A Source-to-Target Constraint Rewriting for Direct Mapping

Ratan Bahadur Thapa and Martin Giese(✉)

Department of Informatics, University of Oslo, Oslo, Norway
{ratanbt,martingi}@ifi.uio.no

Abstract. Most of the existing structured digital information today is still stored in relational databases. That's why it is important for the Semantic Web effort to expose the information in relational databases as RDF, or allow to query it using SPARQL. Direct mapping is a fully automated approach for converting well-structured relational data to RDF that does not require formulating explicit mapping rules [2,8]. Along with the mapped RDF data, it is desirable to have a description of that data. Previous work [3,8] has attempted to describe the RDF graph in terms of OWL axioms, which is problematic, partly due to the open world semantics of OWL. We start from the direct mapping suggested by Sequeda et al. [8], which integrates and extends the functionalities of proposal [10] and the W3C recommendation [2], and present a source-to-target semantics preserving rewriting of constraints in an SQL database schema to equivalent SHACL [7] constraints on the RDF graph. We thus provide a SHACL description of the RDF data generated by the direct mapping without the need to perform a costly validation of those constraints on the generated data. Following the approach of [8], we define the rewriting from SQL constraints to SHACL by a set of Datalog rules. We prove that our source-to-target rewriting of constraints is constraint preserving and weakly semantics preserving.

1 Introduction

Relational *constraints*, also known as *integrity constraints* in relational database theory [1], have traditionally been used to restrict the data in the database to those considered meaningful to the application at hand. Constraints are stated when a relational schema is defined and checked when the stored data is modified or new data is inserted. When relational data is mapped into RDF [4], using Direct Mapping [2] or R2RML [5], the constraints on the original relational data imply certain constraints on the RDF graph, but existing tools do not make these constraints explicit, and the theory behind such constraints on the output of the mapping is not well explored. However, the integrity of the data that is being stored or represented in the RDF graph is a critical piece of information in practice, both to detect problems in the RDF dataset and provide data quality guarantees for the purpose of RDF data exchange and interoperability.

Previous work has attempted to capture the properties of the RDF graph resulting from direct mapping using OWL [8] or as DL-Lite$_{\text{RDFS}}$ axioms with

A. Hotho et al. (Eds.): ISWC 2021, LNCS 12922, pp. 21–38, 2021.
https://doi.org/10.1007/978-3-030-88361-4_2

identification constraints [3]. However, as Sequeda et al. [8, Theorem 3] established, OWL axioms alone cannot provide a mapping that has both of the desirable properties of being *monotone*, i.e., an insertion of new data to the database does not require the alteration of already computed RDF triples, and *semantics preserving*, i.e., one-to-one correspondence between legal relational data and RDF graph satisfying OWL axioms. This is due to (1) DL semantics following the Open World Assumption, and (2) OWL not adopting the Unique Name Assumption (UNA). In our work, we transform integrity constraints on the source data into integrity constraints on the RDF graph, expressed in SHACL, the Shapes Constraint Language [7]. SHACL, as opposed to OWL, subscribes to the Closed World Assumption and is based on the UNA, which makes it a more suitable candidate than OWL for expressing integrity constraints on an RDF graph.

Our work is based on the direct mapping of [8] which is similar to that of the W3C recommendation [2], but has a better treatment of SQL tables that correspond to many-to-many binary relations. The transformation of both the SQL schema and the database instance is described as a set of Datalog rules in [8], which we exploit by describing our generation of SHACL constraints also in terms of Datalog rules, on the same vocabulary. We preserve the original properties of the mapping [8], such as *information preservation*, i.e., the original relational data can be reconstructed from the mapped RDF, and *query preservation*, i.e., SQL queries over source relational data can be rewritten to equivalent SPARQL queries over the mapped RDF.

Our transformation takes into account data types, primary and foreign key constraints, as well as not null and uniqueness constraints in an SQL Schema definition. Under certain reasonable assumptions such that all relations in the relational schema have a primary key and database instances satisfy their primary and foreign key constraints, our proposed SHACL constraint rewriting for direct mapping is *constraint preserving*, i.e., all the original SQL constraints of source database can be reconstructed from the output SHACL constraints, and *weakly semantics preserving*, which means that it exhibits all of the desirable properties proposed in [8].

In Sect. 2, we review central notions of relational databases, SQL constraints, RDF, SHACL and Direct mapping. Section 3 introduces the central notion of constraint rewriting and the properties: constraint preservation and semantics preservation. In Sect. 4, we give the Datalog rules for the proposed rewriting. Section 5 states the properties of the proposed rewriting. Section 6 discusses shortcoming of the rewriting and Sect. 7 concludes the paper.

2 Preliminaries

In this section, we fix notions and notations fundamental to the definition of direct mapping [8] and SHACL constraints [7].

Databases. Let Δ be a countably infinite set of constants, including the reserved symbol null. A *relational schema* $\mathcal{R}$ is a finite set of relation names, known as *relation schemas*. We associate with each relation schema $R \in \mathcal{R}$ a finite, non-empty *set of named attributes*, denoted by $\mathrm{att}(R)$. An *instance* $\mathcal{D}$ of $\mathcal{R}$ assigns to

each relation schema $R \in \mathcal{R}$ a finite set of tuples $R^{\mathcal{D}}$, where each *tuple* $t \in R^{\mathcal{D}}$ is a function that assigns to each attribute in $\mathrm{att}(R)$ a value from the domain Δ. We write $\bar{X}$ as shorthand for a sequence $X_1, \ldots X_n$ of attributes for some $n > 0$, and $X \in \bar{X}$ to say that X is one of the elements of the sequence. $|\bar{X}| = n$ denotes the length of the sequence. Further, we write $\bar{X} \lhd R$ to denote that $\bar{X}$ is a non-empty sequence of attributes of R, i.e. $\emptyset \subsetneq \{X_1, \ldots, X_n\} \subseteq \mathrm{att}(R)$. We write $t(\bar{X})$ to denote the restriction of a tuple $t \in R^{\mathcal{D}}$ to a sequence $\bar{X} \lhd R$ of attributes. Finally, a *database* is a pair $(\mathcal{R}, \mathcal{D})$, where $\mathcal{R}$ is a relational schema and $\mathcal{D}$ is an instance of $\mathcal{R}$.

