

DIRECT MAPPING SQL DATABASES TO THE SEMANTIC WEB: A SURVEY

JUAN F. SEQUEDA

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN
JSEQUEDA@CS.UTEXAS.EDU

SYED H. TIRMIZI

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN
HAMID@CS.UTEXAS.EDU

OSCAR CORCHO

DEPARTAMENTO DE INFORMATICA
UNIVERSIDAD POLITECNICA DE MADRID
OCORCHO@FI.UPM.ES

DANIEL P. MIRANKER

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN
MIRANKER@CS.UTEXAS.EDU

Abstract

The Semantic Web anticipates rich, integrated access to a large number of information sources on the Internet. Since a large number of websites are backed by SQL databases, methods that automate the integration of these databases with the Semantic Web are crucial. This paper surveys methods used to create ontological descriptions of databases by examining the database schema, normally known as direct mapping methods. Compared to wrapping methods, which map databases to existing legacy ontologies, direct mapping methods have the advantage that they are intrinsically automated.

In this review we first identify a correspondence between individual syntactic structures in SQL data description language (DDL) and layers of the Semantic Web stack, and we then characterize each research effort's transformation methods with respect to the expressive hierarchy of the Semantic Web languages themselves. Much of the prior work on direct mapping is expository in nature, relying heavily on examples. From this foundation, we define a unifying transformation system using transformation rules defined using first-order-logic. We show that the unified system is complete with respect to all possible associations formed by combinations of primary-key and foreign-key constructs. The unified transformation system has been implemented using the JESS rule engine, and has been made available to the community as a web-service.

1. Introduction

A goal of the Semantic Web is to add semantics to the existing data on the web and thus create an integrated web of data. Where much of the effort on the Semantic Web concerns annotating HTML pages with semantic tags such as RDFa or Microformat, or extracting RDF from XML or XHTML with GRDDL, the Deep Web [Ber01] must be annotated as well, where Deep Web refers to dynamic web pages generated from relational databases. He et al. determined in 2007 that Internet accessible databases contained up to 500 times more data compared to the static Web and three-quarters of these databases are managed by relational databases [HeP07]. Thus, the success of the Semantic Web hinges on developing methods for making relational databases accessible to the Semantic Web. The number of web sites further suggests that it is paramount to develop automatic methods for integrating relational databases with the Semantic Web.

Taking this objective into account, the content of a relational database will be made available in RDF, either in the form of results from an RDF query language like SPARQL or as RDF documents. This can be done by relying on the existence of an ontology (normally in RDF Schema or OWL) so that the RDF that is generated is related to an existing model or to a model that is created. However, this does not mean that the ontology that is referenced from the RDF triples generated from the RDB has to exist actually.

Broadly stated, there are two architectural approaches to creating such mapping.

- 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. [AnB05] [AnM06] [BaC04] [BaG06] [Biz04] [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 and its SQL schema are directly translated to representations in Semantic Web languages i.e. RDF and OWL, taking in account that the schema is in 3NF. [Ast04] [Astr07] [Buc04][LiD05][She06][Sto02]

A criticism of direct mapping is that the synthesized ontology will still require integration with some global domain ontology. While this is true, in this context, direct mapping, shifts the mapping problem from a relational database to ontology matching problem to an ontology to ontology matching problem. In isolation of RDBMSs successful development of the Semantic Web will see the development of successful systems that implement ontology-to-ontology mappings [Vra08]. That development combined with direct mapping will result in RDBMS to Ontology mapping.

An advantage of the ontology mapping approach is that existing domain ontologies can be widely reused, enabling standard representations of both basic concepts (e.g. units of measure) and complex relationships (e.g. doctor, patient). The construction of a wrapper may range from strictly procedural programming to sophisticated mapping languages. A downside of this approach is that the mapping between the relational schema and the existing ontology is a programming process. Although these systems often include effective mapping languages, such languages themselves usually exploit knowledge-base principles. In standard IT organizations the individual most knowledgeable about the contents of a database and its semantics will be the database administrator. Outside of research environments, database administrators are rarely familiar with even basic elements of knowledge-base systems. Without encapsulation of the mapping languages in graphics-user interfaces, enabling existing web sites onto the Semantic Web using this approach will likely be the province of specialized consultants.

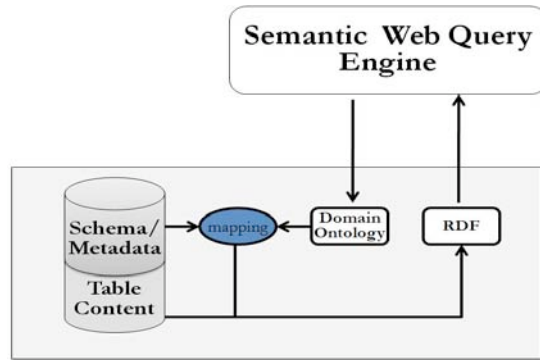


Figure 1. Relational Database to Ontology Mapping

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. In simplified form this often means that a database's domain semantics as encoded in its SQL-database schema are identified and translated into equivalent OWL expressions, (Web Ontological Language). Consequently, direct-mapping may include synthesizing an ontology from a database. The database contents can then be translated into consistent RDF (Resource Description Framework) triple representation.

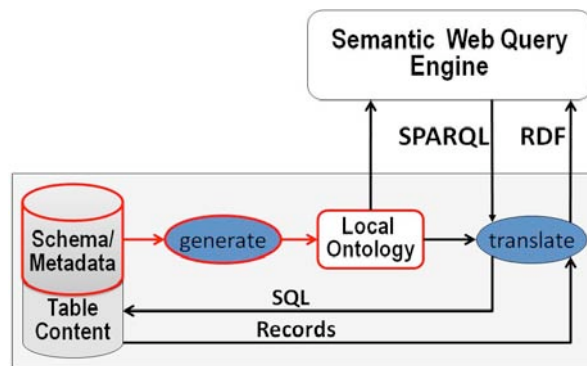


Figure 2. Direct Mapping of a Relational Database to the Semantic Web

A series of examples follow to demonstrate the capability of a SQL database to encode domain semantics using the SQL data definition language (SQL-DDL) and at the same time to expose downside of this approach. If we take a relational database schema, one can observe the implicit domain semantics that are represented in that schema. Take for example a University's relational database. The schema defines table structures concerning students, their enrollment, and courses being offered by a specific department. Figure 3 illustrates the database.

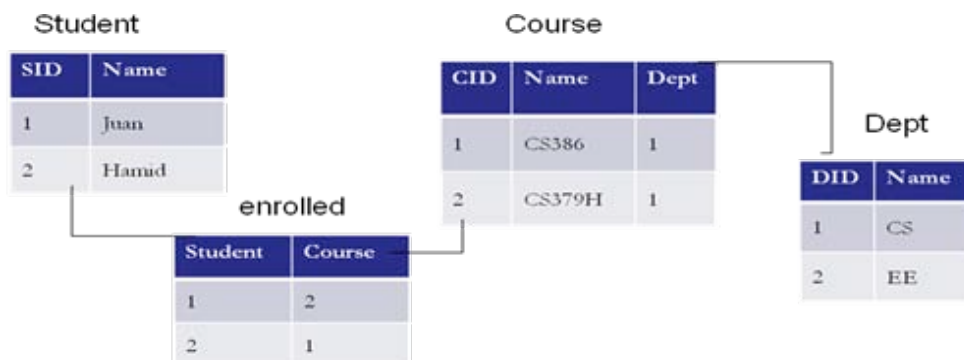


Figure 3. Direct Mapping of a Relational Database to the Semantic Web

By looking at the SQL schema, simple analogies can be observed between the relational schema and an ontology and possible transformations can be easily proposed. The central focus of direct mapping efforts to date concerns the semantic interpretation of the combinations of foreign keys. Any direct mapping system must anticipate every possible association that might be formed by a foreign key and the implication of every possible composition of those constraints. In section 4 we develop an abstract SQL grammar capable of generating all possible primary and foreign key combinations, of which there are 10. As we survey the work on direct mapping, we will use the coverage of these 10 cases, as one of features we use to assess the efforts.

