On Directly Mapping Relational Databases to RDF and OWL

Juan F. Sequeda
University of Texas at Austin
jsequeda@cs.utexas.edu

Marcelo Arenas
PUC Chile
marenas@ing.puc.cl

Daniel P. Miranker
University of Texas at Austin
miranker@cs.utexas.edu

ABSTRACT

Mapping relational databases to RDF is a fundamental problem for the development of the Semantic Web. We present a solution, inspired by draft methods defined by the W3C where relational databases are directly mapped to RDF and OWL. Given a relational database schema and its integrity constraints, this direct mapping produces an OWL ontology, which, provides the basis for generating RDF instances. The semantics of this mapping is defined using Datalog. Two fundamental properties are information preservation and query preservation. We prove that our mapping satisfies both conditions, even for relational databases that contain null values. We also consider two desirable properties: monotonicity and semantics preservation. We prove that our mapping is monotone and also prove that no monotone mapping, including ours, is semantic preserving. We realize that monotonicity is an obstacle for semantic preservation and thus present a non-monotone direct mapping that is semantics preserving.

Categories and Subject Descriptors

H.2.5 [**Heterogeneous Databases**]: Data translation; H.3.5 [**Online Information Services**]: Web-based services

Keywords

Relational Databases, Semantic Web, Direct Mapping, RDB2RDF, SQL, SPARQL, RDF, OWL

1. INTRODUCTION

In this paper, we study the problem of directly mapping a relational database to an RDF graph with OWL vocabulary. A direct mapping is a default and automatic way of translating a relational database to RDF. One report suggests that Internet accessible databases contained up to 500 times more data compared to the static Web and roughly 70% of websites are backed by relational databases, making automatic translation of relational database to RDF central to the success of the Semantic Web [13].

We build on an existing direct mapping of relational database schema to OWL DL [23] and the current draft of the W3C Direct Mapping standard [5]. We study two properties that are fundamental to a direct mapping: information preservation and query preservation. Additionally we study two desirable properties: monotonicity and semantics preservation. To the best of our knowledge, we are presenting the first direct mapping from a relational database to an RDF graph with OWL vocabulary that has been thoroughly studied with respect to these fundamental and desirable properties.

Information preservation speaks to the ability of reconstructing the original database from the result of the direct mapping. Query preservation means that every query over a relational database can be translated into an equivalent query over the result of the direct mapping. Monotonicity is a desired property because it assures that a re-computation of the entire mapping is not needed after any updates to the database. Finally, a direct mapping is semantics preserving if the satisfaction of a set of integrity constraints are encoded in the mapping result.

Our proposed direct mapping is monotone, information preserving and query preserving even in the general and practical scenario where relational databases contain null values. However, given a database that violates an integrity constraint, our direct mapping generates a consistent RDF graph, hence, it is not semantics preserving.

We analyze why our direct mapping is not semantics preserving and realize that monotonicity is an obstacle. We first show that if we only consider primary keys, we can still have a monotone direct mapping that is semantics preserving. However this result is not sufficient because it dismisses foreign keys. Unfortunately, we prove that no monotone direct mapping is semantics preserving if foreign keys are considered, essentially because the only form of constraint checking in OWL is satisfiability testing. This result has an important implication in real world applications: if you migrate your relational database to the Semantic Web using a monotone direct mapping, be prepared to experience consistency when what one would expect is inconsistency.

Finally, we present a non-monotone direct mapping that overcomes the aforementioned limitation. We foresee the existence of monotone direct mappings if OWL is extended with the epistemic operator. Due to lack of space, the paper does not include the proofs of the results. We refer the reader to [19] for these proofs.

2. PRELIMINARIES

In this section, we define the basic terminology used in the paper.

2.1 Relational databases

Assume, a countably infinite domain $\mathbf{D}$ and a reserved symbol NULL that is not in $\mathbf{D}$. A *schema* $\mathbf{R}$ is a finite set of relation names, where for each $R \in \mathbf{R}$, $att(R)$ denotes the nonempty finite set of attributes names associated to R. An instance I of $\mathbf{R}$ assigns to each relation symbol $R \in \mathbf{R}$ a finite set $R^I = \{t_1, \ldots, t_\ell\}$ of tuples, where each tuple t_j ($1 \leq j \leq \ell$) is a function that assigns to each attribute in $att(R)$ a value from $(\mathbf{D} \cup \{\texttt{NULL}\})$. We use notation $t.A$ to refer to the value of a tuple t in an attribute A.

Relational algebra: To define some of the concept studied in this

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

paper, we use relational algebra as a query language for relational databases. Given that we consider relational databases containing null values, we present in detail the syntax and semantics of a version of relational algebra that formalizes the way nulls are treated in practice in database systems. Formally, assume that $\mathbf{R}$ is a relational schema. Then a relational algebra expression φ over $\mathbf{R}$ and its set of attributes $att(\varphi)$ are recursively defined as follows:

1. If $\varphi = R$ with $R \in \mathbf{R}$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(R)$.
2. If $\varphi = \texttt{NULL}_A$, where A is an attribute, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = \{A\}$.
3. If ψ is a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$, $a \in \mathbf{D}$ and φ is any of the expressions $\sigma_{A=a}(\psi)$, $\sigma_{A\neq a}(\psi)$, $\sigma_{\texttt{IsNull}(A)}(\psi)$ or $\sigma_{\texttt{IsNotNull}(A)}(\psi)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(\psi)$.
4. If ψ is a relational algebra expression over $\mathbf{R}$, $U \subseteq att(\psi)$ and $\varphi = \pi_U(\psi)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = U$.
5. If ψ is a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$, B is an attribute such that $B \notin att(\psi)$ and $\varphi = \delta_{A\to B}(\psi)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = (att(\psi) \setminus \{A\}) \cup \{B\}$.
6. If ψ_1, ψ_2 are relational algebra expressions over $\mathbf{R}$ and $\varphi = (\psi_1 \bowtie \psi_2)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = (att(\psi_1) \cup att(\psi_2))$.
7. If ψ_1, ψ_2 are relational algebra expressions over $\mathbf{R}$ such that $att(\psi_1) = att(\psi_2)$ and φ is either $(\psi_1 \cup \psi_2)$ or $(\psi_1 \setminus \psi_2)$, then φ is a relational algebra expression over $\mathbf{R}$ such that $att(\varphi) = att(\psi_1)$.

Let $\mathbf{R}$ be a relational schema, I an instance of $\mathbf{R}$ and φ a relational algebra expression over $\mathbf{R}$. The evaluation of φ over I, denoted by $[\![\varphi]\!]_I$, is defined recursively as follows:

1. If $\varphi = R$ with $R \in \mathbf{R}$, then $[\![\varphi]\!]_I = R^I$.
2. If $\varphi = \texttt{NULL}_A$, where A is an attribute, then $[\![\varphi]\!]_I = \{t\}$, where $t : \{A\} \to (\mathbf{D} \cup \{\texttt{NULL}\})$ is a tuple such that $t.A = \texttt{NULL}$.
3. Let ψ be a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$ and $a \in \mathbf{D}$. If $\varphi = \sigma_{A=a}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A = a\}$. If $\varphi = \sigma_{A\neq a}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A \neq \texttt{NULL}$ and $t.A \neq a\}$. If $\varphi = \sigma_{\texttt{IsNull}(A)}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A = \texttt{NULL}\}$. If $\varphi = \sigma_{\texttt{IsNotNull}(A)}(\psi)$, then $[\![\varphi]\!]_I = \{t \in [\![\psi]\!]_I \mid t.A \neq \texttt{NULL}\}$.
4. If ψ is a relational algebra expression over $\mathbf{R}$, $U \subseteq att(\psi)$ and $\varphi = \pi_U(\psi)$, then $[\![\varphi]\!]_I = \{t : U \to (\mathbf{D} \cup \{\texttt{NULL}\}) \mid$ there exists $t' \in [\![\psi]\!]_I$ such that for every $A \in U$: $t.A = t'.A\}$.
5. If ψ is a relational algebra expression over $\mathbf{R}$, $A \in att(\psi)$, B is an attribute such that $B \notin att(\psi)$ and $\varphi = \delta_{A\to B}(\psi)$, then $[\![\varphi]\!]_I = \{t : att(\varphi) \to (\mathbf{D} \cup \{\texttt{NULL}\}) \mid$ there exists $t' \in [\![\psi]\!]_I$ such that $t.B = t'.A$ and for every $C \in (att(\varphi) \setminus \{B\})$: $t.C = t'.C\}$.
6. If ψ_1, ψ_2 are relational algebra expressions over $\mathbf{R}$ and $\varphi = (\psi_1 \bowtie \psi_2)$, then $[\![\varphi]\!]_I = \{t : att(\varphi) \to (\mathbf{D} \cup \{\texttt{NULL}\}) \mid$ there exist $t_1 \in [\![\psi_1]\!]_I$ and $t_2 \in [\![\psi_2]\!]_I$ such that for every $A \in (att(\psi_1) \cap att(\psi_2))$: $t.A = t_1.A = t_2.A \neq \texttt{NULL}$, for every $A \in (att(\psi_1) \setminus att(\psi_2))$: $t.A = t_1.A$, and for every $A \in (att(\psi_2) \setminus att(\psi_1))$: $t.A = t_2.A\}$.
7. Let ψ_1, ψ_2 be relational algebra expressions over $\mathbf{R}$ such that $att(\psi_1) = att(\psi_2)$. If $\varphi = (\psi_1 \cup \psi_2)$, then $[\![\varphi]\!]_I = [\![\psi_1]\!]_I \cup [\![\psi_2]\!]_I$. If $\varphi = (\psi_1 \setminus \psi_2)$, then $[\![\varphi]\!]_I = [\![\psi_1]\!]_I \setminus [\![\psi_2]\!]_I$.