SQL Constraints. We now define constraints on a relational schema, similarly to the SQL Data Definition Language. The direct mapping of [8] considers only *primary key* (PK) and *foreign key* (FK) constraints, which they refer to as *key constraints*. In addition to these key constraints, we also take account of *not null* (NN) and *unique* (UNQ) entity integrity constraints, as well as *SQL data types*, on a relational schema $\mathcal{R}$, which we refer to as *data constraints*. We write δ for sets of data constraints and σ for sets of key constraints. When there is no risk of confusion, we will often write $\Sigma = \sigma \cup \delta$ to say that Σ is a set of constraints, consisting of key constraints σ and data constraints δ.

A NN constraint on a relational schema $\mathcal{R}$ is an expression of the form $\mathrm{NN}(X, R)$ for any $X \in \mathrm{att}(R)$ with $R \in \mathcal{R}$. Similarly, a UNQ or PK constraint on a relational schema $\mathcal{R}$ is an expression of the form $\mathrm{UNQ}(\bar{X}, R)$ or $\mathrm{PK}(\bar{X}, R)$, respectively, for any $\bar{X} \lhd R$ with $R \in \mathcal{R}$. An instance $\mathcal{D}$ of $\mathcal{R}$ satisfies:

- $\mathrm{NN}(X, R)$ if for every $t \in R^{\mathcal{D}}$, $t(X) \neq \mathsf{null}$.
- $\mathrm{UNQ}(\bar{X}, R)$ if for every $t, t' \in R^{\mathcal{D}}$, if $t(X) = t'(X) \neq \mathsf{null}$ for every $X \in \bar{X}$, then $t = t'$.
- $\mathrm{PK}(\bar{X}, R)$ if (1) for every $t \in R^{\mathcal{D}}$ and $X \in \bar{X}$, $t(X) \neq \mathsf{null}$, and (2) for every $t, t' \in R^{\mathcal{D}}$, if $t(\bar{X}) = t'(\bar{X})$, then $t = t'$.

An FK constraint on $\mathcal{R}$ is an expression of the form $\mathrm{FK}(\bar{X}, R, \bar{Y}, S)$ for any $\bar{X} \lhd R$ and $\bar{Y} \lhd S$ with $|\bar{X}| = |\bar{Y}|$ and $R, S \in \mathcal{R}$. An instance $\mathcal{D}$ of $\mathcal{R}$ satisfies $\mathrm{FK}(\bar{X}, R, \bar{Y}, S)$ if for every $t \in R^{\mathcal{D}}$: either (1) $t(X) = \mathsf{null}$ for some $X \in \bar{X}$, or (2) there is a tuple $t' \in S^{\mathcal{D}}$ such that $t(\bar{X}) = t'(\bar{Y})$. Next, to handle SQL data types, let the domain of each data type T be given as a subset $\Delta_T \subseteq \Delta$ with $\mathsf{null} \in \Delta_T$. An SQL data type declaration on $\mathcal{R}$ is an expression of the form $\textsc{Type}(X, R, T)$ for an attribute $X \in \mathrm{att}(R)$ with $R \in \mathcal{R}$, where T is an SQL data type. An instance $\mathcal{D}$ of $\mathcal{R}$ satisfies $\textsc{Type}(X, R, T)$ on $X \in \mathrm{att}(R)$, if $t(X) \in \Delta_T$ for every $t \in R^{\mathcal{D}}$. Given an instance $\mathcal{D}$ of a relational schema $\mathcal{R}$ and a constraint C on $\mathcal{R}$, we write $\mathcal{D} \vDash C$ to denote that $\mathcal{D}$ satisfies C.

A *relational schema $\mathcal{R}$ with constraints Σ* consists of a relational schema $\mathcal{R}$ and a set of constraints $\Sigma = \sigma \cup \delta$ on $\mathcal{R}$, such that (1) σ contains exactly one primary key declaration $\mathrm{PK}(\bar{X}, R)$ for each $R \in \mathcal{R}$, and (2) $\mathrm{UNQ}(\bar{Y}, S) \in \delta$ for all $\mathrm{FK}(\bar{X}, R, \bar{Y}, S) \in \sigma$, as usual in all SQL implementations. W.l.o.g., we also assume that (3) for every $\mathrm{PK}(\bar{X}, R) \in \sigma$, $\mathrm{UNQ}(\bar{X}, R) \in \delta$ and $\mathrm{NN}(X, R) \in \delta$ for every $X \in \bar{X}$. These data constraints are clearly implied by the PK constraint, but making them explicit will simplify the presentation. Given a relational schema $\mathcal{R}$ with constraints Σ and an instance $\mathcal{D}$ of $\mathcal{R}$, we call $\mathcal{D}$ a *legal instance* of $\mathcal{R}$ with Σ denoted by $\mathcal{D} \vDash \Sigma$, if $\mathcal{D}$ satisfies all constraints in Σ.

Example 1. *For a running example, consider a relational database consisting of relation schemas:* `Emp` *for employees,* `Acc` *for expense accounts,* `Prj` *for research projects, as well as* `Asg` *for the m:n relation that assigns employees to projects:*

```
create table Emp (E_id integer primary key, Name varchar not null,
                  Post varchar);
create table Acc (A_id integer primary key, Name varchar unique);
create table Prj (P_id integer primary key, Name varchar not null,
        ToAcc integer not null unique foreign key references Acc(A_id));
create table Asg (ToEmp integer foreign key references Emp(E_id),
                  ToPrj integer foreign key references Prj(P_id),
                  primary key (ToEmp,ToPrj));
```

RDF Graph. Assume that $\mathcal{I}, \mathcal{B}$ and $\mathcal{L}$ are countably infinite disjoint sets of *Internationalized Resource Identifiers* (IRIs), *blank nodes* and *literals*, respectively. An *RDF triple* is defined as a triple $\langle s, p, o\rangle$, where $s \in \mathcal{I} \cup \mathcal{B}$ is called the subject, $p \in \mathcal{I}$ is called the predicate and $o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ is called the object. An *RDF graph* $G \subset (\mathcal{I} \cup \mathcal{B}) \times \mathcal{I} \times (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$ is a finite set of RDF triples.

Definition 1 (Nodes). *The set of* nodes *of an RDF graph G is the set of subjects and objects of triples in the graph, i.e.* $\{s, o \mid \langle s, p, o\rangle \in G\}$.

SHACL Constraints. SHACL [7] is a language to describe a set of syntactic conditions on RDF graphs. A SHACL document is a set of shapes, called the *shape graph*. When we validate an RDF graph with respect to a shape graph, we call the former the *data graph*.