For example, if we take a closer look at the relation between Course and Department, a foreign key between these two relations exist encoding a one-to-many relationship between the relations. For a person, it is understandable that this relationship means that a Course is offered by a Department. This exact knowledge can be represented in an ontology. Figure 4 shows the SQL-DDL of the relationship between Course and Dept through a foreign key, and its equivalent ontological representation.

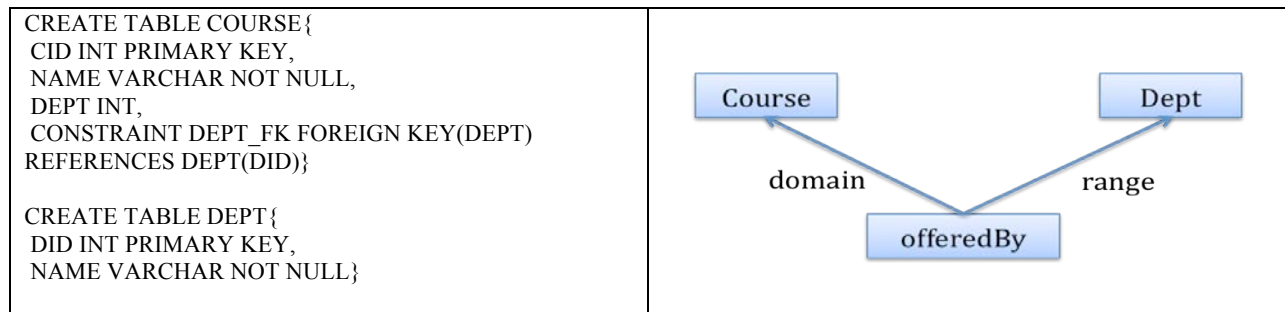


Figure 4. SQL-DDL and Ontology for Course and Dept

An ontology can have two classes, Course and Department, and a property, “offeredBy”, that relates the two classes, having as the domain the Course class and as a range the Department class. This is a simple example of how explicit semantics that are represented in an ontology are represented implicitly in a relational schema. As a result, some examples in a relational schema can have obvious transformations to an ontology. One of the key problems in this transformation is how to obtain the name of the relationship. Even though, in the SQL DDL, the relationship between Course and Dept is explicit, the name of this relationship is not. Therefore, it is necessary to have human involvement to name these relationships. The first approach to exploit domain semantics from SQL to RDFS, as shown in the previous example, was by Stojanovic et al [StS02].

However, just because in this example a transformation can be done does not mean that all examples of a syntactic form work the same way. One particular example is modeling inheritance. By looking at the content and structure of the database, it can be obvious that it maps to an inheritance relationship, but the syntactic structure does not always work the same. The syntactic constructs can be exactly the same for two different representations, but mean different things. Take for example representing inheritance between Student and Person using a Foreign Key as a subset of a Primary Key, as shown in Figure 5.

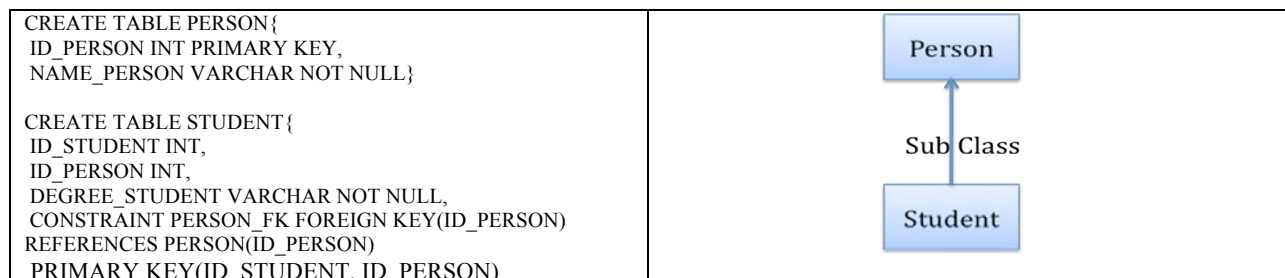


Figure 5. SQL-DDL and Ontology for Person and Student

In this case, the table Person becomes the class Person and the table Student becomes a subclass Person. Due to the fact that one of the primary keys of the table Student is also a foreign key to the table Person, one can say that this is an inheritance.

However, this same syntactic construct, which represented inheritance in the previous examples, models a different type of relationship, as shown in Figure 6. Therefore it is not correct to state that a specific syntactic construct represents inheritance.

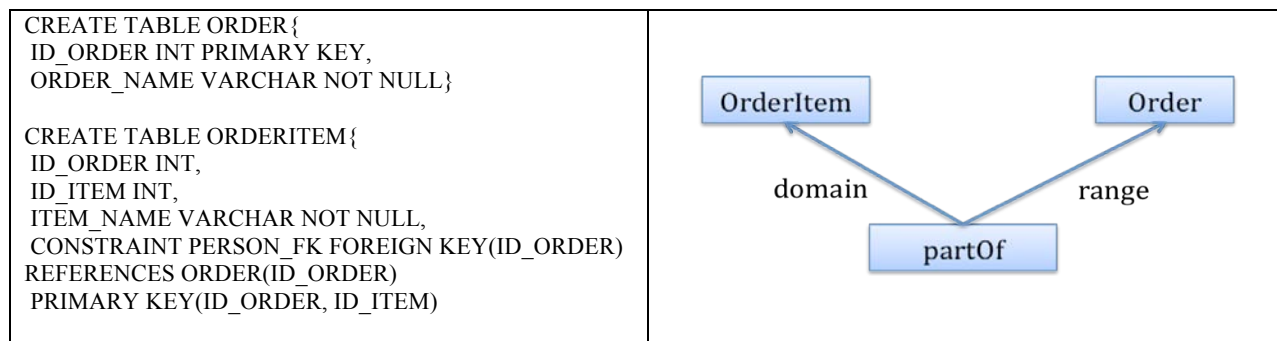


Figure 6. SQL-DDL and Ontology for Order and Order Item

The previous example models the relationship between an Order Item, which is part of an Order, which does not represent inheritance. Therefore, it is easy to make mistakes while thinking that some specific syntactic construct may represent a specific ontological construct, when indeed there is a counter example to it. Hence, not all explicit syntactic constructs may be correctly transformed.

Building on the evolution of SQL, Section 2 presents a mapping between SQL and the Semantic Web layer cake. In Section 3 we present transformation patterns and acknowledge other “hard examples” which include the identification of symmetric and transitive properties, the use of OWL’s someValuesFrom and allValuesFrom and the effects of Open and Closed World Assumptions. In Section 4 we present a theorem of completeness based on all the possible combination of primary keys and foreign keys. The survey of direct mapping approaches is presented in Section 5. Building on the surveyed approaches, we present a consolidated direct mapping system in Section 6, which contains all the transformation rules in First Order Logic (FOL). Section 7 is an analysis of all the surveyed direct mapping approaches. Section 8 describes future directions.

2. SQL-DDL and the Semantic Web Languages

This section discusses the evolution of SQL standards and the growth of the expressiveness of the language as it pertains to enable the encoding of domain semantics. It is arguable that the initial SQL language standard was not expressive enough to explicitly encode the database’s application semantics. However, the SQL language standard has changed significantly from its first release [Pra90, Pra95, Seq07].

Evident in the sequence of SQL standards are increasingly powerful elements for capturing application domain semantics. Direct mapping of a relational database to the Semantic Web is possible, particularly when one considers a relational schema is now defined using the SQL data definition language (SQL DDL) which is in at least third normal form (3NF) and includes explicit definition of consistency constraints, database views and triggers. Further, the inclusion of trigger definitions and view definitions in SQL was initially inspired by knowledge-base systems that capture and execute both for forward and backward rule systems [Sto86]. This is not to say that SQL-DDL is a knowledge representation language. The semantics of most of its constructs are not directly comparable with constructs of an ontology language. 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 grammatical structure.