It is important to notice that the operators left-outer join, right-outer join and full-outer join are all expressible with the previous operators. For more details, we refer the reader to [19].

Integrity constraints: We consider two types of integrity constraints: keys and foreign keys. Let $\mathbf{R}$ be a relational schema. A key φ over $\mathbf{R}$ is an expression of the form $R[A_1, \ldots, A_m]$, where $R \in \mathbf{R}$ and $\emptyset \subsetneq \{A_1, \ldots, A_m\} \subseteq att(R)$. Given an instance I of $\mathbf{R}$, I satisfies key φ, denoted by $I \models \varphi$, if: (1) for every $t \in R^I$ and $k \in \{1, \ldots, m\}$, it holds that $t.A_k \neq \texttt{NULL}$, and (2) for every $t_1, t_2 \in R^I$, if $t_1.A_k = t_2.A_k$ for every $k \in \{1, \ldots, m\}$, then $t_1 = t_2$. A foreign key over $\mathbf{R}$ is an expression of the form $R[A_1, \ldots, A_m] \subseteq_{\text{FK}} S[B_1, \ldots, B_m]$, where $R, S \in \mathbf{R}$, $\emptyset \subsetneq \{A_1, \ldots, A_m\} \subseteq att(R)$ and $\emptyset \subsetneq \{B_1, \ldots, B_m\} \subseteq att(S)$. Given an instance I of $\mathbf{R}$, I satisfies foreign key φ, denoted by $I \models \varphi$, if $I \models S[B_1, \ldots, B_m]$ and for every tuple t in R^I: either (1) there exists $k \in \{1, \ldots, m\}$ such that $t.A_k = \texttt{NULL}$, or (2) there exists a tuple s in S^I such that $t.A_k = s.B_k$ for every $k \in \{1, \ldots, m\}$.

Given a relational schema $\mathbf{R}$, a set Σ of keys and foreign keys is said to be a *set of primary keys (PKs) and foreign keys (FKs) over* $\mathbf{R}$ if: (1) for every $\varphi \in \Sigma$, it holds that φ is either a key or a foreign key over $\mathbf{R}$, and (2) there are no two distinct keys in Σ of the form $R[A_1, \ldots, A_m]$ and $R[B_1, \ldots, B_n]$ (that is, that mention the same relation name R). Moreover, an instance I of $\mathbf{R}$ satisfies Σ, denoted by $I \models \Sigma$, if for every $\varphi \in \Sigma$, it holds that $I \models \varphi$.

2.2 RDF and OWL

Assume there are pairwise disjoint infinite sets $\mathbf{I}$ (IRIs), $\mathbf{B}$ (blank nodes) and $\mathbf{L}$ (literals). A tuple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object. A finite set of RDF triples is called an RDF graph. Moreover, assume the existence of an infinite set $\mathbf{V}$ of variables disjoint from the above sets, and assume that every element in $\mathbf{V}$ starts with the symbol ?.

In this paper, we consider RDF graphs with OWL vocabulary [1], which is the W3C standard ontology language based on description logics, without datatypes. In particular, we say that an RDF graph G is consistent under OWL semantics if a model of G with respect to the OWL vocabulary exists (see [1] for a precise definition of the notion of model and the semantics of OWL).

2.3 SPARQL

In this paper, we use SPARQL as a query language for RDF graphs. The official syntax of SPARQL [17, 12] considers operators `OPTIONAL`, `UNION`, `FILTER`, `SELECT`, `AS` and concatenation via a point symbol (`.`), to construct graph pattern expressions. The syntax of the language also considers `{ }` to group patterns, and some implicit rules of precedence and association. In order to avoid ambiguities in the parsing, we follow the approach proposed in [16], and we present the syntax of SPARQL graph patterns in a more traditional algebraic formalism, using operators AND (`.`), UNION (`UNION`), OPT (`OPTIONAL`), MINUS (`MINUS`), FILTER (`FILTER`), SELECT (`SELECT`) and AS (`AS`). More precisely, a SPARQL graph pattern expression is defined recursively as follows.

1. { } is a graph pattern (the empty graph pattern).
2. A tuple from $(\mathbf{I} \cup \mathbf{L} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ is a graph pattern (a triple pattern).
3. If P_1 and P_2 are graph patterns, then expressions $(P_1$ AND $P_2)$, $(P_1$ OPT $P_2)$, $(P_1$ UNION $P_2)$ and $(P_1$ MINUS $P_2)$ are graph patterns.

4. If P is a graph pattern and R is a SPARQL built-in condition, then the expression $(P \text{ FILTER } R)$ is a graph pattern.
5. If P is a graph pattern and $?A_1, \ldots, ?A_m, ?B_1, \ldots, ?B_m, ?C_1, \ldots, ?C_n$ is a sequence of pairwise distinct elements from $\mathbf{V}$ ($m \geq 0$ and $n \geq 0$) such that none of the variables $?B_i$ ($1 \leq i \leq m$) is mentioned in P, then

   $$(\text{SELECT } \{?A_1 \text{ AS } ?B_1, \ldots, ?A_m \text{ AS } ?B_m, ?C_1, \ldots, ?C_n\} \ P)$$

   is a graph pattern.

A SPARQL built-in condition is constructed using elements of the set $(\mathbf{I} \cup \mathbf{V})$ and constants, logical connectives ($\neg, \wedge, \vee$), inequality symbols ($<, \leq, \geq, >$), the equality symbol ($=$), unary predicates such as bound, isBlank, and isIRI (see [17, 12] for a complete list). In this paper, we restrict to the fragment where the built-in condition is a Boolean combination of terms constructed by using $=$ and bound, that is: (1) if $?X, ?Y \in \mathbf{V}$ and $c \in \mathbf{I}$, then $\text{bound}(?X)$, $?X = c$ and $?X = ?Y$ are built-in conditions, and (2) if R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$ and $(R_1 \wedge R_2)$ are built-in conditions.

The version of SPARQL used in this paper includes the following SPARQL 1.1 features: the operator MINUS, the possibility of nesting the SELECT operator and the operator AS [12].

The answer of a SPARQL query P over an RDF graph G is a finite set of *mappings*, where a mapping μ is a partial function from the set $\mathbf{V}$ of variables to $(\mathbf{I} \cup \mathbf{L} \cup \mathbf{B})$. We define the semantics of SPARQL as a function $[\![\cdot]\!]_G$ that, given an RDF graph G, takes a graph pattern expression and returns a set of mappings. We refer the reader to [19] for more detail.

3. DIRECT MAPPINGS: DEFINITION AND PROPERTIES

A direct mapping is a default way to translate relational databases into RDF (without any input from the user on how the relational data should be translated). The input of a direct mapping $\mathcal{M}$ is a relational schema $\mathbf{R}$, a set Σ of PKs and FKs over $\mathbf{R}$ and an instance I of $\mathbf{R}$. The output is an RDF graph with OWL vocabulary.

Assume $\mathcal{G}$ is the set of all RDF graphs and $\mathcal{RC}$ is the set of all triples of the form $(\mathbf{R}, \Sigma, I)$ such that $\mathbf{R}$ is a relational schema, Σ is a set of PKs and FKs over $\mathbf{R}$ and I is an instance of $\mathbf{R}$.

Definition 1 (Direct mapping) *A direct mapping $\mathcal{M}$ is a total function from $\mathcal{RC}$ to $\mathcal{G}$.*

We now introduce two fundamental properties of direct mappings: information preservation and query preservation; and two desirable properties of these mappings: monotonicity and semantic preservation. Information preservation is a fundamental property because it guarantees that the mapping does not lose information, which is fundamental in an Extract-Transform-Load process. Query preservation is also a fundamental property because it guarantees that everything that can be extracted from the relational database by a relational algebra query, can also be extracted from the resulting RDF graph by a SPARQL query. This property is fundamental for workloads that involve translating SPARQL to SQL. Monotonicity is a desirable property because it would avoid recalculating the mapping for the entire database after inserting new data. In addition to practical considerations when translating relational data to RDF graphs, we must deal with the closed-world database semantics and open world RDF/OWL semantics. Understanding the expressive power of a mapping and, its ability to properly deal with integrity constraints is important. Thus our choice of examining semantics preservation.

3.1 Fundamental properties

Information preservation: A direct mapping is information preserving if it does not lose any information about the relational instance being translated, that is, if there exists a way to recover the original database instance from the RDF graph resulting from the translation process. Formally, assuming that $\mathcal{I}$ is the set of all possible relational instances, we have that:

Definition 2 (Information preservation) *A direct mapping $\mathcal{M}$ is* information preserving *if there is a computable mapping $\mathcal{N} : \mathcal{G} \to \mathcal{I}$ such that for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$, and instance I of $\mathbf{R}$ satisfying Σ: $\mathcal{N}(\mathcal{M}(\mathbf{R}, \Sigma, I)) = I$.*

Recall that a mapping $\mathcal{N} : \mathcal{G} \to \mathcal{I}$ is computable if there exists an algorithm that, given $G \in \mathcal{G}$, computes $\mathcal{N}(G)$.

