mathit{NonBinFK}(x,s,y,t) \wedge \neg\mathit{NN}(x) \wedge \mathit{Unq}(x)$

d. $\mathit{ObjP}(x,s,t), \mathit{FP}(x), \mathit{Crd}(x,s,1), \mathit{FP}(x') \leftarrow \mathit{NonBinFK}(x,s,y,t) \wedge \mathit{NN}(x) \wedge \mathit{Unq}(x) \wedge \neg\mathit{PK}(x,s)$

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 can reference at most one record (instance) of the range, the object property is functional. This entails that inverse of that object property is inverse functional. One example from our university schema is the foreign key from *Study* to *Student* which gives us: *ObjP(RNO,STUDY,STUDENT)*, *FP(RNO)*, *Inv(RNO',RNO)*, *ObjP(RNO',STUDENT,STUDY)*, *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.

Notice that we do not create an object property if the foreign is being equal to the primary key. Instead, we consider it as a pattern for inheritance and propose a rule for inheritance mapping. On the other hand, we are unable to capture the inheritance between *Student* and *Person*, since it is not a unique inheritance pattern, and we transform this relationship into an object property.

A list showing some object properties and their characteristics obtained from the sample relational schema by applying Rule Sets 3 and 4 are presented in Table 3.

Object Properties
ObjP(REG,STUDENT,SEMESTER), ObjP(REG',SEMESTER,STUDENT), Inv(REG,REG')
ObjP(ID1,STUDENT,PERSON), FP(ID1), FP(ID1'), Crd(ID1,STUDENT,1)
ObjP(CODE,COURSE,DEPT), FP(CODE), IFP(CODE'), Crd(CODE,COURSE,1), MinC(CODE',DEPT,0)

ObjP(CNO,OFFER,COURSE), FP(CNO), IFP(CNO’), MinC(CNO’,COURSE,0)
ObjP(CONO,OFFER,OFFER), FP(CONO), IFP(CONO’), MinC(CONO’,OFFER,0)
ObjP(RNO,STUDY,STUDENT), FP(RNO), IFP(RNO’), MinC(RNO’,STUDENT,0)

Table 3. Some object properties identified from the relational schema by applying Rule Sets 3 and 4. An object property *P* implies the existence of an inverse property *P’*. Due to lack of space, we explicitly specify the inverse property only for the first property.

11.1.3.5 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 in a relational schema.

Rule Set 5:

a.	*DTP(x,r,type(x)), FP(x)*	←	*NonFK(x,r)*
b.	*DTP(x,r,type(x)), FP(x), Crd(x,r,1)*	←	*NonFK(x,r) ∧ NN(x,r)*
c.	*DTP(x,r,type(x)∩list(x)), FP(x)*	←	*NonFK(x,r) ∧ 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.

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 listed in Table 4.

Data Type Properties
DTP(ID1,PERSON,xsd:integer), FP(ID1), Crd(ID1,PERSON,1)
DTP(NAME1,PERSON,xsd:string), FP(NAME1), Crd(NAME1,PERSON,1)
DTP(ROLLNO,STUDENT,xsd:integer), FP(ROLLNO), Crd(ROLLNO,STUDENT,1)
DTP(DEGREE,STUDENT,xsd:string), FP(DEGREE)
DTP(SNO,SEMESTER,xsd:integer), FP(SNO), Crd(SNO,SEMESTER,1)
DTP(YEAR,SEMESTER,xsd:date), FP(YEAR), Crd(YEAR,SEMESTER,1)
DTP(SESSION,SEMESTER,xsd:string∩{SPRING,SUMMER,FALL}), FP(SESSION)
DTP(GRADE,STUDY,xsd:string), FP(GRADE)

Table 4. Some data type properties identified from the relational schema by applying Rule Set 5.

11.1.3.6 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. As discussed earlier, since most of these models are not limited to expressing inheritance alone, sometimes it is hard to identify subclass relationships in a relational schema.

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

Rule Set 6:

$$Subclass(r,s) \quad \leftarrow \quad Rel(r) \wedge Rel(s) \wedge PK(x,r) \wedge FK(x,r,\_,s)$$

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 5.

A comparison of the ontologies produced by the domain expert with the one produced automatically using our rules (Table 5) 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.

Automatically Produced Ontology

```
Ontology(<urn:sql2owl>

 ObjectProperty(<REG> domain(<STUDENT>) range(<SEMESTER>))
 ObjectProperty(<REG_I> inverseOf(<REG>))
 ObjectProperty(<COURSE.DEPTCODE> Functional domain(<COURSE>) range(<DEPT>))
 ObjectProperty(<COURSE.DEPTCODE_I> InverseFunctional inverseOf(<COURSE.DEPTCODE>))
 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>))
 ObjectProperty(<STUDENT.ID_I> Functional InverseFunctional
  inverseOf(<STUDENT.ID>))

 ...

 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)) ...)
 ...
)
```

Table 5. Parts of an ontology corresponding to the University Database, produced automatically by applying the rules on the given SQL DDL. The output format is OWL Abstract Syntax. The underlined sections show the differences compared to the human-developed ontology shown earlier in Figure 7.