Automatic methods to translate relational database content to the Semantic Web are possible by identifying correspondences between SQL-DDL syntactic constructs and elements of the Semantic Web. In this section we present the evolution of the SQL Data Models. We consider this a first step of understanding what parts of SQL-DDL can be translated to specific parts of the Semantic Web. Furthermore, we present a SQL layer cake and observed an association between it and the elements of the Semantic Web stack. These associations are direct mappings between SQL and the Semantic Web layer cake, which allows us to understand how SQL can encode semantics and how it can be translated into the different Semantic Web languages. Finally we present an example that uses these direct mapping transformations to create an OWL ontology from the relational database schema.

2.1 The Evolution of SQL Data Models

Prior to the creation of relational query languages, development of the relational model was largely concerned with the representation of data (Figure 7a) as n-ary relations and the correctness of syntactic transformations leveraging relational algebra [Mei83]. The central concern was the theory of normal forms. A primary goal was to reduce storage overhead. Similarly early SQL dialects had little concern with application specific semantics. Over the subsequent two decades, the SQL language has grown and relational database management systems have become increasingly sophisticated. An evolving sequence of standards, SQL 86-89, 92, 99, 2003, documents the progression [Pra90, Pra95, SQL3].

SQL is organized in three parts. The SQL data definition language (DDL), the SQL data manipulation language (DML) and the SQL data query language (DQL). The DDL is used to define both the logical and physical representation of data in a database. To date direct-mapping systems have been concerned with only the DDL. In this section, we highlight that as SQL has evolved; each new standard is a superset of the prior standard. Each such expansion enables a database administrator to express additional invariant details of the application domain, i.e. constraints, such that the database management system proactively maintains the semantic correctness of the contents of the database. We first provide an overview of the DDL features as they have evolved and show they parallel the expressive hierarchy of Semantic Web languages. Taking just some literary license, we can structure the chronological development of increasingly powerful SQL dialects as layers of expressiveness that correspond precisely to the Semantic Web layer cake.

The first SQL standard, SQL86-89 was limited. For the purpose of this paper, the following succinct description suffices; starting from relational algebra, 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 7b). 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 first extension already enables DBA's to encode application 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 7c.

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 7d is a trigger that maintains an invariant on the database concerning salary inversion. SQL 2003 introduces XML-related features. We find, so far, that the XML-related extensions enable only the encoding of syntactic properties specific to 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 7. Evolution of Relational DDL

2.2 SQL and the Semantic Web Layer Cake

Knowing the historical context, it is easy to decompose SQL-DDL into a stack representing increasing expressive power. While analyzing the evolution of SQL-DDL with respect to the Semantic Web stack, we suggest a correspondence between individual syntactic structures in SQL-DDL and the layers in the Semantic Web stack (Figure 8). A material result is a hierarchy of SQL-DDL grammars whose expressive power corresponds to the stack of Semantic Web languages. We assert that the layering of the Semantic Web is larger than just the RDF/RDFS/OWL hierarchy, but represents a generic representation (a la Newell and Simon) of the Semantic Web.

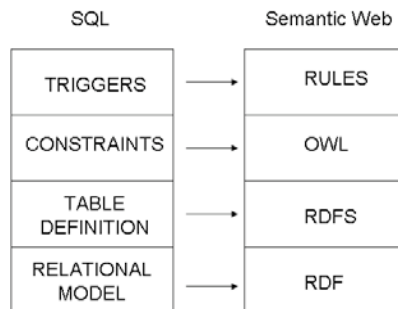


Figure 8. Layer Cake Mapping between SQL and the Semantic Web

It appears that the layers of that stack correspond to the layers of the Semantic Web stack (Figure 8). 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. 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.

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 3, thus demonstrating how a relational schema and ontology, of the same domain, are related. We continue to discuss incompatibilities between relational databases and ontologies.

2.2.1 Relational Model and RDF: Mapping Relational Tuples to 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.

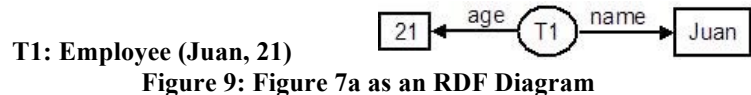


Figure 9: Figure 7a as an RDF Diagram

2.2.2 Table Definition and RDFS

The relational model offers sufficient semantics to create relations between the data. 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 be mapped only to the RDFS layer. RDFS can represent the table definition (Figure 7b) by having a class Employee with properties name and age. In this example, the column name “age” becomes and rdf:Property as shown in Figure 10.

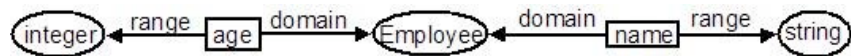


Figure 10. Figure 7b has an RDFS Diagram

2.2.3 Constraints and OWL

Climbing the Semantic Web layer cake, more expressive power is encountered. OWL offers expressiveness that can not be expressed 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). 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 7c, 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 11.

```
<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 11. “Age” attribute of Figure 1c in OWL format

2.2.4 Triggers and Rules

The latest version of SQL (SQL 2003) introduces a new semantic component: triggers. This new component can be considered as business rules that guarantee the integrity of data in a relational database and derive new information

materializing it. Even though these rules are not used for inference, they may expose rules about the domain. For example, the trigger in Figure 3d, explains that the maximum salary increase is of 20%. Triggers like the previous one may be analyzed to extract the rules and transform them to Semantic Web format. However, no direct mapping approach integrates triggers in the transformation process. The integration of triggers into direct mapping is still an open research question.

2.3 An example of a direct mapping transformation

SQL DDL is a language that defines structure to store data and not a language to encode knowledge. However, in the process of defining a relational schema, some domain semantics are introduced in the relationships between different entities of the relational databases. Hence, SQL DDL is capable of capture some semantics of the domain modeled in a SQL application. To further show that SQL DDL can encode semantics and that relational databases and ontologies have a correspondence, we show an example of how an ontology can be extracted out of the relational schema. Consider a relational database in 3NF for a university. SQL DDL for this database is given in Figure 12.

University Database Schema
<pre> 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) } </pre>

Figure 12. 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 that is in 3NF, and to identify the semantics of their properties and different kinds of relationships that exist between these concepts. Figure 13 shows an ontology corresponding to the given schema, developed by a domain expert.

Domain Expert's Ontology
<pre> 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)) ...) ...) </pre>

Figure 13. Parts of an ontology corresponding to the schema in Figure 12, 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.

It is apparent that an ontology does not correspond exactly to a relational databases. Ontologies can and must encode domain semantics that are not possible to be encode in a relational database, or even in SQL-DDL. Therefore, a translation of a relational schema to an ontology will suffer of ambiguity, and several disparities, which are explained in the following section.

3 Transformation Patterns 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, it is reasonable to expect some disparities among them in terms of basic assumptions.

It is important to understand the mismatches 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.

3.1 Basic Transformations

Several components of relational databases can be seen as obvious transformations to an ontology. In most cases, a relation is mapped to an ontological concept unless it is a binary relation. In this case, the binary relation would be mapped to a property that connects two concepts. A foreign key connects two relations, therefore, the foreign key is mapped to a property. All attributes that are not foreign keys are also mapped to properties. All direct mapping systems follow these basic transformation patterns. As a result, we believe that this overlap indicate that the majority of direct mapping approaches share rules that can be standardized.

Relational Database	RDFS Ontology	OWL Ontology
Non-binary Relation	RDFS Class	OWL Class
Binary Relation	RDFS Property	OWL Object Property
Column	RDFS Property	OWL Datatype Property
Foreign Key	RDFS Property	OWL Object Property

Table 1. Summary of Relational Database direct mapping to RDFS and OWL

3.2 Inheritance Modeling

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.

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. We present some inheritance modeling possibilities through possible foreign key patterns and discuss why some inheritance modeling choices are harder to identify automatically.

- 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.

- 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.

3.3 Symmetric and Transitive Relationships

SQL lack the expressive power to define symmetric and transitive relationships. Expressing such properties of relationships is natural to ontology languages like OWL.

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.

3.4 Value Constraints

The OWL language has a set of value constraints which puts constraints on the range of a property when applied to a particular class description. The three value constraints are allValuesFrom, someValuesFrom and hasValue. The OWL allValuesFrom is analogous to the universal quantifier of predicate logic while, the OWL someValuesFrom is analogous to the existential quantifier of predicate logic. Consider the following example:

Employee(id, id_role) Admin(id, id_role) Role(id_role,role_name)

Employee(id_role) and Admin(id_role) are foreign keys to Role(id_role). One can consider this example has having two different relationships: Employee has a role and an Admin has a role. Therefore, two Object Properties would be created. Nevertheless, if both object properties result in meaning the same thing, then it would be sufficient to have only one Object Property and have an owl:allValuesFrom with the union of the Employee and Admin table. However, it is unclear on how to determine if these two Object Properties are the same.

Furthermore, having different value constraints may lead to different interpretations. For example, if we consider an Open World Assumption, then a property should only a domain. If a property has a domain and a range, then we are considering a Closed World Assumption. We could tighten the closed world in a property by not only having a domain and range, but also having a allValuesFrom constraint.

3.5 Check Constraint

Check constraints are conditions that validate the data that is being inserted in a table. The enumerated CHECK constraint can be represented in OWL using owl:oneOf and rdf:List. For example, if we take attribute Session from the Semester table in Figure 12; its representation in OWL would be the following:

```
oneof('Spring' 'Summer' 'Fall')
```

Nevertheless, the check constraint could represent inheritance. In this case, a “Spring Session” is also a type of “Session”. On the other hand, OWL is not expressive enough to represent all the possibilities that the semantics of CHECK offers. Consider having a CHECK constraint to guarantee that the value of a column degree is between 0 and 360. This type of constraint cannot be represented in OWL.

3.6 Open World vs. Closed World

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.

4 Completeness of a Direct Mapping Transformation

Since we are inferring semantic properties formed by foreign keys, we have exhaustively considered all possible primary and foreign key combinations. This issue is complicated by the possible presence of compound keys (keys composed of multiple attributes) and the possibilities of multiple overlapping keys.

We present a notion of completeness of a SQL DDL to ontology transformation where 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.

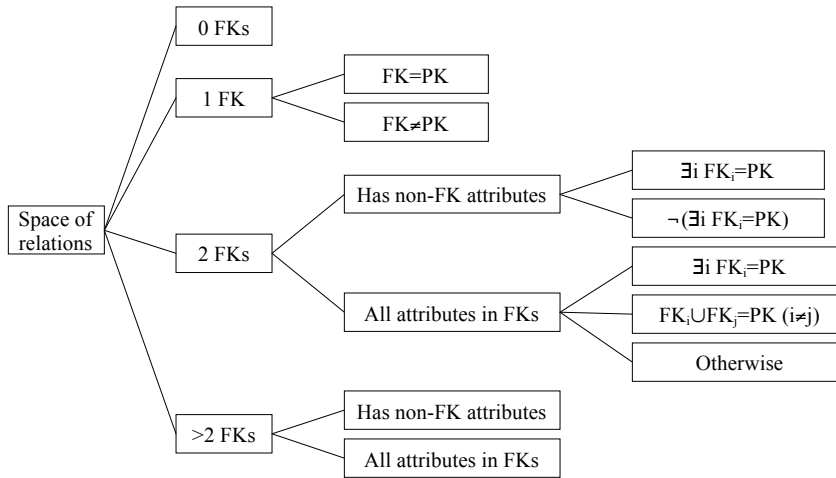


Figure14. The tree describes the complete space of relations when all possible combinations of primary and foreign keys are considered. For each branch, applicable

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. Figure14 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 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 a number of attributes, x , where $x = 0 \dots n$. 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 for LR(0) item sets, 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) $PK \cap S\text{-FK} = 0$: the Foreign Key and the Primary Key do not share any attributes
- 6. PK + N-FK: a relation has a Primary Key and two (2) Foreign Keys
 - a) $PK \cap N\text{-FK} = 0$: the Foreign Key and the Primary Key do not share any attributes
 - b) $PK \subset N\text{-FK}$: one of the Foreign Keys is also the Primary Key
- 7. PK + N-FK: a relation has a Primary Key and more than two (> 2) Foreign Keys
 - c) $PK \cap N\text{-FK} = 0$: the Foreign Key and the Primary Key do not share any attributes
 - d) $PK \subset N\text{-FK}$: one of the Foreign Keys is also the Primary Key
- 8. C-PK + S-FK: a relation has a Composite Primary Key and only one Foreign Key.
 - a) $C\text{-PK} \cap S\text{-FK} = 0$: the Foreign Key and the Primary Key do not share any attributes
 - b) $S\text{-FK} \subset C\text{-PK}$: the Foreign Key is part of the Primary Key
- 9. C-PK + N-FK: a relation has a Composite Primary Key and two (2) Foreign Keys
 - a) $C\text{-PK} \cap N\text{-FK} = 0$: all the Foreign Keys and the Primary Key do not share any attributes
 - b) $N\text{-FK} \subseteq C\text{-PK}$: all the Foreign Keys are part of the Primary Key
 - c) $C\text{-PK} \cap N\text{-FK} \neq 0, C\text{-PK} - N\text{-FK} \neq 0, N\text{-FK} - C\text{-PK} \neq 0$: The Foreign Keys and Primary Key share common attributes
- 10. C-PK + N-FK: a relation has a Composite Primary Key and more than two (> 2) Foreign Keys
 - d) $C\text{-PK} \cap N\text{-FK} = 0$: all the Foreign Keys and the Primary Key do not share any attributes
 - e) $N\text{-FK} \subseteq C\text{-PK}$: all the Foreign Keys are part of the Primary Key
 - f) $C\text{-PK} \cap N\text{-FK} \neq 0, C\text{-PK} - N\text{-FK} \neq 0, N\text{-FK} - C\text{-PK} \neq 0$: 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.

5 Direct Mapping Approaches

As we survey the different Direct Mapping approaches, we characterize and compare them utilizing the suggested correspondence between SQL and the Semantic Web. Furthermore, in the previous section we have presented a notion of completeness of transformation based on all the possible combinations of primary and foreign keys, which exemplifies all the possible combinations that are needed to be taken in consideration for a complete SQL to Semantic Web transformation system. This will be the criteria to evaluate the completeness of each transformation system and recognize if each system exploits all the possible primary and foreign key combinations. This evaluation will be presented in Section 7. Some surveyed approaches are not formally defined relying heavily on examples and written descriptions of the transformations; very much like the introduction of this paper. As a result, determining if the transformation system is complete is sometimes difficult, at best. We acknowledge correct correspondences and recognize the overlap between the surveyed systems. In addition, we offer counter-examples to any specific rule that may be erroneous or ambiguous. In Table 7 we present all the surveyed approaches and their outputs given a SQL-DDL construct.

It is important to note that 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. After creating an ontology from the database schema, the data in the relational database can be extracted and transformed to ontological instances. All the following surveyed approaches accomplish this objective.

The solution fails when the relational database is updated. Upon an update, 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.

	Stojanovic et al	Astrova et al (1)	Buccella et al.	Li et al.	Shen et al.	Astrova et al (2)	Consolidated System
Ontology Language	RDFS / F-Logic	RDFS / F-Logic	OWL	OWL	OWL	OWL Full	OWL DL
<table name>	RDFS Class RDFS Subclass RDFS Property RDFS Domain RDFS Range	RDFS Class RDFS Subclass RDFS Property RDFS Domain RDFS Range	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	OWL Class RDFS Subclass OWL ObjectProperty RDFS Domain RDFS Range	OWL Class RDFS Subclass OWL ObjectProperty RDFS Domain RDFS Range
<column name>	RDFS Property RDFS Domain	RDFS Property RDFS Domain	OWL Datatype Property OWL Functional Property OWL allValuesFrom	OWL Datatype Property OWL Object Property RDFS Domain	OWL Datatype Property OWL Object Property RDFS Domain	OWL Datatype Property OWL Object Property RDFS Domain	OWL Datatype Property OWL Object Property OWL Functional Property RDFS Domain
<data type>	RDFS Range	RDFS Range	OWL Functional Property OWL AllValuesFrom XML Schema Datatypes	RDFS range	RDFS range	OWL Maximum Cardinality of 1 RDFS range	OWL Maximum Cardinality of 1 OWL Functional Property RDFS range XML Schema Datatypes
<referential constraint definition>	RDFS Property	RDFS Property	OWL Object Property OWL Functional Property OWL InverseOf	OWL Object Property OWL Minimum Cardinality of 1 OWL Maximum Cardinality of 1	OWL Object Property OWL Minimum Cardinality of 1 OWL Maximum Cardinality of 1 OWL AllValuesFrom	OWL Object Property OWL Cardinality of 1 OWL Symmetric Property OWL Transitive Property	OWL Object Property OWL Functional Property OWL InverseOf OWL Minimum Cardinality of 0
<primary key definition>	F-Logic axioms	F-Logic axioms	OWL Cardinality of 1	OWL Minimum Cardinality of 1 OWL Maximum Cardinality of 1	OWL Minimum Cardinality of 1 OWL Maximum Cardinality of 1	OWL Inverse Functional Property OWL Minimum Cardinality of 1	OWL Cardinality of 1 OWL Functional Property