Query preservation: A direct mapping is query preserving if every query over a relational database can be translated into an equivalent query over the RDF graph resulting from the mapping. That is, query preservation ensures that every relational query can be evaluated using the mapped RDF data.

To formally define query preservation, we focus on relational queries that can be expressed in relational algebra [3] and RDF queries that can be expressed in SPARQL [17, 16]. In Section 2.1, we introduced a version of relational algebra that formalizes the semantics of null values in practice. In Section 2.3, we introduce an algebraic version of SPARQL that follows the approach proposed in [16]. Given the mismatch in the formats of these query languages, we introduce a function *tr* that converts tuples returned by relational algebra queries into mappings returned by SPARQL. Formally, given a relational schema $\mathbf{R}$, a relation name $R \in \mathbf{R}$, an instance I of $\mathbf{R}$ and a tuple $t \in R^I$, define $tr(t)$ as the mapping μ such that: (1) the domain of μ is $\{?A \mid A \in att(R) \text{ and } t.A \neq \texttt{NULL}\}$, and (2) $\mu(?A) = t.A$ for every A in the domain of μ.

Example 1 Assume that a relational schema contains a relation name `STUDENT` and attributes `ID`, `NAME` and `AGE`. Moreover, assume that t is a tuple in this relation such that $t.\texttt{ID} = 1$, $t.\texttt{NAME} =$ John and $t.\texttt{AGE} = \texttt{NULL}$. Then, $tr(t) = \mu$, where the domain of μ is $\{?\texttt{ID}, ?\texttt{NAME}\}$, $\mu(?\texttt{ID}) = 1$ and $\mu(?\texttt{NAME}) =$ John. □

Definition 3 (Query preservation) *A direct mapping $\mathcal{M}$ is* query preserving *if for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and relational algebra query Q over $\mathbf{R}$, there exists a SPARQL query $Q^\star$ such that for every instance I of $\mathbf{R}$ satisfying Σ: $tr([\![Q]\!]_I) = [\![Q^\star]\!]_{\mathcal{M}(\mathbf{R},\Sigma,I)}$.*

It is important to notice that information preservation and query preservation are incomparable properties in our setting. On one side, if a direct mapping $\mathcal{M}$ is information preserving, this does not guarantee that every relational algebra query Q can be rewritten into an equivalent SPARQL query over the translated data, as $\mathcal{M}$ could transform source relational databases in such a way that a more expressive query language is needed to express Q over the generated RDF graphs. On the other side, a mapping $\mathcal{M}$ can be query preserving and not information preserving if the information about the schema of the relational database being translated is not stored. For example, we define in Section 4 a direct mapping $\mathcal{DM}$ that includes information about these relational schemas. It will become clear in Sections 4 and 5 that if such information is not stored, then $\mathcal{DM}$ would be query preserving but not information preserving.

3.2 Desirable properties

Monotonicity: Given two database instances I_1 and I_2 over a relational schema $\mathbf{R}$, instance I_1 is said to be contained in instance I_2,

denoted by $I_1 \subseteq I_2$, if for every $R \in \mathbf{R}$, it holds that $R^{I_1} \subseteq R^{I_2}$. A direct mapping $\mathcal{M}$ is considered monotone if for any such pair of instances, the result of mapping I_2 contains the result of mapping I_1. In other words, if we insert new data to the database, then the elements of the mapping that are already computed are unaltered.

Definition 4 (Monotonicity) *A direct mapping $\mathcal{M}$ is monotone if for every relational schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$, and instances I_1, I_2 of $\mathbf{R}$ such that $I_1 \subseteq I_2$: $\mathcal{M}(\mathbf{R}, \Sigma, I_1) \subseteq \mathcal{M}(\mathbf{R}, \Sigma, I_2)$.*

Semantics preservation: A direct mapping is semantics preserving if the satisfaction of a set of PKs and FKs by a relational database is encoded in the translation process. More precisely, given a relational schema $\mathbf{R}$, a set Σ of PKs and FKs over $\mathbf{R}$ and an instance I of $\mathbf{R}$, a semantics preserving mapping should generate from I a consistent RDF graph if $I \models \Sigma$, and it should generate an inconsistent RDF graph otherwise.

Definition 5 (Semantics preservation) *A direct mapping $\mathcal{M}$ is semantics preserving if for every relation schema $\mathbf{R}$, set Σ of PKs and FKs over $\mathbf{R}$ and instance I of $\mathbf{R}$: $I \models \Sigma$ iff $\mathcal{M}(\mathbf{R}, \Sigma, I)$ is consistent under OWL semantics.*

4. THE DIRECT MAPPING $\mathcal{DM}$

We introduce a direct mapping $\mathcal{DM}$, that integrates and extends the functionalities of the direct mappings proposed in [23, 5]. $\mathcal{DM}$ is defined as a set of Datalog rules[1], which are divided in two parts: translate relational schemas and translate relational instances.

In Section 4.1, we present the predicates that are used to store a relational database, the input of $\mathcal{DM}$. In Section 4.2, we present predicates that are used to store an ontology and Datalog rules to generate an ontology from the relational schema and the set of PKs and FKs. In Section 4.3, we present the Datalog rules that generate the OWL vocabulary from the ontology that was derived from the relational schema and a set of PKs and FKs. Finally, we present in Section 4.4 the Datalog rules that generates RDF triples from a relational instance.

Throughout this section, we use the following running example. Consider a relational database for a university. The schema of this database consists of tables `STUDENT(SID,NAME)`, `COURSE(CID,TITLE,CODE)`, `DEPT(DID,NAME)` and `ENROLLED(SID,CID)`. Moreover, we have the following constraints about the schema of the university: `SID` is the primary key of `STUDENT`, `CID` is the primary key of `COURSE`, `DID` is the primary key of `DEPT`, `(SID,CID)` is the primary key of `ENROLLED`, `CODE` is a foreign key in `COURSE` that references attribute `DID` in `DEPT`, `SID` is a foreign key in `ENROLLED` that references attribute `SID` in `STUDENT`, and `CID` is a foreign key in `ENROLLED` that references attribute `CID` in `COURSE`.

4.1 Storing relational databases

Given that the direct mapping $\mathcal{DM}$ is specified by a set of Datalog rules, its input $(\mathbf{R}, \Sigma, I)$ has to be encoded as a set of relations. We define the predicates that are used to store the triples of the form $(\mathbf{R}, \Sigma, I)$. More precisely, the following predicates are used to store a relational schema $\mathbf{R}$ and a set Σ of PKs and FKs over $\mathbf{R}$.

- $\text{REL}(r)$: Indicates that r is a relation name in $\mathbf{R}$; e.g. $\text{REL}(\texttt{"STUDENT"})$ indicates that `STUDENT` is a relation name.[2]
- $\text{ATTR}(a, r)$: Indicates that a is an attribute in the relation r in $\mathbf{R}$; e.g. $\text{ATTR}(\texttt{"NAME"}, \texttt{"STUDENT"})$ holds.
- $\text{PK}_n(a_1, \ldots, a_n, r)$: Indicates that $r[a_1, \ldots, a_n]$ is a primary key in Σ; e.g. $\text{PK}_1(\texttt{"SID"}, \texttt{"STUDENT"})$ holds.
- $\text{FK}_n(a_1, \ldots, a_n, r, b_1, \ldots, b_n, s)$: Indicates that $r[a_1, \ldots, a_n] \subseteq_{\text{FK}} s[b_1, \ldots, b_n]$ is a foreign key in Σ; e.g. $\text{FK}_1(\texttt{"CODE"}, \texttt{"COURSE"}, \texttt{"DID"}, \texttt{"DEPT"})$ holds.

Moreover, the following predicate is used to store the tuples in an relational instance I of a relational schema $\mathbf{R}$.

- $\text{VALUE}(v, a, t, r)$: Indicates that v is the value of an attribute a in a tuple with identifier t in a relation r (that belongs to $\mathbf{R}$); e.g. a tuple t_1 of table `STUDENT` such that t_1.`SID` = `"1"` and t_1.`NAME` = `NULL` is stored by using the facts $\text{VALUE}(\texttt{"1"}, \texttt{"SID"}, \texttt{"id1"}, \texttt{"STUDENT"})$ and $\text{VALUE}(\texttt{NULL}, \texttt{"NAME"}, \texttt{"id1"}, \texttt{"STUDENT"})$, assuming that `id1` is the identifier of tuple t_1.

4.2 Storing an ontology

In order to translate a relational database into an RDF graph with OWL vocabulary, we first extract an ontology from the relational schema and the set of PKs and FKs given as input. In particular, we classify each relation name in the schema as a class or a binary relation (which is used to represent a many-to-many relationship between entities in an ER/UML diagram), we represent foreign keys as object properties and attributes of relations as data type properties. More specifically, the following predicates are used to store the extracted ontology:

- $\text{CLASS}(c)$: Indicates that c is a class.
- $\text{OP}_n(p_1, \ldots, p_n, d, r)$: Indicates that $p_1, \ldots, p_n$ $(n \geq 1)$ form an object property with domain d and range r.
- $\text{DTP}(p, d)$: Indicates that p is a data type property with domain d.