12. Conclusions

Relational database systems need to fulfill the emerging requirements that are data integration and knowledge representation. Embracing the technology provided by the Semantic Web is a practical way of accomplishing these objectives.

SQL DDL is a standard language for representation of relational schemas. While it is not a knowledge representation language, we have shown that it is capable of capturing some semantics of the domain represented by the relational model.

OWL ontologies are capable of capturing the domain semantics offered by relational models.. These ontologies provide useful inputs to data integration and knowledge extraction applications. 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.

A survey of existing approaches of direct mapping of relational databases to ontologies has been presented. Counterexamples of the rules of the existing approaches have also been presented to prove the need of a formal transformation system. In this paper we also have defined a system for automatic transformation of normalized relational schemas represented in SQL DDL into OWL ontologies. We have defined our entire set of transformation rules in first order logic eliminating the possibility of syntactic and semantic ambiguities. The use of first order logic also allows for easy implementation of the system in languages like Prolog and Datalog. In defining our rules, we have ensured compatibility with description logics based OWL sublanguage (OWL DL), which is essential to assuring decidability for reasoning. In the appendix, we show a comparison of the surveyed approaches.

We have demonstrated that an automatic transformation system has its deficiencies when it comes to identifying inheritance and other richer semantics. These deficiencies stem from the limitations of the expressive power of SQL DDL, and become evident when an ontology produced by the system is compared to an ontology produced by a domain expert. Also, the difference of open world assumption (in ontologies) and closed world assumption (in relational databases) is a crucial factor in the output of such systems.

We also believe that the following are fundamental ingredients for a transformation system aiming for the migration of relational data to the Semantic Web:

1. Reasoning over ontologies is a key goal of the Semantic Web [Hor03]. While it is useful to map relational databases to the Semantic Web ontologies, it is important to stay within a framework that assures decidability for an OWL reasoner. OWL DL is a sublanguage of OWL that guarantees decidability. Therefore, transformation rules for such systems should guarantee producing OWL DL.
2. The rules for a transformation system should be specified formally to avoid any syntactic or semantic ambiguities in the specifications. Also, rules defined in formal systems like first order logic can be easily implemented in languages like Prolog or Datalog.
3. When developing rules for automatic translation of a relational database to an ontology, special care should be taken to avoid the influence of domain specific examples. An automated transformation system should try to capture the semantics offered by the schema definition language alone. Sometimes, the

influence of examples from a particular domain can result in incorrect rules that are based on enumerating examples instead of focusing on formal power.

We admit that the scope and frequency of success of direct transformation is very unlikely to achieve the scope of description ontology mapping methods can achieve, in part because the amount of domain semantics captured in SQL DDL models is highly variable. However, such methods are intrinsically completely automated and offer solutions to the problems that *relational database to ontology mapping* have.

In the future, we plan to extend the automatic transformation system to unnormalized databases by learning functional dependencies from the database content. We would also like to reduce the limitations of a transformation system by identifying the cases where human intervention becomes unavoidable and investigating whether efficient semi-automatic solutions can be developed and deployed.