<unique constraint definition>	F-Logic axioms	F-Logic axioms		OWL Maximum Cardinality of 1	OWL Maximum Cardinality of 1	OWL Inverse Functional Property	OWL Inverse Functional Property
<not null constraint definition>	F-Logic axioms	F-Logic axioms	OWL Cardinality of 1	OWL Minimum Cardinality of 1	OWL Minimum Cardinality of 1	OWL Minimum Cardinality of 1	OWL Cardinality of 1
<check constraint definition>						OWL Datatype Property OWL hasValue OWL oneOf	OWL Datatype Property OWL oneOf

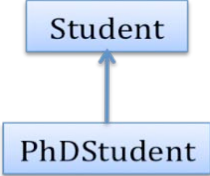
Table 2. Output of each direct mapping approach given the SQL input

Stojanovic et al.

Stojanovic et al. [Sto02] published the first effort to extract domain semantics from SQL-DDL and transform it to a Semantic Web representation. Their 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 F-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.

Stojanovic et al.’s work formally defines rules for identification of classes and properties in relational schemas; the target language is RDF Schema. Its mapping process consists of four steps.

First; capture information from a relational schema through reverse engineering (relations, attributes, attributes types, primary keys, and foreign keys/inclusion dependencies). Continue 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. Furthermore, 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 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 16 and 17 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.

<pre>create table STUDENT { STUDID integer primary key, NAME varchar} create table PHDSTUDENT { STUDID integer primary key, YEAR date not null }</pre>	
<p align="center">Figure 16. SQL-DDL representation of Student and PhD Student</p>	<p align="center">Figure 17. RDFS result of translating the SQL-DDL in Figure 16 with Stojanovic et al.’s rules and the SQL-DDL in Figure 17 with Astrova et al.’s rules</p>

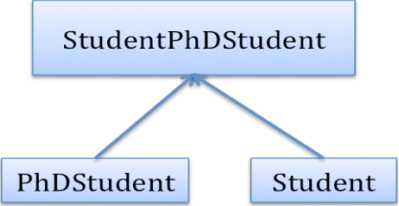
For relationships, a many-to-many relationship, one-to-many relationship (use of foreign keys), one-to-one relationship, and in the last case, a relational attribute is converted into a relationship. If n-ary relationships exist, these must be transformed into further concepts and a set of binary relations. 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 (1).

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).

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.

Relations can be mapped by a combination of key, data and attribute correlation. Analyzing the different foreign and primary key combinations of attributes lead to determine if a relation gets mapped to a concepts, properties or relationships. When the key of the relation and its data are equal across two different relations, but the attributes differ (Key and Data equality and Attribute disjointedness), the relationship expresses vertical partitioning and both relations become one concept. However, when the key and attributes are equal across two different relations but the data is different (Key and Attribute equality and Data disjointedness), the relationship expresses horizontal partitioning and both relations become one concept. Furthermore, when the keys between two relations are equal and the data of one relation is included in the other relation (Key equality and Data inclusion), the relationship between both relations expresses single inheritance. For example, the SQL-DDL in Figure 18 is would be transformed to the RDFS in Figure 17. If the keys are equal across two relations and there is data overlap and data inclusion between both relations (Key equality, Data overlap and Data inclusion), a new concept is discovered which would express multiple inheritance. Figure 16 is an example of the SQL-DDL that represents key equality, data overlap and data inclusion and it would be transformed to the RDFS in Figure 19. As it can be seen, Figure 19 creates a class that is not necessary, therefore this rule may lead into ambiguity. Moreover, a new concept is also discovered if the keys are equal, the data is disjoint and the attributes overlap between two relations. In addition, equal keys, data and attributes between relations expresses a diamond-shape inheritance.

<pre>create table STUDENT { STUDID integer primary key, NAME varchar} create table PHDSTUDENT { STUDID integer primary key references STUDENT, YEAR date not null }</pre>	 <pre> classDiagram StudentPhDStudent < -- PhDStudent StudentPhDStudent < -- Student </pre>
<p>Figure 18. Another SQL-DDL representation of Student and PhD Student.</p>	<p>Figure 19. RDFS result of translating the SQL-DDL in Figure 17 with Astrova et al.'s rules</p>

In summary, we have explained the two different approaches to transform SQL-DDL to RDFS. These approaches are very similar because given the same SQL-DDL input; the resulting RDFS is the same. However there are a couple of mismatches that have been explained in the previous 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 because they are not formalized transformation systems.

Buccella et al.

Buccella et al published the first approach that takes SQL-DDL and transforms it to OWL. The transformation system presented is not formalized and based on some expository examples. This approach analyzes five cases: tables without constraint, tables with exactly one foreign key, tables with exactly two foreign keys, tables with more than two foreign keys, and check constraints.

The first case translates tables without constraints are mapped to an OWL Class. All the attributes of the table are mapped to OWL Functional Datatype Property. If the attribute is NOT NULL, it is mapped to a cardinality constraint of 1. Primary Keys already have the NOT NULL implicit, therefore they are also mapped to a cardinality constraint of 1. The Datatype properties do not have domain and range defined because they can be used by other

classes. Furthermore, the restriction owl:allValuesFrom is applied with a String. This approach would lead to ambiguity. Take the following example in Figure 20.

<pre>create table PERSON { ID integer primary key, NAME varchar not null }</pre>	<pre>create table DEPT { CODE varchar primary key, NAME varchar unique not null }</pre>
--------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------

Figure 20. SQL-DDL of Person table and Dept table

Following this rule, when a Datatype Property “hasName” has been created for the Class Person, it would not have the domain Person and Range string, because another class could reuse it. The table DEPT also has an attribute name, therefore the previous Datatype Property created from the table Person could be reused for the new class Dept. Because it is not explicit that the class Person and class Dept are disjoint, it may be inferred that a name of a Dept is also a name of a Person which is false. Hence, due to the Open World Assumption, this rule is ambiguous.