The above predicates are defined by the Datalog rules described in the following sections.

Identifying binary relations: We define auxiliary predicates that identify binary relations to facilitate identifying classes, object properties and data type properties. Informally, a relation R is a binary relation between two relations S and T if (1) both S and T are different from R, (2) R has exactly two attributes A and B, which form a primary key of R, (3) A is the attribute of a foreign key in R that points to S, (4) B is the attribute of a foreign key in R that points to T, (5) A is not the attribute of two distinct foreign keys in R, (6) B is not the attribute of two distinct foreign keys in R, (7) A and B are not the attributes of a composite foreign key in R, and (8) relation R does not have incoming foreign keys. In Datalog this becomes:

$$\begin{aligned}\text{BINREL}(R, A, B, S, C, T, D) \leftarrow\ & \\ & \text{PK}_2(A, B, R), \neg\text{THREEATTR}(R), \\ & \text{FK}_1(A, R, C, S), R \neq S, \text{FK}_1(B, R, D, T), R \neq T, \\ & \neg\text{TWOFK}(A, R), \neg\text{TWOFK}(B, R), \\ & \neg\text{ONEFK}(A, B, R), \neg\text{FKTO}(R).\end{aligned} \quad (1)$$

In a Datalog rule, negation is represented with the symbol $\neg$ and upper case letters are used to denote variables. Thus, the previous rule states that the relation R is a binary relation between two relations S and T if the following conditions are satisfied. (a) Expression $\text{PK}_2(A, B, R)$ in (1) indicates that attributes A and B form a primary key of R. (b) Predicate THREEATTR checks whether a relation has at least three attributes, and it is defined as follows from the base predicate ATTR:

$$\begin{aligned}\text{THREEATTR}(R) \leftarrow\ & \text{ATTR}(X, R), \text{ATTR}(Y, R), \\ & \text{ATTR}(Z, R), X \neq Y, X \neq Z, Y \neq Z.\end{aligned}$$

[1] We refer the reader to [3] for the syntax and semantics of Datalog.
[2] As is customary, we use double quotes to delimit strings.

Thus, expression $\neg$THREEATTR(R) in (1) indicates that R has at least two attributes. Notice that by combining this expression with $\text{PK}_2(A, B, R)$, we conclude that A, B are exactly the attributes of R. (c) Expressions $\text{FK}_1(A, R, C, S)$ and $\text{FK}_1(B, R, D, T)$ in (1) indicate that A is the attribute of a foreign key in R that points to S and B is the attribute of a foreign key in R that points to T, respectively. (d) Expressions $R \neq S$ and $R \neq T$ in (1) indicate that both S and T are different from relation R. (e) Predicate TWOFK checks whether an attribute of a relation is the attribute of two distinct foreign keys in that relation, and it is defined as follows from the base predicate FK_1:

$$\begin{aligned}\text{TWOFK}(X, Y) \quad &\leftarrow \quad \text{FK}_1(X, Y, U_1, V_1), \text{FK}_1(X, Y, U_2, V_2), \\ &\qquad U_1 \neq U_2 \\ \text{TWOFK}(X, Y) \quad &\leftarrow \quad \text{FK}_1(X, Y, U_1, V_1), \text{FK}_1(X, Y, U_2, V_2), \\ &\qquad V_1 \neq V_2\end{aligned}$$

Thus, expressions $\neg$TWOFK(A, R) and $\neg$TWOFK(B, R) in (1) indicate that attribute A is not the attribute of two distinct foreign keys in R and B is not the attribute of two distinct foreign keys in R, respectively. (f) Predicate ONEFK checks whether a pair of attributes of a relation are the attributes of a composite foreign key in that relation:

$$\begin{aligned}\text{ONEFK}(X, Y, Z) \quad &\leftarrow \quad \text{FK}_2(X, Y, Z, U, V, W) \\ \text{ONEFK}(X, Y, Z) \quad &\leftarrow \quad \text{FK}_2(Y, X, Z, U, V, W)\end{aligned}$$

Thus, expression $\neg$ONEFK(A, B, R) in (1) indicates that attributes A, B of R are not the attributes of a composite foreign key in R. (g) Finally, predicate FKTO checks whether a relation with two attributes has incoming foreign keys:

$$\begin{aligned}\text{FKTO}(X) \quad &\leftarrow \quad \text{FK}_1(U_1, Y, V, X) \\ \text{FKTO}(X) \quad &\leftarrow \quad \text{FK}_2(U_1, U_2, Y, V_1, V_2, X)\end{aligned}$$

Thus, expression $\neg$FKTO(R) in (1) indicates that relation R does not have incoming foreign keys.

For instance, BINREL("ENROLLED", "SID", "CID", "STUDENT", "SID", "COURSE", "CID") holds in our example. Note that there is no condition in the rule (1) that requires S and T to be different, allowing binary relations that have their domain equal to their range. Also note that, for simplicity, we assume in the rule (1) that a binary relation R consists of only two attributes A and B. However, this rule can be easily extended to deal with binary relations generated from many-to-many relationships between entities in an ER/UML diagram that have more than two attributes.

Identifying classes: In our context, a class is any relation that is not a binary relation. That is, predicate CLASS is defined by the following Datalog rules:

$$\begin{aligned}\text{CLASS}(X) \quad &\leftarrow \quad \text{REL}(X), \neg\text{ISBINREL}(X) \\ \text{ISBINREL}(X) \quad &\leftarrow \quad \text{BINREL}(X, A, B, S, C, T, D)\end{aligned}$$

In our example, CLASS("DEPT"), CLASS("STUDENT") and CLASS("COURSE") hold.

Identifying object properties: For every $n \geq 1$, the following rule is used for identifying object properties that are generated from foreign keys: [3]

$$\begin{aligned}&\text{OP}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T) \leftarrow \\ &\qquad \text{FK}_n(X_1, \ldots, X_n, S, Y_1, \ldots, Y_n, T), \neg\text{ISBINREL}(S)\end{aligned}$$

[3]Notice that although we consider an infinite number of rules in the definition of $\mathcal{DM}$, for every concrete relational database we will need only a finite number of these rules.

This rule states that a foreign key represents an object property from the entity containing the foreign key (domain) to the referenced entity (range). It should be noticed that this rule excludes the case of binary relations, as there is a special rule for this type of relations (see rule (1)). In our example, OP_2("CODE", "DID", "COURSE", "DEPT") holds as CODE is a foreign key in the table COURSE that references attribute DID in the table DEPT.

Identifying data type properties: Every attribute in a non-binary relation is mapped to a data type property:

$$\text{DTP}(A, R) \quad \leftarrow \quad \text{ATTR}(A, R), \neg\text{ISBINREL}(R)$$

For instance, we have that DTP("NAME", "STUDENT") holds in our example, while DTP("SID", "ENROLLED") does not hold as ENROLLED is a binary relation.

4.3 Translating a relational schema into OWL

We now define the rules that translates a relational database schema into an OWL vocabulary.

4.3.1 *Generating IRIs for classes, object properties and data type properties*

We introduce a family of rules that produce IRIs for classes, binary relations, object properties and data type properties identified by the mapping (which are stored in the predicates CLASS, BINREL, OP_n and DTP, respectively). Note that the IRIs generated can be later on replaced or mapped to existing IRIs available in the Semantic Web. Assume given a base IRI base for the relational database to be translated (for example, "http://example.edu/db/"), and assume given a family of built-in predicates CONCAT_n ($n \geq 2$) such that CONCAT_n has $n+1$ arguments and $\text{CONCAT}_n(x_1, \ldots, x_n, y)$ holds if y is the concatenation of the strings x_1, ..., x_n. Then by following the approach proposed in [5], $\mathcal{DM}$ uses the following Datalog rules to produce IRIs for classes and data type properties:

$$\begin{aligned}&\text{CLASSIRI}(R, X) \leftarrow \text{CLASS}(R), \text{CONCAT}_2(\text{base}, R, X) \\ &\text{DTP\_IRI}(A, R, X) \leftarrow \text{DTP}(A, R), \text{CONCAT}_4(\text{base}, R, \text{"\#"}, A, X)\end{aligned}$$

For instance, http://example.edu/db/STUDENT is the IRI for the STUDENT relation in our example, and http://example.edu/db/STUDENT#NAME is the IRI for attribute NAME in the STUDENT relation (recall that DTP("NAME", "STUDENT") holds in our example). Moreover, $\mathcal{DM}$ uses the following family of Datalog rules to generate IRIs for object properties. First, for object properties generated from binary relations, the following rules is used:

$$\begin{aligned}&\text{OP\_IRI}_1(R, A, B, S, C, T, D, X) \leftarrow \\ &\qquad \text{BINREL}(R, A, B, S, C, T, D), \\ &\qquad \text{CONCAT}_{10}(\text{base}, R, \text{"\#"}, A, \text{","}, B, \text{","}, C, \text{","}, D, X)\end{aligned}$$

Thus, http://example.edu/db/ENROLLED#SID,CID,SID,CID is the IRI for binary relation ENROLLED in our example. Second, for object properties generated from a foreign key consisting of n attributes ($n \geq 1$), the following rule is used:

$$\begin{aligned}&\text{OP\_IRI}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T, X) \leftarrow \\ &\quad \text{OP}_{2n}(X_1, \ldots, X_n, Y_1, \ldots, Y_n, S, T), \\ &\quad \text{CONCAT}_{4n+4}(\text{base}, S, \text{","}, T, \text{"\#"}, X_1, \text{","}, \text{","}, \ldots, X_{n-1}, \text{","}, \text{","}, \\ &\qquad\qquad X_n, \text{","}, \text{","}, Y_1, \text{","}, \text{","}, \ldots, Y_{n-1}, \text{","}, \text{","}, Y_n, X)\end{aligned}$$

Thus, given that OP_2("CODE", "DID", "COURSE", "DEPT") holds in our example, IRI http://example.edu/db/COURSE,DEPT#CODE,DID is generated to represented the fact that CODE is a foreign key in the table COURSE that references attribute DID in the table DEPT.