For the purpose of this paper, each shape in a shape graph S can be expressed as a triple $\langle s, \tau, \phi\rangle$ consisting of a shape IRI s, a *target definition* τ, and a *constraint definition* ϕ. The τ and ϕ are expressions that determine for every data graph G and node n of G, whether n is a target of the shape, $G \models \tau(n)$, resp., whether n satisfies the constraint, $G \models \phi(n)$. All shapes generated by our transformation have an 'implicit target class,' which means that s is also the IRI of a class and $G \models \tau(n)$ iff n is a SHACL instance of class s.[1]

A data graph G validates against a shape $\langle s, \tau, \phi\rangle$ if for all nodes n of G, if $G \models \tau(n)$, then $G \models \phi(n)$. A data graph G validates against a shape graph S, written $G \vDash S$, iff G validates against all shapes in S.

In addition to the *core constraint components* of SHACL (namespace `sh:`), we introduce a SPARQL-based constraint component `uq:uniqueValuesForClass` in Sect. 4, p. 10 to translate UNQ constraints equivalent SHACL constraints.

Example 2. *Consider the following SHACL node shape (left) and RDF graph (right):*

[1] https://www.w3.org/TR/shacl/#implicit-targetClass.

```
:Employee a sh:NodeShape, rdfs:Class;
 sh:property [ sh:path :hasID;                    :Julie a :Employee;
             sh:nodeKind sh:Literal;                :hasID "001"^^xsd:int.
             sh:maxCount 1;  sh:minCount 1;       :Magnus a :Employee;
             sh:datatype xsd:int ];                 :hasID "002"^^xsd:int.
 uq:uniqueValuesForClass [ uq:unqProp :hasID;
             uq:unqForClass :Employee ].
```

The shape for :Employee has (1) an implicit class target declaration, meaning that all the members of the :Employee class are target nodes of the node shape, (2) a property shape that declares that all employees must have exactly one ID with data type xsd:int, and (3) a `uq:uniqueValuesForClass` *constraint that declares that there is no other employee with the same ID. An instance of :Employee validates against the node shape if it satisfies both constraints. The data graph on the right validates against the shape, but can be made invalid by changing the ID of Julie to "002"^^xsd:int.*

Direct Mapping. We now briefly review the direct mapping $\mathcal{DM}$ as defined by Sequeda et al. [8], which integrates and extends the functionalities of the proposal of [10] and the W3C recommendation [2]. The input of $\mathcal{DM}$ consists of a relational schema $\mathcal{R}$, a set σ of PK and FK constraints on $\mathcal{R}$, as well as an instance $\mathcal{D}$ of $\mathcal{R}$. The output is an RDF graph with OWL axioms. $\mathcal{DM}$ is described as a set of Datalog rules. Section 4.1 of [8] defines the following Datalog predicates to represent $\mathcal{R}$, σ and $\mathcal{D}$.

1. $\textsc{Rel}(R)$: Indicates that R is a *relation schema* in $\mathcal{R}$.
2. $\textsc{Attr}_n(\bar{X}, R)$: Indicates that $\bar{X} \triangleleft R$, with $|\bar{X}| = n$.
3. $\mathrm{PK}_n(\bar{X}, R)$ and $\mathrm{FK}_n(\bar{X}, R, \bar{Y}, S)$ represent key constraints, as introduced previously, with $|\bar{X}| = |\bar{Y}| = n$.
4. $\textsc{Value}(V, X, t, R)$: V is the value of $X \in \mathrm{att}(R)$ in a tuple with identifier t in $R^{\mathcal{D}}$.

As is usual in Datalog, subscripts are added to predicate symbols to distinguish variants with different arities.

Section 4.2 of [8] gives rules that determine the RDFS classes and properties to be generated, as well as their ranges and domains, from the relational schema: If a relation $R \in \mathcal{R}$ is *not* identified as representing a binary many-to-many relation,[2] it is translated to a class, i.e., $\textsc{Class}(R)$. Each foreign key reference of R is translated to an object property, represented by an OP_{2n} fact, and each attribute $X \in \mathrm{att}(R)$ to a datatype property, i.e., a DTP fact. If a relation $Q \in \mathcal{R}$ *is* identified as a binary relation $\textsc{BinRel}(Q)$, i.e., $\mathrm{att}(Q) = \{\mathrm{X}, \mathrm{Y}\}$ with $\mathrm{PK}_2(X, Y, Q)$ such that X and Y are foreign key references to tables R and T, then the translation will be an object property between R and T.

Section 4.3.1 of [8] gives rules that generate the IRIs for the classes and properties determined by the previous mapping rules: It generates facts with

[2] This identification of binary relations is the main technical difference between the direct mapping of Sequeda et al. [8] and the W3C recommendation [2].

the predicates ClassIRI for the classes, OP_IRI_1 for object properties from binary relations, OP_IRI_{2n} for the other object properties and DTP_IRI for the datatype properties. Figure 1 gives a summary of the predicates and their arguments.

Section 4.3.2 of [8] gives rules that output OWL axioms as OWL/RDF triples from IRIs for classes and properties. We will ignore these and generate SHACL constraints instead. The rules in Sect. 4.4 of [8] generate RDF triples from the instance $\mathcal{D}$ (encoded as Value facts), based on the previously determined classes and properties, and the rules in Sect. 6 generate unsatisfiable OWL axioms in the case where the database instance violates its key constraints.

In our work, we generate SHACL constraints instead of OWL axioms, so we ignore the rules of [8] that output OWL axioms. The remaining rules of [8] can be divided into (1) the set $\mathcal{M}^s$ of rules that generate the classes and properties and their IRIs from the relational schema $\mathcal{R}$ and the set σ of PK and FK constraints, and (2) the set $\mathcal{M}^i$ of rules that transform an instance $\mathcal{D}$ of $\mathcal{R}$ into an RDF graph based on the facts produced by $\mathcal{M}^s$, as well as the constraints σ.