The next case is when a table has exactly one foreign key. This table is then mapped to an OWL Class. The non foreign key attributes are mapped following the previous rules. There are now two sub cases: 1) if the foreign key is a subset of the primary key and 2) if the foreign key is not a subset of the primary key. For the first sub case, the primary key that is not a foreign key is mapped to a Functional Datatype property with cardinality of 1 and with the owl:allValuesFrom restriction, with the attributes datatype. The foreign key is mapped to a Functional Object Property. This object property does have a domain and range which is the current table as the domain, and the relationship of the foreign key as the range. The cardinality of 1 is also added and the Inverse Object Property is also created. This rule is one of the incompatibilities discussed in section 8.1 (Inheritance Modeling). For the second sub case, the foreign key is mapped to a Functional Object Property with minimum cardinality of 1, the domain is the current table and the range is the foreign key table. The Inverse property is also created which is not functional.

The third case is when a table has exactly two foreign keys. There are two sub cases: 1) it does not have any additional attributes or 2) it does have additional attributes. For the first sub case, the table becomes an object property with domain and range being the foreign key tables respectively. The second sub case would map the table to an OWL Class and map the attributes to properties by following the respective rule.

If a table has more than two foreign keys, it is mapped to an OWL class and all the attributes are also mapped to Datatype properties by following the respective rule.

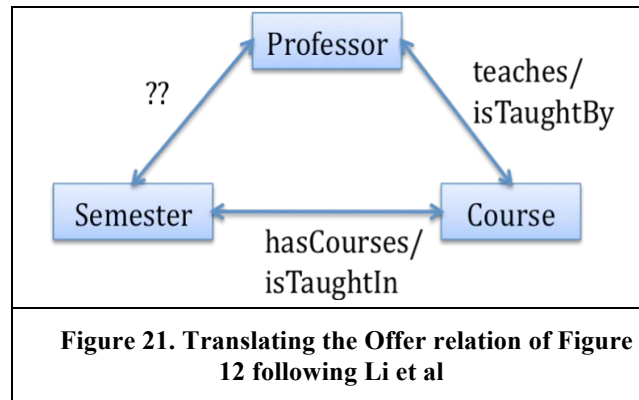
Finally the CHECK constraint is considered. If check constraint has LIKE A%, a class “beginning_with_A” can be created and that attribute can have a restriction that all values from that property have to be from that new class. Another possibility is to have the enumerated check constraints which is then mapped to owl:oneOf

Li et al.

Li et al’s approach is the first one to transform SQL-DDL to OWL with a combination of some formal notation and English language. The rules learn ontological classes, properties, hierarchy, cardinality and instances.

Li et al presents a rule to learn an OWL Class which is if several relations are used to describe one entity by sharing the same primary key, then they are all integrated into one ontological class. This rule is taken in account if vertical partitioning is considered in the relational schema. However, this same rule is used to learn inheritance. Thus, applying this rule is subject to human interaction.

A rule is presented which creates a “has-part” and “is-part-of” Object Property, if a relation has a foreign key as a primary key (Figure 5). Nevertheless, this rule can also be used to represent hierarchy (Figure 6). Therefore this rule can have a double meaning. Furthermore, if a 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. Taking this approach can produce unnecessary relationships. Consider the example with a 3-ary relation from Figure 12: Professor, Course and Semester under the relation Offer shown in Figure 21. This rule would create the relationships Professor-Course, Professor-Semester, Course-Semester. Is the relationship Professor-Semester needed? Maybe the other relationships were already created, and now it is repeated.



OWL 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.

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

Benslimane et al. presents an approach of acquiring an OWL ontology from data-intensive web sites by combining Li et al’s rule and analyzing the HTML page. They share the exact rules except for Li’s Rule 1 which is not included in Benslimane’s approach.

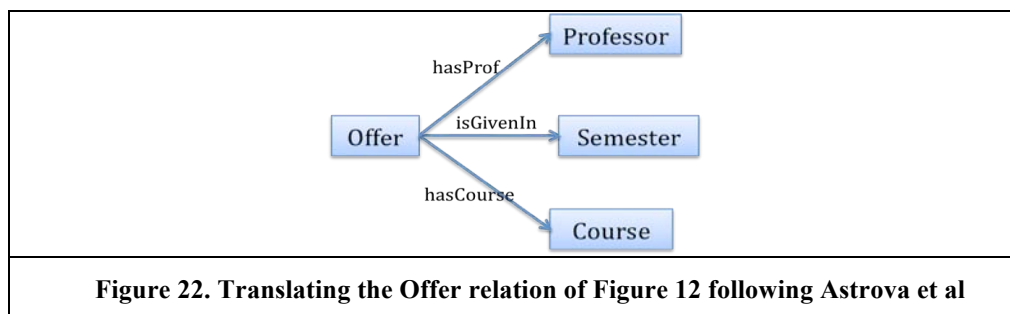
Shen et al.

Shen et al. follows Li et al’s learning rules with slight variations. Shen et al does not include Li et al’s rules of identifying a “has-part” relationship. Furthermore, it only includes a rule of identifying 3-ary relationships. However it adds the use of OWL allValuesFrom restriction. This is possible if a foreign key is mapped to an OWL Object Property, then the object property has an OWL allValuesFrom restriction, which refers to the corresponding inclusion dependency. This may lead to a disparity discussed previously in Section 3.4.

Astrova et al (2).

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 their previous effort [Ast04]. 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.

A main difference between Astrova et al. and Li et al’s rules is the mapping of n-ary relationships. If the table is composed of foreign keys to three other tables (3-ary relationship), this table is mapped to a class, instead of creating object properties between the relationships, as shown in Figure22.



Columns become OWL Datatype properties with a maximum cardinality of 1 unless it is a foreign key. The property can also be defined as functional, which is the same as saying that the maximum cardinality is 1.

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`.

SQL constraints are mapped as follows. The attribute with a UNIQUE constraint is mapped to an inverse functional property. This rule can not be true if the target ontology is OWL DL because Inverse Functional Properties can only be applied to Object Properties and not Data Type Properties [OWLGuide]. 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]. If the ontology is OWL Full, then it can be accepted.

Attributes with a NOT NULL constraint are mapped to a minimum cardinality of 1. PRIMARY KEY constraints are mapped to OWL Inverse Functional Property (which encounters the same problem explained previously) and a minimum cardinality 1.

If the foreign key is part of the primary key, it is also mapped to an object property accompanied with its cardinality of 1. However, Li et al assigns this case by translating it into a “has-part” Object Property. This same model can be used for inheritance, as explained in section 8.1. Finally, by taking vertical partitioning in account, 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. This rule can not be always applied, which has been explained in Section 8.2. If a column in a table is a foreign key to the same table accompanied by ON DELETE CASCADE, the relationship consists of a whole part, where the part cannot exist without the whole; therefore it is a transitive property.

CHECK constraints can be mapped several ways. If a column has a CHECK constraint to a single same value for all instances, then it is mapped to OWL `hasValue` restriction. Even though this translation makes sense, this rule appears to be ambiguous because it depends on a specific example, which is presented in the following example.