4.3.2 Translating relational schemas

The following Datalog rules are used to generate the RDF representation of the OWL vocabulary. First, a rule is used to collect all the classes:

$$\text{TRIPLE}(U, \texttt{"rdf:type"}, \texttt{"owl:Class"}) \leftarrow \text{CLASS}(R), \text{CLASSIRI}(R, U)$$

Predicate TRIPLE is used to collect all the triples of the RDF graph generated by the direct mapping $\mathcal{DM}$. Second, the following family of rules is used to collect all the object properties ($n \geq 1$):

$$\text{TRIPLE}(U, \texttt{"rdf:type"}, \texttt{"owl:ObjectProperty"}) \leftarrow \text{OP}_n(X_1, \ldots, X_n, S, T), \text{OP\_IRI}_n(X_1, \ldots, X_n, S, T, U)$$

Third, the following rule is used to collect the domains of the object properties ($n \geq 1$):

$$\text{TRIPLE}(U, \texttt{"rdfs:domain"}, W) \leftarrow \text{OP}_n(X_1, \ldots, X_n, S, T), \text{OP\_IRI}_n(X_1, \ldots, X_n, S, T, U), \text{CLASSIRI}(S, W)$$

Fourth, the following rule is used to collect the ranges of the object properties ($n \geq 1$):

$$\text{TRIPLE}(U, \texttt{"rdfs:range"}, W) \leftarrow \text{OP}_n(X_1, \ldots, X_n, S, T), \text{OP\_IRI}_n(X_1, \ldots, X_n, S, T, U), \text{CLASSIRI}(T, W)$$

Fifth, the following rule is used to collect all the data type properties:

$$\text{TRIPLE}(U, \texttt{"rdf:type"}, \texttt{"owl:DatatypeProperty"}) \leftarrow \text{DTP}(A, R), \text{DTP\_IRI}(A, R, U)$$

Finally, the following rule is used to collect the domains of the data type properties:

$$\text{TRIPLE}(U, \texttt{"rdfs:domain"}, W) \leftarrow \text{DTP}(A, R), \text{DTP\_IRI}(A, R, U), \text{CLASSIRI}(R, W)$$

4.4 Translating a database instance into RDF

We now define the rules that map a relational database instance into RDF. More specifically, we first introduce a series of rules for generating IRIs, and then we present the Datalog rules that generate RDF.

4.4.1 Generating IRIs for tuples

We introduce a family of predicates that produce IRIs for the tuples being translated, where we assume a given a base IRI `base` for the relational database (for example, `"http://example.edu/db/"`). First, $\mathcal{DM}$ uses the following Datalog rule to produce IRIs for the tuples of the relations having a primary key:

$$\begin{aligned}\text{ROWIRI}_n(V_1, V_2, \ldots, V_n, A_1, A_2, \ldots, A_n, T, R, X) \leftarrow\ & \text{PK}_n(A_1, A_2, \ldots, A_n, R), \text{VALUE}(V_1, A_1, T, R), \\ & \text{VALUE}(V_2, A_2, T, R), \ldots, \text{VALUE}(V_n, A_n, T, R), \\ & \text{CONCAT}_{4n+2}(\texttt{base}, R, \texttt{"\#"}, A_1, \texttt{"="}, V_1, \texttt{","}, \\ & A_2, \texttt{"="}, V_2, \texttt{","}, \ldots, \texttt{","}, A_n, \texttt{"="}, V_n, X)\end{aligned}$$

Thus, given that the facts $\text{PK}_1(\texttt{"SID"}, \texttt{"STUDENT"})$ and $\text{VALUE}(\texttt{"1"}, \texttt{"SID"}, \texttt{"id1"}, \texttt{"STUDENT"})$ hold in our example, the IRI `http://example.edu/db/STUDENT#SID=1` is the identifier for the tuple in table `STUDENT` with value `1` in the primary key. Moreover, $\mathcal{DM}$ uses the following rule to generate blank nodes for the tuples of the relations not having a primary key:

$$\text{BLANKNODE}(T, R, X) \leftarrow \text{VALUE}(V, A, T, R), \text{CONCAT}_3(\texttt{"\_:"}, R, T, X)$$

4.4.2 Translating relational instances

The direct mapping $\mathcal{DM}$ generates three types of triples when translating a relational instance: Table triples, reference triples and literal triples [5]. Following are the Datalog rules for each one of these cases.

For table triples, $\mathcal{DM}$ produces for each tuple t in a relation R, a triple indicating that t is of type r. To construct these tuples, $\mathcal{DM}$ uses the following auxiliary rules:

$$\begin{aligned}\text{TUPLEID}(T, R, X) \leftarrow\ & \text{CLASS}(R), \text{PK}_n(A_1, \ldots, A_n, R), \\ & \text{VALUE}(V_1, A_1, T, R), \ldots, \text{VALUE}(V_n, A_n, T, R), \\ & \text{ROWIRI}_n(V_1, \ldots, V_n, A_1, \ldots, A_n, T, R, X)\end{aligned}$$

$$\begin{aligned}\text{TUPLEID}(T, R, X) \leftarrow\ & \text{CLASS}(R), \neg\text{HASPK}_n(R), \\ & \text{VALUE}(V, A, T, R), \text{BLANKNODE}(T, R, X)\end{aligned}$$

That is, $\text{TUPLEID}(T, R, X)$ generates the identifier X of a tuple T of a relation R, which is an IRI if R has a primary key or a blank node otherwise. Notice that in the preceding rules, predicate HASPK_n is used to check whether a table R with n attributes has a primary key (thus, $\neg\text{HASPK}_n(R)$ indicates that R does not have a primary key). Predicate HASPK_n is defined by the following n rules:

$$\text{HASPK}_n(X) \leftarrow \text{PK}_i(A_1, \ldots, A_i, X) \qquad i \in \{1, \ldots, n\}$$

The following rule generates the table triples:

$$\text{TRIPLE}(U, \texttt{"rdf:type"}, W) \leftarrow \text{VALUE}(V, A, T, R), \text{TUPLEID}(T, R, U), \text{CLASSIRI}(R, W)$$

For example, the following is a table triple in our example:

$$\begin{aligned}\text{TRIPLE}(&\texttt{"http://example.edu/db/STUDENT\#SID=1"}, \\ &\texttt{"rdf:type"}, \\ &\texttt{"http://example.edu/db/STUDENT"})\end{aligned}$$

For reference triples, $\mathcal{DM}$ generates triples that store the references generated by binary relations and foreign keys. More precisely, the following Datalog rule is used to construct reference triples for object properties that are generated from binary relations:

$$\begin{aligned}\text{TRIPLE}(U, V, W) \leftarrow\ & \text{BINREL}(R, A, B, S, C, T, D), \\ & \text{VALUE}(V_1, A, T_1, R), \text{VALUE}(V_1, C, T_2, S), \\ & \text{VALUE}(V_2, B, T_1, R), \text{VALUE}(V_2, D, T_3, T), \\ & \text{TUPLEID}(T_2, S, U), \\ & \text{OP\_IRI}_1(R, A, B, S, C, T, D, V), \\ & \text{TUPLEID}(T_3, T, W)\end{aligned}$$

Moreover, the following Datalog rule is used to construct reference triples for object properties that are generated from foreign keys ($n \geq 1$):

$$\begin{aligned}\text{TRIPLE}(U, V, W) \leftarrow\ & \text{OP}_{2n}(A_1, \ldots, A_n, B_1, \ldots, B_n, S, T), \\ & \text{VALUE}(V_1, A_1, T_1, S), \ldots, \text{VALUE}(V_n, A_n, T_1, S), \\ & \text{VALUE}(V_1, B_1, T_2, T), \ldots, \text{VALUE}(V_n, B_n, T_2, T), \\ & \text{TUPLEID}(T_1, S, U), \text{TUPLEID}(T_2, T, W), \\ & \text{OP\_IRI}_{2n}(A_1, \ldots, A_n, B_1, \ldots, B_n, S, T, V)\end{aligned}$$

Finally, $\mathcal{DM}$ produces for every tuple t in a relation R and for every attribute A of R, a triple storing the value of t in A, which is called a literal triple. The following Datalog rule is used to generate such triples:

$$\text{TRIPLE}(U, V, W) \leftarrow \text{DTP}(A, R), \text{VALUE}(W, A, T, R), \\ W \neq \texttt{NULL}, \text{TUPLEID}(T, R, U), \text{DTP\_IRI}(A, R, V)$$

Notice that in the above rule, we use the condition $W \neq \texttt{NULL}$ to check that the value of the attribute A in a tuple T in a relation R is not null. Thus, literal triples are generated only for non-null values. The following is an example of a literal triple:

```
TRIPLE("http://example.edu/db/STUDENT#SID=1",
       "http://example.edu/db/STUDENT#NAME","John")
```

5. PROPERTIES OF $\mathcal{DM}$

We now study our direct mapping $\mathcal{DM}$ with respect to the two fundamental properties (information preservation and query preservation) and the two desirable properties (monotonicity and semantics preservation) defined in Section 3.