13. References

[AnB05] An, Y., Borgida, A., & Mylopoulos, J. (2005). Inferring Complex Semantic Mappings between Relational Tables and Ontologies from Simple Correspondences. *On The Move to Meaninful Internet Systems.*
[AnM06] An, Y., Mylopoulos, J., & Borgida, A. (2006). Building Semantic Mappings from Databases to Ontologies. *21th National Conference on Artificial Intelligence.*
[Ast04] Astrova, I. (2004). Reverse Engineering of Relational Databases to Ontologies. In *The Semantic Web: Research and Applications.* Springer Berlin / Heidelberg.
[Ast07] Astrova, I., Korda, N., & Kalja, A. (2007). Rule-Based Transformation of SQL Relational Databases to OWL Ontologies. *2nd International Conference on Metadata & Semantic Research.*
[BaM07] Barbancon, F., & Miranker, D. P. (2007). SPHINX: Schema integration by example. *Journal of Intelligent Information Systems , 29* (2).
[BaG06] Barrasa, J., & Gomez-Perez, A. (2006). Upgrading relational legacy data to the semantic web. *15th international conference on World Wide Web.*
[BaC04] Barrasa, J., Corcho, O., & Gomez-Perez, A. (2004). R2O, an Extensible and Semantically Based Database-to-Ontology Mapping Language. *Second Workshop on Semantic Web and Databases.*
[XML] Biron, P. V., Permanente, K., & Malhotra, A. (2004, October 28). *XML Schema Part 2: Datatypes Second Edition*. Retrieved May 2, 2008, from W3C Recommendation: http://www.w3.org/TR/xmlschema-2/
[Biz04] Bizer, C., & Seaborne, A. (2004). D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. *3rd International Semantic Web Conference.*
[Che06] Chen, H., Wang, Y., Wang, H., Mao, Y., Tang, J., Zhou, C., et al. (2006). Towards a Semantic Web of Relational Databases: a Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine. *5th International Semantic Web Conference.*
[Lab06] de Laborda, C. P., & Conrad, S. (2006). Database to Semantic Web Mapping using RDF Query Languages. *25th International Conference on Conceptual Mapping.*
[Lab05] de Laborda, C. P., & Conrad, S. (2005). Relational.OWL: a data and schema representation format based on OWL. *2nd Asia-Pacific Conference on Conceptual Modelling*, *43.*
[OWLRef] Dean, M., & Schreiber, G. (2004, February 10). *OWL Web Ontological Language Reference*. Retrieved May 2, 2008, from W3C Recommendation: http://www.w3.org/TR/owl-ref/
[Dru06] Drummond, N., & Shearer, R. (2006). The Open World Assumption. *eSI Workshop: The Closed World of Databases meets the Open World of the Semantic Web.*
[DuW99] Du, H., & Wery, L. (1999). Micro: A normalization tool for relational database engineers. *Journal of Networkand Computer Applications* .
[Duo06] Duo, D., Pan, J., Qin, H., & LePendu, P. (2006). Towards Populating and Querying the Semantic Web. *Workshop on Scalable Semantic Web Knowledge Base Systems.*
[Gru93] Gruber, T. (1993). A translation approach to portable ontology specifications. In *Knowledge Acquisition* (Vol. 5). Academic Press Ltd.
[RDFSem] Hayes, P. (2004, February 10). *RDF Semantics*. Retrieved May 2, 2008, from W3C Recommendation: http://www.w3.org/TR/rdf-mt/
[HeP07] He, B., Patel, M., Zhang, Z., & Chang, K. (2007, May). Accessing the deep web. *Communications of the ACM , 50* (5), pp. 94-101.
[Hor03] Horrocks, I., & Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. *2nd International Semantic Web Conference.*
[Kor04] Korotkiy, M., & Top, J. (2004). From Relational Data to RDFS Models. *International Conference on Web Engineering.*
[Lac06] Laclavik, M. (2006). RDB2Onto: Relational Database Data to Ontology Individual Mapping. *Tools for Acquisition, Organisation and Presenting of Information and Knowledge.*
[LiD05] Li, M., Du, X., & Wang, S. (2005). Learning ontology from relational database. *4th International Conference on Machine Learning and Cybernetics.*
[Mei83] Meier, D. (1983). *The Theory of Relational Databases.* Computer Science Press.
[Mil0] Miller, R., Haas, L., & Hernandez, M. (2000). Schema mapping as query discovery. *Very Large Database Conference.*

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

[Noy06] Noy, N., & Rector, A. (2006, April). *Defining N-ary Relations on the Semantic Web*. Retrieved May 2, 2007, from W3C Working Group Note 12: http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/

[Pra90] Pratt, P. J. (1990). *A Guide to SQL.* Boston: Boyd & Fraser Publishing Company.

[Pra95] Pratt, P. J. (1995). *A Guide to SQL* (3rd Edition ed.). Boston: Boyd & Fraser Publishing Company.

[OWLG]Smith, M. K., Welty, C., & McGuinness, D. L. (2004, February 10). *OWL Web Ontology Language Guide*. Retrieved May 2, 2008, from W3C Recommendation: http://www.w3.org/TR/owl-guide/

[SQL3] *SQL3 (ISO-ANSI Working Draft)*. (n.d.). Retrieved May 2, 2008, from http://www.inf.fu-berlin.de/lehre/SS94/einfdb/SQL3/sqlindex.html

[Sto02] Stojanovic, L., Stojanovic, N., & Volz, R. (2002). Migrating data-intensive web sites into the Semantic Web. *ACM Symposium on Applied computing.*

[Stu98] Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge Engineering: Principles and Methods. In *Data and Knowledge Engineering* (Vol. 25). Elsevier.