```
CREATE TABLE Project{
type VARCHAR CHECK (type ='Software')}

<owl:hasValue rdf:datatype="xsd:string">Software</owl:hasValue>
```

If the CHECK constraint is with enumeration, then it can be mapped to an enumerated data type `owl:oneOf`

Astrova et al. identifies more semantics from a relational databases schema as 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.

6 A Consolidated System of Direct Mapping of a Relational Database to the Semantic Web

Building on the related work described in this paper, we define a consolidated system for automatic transformation of relational databases into OWL ontologies defined in first order logic (FOL) eliminating syntactic and semantic.

6.1 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].

6.2 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 a 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:

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

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.

6.3 Transformation Rules and Examples

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

6.3.1 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.

6.3.2 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 3 we present a list of commonly used SQL data types along with their corresponding XML Schema types. During transformation of data type properties, the SQL data types are transformed into the corresponding XML Schema types.

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

6.3.3 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:

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

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 4) from our relational schema.

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

Table 4. Classes identified from the relational schema by applying Rule Set 2

6.3.4 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:

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

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:

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

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

Each rule in Rule Set 4 states that a foreign key represents an object property from the entity containing the foreign key (domain) to the referenced entity (range). Since a foreign key can reference at most one record (instance) of the range, the object property is functional. This requires 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 the NOT NULL constraint. Like one-to-many relationships mentioned above, if NOT NULL is present, it 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.

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 5.

Object Properties
$ObjP(REG,STUDENT,SEMESTER), ObjP(REG',SEMESTER,STUDENT), Inv(REG,REG')$
$ObjP(IDI,STUDENT,PERSON), FP(IDI), FP(IDI'), Crd(IDI,STUDENT,1)$
$ObjP(CODE,COURSE,DEPT), FP(CODE), IFP(CODE'), Crd(CODE,COURSE,1), MinC(CODE',DEPT,0)$
$ObjP(CNO,OFFER,COURSE), FP(CNO), IFP(CNO'), MinC(CNO',COURSE,0)$
$ObjP(CONO,OFFER,OFFER), FP(CONO), IFP(CONO'), MinC(CONO',OFFER,0)$
$ObjP(RNO,STUDY,STUDENT), FP(RNO), IFP(RNO'), MinC(RNO',STUDENT,0)$

Table 5. 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.

6.3.5 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. $DTP(x,r,type(x)), FP(x) \leftarrow NonFK(x,r)$
- b. $DTP(x,r,type(x)), FP(x), Crd(x,r,1) \leftarrow NonFK(x,r) \quad NN(x,r)$
- c. $DTP(x,r,type(x) \cap list(x)), FP(x) \leftarrow NonFK(x,r) \quad Chk(x,r)$

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 6.

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

Table 6. Some data type properties identified from the relational schema by applying Rule Set 5.

6.3.6 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:

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

This rule states that an entity represented by a relation r is a subclass of an entity represented by relation s , if the primary key of r is a foreign key to s . In our sample university schema, we can clearly identify that $Subclass(PROFESSOR, PERSON)$ holds.

6.4 Implementation

As a result of applying our rules on the given relational schema, we get the ontology shown in Table 7.

A comparison of the ontologies produced by the domain expert with the one produced automatically using our rules (Table) shows a number of differences. For example, the rules of the consolidated system 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.

As a result of presenting this consolidated system in FOL, these rules can be implemented in any rule engine. For sake of this example, we have implemented the rules in JESS. To obtain the SQL-DDL, we analyze only the data dictionary of MySQL databases. The ontology is then generated using the Jena framework.¹

¹ www.cs.utexas.edu/~jsequeda/sql2sw

Automatically Produced Ontology
<pre> 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> <u>Functional</u> domain(<OFFER>) range(<OFFER>)) ObjectProperty(<OFFER.CONO_I> <u>InverseFunctional</u> inverseOf(<OFFER.CONO>)) ObjectProperty(<STUDENT.ID> <u>Functional InverseFunctional</u> 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 <u>restriction(<STUDENT.ID> cardinality(1))</u> restriction(<STUDY.RNO_I> minCardinality(0)) ...) Class(<COURSE> partial restriction(<COURSE.DEPTCODE> cardinality(1)) restriction(<COURSE.CNO> cardinality(1)) ...) ...) </pre>

Table 7. 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.

7 A Comparative Evaluation of Direct Mapping Approaches

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

- 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 implemented in rule languages such as Prolog, Datalog or Jess.
- 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.
- The transformation system should infer semantic properties by considering all the possible combination of foreign and primary keys, hence being complete.

We have surveyed existing approaches of direct mapping relational databases to RDFS and OWL ontologies. Stojanovic et al and Astrova et al were the first approaches to consider extracting the domain semantics from the SQL-DDL and transforming them into RDFS. The majority of their rules overlap, meaning that given the same SQL-DDL input, it would output the same RDFS. However, we presented a difference between these two approaches on how they each identify inheritance. Stojanovic et al presents the transformation rules in a formal manner. As shown in Table 8, Stojanovic et al's rules are not complete in reference when a primary key is a subset of a foreign key; therefore this approach is not complete. On the other hand, Astrova et al presents the transformations through examples without any formality, which may lead to ambiguity; for that reason, the completeness of Astrova et al's approach cannot be determined.

Another series of surveyed direct mapping approaches transform the SQL-DDL to OWL ontologies. 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. Table 8 shows that there are Li et al's approach has rules to satisfy each case, and therefore is complete. Furthermore, Shen et al extends on Li et al's approach, which is also presented in a combination of formal notation and English language. However, Shen et al does not present rules to satisfy case 7 and 10, as seen in Table 8, hence this approach is not complete. On the other hand, Bucella et al and Astrova et al only present examples of transformation without any formalism at all, therefore it is ambiguous and there is no formal way to guarantee completeness, thus not showing up in Table 8. Building on the surveyed work, we also present a consolidated direct mapping system for automatic transformation of normalized relational schemas represented in SQL DDL into OWL ontologies, which builds upon the surveyed approaches. We have defined the entire set of transformation rules in first order logic eliminating the possibility of syntactic and semantic ambiguities and presenting a complete solution. The use of first order logic also allows for easy implementation of the system in rule-based languages. In defining the consolidated rules, we have ensured compatibility with description logics based OWL sublanguage (OWL DL), which is essential to assuring decidability for reasoning.

Relation with	Stojanovic <i>et al.</i> ²	Li <i>et al.</i> ³	Shen <i>et al.</i> ⁴	Consolidated System ⁵
1) PK	Rule C1, C2, C3	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 5
2) C-PK	Rule C1, C2, C3,	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 5
3) S-FK	Rule C1, C2, C3,	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
4) N-FK	Rule C1, C2, C3,	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
5) PK + S-FK				
a) PK = S-FK	Rule C1, C2, C3, A3, I1	Rule 2, 7, 8, 9, 10, 11	Rule C-1, C-3, P-1.2, R-2, R-3, R-4	Rule 2, 5, 6
b) $PK \cap S-FK = 0$	Rule C1, C2, C3, A2	Rule 2, 3, 7, 9, 10, 11	Rule C-1, P-1.1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
6) PK + N-FK N = 2				
a) $PK \cap N-FK = 0$	Rule C1, C2, C3, A1	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
b) $PK \subset N-FK$	Rule C1, C2, C3, A4	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5, 6
7) PK + N-FK N > 2				
a) $PK \cap N-FK = 0$	Rule C1, C2, C3, A5	Rule 2, 7, 9, 10, 11	-	Rule 2, 4, 5
b) $PK \subset N-FK$	-	Rule 2, 7, 9, 10, 11	-	Rule 2, 4, 5, 6
8) C-PK + S-FK				
a) $C-PK \cap S-FK =$	Rule C1, C2, C3, A2	Rule 2, 3, 7, 9, 10, 11	Rule C-1, P-1.1, P-1.2,	Rule 2, 5, 6

² The rule numbers correspond to the same rule numbers of [Sto06]

³ The rule numbers correspond to the same rule numbers of [LiD05]

⁴ The rule numbers correspond to the same rule numbers of [She06]

⁵ The rule numbers correspond to the same rule numbers of Section 6

0			R-2, R-3, R-4	
b) $S\text{-FK} \subset C\text{-PK}$:	-	Rule 2, 4, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
9) $C\text{-PK} + N\text{-FK} \mid N = 2$				
a) $C\text{-PK} \cap N\text{-FK} = 0$	Rule C1, C2, C3, A2	Rule 2, 3, 7, 9, 10, 11	Rule C-1, P-1.1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
b) $N\text{-FK} \subseteq C\text{-PK}$	Rule C1, C2, C3, A4	Rule 2, 5, 6, 7, 9, 10, 11	Rule C-1, P-2, P-1.2, R-2, R-3, R-4	Rule 2, 3, 4, 5
c) $C\text{-PK} \cap N\text{-FK} \neq 0$, $C\text{-PK} - N\text{-FK} \neq 0$, $N\text{-FK} - C\text{-PK} \neq 0$	-	Rule 2, 7, 9, 10, 11	Rule C-1, P-1.2, R-2, R-3, R-4	Rule 2, 4, 5
10) $C\text{-PK} + N\text{-FK} \mid N > 2$				
a) $C\text{-PK} \cap N\text{-FK} = 0$	Rule C1, C2, C3, A2	Rule 2, 3, 7, 9, 10, 11	-	Rule 2, 4, 5
b) $N\text{-FK} \subseteq C\text{-PK}$	Rule C1, C2, C3, A4	Rule 2, 5, 6, 7, 9, 10, 11	-	Rule 2, 3, 4, 5