5.1 Information preservation of $\mathcal{DM}$

First, we show that $\mathcal{DM}$ does not lose any piece of information in the relational instance being translated:

Theorem 1 *The direct mapping $\mathcal{DM}$ is information preserving.*

The proof of this theorem is straightforward, and it involves providing a computable mapping $\mathcal{N} : \mathcal{G} \rightarrow \mathcal{I}$ that satisfies the condition in Definition 2, that is, a computable mapping $\mathcal{N}$ that can reconstruct the initial relational instance from the generated RDF graph.

5.2 Query preservation of $\mathcal{DM}$

Second, we show that the way $\mathcal{DM}$ maps relational data into RDF allows one to answer a query over a relational instance by translating it into an equivalent query over the generated RDF graph.

Theorem 2 *The direct mapping $\mathcal{DM}$ is query preserving.*

In [4], it was proved that SPARQL has the same expressive power as relational algebra. Thus, one may be tempted to think that this result could be used to prove Theorem 2. However, the version of relational algebra considered in [4] does not include the null value NULL, and hence cannot be used to prove our result. In addition to this, other researchers have addressed the issue of querying answering on DL ontologies with relational databases [21]. Our work is similar in the sense that we address the issue of query preservation between a database and an ontology. However, the main difference is that rather than a domain ontology, the ontology we use is synthesized in a standard way from the database schema. Therefore, their results cannot be directly applied to our setting.

We present an outline of the proof of this theorem, and refer the reader to [19] for the details. Assume given a relational schema $\mathbf{R}$ and a set Σ of PKs and FKs over $\mathbf{R}$. Then we have to show that for every relational algebra query Q over $\mathbf{R}$, there exists a SPARQL query $Q^\star$ such that for every instance I of $\mathbf{R}$ (possibly including null values) satisfying Σ:

$$tr(\llbracket Q \rrbracket_I) \quad = \quad \llbracket Q^\star \rrbracket_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \tag{2}$$

Interestingly, the proof that the previous condition holds is by induction on the structure of Q, and thus it gives us a bottom-up algorithm for translating Q into an equivalent SPARQL query $Q^\star$, that is, a query $Q^\star$ satisfying condition (2). In what follows, we consider the database used as example in Section 4 and the relational algebra query $\sigma_{\texttt{Name=Juan}}(\texttt{STUDENT}) \bowtie \texttt{ENROLLED}$, which we will use as a running example and translate it step by step to SPARQL, showing how the translation algorithm works.

For the sake of readability, we introduce a function ν that retrieves the IRI for a given relation R, denoted by $\nu(R)$, and the IRI for a given attribute A in a relation R, denoted by $\nu(A, R)$. The inductive proof starts by considering the two base relational algebra queries: the identity query R, where R is a relation name in the relational schema $\mathbf{R}$, and the query $\texttt{NULL}_A$. These two base queries give rise to the following three base cases for the inductive proof.

Non-binary relations: Assume that Q is the identity relational algebra query R, where $R \in \mathbf{R}$ is a non-binary relation (that is, $\text{ISBINREL}(R)$ does not hold). Moreover, assume that $att(R) = \{A_1, \ldots, A_\ell\}$, with the corresponding IRIs $\nu(R) = r, \nu(A_1, R) = a_1, \ldots, \nu(A_\ell, R) = a_\ell$. Then a SPARQL query $Q^\star$ satisfying (2) is constructed as follows:

$$\begin{aligned}\text{SELECT } \{?A_1, \ldots, ?A_\ell\} \Big[\cdots \Big(\Big(\Big(&(?X, \texttt{"rdf:type"}, r) \\ &\text{OPT } (?X, a_1, ?A_1) \Big) \text{ OPT } (?X, a_2, ?A_2) \Big) \\ &\text{OPT } (?X, a_3, ?A_3) \Big) \cdots \text{ OPT } (?X, a_\ell, ?A_\ell) \Big].\end{aligned}$$

Notice that in order to not lose information, the operator OPT is used (instead of AND) because the direct mapping $\mathcal{DM}$ does not translate NULL values. In our example, the relation name STUDENT is a non-binary relation. Therefore the following equivalent SPARQL query is generated with input STUDENT:

$$\begin{aligned}\text{SELECT } \{\texttt{?SID}, \texttt{?NAME}\} \Big[\Big(&(?X, \texttt{"rdf:type"}, \texttt{:STUDENT}) \\ &\text{OPT } (?X, \texttt{:STUDENT\#SID}, \texttt{?SID}) \Big) \\ &\text{OPT } (?X, \texttt{:STUDENT\#NAME}, \texttt{?NAME}) \Big]\end{aligned}$$

It should be noticed that in the previous query, the symbol : has to be replaced by the base IRI used when generating IRIs for relations and attributes in a relation (see Section 4.3.1) [4].

Binary relations: Assume that Q is the identity relational algebra query R, where $R \in \mathbf{R}$ is a binary relation (that is, $\text{ISBINREL}(R)$ holds). Moreover, assume that $att(R) = \{A_1, A_2\}$, where A_1 is a foreign key referencing the attribute B of a relation S, and A_2 is a foreign key referencing the attribute C of a relation T. Finally, assume that $\nu(R) = r$, $\nu(B, S) = b$ and $\nu(C, T) = c$, Then a SPARQL query $Q^\star$ satisfying (2) is defined as follows:

$$\begin{aligned}\text{SELECT } \{?A_1, ?A_2\} \; ((?T_1, r, ?T_2) \;\; &\text{AND} \\ &(?T_1, b, ?A_1) \text{ AND } (?T_2, c, ?A_2)).\end{aligned}$$

Given that a binary relation is mapped to an object property, the values of a binary relation can be retrieved by querying the datatype properties of the referenced attributes. In our example, the relational name ENROLLED is a binary relation. Therefore the following equivalent SPARQL query is generated with input ENROLLED:

```
SELECT {?SID, ?CID}(
        (?T1, :ENROLLED#SID,CID, SID,CID, ?T2) AND
        (?T1, :STUDENT#SID, ?SID) AND
        (?T2, :COURSE#CID, ?CID)).
```

[4] In SPARQL terminology, we have included the following prefix in the query: `@prefix : <http://example.edu/db/>`, if the base IRI is `<http://example.edu/db/>`.

Empty relation: Assume that $Q = \texttt{NULL}_A$, and define $Q^\star$ as the empty graph pattern $\{\ \}$. Then we have that condition (2) holds because of the definition of the function *tr*, which does not translate NULL values to mappings.

We now present the inductive step in the proof of Theorem 2. Assume that the theorem holds for relational algebra queries Q_1 and Q_2. That is, there exists SPARQL queries $Q_1^\star$ and $Q_2^\star$ such that:

$$tr([\![Q_1]\!]_I) = [\![Q_1^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}, \quad (3)$$

$$tr([\![Q_2]\!]_I) = [\![Q_2^\star]\!]_{\mathcal{DM}(\mathbf{R},\Sigma,I)}. \quad (4)$$

The proof continues by presenting equivalent SPARQL queries for the following relational algebra operators: selection (σ), projection (π), rename (δ), join ($\bowtie$), union ($\cup$) and difference ($\setminus$). It is important to notice that the operators left-outer join, right-outer join and full-outer join are all expressible with the previous operators, hence we do not present cases for these operators.

Selection: We need to consider four cases to define query $Q^\star$ satisfying condition (2). In all these cases, we use the already established equivalence (3).