$\text{BinRel}(Q, X, Y, R, X', T, Y')$	$\text{Rel}(Q)$ is a binary schema, i.e.,att(Q)={X,Y} with $\text{PK}_2(X, Y, Q)$, $\text{FK}_1(X, Q, X', R)$ and $\text{FK}_1(Y, Q, Y', T)$
$\text{Class}(R)$	$\text{Rel}(R)$ is a non-binary schema
$\text{OP}_{2n}(\bar{X}, \bar{Y}, R, T)$	Non-binary schema $\text{Rel}(R)$ has $\text{FK}_n(\bar{X}, R, \bar{Y}, T)$ consisting of $n \geq 1$ attributes, i.e., $\lvert\bar{X}\rvert = \lvert\bar{Y}\rvert = n$.
$\text{DTP}(X, R)$	Non-binary schema $\text{Rel}(R)$ has $X \in \text{att}(R)$
$\text{ClassIRI}(R, R^{\text{IRI}})$	R^{IRI} is the IRI of the class generated for $\text{Class}(R)$
$\text{DTP\_IRI}(X, R, X^{\text{IRI}})$	X^{IRI} is the IRI of the datatype property for $\text{DTP}(X, R)$
$\text{OP\_IRI}_1(Q, X, Y, R, X', T, Y', Q^{\text{IRI}})$	Q^{IRI} is the IRI of the object property for the binary schema $\text{BinRel}(Q, X, Y, R, X', T, Y')$
$\text{OP\_IRI}_{2n}(\bar{X}, \bar{Y}, R, T, W^{\text{IRI}})$	W^{IRI} is the IRI of the obj. prop. $\text{OP}_{2n}(\bar{X}, \bar{Y}, R, T)$ generated from $\text{FK}_n(\bar{X}, R, \bar{Y}, T)$ of $\text{Rel}(R)$, $\lvert\bar{X}\rvert = n \geq 1$

Fig. 1. The Datalog predicates used to represent classes and properties in [8].

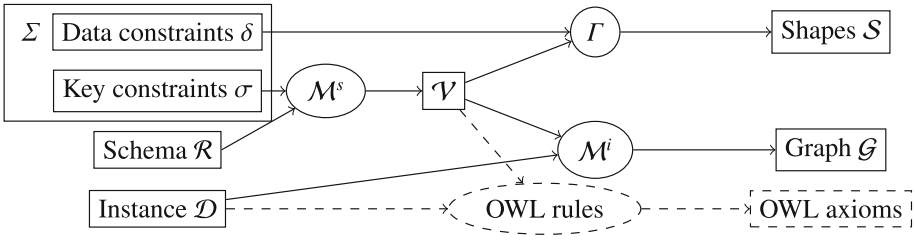

Fig. 2. An overview of direct mapping $\mathcal{M}$. The $\mathcal{DM}$ rules for OWL axioms are not relevant to our work. Γ is the constraint rewriting to be defined in Sect. 4.

Definition 2 (Directly mapped schema). *We denote by* $\mathcal{V} = \mathcal{M}^s(\mathcal{R}, \sigma)$ *the set of all facts with predicates* Class, BinRel, OP_{2n}, DTP, ClassIRI, OP_IRI_1, OP_IRI_{2n} *and* DTP_IRI *that result from (a) representing* $\mathcal{R}$ *and* σ *as Datalog facts, and (b) applying the rules in* $\mathcal{M}^s$ *exhaustively. We call* $\mathcal{V}$ *the* directly mapped schema *given by the direct mapping.*

For illustration, Fig. 3 gives $\mathcal{V}$ for the relational schema of Example 1.

Predicates for schema	Class and property predicates	IRI predicates
Rel(Emp)	Class(Emp)	ClassIRI(Emp, *:Emp*)
Attr_1(Emp, E_id)	DTP(E_id, Emp)	DTP_IRI(E_id, Emp, *:Emp#E_id*)
Attr_1(Emp, Name)	DTP(Name, Emp)	DTP_IRI(Name, Emp, *:Emp#Name*)
Attr_1(Emp, Post)	DTP(Post, Emp)	DTP_IRI(Post, Emp, *:Emp#Post*)
Rel(Prj)	Class(Prj)	ClassIRI(Prj, *:Prj*)
Attr_1(Prj, P_id)	DTP(P_id, Prj)	DTP_IRI(P_id, Prj, *:Prj#P_id*)
Attr_1(Prj, Name)	DTP(Name, Prj)	DTP_IRI(Name, Prj, *:Prj#Name*)
Attr_1(Prj, ToAcc)	DTP(ToAcc, Prj)	DTP_IRI(ToAcc, Prj, *:Prj#ToAcc*)
FK_1(ToAcc, Prj, A_id, Acc)	OP_2(ToAcc, A_id, Prj, Acc)	OP_IRI_2(ToAcc, A_id, Prj, Acc, *:Prj,Acc#ToAcc,A_id*)
Rel(Acc)	Class(Acc)	ClassIRI(Acc, *:Acc*)
Attr_1(Acc, A_id)	DTP(A_id, Acc)	DTP_IRI(A_id, Acc, *:Acc#A_id*)
Attr_1(Acc, Name)	DTP(Name, Acc)	DTP_IRI(Name, Acc, *:Acc#Name*)
Rel(Asg)	BinRel(Asg,ToEmp,ToPrj, Emp,E_id,Prj,P_id)	OP_IRI_1(Asg,ToEmp,ToPrj,Emp,E_id,Prj,P_id, *:Asg#ToEmp,ToPrj,E_id,P_id*)

Fig. 3. The directly mapped schema $\mathcal{V}$ for the relational schema in Example 1. We use QNames, i.e., abbreviations starting with a colon, such as '*:Emp* #E_id', for the IRIs generated by the mappings $\mathcal{M}^s$.

Definition 3 (Directly mapped RDF graph). *We denote by* $\mathcal{G} = \mathcal{M}^i(\mathcal{V}, \sigma, \mathcal{D})$ *the RDF graph that contains one triple* $\langle s, p, o\rangle$ *for each fact* $\text{Triple}(s, p, o)$ *generated by (a) representing* $\mathcal{D}$ *and* σ *as Datalog facts, (b) applying the rules in* $\mathcal{M}^i$ *exhaustively to these facts and the ones in* $\mathcal{V}$*. We call* $\mathcal{G}$ *the* directly mapped RDF graph.

See Fig. 2 for an overview of the different components of the direct mapping. We will write $\mathcal{M}$ for the entire direct mapping defined by applying first $\mathcal{M}^s$ and then $\mathcal{M}^i$. Note that [8, Sect. 4.4.1] contains a rule to generate *blank nodes* for the tuples of relation schemas not having a primary key. This is not needed in our setting since we have assumed that σ contains a PK constraint for every relation $R \in \mathcal{R}$. Finally, note that $\mathcal{M}$ does not interfere with the RDF generation process of $\mathcal{DM}$ except ignoring the rules that generate OWL axioms. Therefore, all the properties [8, Theorem 1 and 2] of $\mathcal{DM}$, except the semantics preservation, are transferable to the $\mathcal{M}$, i.e., $\mathcal{M}$ is information preserving, query preserving and monotonic.

3 Constraint Rewriting: Definition and Properties

We will define a source-to-target constraint rewriting for the direct mapping $\mathcal{M}$ described above. The input of this rewriting is the directly mapped schema $\mathcal{V}$ of a relational database and the set δ of SQL data constraints declared on the schema $\mathcal{R}$ of the database. The output is a set of SHACL shapes.