c) $C\text{-PK} \cap N\text{-FK} \neq 0$, $C\text{-PK} - N\text{-FK} \neq 0$, $N\text{-FK} - C\text{-PK} \neq 0$	-	Rule 2, 7, 9, 10, 11	-	Rule 2, 4, 5

Table 8. Rules that apply to each of the Closure set item

In Table 9, we present the output of each direct mapping transformation system given the SQL-DDL input. As we can see, there is a major overlap in the output of all direct mapping systems as presented in Section 3.1. Stojanovic et al and Astrova et al (1) only output RDFS ontologies. Buccella et al and Shen et al are the only approaches that considers using owl:AllValuesFrom, however they are used differently. This leads us to acknowledge that identifying the use of owl:allValuesFrom and owl:someValuesFrom directly from a relational schema is still an open problem, as expressed in Section 3.4. In addition there is a difference in the usage of the cardinalities that are identified. Some approaches present the use of owl:maxCardinality and owl:minCardinality while others use only owl:cardinality.

In addition in Table 10, we also present the specific RDFS and OWL constructs that are created through each direct mapping approach. We can realize that Stojanovic et al and Astrova et al (1) output the same RDFS constructs. Li et al and Shen et al also produce the same OWL outputs, for the exception that Shen et al is able to use owl:allValuesFrom. Astrova et al (2) is the only approach able to output owl:SymmetricProperty, owl:TransitiveProperty and owl:hasValue. However, the rules to output these specific constructs are ambiguous and therefore not recommendable. Hence, we acknowledge that identifying symmetric and transitive properties from a relational schema is also an open problem.

	Stojanovic et al.	Astrova et al. (1)	Buccella et al.	Li et al.	Shen et al.	Astrova et al. (2)	Consolidated System
owl:class			x	x	x	x	x
rdfs:class	x	x					
owl:allValuesFrom			x		x		
owl:someValuesFrom							
owl:hasValue						x	
Cardinality constraint							
owl:maxCardinality			x	x	x	x	x

owl:minCardinality			X	X	X	X	X
owl:cardinality			X				X
owl:intersectionOf							
owl:unionOf				X	X		
owl:complementOf							
rdfs:subClassOf	X	X		X	X	X	X
owl:equivalentClass							
owl:disjointWith							
owl:oneOf						X	X
owl:DatatypeProperty			X	X	X	X	X
owl:ObjectProperty			X	X	X	X	X
RDFS Properties							
rdfs:subPropertyOf							
rdfs:domain	X	X	X	X	X	X	X
rdfs:range	X	X	X	X	X	X	X
owl:equivalentProperty							
owl:inverseOf			X	X	X	X	X
owl:FunctionalProperty			X				X
owl:InverseFunctionalProperty						X	X
owl:TransitiveProperty						X	
owl:SymmetricProperty						X	

Table 10. RDFS/OWL constructs that are outputted by each direct mapping approach

8 Discussion and Future Directions

The objective of the Semantic Web is to create a web of data with well-defined meaning. Due to the fact that the majority of the data on the web resides in SQL databases, the success of the Semantic Web hinges on offering efficient ways of integrating relational databases with the semantic web.

It is rare that integrating a database with the Semantic Web accrues obvious benefit yet to the owners of a database. Furthermore, for common Internet users, Semantic Web data or Linked Data, does not have great significance yet. Nevertheless, in short time, Linked Data will be more accessible for the common Internet users and at the same time they will realize the advantages of having Linked Data. Therefore database owners are going to want to be part of the Semantic Web and get their relational database content on it. However, at this moment, 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.

In this paper we have surveyed direct mapping approaches which try to bridge the gap between relational databases and the Semantic Web in an easy and automatic way. As a result of the direct mapping, an ontology is derived from the relational database schema. Conversely, the expressiveness of this ontology depends on the structure of the relational database schema. If the schema has been developed with sophisticated tools and in a normalized manner, the schema can portray enough semantics to create a semantically rich ontology. If this is the case, then direct mapping is the easiest way to create RDF from relational database data, by virtue of being completely automated. Therefore, if only a small portion of the SQL databases on the web are created with a rich schema, then these can easily benefit of direct mapping, by easily implementing a software package on top of their database software.

On the other hand, the amount of domain semantics captured in SQL-DDL models is highly variable and occasionally does not contain rich structure. If these types of models are used in a direct mapping system, the ontology that is derived will lack rich semantics. Therefore, we call the output of a direct mapping transformation system a database derived *“putative ontology”*. The putative ontology represents the basic implicit domain semantics of a relational database, which is obtained by applying direct mapping rules. Even though the putative ontology is not semantically equivalent to a domain ontology, it is not semantically incorrect, hence putative. The

deficiencies of a putative ontology become evident when an ontology produced by the system is compared to an ontology produced by a domain expert.

In Section 3 we have exposed patterns between relational databases and ontologies, and the deficiencies when it comes to identifying richer semantics in a relational database schema. As a result, we outline several open research. Firstly we acknowledge that inheritance in a relational database schema can be modeled in different ways; therefore a single rule to identify inheritance is not possible. In addition, symmetric and transitive relationships may be implicit the domain but are not explicit in the relational schema. Furthermore, identifying specific value constraints such as “for all” or “there exist” may imply identifying new properties or reusing existing properties. Continuously, it may also have implications in the open and closed world assumptions. Additionally, we admit that SQL’s check constraint can represent different types of semantics such as enumeration, inheritance or business type rules. Finally, we recognize that analyzing triggers to extract semantics applicable to ontologies is still an open problem. Overall, there is also a need to evaluate the quality of a putative ontology and a domain ontology.

We assert that the scope and frequency of success of direct mapping transformation systems is very unlikely to achieve the scope of description database-ontology wrapper mapping methods can achieve, however we consider direct mapping the first step in a two step process. As already mentioned, direct mapping converts the relational database to ontology mapping problem to an ontology to ontology mapping problem. Therefore, we consider that the next step in this area is to bridge the gap between putative and domain ontologies.

In order to bridge this gap, we propose to refine the putative ontology through a bootstrapping architecture [Seq08]. This refinement process consists of initially matching the putative ontology with an existing domain ontology. Due to the fact that the putative ontology lacks semantics and may have some ambiguity with respect to the domain ontology, we treat the alignment between the putative and domain ontology as an alignment hypothesis. Querying and analyzing the relational database data can test the alignment hypothesis. After the hypothesis has been tested, the new knowledge is used to refine the putative ontology. In this process, the putative ontology would be automatically matched to other domain ontologies. This would be an iteratively process different domain ontologies, until the putative ontology has been refined and it completely represents the domain semantics of the relational database.

9 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 Meaningful 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*.
- [Ber01] Bergman, M. (2001) The Deep Web: Surfacing Hidden Value. *Scholarly Publishing Office, University of Michigan University Library Journal of Electronic Publishing*.
- [Biz04] Bizer, C., & Seaborne, A. (2004). D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. *3rd International Semantic Web Conference*.
- [Buc04] Buccella, A., Penabaz, M. R., Rodríguez, F. J., Fariña, A., Cechich, A. From Relational Databases to OWL Ontologies. *Procs of the 6th Russian Conference on Digital Libraries*. Pushchino (Russia), 2004.
- [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*.
- [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.
- [Lab06] de Laborda, C. P., & Conrad, S. (2006). Database to Semantic Web Mapping using RDF Query Languages. *25th International Conference on Conceptual Mapping*.
- [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 Network and 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/>
- [Seq07] J.F. Sequeda, S.H. Tirmizi and D.P. Miranker. (2007) SQL Databases are a Moving Target. Position Paper for W3C Workshop on RDF Access to Relational Databases, October 2007.'
- [Seq08] J.F. Sequeda, S.H. Tirmizi and D.P. Miranker. (2008) A Bootstrapping Architecture for Integration of Relational Databases to the Semantic Web. In Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC08),
- [She06] Shen, G., Huang, Z., Zhu, X., Zhao, X.: Research on the Rules of Mapping from Relational Model to OWL. In: Proceedings of the Workshop on OWL: Experiences and Directions. Vol. 216 (2006)
- [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>
- [Sto86] Stonebraker, M. (1986). Triggers and inference in database systems. In on Knowledge Base Management Systems: integrating Artificial intelligence and Database Technologies
- [Sto02] Stojanovic, L., Stojanovic, N., & Volz, R. (2002). Migrating data-intensive web sites into the Semantic Web. *ACM Symposium on Applied computing*.
- [StS02] Stojanovic, L., Stojanovic, N., & Volz, R. (2002). A reverse engineering approach for migrating data-intensive web sites to the Semantic Web. *Proceedings of the IFIP 17th World Computer Congress*.
- [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*.
- [Vra08] Vrandecic, D., & Sure, Y. (2008). How to Design Better Ontology Metrics
- [Wan00] Wang, S., Shen, J., & Hong, T. (2000). Mining fuzzy functional dependencies from quantitative data. *IEEE International Conference on Systems, Man and Cybernetics*, 5.
- [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/>
- [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*.