1. If Q is $\sigma_{A_1=a}(Q_1)$, then
$$Q^\star = (Q_1^\star \text{ FILTER } (?A_1 = a)).$$
2. If Q is $\sigma_{A_1\neq a}(Q_1)$, then
$$Q^\star = (Q_1^\star \text{ FILTER } (\neg(?A_1 = a) \wedge \text{bound}(?A_1))).$$
3. If Q is $\sigma_{\texttt{IsNull}(A_1)}(Q_1)$, then
$$Q^\star = (Q_1^\star \text{ FILTER } (\neg \text{bound}(?A_1))).$$
4. If Q is $\sigma_{\texttt{IsNotNull}(A_1)}(Q_1)$, then
$$Q^\star = (Q_1^\star \text{ FILTER } (\text{bound}(?A_1))).$$

These equivalences are straightforward. However, it is important to note the use of $\text{bound}(\cdot)$ in the second case; as the semantics of relational algebra states that if Q is the query $\sigma_{A_1\neq a}(Q_1)$, then $[\![Q]\!]_I = \{t \in [\![Q_1]\!]_I \mid t.A_1 \neq \texttt{NULL} \text{ and } t.A_1 \neq a\}$, we have that the variable $?A_1$ has to be bound because the values in the attribute A_1 in the answer to $\sigma_{A_1\neq a}(Q_1)$ are different from NULL. Following our example, we have that the following SPARQL query is generated with input $\sigma_{\texttt{Name=Juan}}(\texttt{STUDENT})$:

$$\Big(\text{SELECT } \{?\texttt{SID}, ?\texttt{NAME}\} \Big[\Big((?X, \texttt{"rdf:type"}, \texttt{:STUDENT}) \text{ OPT } (?X, \texttt{:STUDENT\#SID}, ?\texttt{SID})\Big) \text{ OPT } (?X, \texttt{:STUDENT\#NAME}, ?\texttt{NAME})\Big]\Big) \text{ FILTER } (?\texttt{NAME} = \texttt{Juan})$$

Projection: Assume that $Q = \pi_{\{A_1,\ldots,A_\ell\}}(Q_1)$. Then query $Q^\star$ satisfying condition (2) is defined as $(\text{SELECT } \{?A_1, \ldots, ?A_\ell\}\ Q_1^\star)$. It is important to notice that we use nested SELECT queries to deal with projection, as well as in two of the base cases, which is a functionality specific to SPARQL 1.1 [12].

Rename: Assume that $Q = \delta_{A_1\to B_1}(Q_1)$ and $att(Q) = \{A_1, \ldots, A_\ell\}$. Then query $Q^\star$ satisfying condition (2) is defined as $(\text{SELECT } \{?A_1 \text{ AS } ?B_1, ?A_2, \ldots, ?A_\ell\}\ Q_1^\star)$. Notice that this equivalence holds because the rename operator in relational algebra renames one attribute to another and projects all attributes of Q.

Join: Assume that $Q = (Q_1 \bowtie Q_2)$, where $(att(Q_1) \cap att(Q_2)) = \{A_1, \ldots, A_\ell\}$. Then query $Q^\star$ satisfying condition (2) is defined as follows:

$$\Big[\Big(Q_1^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell))\Big) \text{ AND } \Big(Q_2^\star \text{ FILTER } (\text{bound}(?A_1) \wedge \cdots \wedge \text{bound}(?A_\ell))\Big)\Big].$$

Note the use of $\text{bound}(\cdot)$ which is necessary in the SPARQL query in order to guarantee that the variables that are being joined on are not null. Following our example, Figure 1 shows the SPARQL query generated with input $\sigma_{\texttt{Name=Juan}}(\texttt{STUDENT}) \bowtie \texttt{ENROLLED}$.

Union: Assume that $Q = (Q_1 \cup Q_2)$. Then query $Q^\star$ satisfying condition (2) is simply defined as $(Q_1^\star \text{ UNION } Q_2^\star)$. Notice that in this case we are using the already established equivalences (3) and (4).

Difference: We conclude our proof by assuming that $Q = (Q_1 \setminus Q_2)$. In this case, it is also possible to define a SPARQL query $Q^\star$ satisfying condition (2). Due to the lack of space and the complex structure of this query, we refer the reader to [19] for its complete description.

5.3 Monotonicity and semantics preservation of $\mathcal{DM}$

Finally, we consider the two desirable properties identified in Section 3.2. First, it is straightforward to see that $\mathcal{DM}$ is monotone, because all the negative atoms in the Datalog rules defining $\mathcal{DM}$ refer to the schema, the PKs and the FKs of the database, and these elements are kept fixed when checking monotonicity. Unfortunately, the situation is completely different for the case of semantics preservation, as the following example shows that the direct mapping $\mathcal{DM}$ does not satisfy this property.

Example 2 Assume that a relational schema contains a relation with name STUDENT and attributes SID, NAME, and assume that the attribute SID is the primary key. Moreover, assume that this relation has two tuples, t_1 and t_2 such that $t_1.\texttt{SID} = 1$, $t_1.\texttt{NAME} =$ John and $t_2.\texttt{SID} = 1$, $t_2.\texttt{NAME} =$ Peter. It is clear that the primary key is violated, therefore the database is inconsistent. However, it is not difficult to see that after applying $\mathcal{DM}$, the resulting RDF graph is consistent. □

In fact, the result in Example 2 can be generalized as it is possible to show that the direct mapping $\mathcal{DM}$ always generates a consistent RDF graph, hence, it cannot be semantics preserving.

Proposition 1 *The direct mapping $\mathcal{DM}$ is not semantics preserving.*

Does this mean that our direct mapping is incorrect? What could we do to create a direct mapping that is semantics preserving? These problems are studied in depth in the following section.

6. SEMANTICS PRESERVATION OF DIRECT MAPPINGS

We now study the problem of generating a semantics-preserving direct mapping. Specifically, we show in Section 6.1 that a simple extension of the direct mapping $\mathcal{DM}$ can deal with primary keys. Then we show in Section 6.2 that dealing with foreign keys is more difficult, as any direct mapping that satisfies the condition of being monotone cannot be semantics preserving. Finally, we present two possible ways of overcoming this limitation.

$$\Big[\Big(\Big(\text{SELECT}\ \{\texttt{?SID}, \texttt{?NAME}\}\ \Big[\Big((?X, \texttt{"rdf:type"}, \texttt{:STUDENT})\ \text{OPT}\ (?X, \texttt{:STUDENT\#SID}, \texttt{?SID})\Big)\ \text{OPT}$$
$$(?X, \texttt{:STUDENT\#NAME}, \texttt{?NAME})\Big]\Big)\ \text{FILTER}\ (\texttt{?NAME} = \texttt{Juan})\Big)\ \text{FILTER}\ (\text{bound}(\texttt{?SID}))\Big]$$
$$\text{AND}$$
$$\Big[\Big(\text{SELECT}\ \{\texttt{?SID}, \texttt{?CID}\}\Big((?T_1, \texttt{:ENROLLED\#SID,CID,SID,CID},?T_2)\ \text{AND}\ (?T_1, \texttt{:STUDENT\#SID}, \texttt{?SID})\ \text{AND}$$
$$(?T_2, \texttt{:COURSE\#CID}, \texttt{?CID})\Big)\Big)\ \text{FILTER}\ (\text{bound}(\texttt{?SID}))\Big]$$

Figure 1: SPARQL translation of the relational algebra query $\sigma_{\texttt{Name}=\texttt{Juan}}(\texttt{STUDENT}) \bowtie \texttt{ENROLLED}$.

6.1 A semantics preserving direct mapping for primary keys

Recall that a primary key can be violated if there are repeated values or null values. At a first glance, one would assume that owl:hasKey could be used to create a semantics preserving direct mapping for primary keys. If we consider a database without null values, a violation of the primary key would generate an inconsistency with owl:hasKey and the unique name assumption (UNA). However, if we consider a database with null values, then owl:hasKey with the UNA does not generate an inconsistency because it is trivially satisfied for a class expression that does not have a value for the datatype expression. Therefore, we must consider a different approach.

Consider a new direct mapping $\mathcal{DM}_{pk}$ that extends $\mathcal{DM}$ as follows. A Datalog rule is used to determine if the value of a primary key attribute is repeated, and a family of Datalog rules are used to determine if there is a value NULL in a column corresponding to a primary key. If some of these violations are found, then an artificial triple is generated that would produce an inconsistency. For example, the following rules are used to map a primary key with two attributes:

$$\begin{aligned}\text{TRIPLE}(a, \texttt{"owl:differentFrom"}, a) \leftarrow\ & \text{PK}_2(X_1, X_2, R),\\ & \text{VALUE}(V_1, X_1, T_1, R), \text{VALUE}(V_1, X_1, T_2, R),\\ & \text{VALUE}(V_2, X_2, T_1, R), \text{VALUE}(V_2, X_2, T_2, R), T_1 \neq T_2\\ \text{TRIPLE}(a, \texttt{"owl:differentFrom"}, a) \leftarrow\ & \text{PK}_2(X_1, X_2, R),\\ & \text{VALUE}(V, X_1, T, R), V = \texttt{NULL}\end{aligned}$$

$$\begin{aligned}\text{TRIPLE}(a, \texttt{"owl:differentFrom"}, a) \leftarrow\ & \text{PK}_2(X_1, X_2, R),\\ & \text{VALUE}(V, X_2, T, R), V = \texttt{NULL}\end{aligned}$$

In the previous rules, a is any valid IRI. If we apply $\mathcal{DM}_{pk}$ to the database of Example 2, it is straightforward to see that starting from an inconsistent relational database, one obtains an RDF graph that is also inconsistent. In fact, we have that:

Proposition 2 *The direct mapping $\mathcal{DM}_{pk}$ is information preserving, query preserving, monotone, and semantics preserving if one considers only PKs. That is, for every relational schema $\mathbf{R}$, set Σ of (only) PKs over $\mathbf{R}$ and instance I of $\mathbf{R}$: $I \models \Sigma$ iff $\mathcal{DM}_{pk}(\mathbf{R}, \Sigma, I)$ is consistent under OWL semantics.*

Information preservation, query preservation and monotonicity of $\mathcal{DM}_{pk}$ are corollaries of the fact that these properties hold for $\mathcal{DM}$, and of the fact that the Datalog rules introduced to handle primary keys are monotone.