Let $\mathbb{S}$ be the set of all SHACL shapes and $\mathbb{G}$ the set of all pairs of the form $(\mathcal{V}, \delta)$ such that $\mathcal{V} = \mathcal{M}^s(\mathcal{R}, \sigma)$ is a directly mapped schema of a relational schema $\mathcal{R}$ with constraints $\Sigma = \sigma \cup \delta$, i.e., a set of key constraints σ and data constraints δ.

Definition 4 (Constraint rewriting). *A* constraint rewriting *is a function* $\mathcal{T}: \mathbb{G} \rightarrow \mathcal{P}(\mathbb{S})$.

We are now ready to introduce two fundamental properties of a constraint rewriting: *constraint preservation* and *semantics preservation*.

Definition 5 (Constraint preservation). *A constraint rewriting* $\mathcal{T}$ *is* constraint preserving *if there is a computable mapping* $\mathcal{N} : \mathcal{P}(\mathbb{S}) \rightarrow \mathbb{G}$ *such that for every directly mapped schema* $\mathcal{V} = \mathcal{M}^s(\mathcal{R}, \sigma)$ *of any relational schema* $\mathcal{R}$ *with constraints* $\Sigma = \sigma \cup \delta$, $\mathcal{N}(\mathcal{T}(\mathcal{V}, \delta)) = (\mathcal{V}, \delta)$.

The monotonicity [8, Definition 4] property of direct mapping $\mathcal{M}$ ensures that a re-computation of the entire RDF graph from the database is not required when the database is updated after the mapping. Here, it is straightforward to see that a constraint rewriting according to Definition 4 is independent of the database instance, hence, any updates in the database instance do not influence the rewriting of SHACL constraints for the $\mathcal{M}$. This is in contrast to [8], where the rules in Sect. 6.1 have to produce OWL axioms based on the database *instance* to ensure semantics preservation while contradicting the monotonicity property [8, Theorem 3]. Therefore, when we state the additional properties of a constraint rewriting $\mathcal{T}$ for $\mathcal{M}$ that generates a desired one-to-one correspondence between relational databases and the directly mapped RDF graphs with the SHACL constraints, we preserve all the properties of $\mathcal{M}$.

Definition 6 (Semantics preservation). *A constraint rewriting* $\mathcal{T}$ *is* semantics preserving *if for every relational schema* $\mathcal{R}$ *with constraints* $\Sigma = \sigma \cup \delta$ *and arbitrary instance* $\mathcal{D}$ *of* $\mathcal{R}$*:*

$$\mathcal{D} \vDash \Sigma \Longleftrightarrow \mathcal{G} \vDash \mathcal{S},$$

where $\mathcal{V} = \mathcal{M}^s(\mathcal{R}, \sigma)$, $\mathcal{G} = \mathcal{M}^i(\mathcal{V}, \sigma, \mathcal{D})$ *is the directly mapped RDF graph and* $\mathcal{S} = \mathcal{T}(\mathcal{V}, \delta)$ *is the set of SHACL shapes.*

We recall that the direct mapping $\mathcal{M}$ relies on primary keys to generate IRIs for the tuples [8, Sect. 4.4.1] of relation schemas being translated into RDF, and on foreign key references for object properties. The semantics preservation does not hold if these key constraints are violated in the relational database. Sequeda et al. circumvent this problem in [8, Sect. 6.1] by taking the database instance as an extra argument of the rewriting and generating an unsatisfiable OWL axiom if key constraints are violated. To avoid making the generated constraints depend on the instance, we restrict the notion of semantics preservation to database instances that satisfy the key constraints:

Definition 7 (Weak semantics preservation). *A constraint rewriting* $\mathcal{T}$ *is weakly semantics preserving if for every relational schema* $\mathcal{R}$ *with constraints* $\Sigma = \sigma \cup \delta$ *and arbitrary instance* $\mathcal{D}_\sigma$ *of* $\mathcal{R}$ *that satisfies the key constraints* σ *of* $\mathcal{R}$*:*

$$\mathcal{D}_\sigma \vDash \Sigma \Longleftrightarrow \mathcal{G} \vDash \mathcal{S},$$

where $\mathcal{V} = \mathcal{M}^s(\mathcal{R}, \sigma)$, $\mathcal{G} = \mathcal{M}^i(\mathcal{V}, \sigma, \mathcal{D}_\sigma)$ *is the directly mapped RDF graph and* $\mathcal{S} = \mathcal{T}(\mathcal{V}, \delta)$ *is the set of SHACL constraints.*

Constraint and semantics preservation are two independent properties of a constraint rewriting $\mathcal{T}$: the former is a syntactic property and the latter is semantic. It is possible to define a constraint rewriting that is constraint preserving and not semantics preserving, and vice versa. In the following section, we will define a concrete constraint rewriting Γ that is both constraint preserving and weakly semantics preserving.

4 The Constraint Rewriting Γ

We now define a concrete constraint rewriting Γ. According to Definition 4, Γ takes a directly mapped schema $\mathcal{V} = \mathcal{M}^s(\mathcal{R}, \sigma)$ and a set of data constraints δ and produces a set of SHACL shapes. In order to keep our definitions tightly linked to the definition of $\mathcal{M}$, we define Γ in terms of Datalog rules based on the Datalog facts in $\mathcal{V}$ generated by the rules of $\mathcal{M}^s$, as well as a set of Datalog facts representing the constraints in δ. We will first define the predicates used to represent relational schemata and SHACL constraints, and then give the Datalog rules that define Γ.

4.1 Datalog Predicates

Predicates for relational schemata. To represent a relational schema $\mathcal{R}$ with constraints $\Sigma = \sigma \cup \delta$, we reuse the Datalog representation from [8] as introduced in Sect. 2, i.e., $\textsc{Rel}(R)$, $\textsc{Attr}_n(\bar{X}, R)$, $\text{PK}_n(\bar{X}, R)$ and $\text{FK}_n(\bar{X}, R, \bar{Y}, S)$. Since [8] only uses key constraints, we additionally need the following predicates to represent data constraints.

1. $\textsc{Type}(X, R, T)$ indicates that $\textsc{Attr}_1(X, R)$ has an SQL data type T.
2. $\text{NN}_1(X, R)$ and $\text{UNQ}_n(\bar{X}, R)$ represent not null and unique constraints (see Sect. 2).
3. $\textsc{TypeXml}(X, R, T)$ indicates that $\textsc{Attr}_1(X, R)$ has XML Schema datatype T, e.g., xsd:string, xsd:integer. These have to be generated from the $\textsc{Type}(X, R, T)$ constraints of the database to map SQL data types to XML datatypes; we omit the details.