[Svi04] Svihla, M., & Jelinek, I. (2004). Two Layer Mapping from Database to RDF. *Electronic Computers and Informatics.*

[Wan00] Wang, S., Shen, J., & Hong, T. (2000). Mining fuzzy functional dependencies from quantitative data. *IEEE International Conference on Systems, Man and Cybernetics*, *5.*

[XuZ06] Xu, Z., Zhang, S., & Y, D. (2006). Mapping between Relational Database Schema and OWL Ontology for Deep Annotation. *IEEE/WIC/ACM international Conference on Web intelligence.*

14. Appendix

	Li et al.	**Astrova et al (2)**	**Sequeda et al.**
Ontology Language	OWL	OWL Full	OWL-DL
<table name>	OWL Class RDFS Subclass OWL ObjectProperty RDFS Domain RDFS Range	OWL Class RDFS Subclass OWL ObjectProperty RDFS Domain RDFS Range	OWL Class RDFS Subclass OWL ObjectProperty RDFS Domain RDFS Range
<column name>	OWL Datatype Property OWL Object Property RDFS Domain RDFS Range	OWL Datatype Property OWL Object Property RDFS Domain RDFS Range	OWL Datatype Property OWL Object Property RDFS Domain RDFS Range
<data type>	RDFS range	OWL Maximum Cardinality of 1 RDFS range	RDFS range
<referential constraint definition>	OWL Object Property OWL Minimum Cardinality of 1 OWL Maximum Cardinality of 1	RDFS Subclass OWL Object Property OWL Cardinality of 1 OWL Symmetric Property OWL Transitive Property	OWL Object Property OWL Cardinality of 1
<primary key definition>	OWL Minimum Cardinality of 1 OWL Maximum Cardinality of 1	OWL Inverse Functional Property OWL Minimum Cardinality of 1	OWL Functional Property
<unique constraint definition>	OWL Maximum Cardinality of 1	OWL Inverse Functional Property	OWL Functional Property
<not null constraint definition>	OWL Minimum Cardinality of 1	OWL Minimum Cardinality of 1	OWL Functional Property OWL Minimum Cardinality of 1 OWL Minimum Cardinality of 0 OWL Functional Property
<check constraint definition>		OWL hasValue OWL oneOf	OWL hasValue OWL oneOf

	Li *et al.*	**Astrova *et al.* (2)**	**Sequeda et al.**
owl:class	x	x	X
owl:allValuesFrom			
owl:someValuesFrom			
owl:hasValue		x	
Cardinality constraint			
owl:maxCardinality	x	X	X
owl:minCardinality	x	x	X
owl:cardinality			X
owl:intersectionOf			
owl:unionOf	x		
owl:complementOf			
rdfs:subClassOf	x	x	X
owl:equivalentClass			
owl:disjointWith			
owl:oneOf		x	X
owl:DatatypeProperty	x	x	X
owl:ObjectProperty	x	x	X
RDFS Properties			
rdfs:subPropertyOf			
rdfs:domain	x	x	X
rdfs:range	x	x	X
owl:equivalentProperty			
owl:inverseOf	x	X	X
owl:FunctionalProperty			X
owl:InverseFunctionalProperty		X	X
owl:TransitiveProperty		X	
owl:SymmetricProperty		x	

Table with	Li *et al.*	Astrova *et al.* (2)	*Sequeda et al.*
1) PK	Rule 2, 7, 9, 10, 11	X[2]	Rule 2, 5
2) C-PK	Rule 2, 7, 9, 10, 11	X	Rule 2, 5
3) S-FK	Rule 2, 7, 9, 10, 11	X	Rule 2, 4, 5
4) N-FK	Rule 2, 7, 9, 10, 11	X	Rule 2, 4, 5
5) PK + S-FK			
a) PK = S-FK	Rule 2, 7, 8, 9, 10, 11	X	Rule 2, 5, 6
b) PK ∩ S-FK = 0	Rule 2, 3, 7, 9, 10, 11	X	Rule 2, 4, 5
6) PK + N-FK			
a) PK ∩ N-FK = 0	Rule 2, 7, 9, 10, 11	X	Rule 2, 4, 5
b) PK $\subset$ N-FK	Rule 2, 7, 9, 10, 11	X	Rule 2, 4, 5, 6
7) C-PK + S-FK			
a) C-PK ∩ S-FK = 0	Rule 2, 3, 7, 9, 10, 11	X	Rule 2, 5, 6
b) S-FK $\subset$ C-PK:	Rule 2, 4, 7, 9, 10, 11	X	Rule 2, 4, 5
8) C-PK + N-FK			
a) C-PK ∩ N-FK = 0	Rule 2, 3, 7, 9, 10, 11	X	Rule 2, 4, 5
b) N-FK $\subseteq$ C-PK	Rule 2, 5, 6, 7, 9, 10, 11	X	Rule 2, 3, 4, 5
c) C-PK ∩ N-FK $\neq$ 0, C-PK – N-FK $\neq$ 0, N-FK – C-PK $\neq$ 0	Rule 2, 7, 9, 10, 11	X	Rule 2, 4, 5

Rules that apply to each of the Closure set item

[2] Astrova et al. only offers expository examples, therefore it can not be formally assigned to a Primary Key – Foreign Key combination