A natural question at this point is whether $\mathcal{DM}_{pk}$ can also deal with foreign keys. Unfortunately, it is easy to construct an example that shows that this is not the case. Does this mean that we cannot have a direct mapping that is semantics preserving and considers foreign keys? We show in the following section that monotonicity has been one of the obstacles to obtain such a mapping.

6.2 Semantics preserving direct mappings for primary keys and foreign keys

The following theorem shows that the desirable condition of being monotone is, unfortunately, an obstacle to obtain a semantics preserving direct mapping.

Theorem 3 *No monotone direct mapping is semantics preserving.*

It is important to understand the reasons why we have not been able to create a semantics preserving direct mapping. The issue is with two characteristics of OWL: (1) it adopts the Open World Assumption (OWA), where a statement cannot be inferred to be false on the basis of failing to prove it, and (2) it does not adopt the Unique Name Assumption (UNA), where two different names can identify the same thing. On the other hand, a relational database adopts the Closed World Assumption (CWA), where a statement is inferred to be false if it is not known to be true. In other words, what causes an inconsistency in a relational database, can cause an inference of new knowledge in OWL.

In order to preserve the semantics of the relational database, we need to ensure that whatever causes an inconsistency in a relational database, is going to cause an inconsistency in OWL. Following this idea, we now present a non-monotone direct mapping, $\mathcal{DM}_{pk+fk}$, which extends $\mathcal{DM}_{pk}$ by introducing rules for verifying beforehand if there is a violation of a foreign key constraint. If such a violation exists, then an artificial RDF triple is created which will generate an inconsistency with respect to the OWL semantics. More precisely, the following family of Datalog rules are used in $\mathcal{DM}_{pk+fk}$ to detect an inconsistency in a relational database:

$$\begin{aligned}\text{VIOLATION}(S) \leftarrow\ & \\ & \text{FK}_n(X_1, \ldots, X_n, S, Y_1, \ldots, Y_n, T),\\ & \text{VALUE}_n(V_1, X_1, T, S), \ldots, \text{VALUE}(V_n, X_n, T, S),\\ & V_1 \neq \texttt{NULL}, \ldots, V_n \neq \texttt{NULL},\\ & \neg\text{ISVALUE}_n(V_1, \ldots, V_n, Y_1, \ldots, Y_n, T)\end{aligned}$$

In the preceding rule, the predicate ISVALUE_n is used to check whether a tuple in a relation has values for some given attributes. The predicate ISVALUE_n is defined by the following rule:

$$\begin{aligned}\text{ISVALUE}_n(V_1, \ldots, V_n, B_1, \ldots, B_n, S)\ \leftarrow\ & \\ & \text{VALUE}(V_1, B_1, T, S), \ldots, \text{VALUE}(V_n, B_n, T, S)\end{aligned}$$

Finally, the following Datalog rule is used to obtain an inconsistency in the generated RDF graph:

$$\text{TRIPLE}(a, \texttt{"owl:differentFrom"}, a) \leftarrow \text{VIOLATION}(S)$$

In the previous rule, a is any valid IRI. It should be noticed that $\mathcal{DM}_{pk+fk}$ is non-monotone because if new data in the database is added which now satisfies the FK constraint, then the artificial RDF triple needs to be retracted.

Theorem 4 *The direct mapping $\mathcal{DM}_{pk+fk}$ is information preserving, query preserving and semantics preserving.*

Information preservation and query preservation of $\mathcal{DM}_{pk+fk}$ are corollaries of the fact that these properties hold for $\mathcal{DM}$ and $\mathcal{DM}_{pk}$.

A direct mapping that satisfies the four properties can be obtained by considering an alternative semantics of OWL that expresses integrity constraints. Because OWL is based on Description Logic, we would need a version of DL that supports integrity constraints, which is not a new idea. Integrity constraints are epistemic in nature and are about "what the knowledge base knows" [18]. Extending DL with the epistemic operator **K** has been studied [7, 9, 10]. Grimm et al. proposed to extend the semantics of OWL to support the epistemic operator [11]. Motik et al. proposed to write integrity constraints as standard OWL axioms but interpreted with different semantics for data validation purposes [15]. Tao et al. showed that integrity constraint validation can be reduced to SPARQL query answering [22]. Recently, Mehdi et al. introduced a way to answer epistemic queries to restricted OWL ontologies [14]. Thus, it is possible to extend $\mathcal{DM}_{pk}$ to create an information preserving, query preserving and monotone direct mapping that is also semantics preserving, but it is based on a non-standard version of OWL including the epistemic operator **K**.

7. CONCLUDING REMARKS

In this paper, we study how to directly map relational databases to an RDF graph with OWL vocabulary based on two fundamental properties (information preservation and query preservation) and two desirable properties (monotonicity and semantics preservation). We first present a monotone, information preserving and query preserving direct mapping considering databases that have null values. Then we prove that the combination of monotonicity with the OWL semantics is an obstacle to generating a semantics preserving direct mapping. Finally, we overcome this obstacle by presenting a non-monotone direct mapping that is semantics preserving, and also by discussing the possibility of generating a monotone mapping that assumes an extension of OWL with the epistemic operator.

Related Work: Several approaches directly map relational schemas to RDFS and OWL. We refer the reader to the following survey [20]. D2R Server has an option that directly maps the relational database into RDF, however this process is not documented [2]. RDBToOnto presents a direct mapping that mines the content of the relational databases in order to learn ontologies with deeper taxonomies [8]. Currently, the W3C RDB2RDF Working Group is developing a direct mapping standard that focuses on translating relational database instances to RDF [5, 6].

Future Work: We would like to extend our direct mapping to consider datatypes, relational databases under bag semantics and evaluate this rule based approach on large relational databases. The extension of our direct mapping to bag semantics is straightforward. In our setting each tuple has its own identifier, which is represented in the VALUE predicate. Thus, even if repeated tuples exist, each tuple will still have its unique identifier and, therefore, exactly the same rules can be used to map relational data under bag semantics.

8. ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for many helpful comments, and the members of the W3C RDB2RDF Working group for many fruitful discussions. J. F. Sequeda was supported by the NSF Graduate Research Fellowship, M. Arenas by Fondecyt grant #1090565 and D.P. Miranker by NSF grant #1018554.

9. REFERENCES

[1] W3C OWL Working Group. OWL 2 Web ontology language document overview. W3C Recommendation 27 October 2009, http://www.w3.org/TR/owl2-overview/.

[2] D2R Server. Publishing Relational Databases on the Semantic Web http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/.

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] R. Angles and C. Gutierrez. The expressive power of sparql. In *ISWC*, pages 114–129, 2008.

[5] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. Direct mapping of relational data to RDF. W3C Working Draft 20 September 2011, http://www.w3.org/TR/rdb-direct-mapping/.

[6] A. Bertails, and E. Prud'hommeaux. Interpreting relational databases in the RDF domain In *K-CAP*, pages 129–136, 2011.

[7] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Eql-lite: Effective first-order query processing in description logics. In *IJCAI*, pages 274–279, 2007.

[8] F. Cerbah. Mining the Content of Relational Databases to Learn Ontologies with Deeper Taxonomies In *Web Intelligence*, pages 553-557, 2008.

[9] F. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.

[10] F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM TOCL*, 3(2):177–225, 2002.

[11] S. Grimm and B. Motik. Closed world reasoning in the semantic web through epistemic operators. In *OWLED*, 2005.

[12] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C Working Draft 12 May 2011, http://www.w3.org/TR/sparql11-query/.

[13] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50:94–101, May 2007.

[14] A. Mehdi, S. Rudolph, and S. Grimm. Epistemic querying of OWL knowledge bases. In *ESWC (1)*, pages 397–409, 2011.

[15] B. Motik, I. Horrocks, and U. Sattler. Bridging the gap between OWL and relational databases. *J. Web Sem.*, 7(2):74–89, 2009.

[16] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[17] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, http://www.w3.org/TR/rdf-sparql-query/.

[18] R. Reiter. On integrity constraints. In *TARK*, pages 97–111, 1988.

[19] J. F. Sequeda, M. Arenas, and D. P. Miranker. On Directly Mapping Relational Databases to RDF and OWL (Extended Version). arXiv:1202.3667 [cs.DB] (February 2012), http://arxiv.org/abs/1202.3667.

[20] J. F. Sequeda, S. H. Tirmizi, O. Corcho, and D. P. Miranker. Survey of directly mapping sql databases to the semantic web. *Knowledge Eng. Review*, 26(4): 445-486 (2011)

[21] I. Seylan, E. Franconi, and J. De Bruijn. Effective query rewriting with ontologies over DBoxes. In *IJCAI*, pages 923–929, 2009.

[22] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity constraints in OWL. In *AAAI*, 2010.

[23] S. H. Tirmizi, J. Sequeda, and D. P. Miranker. Translating SQL Applications to the Semantic Web. In *DEXA*, pages 450–464, 2008.