Predicates for SHACL Syntax. We introduce a number of predicates to express SHACL shapes. A general vocabulary to encode SHACL shapes would have to take the recursive syntax into account, but this is not needed in our rewriting setting: we only require a limited number of different SHACL constraint components, that are easily represented by a few Datalog predicates.

- We generate exactly one shape for each class in $\mathcal{V}$ mapped by $\mathcal{M}^s$. In fact, we use the class IRI to identify the shape, i.e., *sh:NodeShape*, as is done with implicit target classes in SHACL. This means that the *sh:targetClass* declaration is implicit, i.e., also the class IRI.
- For each class, we generate a number of simple property and node shapes, based on the RDF vocabulary in $\mathcal{V}$ corresponding to each class IRI and the SQL constraints in the relational schema.

We use the following predicates to represent SHACL shapes:

1. SHAPE(R): Indicates that the IRI R designates a node shape with implicit class target, i.e.,

   R a sh:NodeShape, rdfs:Class.

   Note that our transformation uses the same IRI to identify node shape and the class target as is done with implicit target class.
2. PROP(R, P, S): Indicates that the node shape R has a property shape that requires the values of the predicate with IRI P to be instances of the *rdfs:Class* identified by the IRI S, i.e.,

   sh:property [sh:path P; sh:nodeKind sh:IRI; sh:class S].

3. DATA(R, P, T): Indicates that the node shape R has a property shape that requires the values of predicate P to be literals with XML Schema datatype T, i.e.,

   sh:property [sh:path P; sh:nodeKind sh:Literal; sh:datatype T].

4. MAXPROP(R, P, S): Indicates that the node shape R has a property shape that requires the predicate P to have at most one value, which belongs to the class S, i.e.,

   sh:property [sh:path P; sh:nodeKind sh:IRI; sh:maxCount 1; sh:class S].

5. CRDPROP(R, P, S): Indicates that the node shape R has a property shape that requires the predicate P to have exactly one value, which belongs to the class S, i.e.,

   sh:property [sh:path P; sh:nodeKind sh:IRI; sh:minCount 1; sh:maxCount 1; sh:class S].

6. INVPROP(R, P, S): Like PROP(R, P, S) but for the inverse path of P:

   sh:property [sh:path [sh:inversePath P]; sh:nodeKind sh:IRI; sh:class S].

7. INVMAXPROP(R, P, S): Like MAXPROP(R, P, S) but for the inverse path of P:

   sh:property [sh:path [sh:inversePath P]; sh:nodeKind sh:IRI; sh:maxCount 1 sh:class S].

8. MAXDATA(R, P, T) and CRDDATA(R, P, T): Like the predicate DATA(R, P, T) with maximum-one and exactly-one cardinality restrictions, respectively. For instance, predicate MAXDATA(R, P, T):

   sh:property [sh:path P; sh:nodeKind sh:Literal; sh:maxCount 1; sh:datatype T].

9. UNQTUPLE$_n$($R, \bar{P}$): Indicates that the node shape R has a constraint that requires the combination of values of the predicates $\bar{P}$ to be unique among all the members of the class R. This cannot be expressed using the core SHACL constraint components, so we define a SHACL-SPARQL constraint

component for the purpose.[3] A Datalog fact $\text{UNQTUPLE}_n(R, P_1, \ldots, P_n)$ then translates to a node shape

uq:uniqueValuesForClass [uq:unqProp $P_1, \ldots, P_n$; uq:unqForClass R].

The uq:uniqueValuesForClass component is defined as follows:

```
@prefix uq: <http://sirius−labs.no/shapes/unique#>
uq:UniqueValuesConstraintComponent a sh:ConstraintComponent ;
  sh:parameter [sh:path uq:uniqueValuesForClass] ;
  sh:nodeValidator [ a sh:SPARQLSelectValidator ;
     sh: select  """SELECT $this ?other WHERE {
       FILTER NOT EXISTS {
          GRAPH $shapesGraph {$uniqueValuesForClass uq:unqProp ?prop}
          $this  ?prop ?thisVal  .
          ?other ?prop ?otherVal .
          FILTER (?thisVal != ?otherVal)
       }
       FILTER (?other != $this)
       GRAPH $shapesGraph {$uniqueValuesForClass uq:unqForClass ?class}
       ?other rdf:type $class  .
    }""" ] .
```

As per the definition of SHACL-SPARQL, the object of the shape triple with predicate :uniqueValuesForClass is accessible in the SPARQL query as $uniqueValuesForClass, and the :unqProp parameters can be accessed using "GRAPH $shapesGraph". The `sh:select` SPARQL query will be evaluated with `$this` bound to each target node in turn. What it does is to search for ?other!=$this resources that do not (FILTER NOT EXISTS) disagree (?thisVal != ?otherVal) on any of the :unqProp properties $P_1, \ldots, P_n$ given by the shape. Note that this uses the pre-bound variable $shapesGraph, which is an optional feature for SHACL-SPARQL processors.[4]

4.2 The Constraint Rewriting Γ Rules

The following set of Datalog rules for Γ act on the directly mapped schema $\mathcal{V}$ produced by $\mathcal{M}^s \subset \mathcal{M}$ together with the set δ of data constraints expressed as Datalog facts, and generates SHACL shapes by using the shape predicates defined in Sect. 4.1.

First, the rule (1) is used to generate SHACL shapes for all the CLASSIRI vocabularies produced by $\mathcal{M}^s$.

$$\text{SHAPE}(R^{\text{IRI}}) \longleftarrow \text{CLASSIRI}(R, R^{\text{IRI}}) \tag{1}$$

For example, SHACL predicates SHAPE(*:Emp*), SHAPE(*:Prj*) and SHAPE(*:Acc*) hold in our Example 1, assuming that *:Emp*, etc., are the IRIs generated by the mappings $\mathcal{M}^s$ from the relation schema names, CLASSIRI(Emp, *:Emp*), etc.

[3] For $n = 1$ the same requirement could be expressed using dash:uniqueValueForClass from http://datashapes.org/constraints.html, but not for larger n.

[4] See https://www.w3.org/TR/shacl/#sparql-constraints-prebound.

Second, the rules (2) to (5) are used to generate SHACL property shapes for the case where the direct mapping identifies a relation schema Q as binary, therefore mapping it to an object property, as opposed to an RDFS class and one property per attribute. In this case, the directly mapped schema $\mathcal{V}$ contains a fact $\text{OP\_IRI}_1(Q, X, Y, R, X', T, Y', Q^{\text{IRI}})$, which expresses that

- $Q(X, Y)$ is a relation schema with exactly two attributes X and Y, and $\text{PK}_2(X, Y, R)$,
- X is a foreign key reference to attribute X' of R,
- Y is a foreign key reference to attribute Y' of T, and
- Q^{IRI} is the IRI of the property generated from Q.

There will also be facts $\text{ClassIRI}(R, R^{\text{IRI}})$ and $\text{ClassIRI}(T, T^{\text{IRI}})$ that give the IRIs of the classes generated from R and T. For instance, for the relation schema '`Asg`' in Example 1, the directly mapped schema $\mathcal{V}$ in Fig. 3 contains facts $\text{ClassIRI}(\texttt{Emp}, \textit{:Emp})$, $\text{ClassIRI}(\texttt{Prj}, \textit{:Prj})$ and

$$\text{OP\_IRI}_1(\texttt{Asg}, \texttt{ToEmp}, \texttt{ToPrj}, \texttt{Emp}, \texttt{E\_id}, \texttt{Prj}, \texttt{P\_id}, \textit{:Asg\#}ToEmp, ToPrj, E\_id, P\_id).$$

In general, since Q^{IRI} is a many-to-many relation, the only constraints that can be guaranteed on the directly mapped RDF concern the type of the involved nodes. However, if there is a UNQ constraint on X, we can conclude that elements of R^{IRI} can participate in at most one Q^{IRI} triple, and similarly if there is a UNQ constraint for Y. The following two rules generate a property shape for Q^{IRI} with or without maximum cardinality 1, depending on whether is a UNQ constraint on X of Q or not:

$$\text{MaxProp}(R^{\text{IRI}}, Q^{\text{IRI}}, T^{\text{IRI}}) \longleftarrow \text{UNQ}_1(X, Q), \text{OP\_IRI}_1(Q, X, Y, R, X', T, Y', Q^{\text{IRI}}), \text{ClassIRI}(R, R^{\text{IRI}}), \text{ClassIRI}(T, T^{\text{IRI}}) \quad (2)$$

$$\text{Prop}(R^{\text{IRI}}, Q^{\text{IRI}}, T^{\text{IRI}}) \longleftarrow \neg\text{UNQ}_1(X, Q), \text{OP\_IRI}_1(Q, X, Y, R, X', T, Y', Q^{\text{IRI}}), \text{ClassIRI}(R, R^{\text{IRI}}), \text{ClassIRI}(T, T^{\text{IRI}}) \quad (3)$$

Note that some of our rules use negated atoms: e.g., $\neg\text{UNQ}_1(X, Q)$ indicates that there is *no* unique constraint for the attribute $X \triangleleft Q$. The next rules do the same for the inverse direction of Q^{IRI}:

$$\text{InvMaxProp}(T^{\text{IRI}}, Q^{\text{IRI}}, R^{\text{IRI}}) \longleftarrow \text{UNQ}_1(Y, Q), \text{OP\_IRI}_1(Q, X, Y, R, X', T, Y', Q^{\text{IRI}}), \text{ClassIRI}(R, R^{\text{IRI}}), \text{ClassIRI}(T, T^{\text{IRI}}) \quad (4)$$

$$\text{InvProp}(T^{\text{IRI}}, Q^{\text{IRI}}, R^{\text{IRI}}) \longleftarrow \neg\text{UNQ}_1(Y, Q), \text{OP\_IRI}_1(Q, X, Y, R, X', T, Y', Q^{\text{IRI}}), \text{ClassIRI}(R, R^{\text{IRI}}), \text{ClassIRI}(T, T^{\text{IRI}}) \quad (5)$$

Third, the rules (6) to (9) are used to generate the SHACL property shapes for the object properties that stem from foreign key references in relations that were not identified as binary. For these, $\mathcal{M}^s$ generates Datalog facts $\text{OP\_IRI}_{2n}(\bar{X}, \bar{Y}, R, T, W^{\text{IRI}})$ where

- $\bar{X}$ are some attributes of R and $\bar{Y}$ are some attributes of T,
- There is a foreign key constraint $\mathrm{FK}_n(\bar{X}, R, \bar{Y}, T)$ from R to T, and
- W^{IRI} is the IRI constructed for this object property.

E.g., a Datalog fact $\mathrm{OP\_IRI}_2(\texttt{ToAcc,A\_id,Prj,Acc,}\textit{:Prj,Acc}\#\textit{ToAcc}, \textit{A\_id})$ is generated for the foreign key reference of schema '`Prj`' in Fig. 3.

Since the direct mapping produces one resource per tuple in an instance of R, and a W^{IRI} triple only for non-null attribute values, the property W^{IRI} will have a cardinality of 'at most 1.' If there is additionally a non-null constraint for the attributes $\bar{X}$, the cardinality will be 'exactly 1.' We use the notation $\overline{\mathrm{NN}}(\bar{X}, R)$ in a rule to mean that $\mathrm{NN}_1(X, R)$ is present for all $X \in \bar{X}$. The following rules generate property paths with a maximum cardinality, and with or without a minimum cardinality depending on the presence of NN constraints:

$$\textsc{CrdProp}(R^{\mathrm{IRI}}, W^{\mathrm{IRI}}, T^{\mathrm{IRI}}) \longleftarrow \overline{\mathrm{NN}}(\bar{X}, R), \mathrm{OP\_IRI}_{2n}(\bar{X}, \bar{Y}, R, T, W^{\mathrm{IRI}}), \textsc{ClassIRI}(R, R^{\mathrm{IRI}}), \textsc{ClassIRI}(T, T^{\mathrm{IRI}}) \tag{6}$$

$$\textsc{MaxProp}(R^{\mathrm{IRI}}, W^{\mathrm{IRI}}, T^{\mathrm{IRI}}) \longleftarrow \neg\overline{\mathrm{NN}}(\bar{X}, R), \mathrm{OP\_IRI}_{2n}(\bar{X}, \bar{Y}, R, T, W^{\mathrm{IRI}}), \textsc{ClassIRI}(R, R^{\mathrm{IRI}}), \textsc{ClassIRI}(T, T^{\mathrm{IRI}}) \tag{7}$$

The following two rules are for the inverse direction from relation schema T to R. The crucial observation here is that if there is a constraint $\mathrm{UNQ}_n(\bar{X}, R)$, then the inverse property of W^{IRI} has a maximum cardinality of '1'. Otherwise, the typing is the only guarantee we have on the inverse.

$$\textsc{InvMaxProp}(T^{\mathrm{IRI}}, W^{\mathrm{IRI}}, R^{\mathrm{IRI}}) \longleftarrow \mathrm{UNQ}_n(\bar{X}, R), \mathrm{OP\_IRI}_{2n}(\bar{X}, \bar{Y}, R, T, W^{\mathrm{IRI}}), \textsc{ClassIRI}(R, R^{\mathrm{IRI}}), \textsc{ClassIRI}(T, T^{\mathrm{IRI}}) \tag{8}$$

$$\textsc{InvProp}(T^{\mathrm{IRI}}, W^{\mathrm{IRI}}, R^{\mathrm{IRI}}) \longleftarrow \neg\mathrm{UNQ}_n(\bar{X}, R), \mathrm{OP\_IRI}_{2n}(\bar{X}, \bar{Y}, R, T, W^{\mathrm{IRI}}), \textsc{ClassIRI}(R, R^{\mathrm{IRI}}), \textsc{ClassIRI}(T, T^{\mathrm{IRI}}) \tag{9}$$

Fourth, the rules (10) and (11) handle the datatype properties that are generated by $\mathcal{M}^s$ for every attribute of a non-binary relation schema. A fact $\mathrm{DTP\_IRI}(X, R, X^{\mathrm{IRI}})$ in $\mathcal{V}$ denotes mapping of an attribute $X \in \mathrm{att}(R)$ to a datatype property with IRI X^{IRI}. For instance, we have facts like $\mathrm{DTP\_IRI}(\texttt{Name}, \texttt{Emp}, \textit{:Emp}\#\textit{Name})$, etc., for Example 1 in Fig. 3. The following rules treat the case with and without an NN constraint on an attribute X.

$$\textsc{MaxData}(R^{\mathrm{IRI}}, X^{\mathrm{IRI}}, T) \longleftarrow \neg\mathrm{NN}_1(X, R), \mathrm{DTP\_IRI}(X, R, X^{\mathrm{IRI}}), \textsc{TypeXml}(X, R, T), \textsc{ClassIRI}(R, R^{\mathrm{IRI}}) \tag{10}$$

$$\textsc{CrdData}(R^{\mathrm{IRI}}, X^{\mathrm{IRI}}, T) \longleftarrow \mathrm{NN}_1(X, R), \mathrm{DTP\_IRI}(X, R, X^{\mathrm{IRI}}), \textsc{TypeXml}(X, R, T), \textsc{ClassIRI}(R, R^{\mathrm{IRI}}) \tag{11}$$

Finally, rule (12) generates node shapes that reflect UNQ constraints on (combinations of) attributes. For all $n \geq 1$:

$$\textsc{UnqTuple}_n(R^{\text{IRI}}, X_1{}^{\text{IRI}}, \ldots, X_n{}^{\text{IRI}}) \longleftarrow \text{UNQ}_n(\bar{X}, R), \quad (12)$$
$$\text{DTP\_IRI}(X_1, R, X_1{}^{\text{IRI}}), \ldots, \text{DTP\_IRI}(X_n, R, X_n{}^{\text{IRI}}),$$
$$\textsc{ClassIRI}(R, R^{\text{IRI}})$$

Example 3. *The following SHACL predicates result from the application of rewriting rules (1)–(12) on the relations schemas stated in Example 1.*

$\textsc{Shape}(:Emp)$	*by* Γ *rule 1*
$\textsc{CrdData}(:Emp,\ :Emp\#E\_id,\ xsd{:}integer)$;	*by* Γ *rule 11*
$\textsc{UnqTuple}_1(:Emp,\ :Emp\#E\_id)$;	*by* Γ *rule 12*
$\textsc{CrdData}(:Emp,\ :Emp\#Name,\ xsd{:}string)$;	*by* Γ *rule 11*
$\textsc{MaxData}(:Emp,\ :Emp\#Post,\ xsd{:}string)$;	*by* Γ *rule 10*
$\textsc{Prop}(:Emp,\ :Asg\#ToEmp,ToPrj,E\_id,P\_id,\ :Prj)$.	*by* Γ *rule 3*
$\textsc{Shape}(:Prj)$	*by* Γ *rule 1*
$\textsc{CrdData}(:Prj,\ :Prj\#P\_id,\ xsd{:}integer)$;	*by* Γ *rule 11*
$\textsc{UnqTuple}_1(:Prj,\ :Prj\#P\_id)$;	*by* Γ *rule 12*
$\textsc{CrdData}(:Prj,\ :Prj\#Name,\ xsd{:}string)$;	*by* Γ *rule 11*
$\textsc{CrdData}(:Prj,\ :Prj\#ToAcc,\ xsd{:}integer)$;	*by* Γ *rule 11*
$\textsc{UnqTuple}_1(:Prj,\ :Prj\#ToAcc)$;	*by* Γ *rule 12*
$\textsc{CrdProp}(:Prj,\ :Prj,Acc\#ToAcc,A\_id,\ :Acc)$;	*by* Γ *rule 6*
$\textsc{InvProp}(:Prj,\ :Asg\#ToEmp,ToPrj,E\_id,P\_id,\ :Emp)$.	*by* Γ *rule 5*
$\textsc{Shape}(:Acc)$	*by* Γ *rule 1*
$\textsc{CrdData}(:Acc,\ :Acc\#A\_id,\ xsd{:}integer)$;	*by* Γ *rule 11*
$\textsc{UnqTuple}_1(:Acc,\ :Acc\#A\_id)$;	*by* Γ *rule 12*
$\textsc{MaxData}(:Acc,\ :Acc\#Name,\ xsd{:}string)$;	*by* Γ *rule 10*
$\textsc{InvMaxProp}(:Acc,\ :Prj,Acc\#ToAcc,A\_id,\ :Prj)$;	*by* Γ *rule 8*

We refer to our technical report [9, Appendix C.4] for the complete translation of these SHACL predicates into the SHACL document.

5 Properties of the Constraint Rewriting Γ

We now study the properties of our constraint rewriting Γ for the direct mapping $\mathcal{M}$: constraint preservation and semantics preservation, defined in Sect. 3.

First, we show that the constraint rewriting Γ does not lose any SQL data constraints of the relational database that is being translated into the RDF graph:

Theorem 1. *The constraint rewriting Γ is constraint preserving.*

Proof Outline: We explicitly define an inverse mapping $\mathcal{N} : \mathcal{P}(\mathbb{S}) \to \mathbb{G}$ of Γ. Then, letting $(\mathcal{V}', \delta') = \mathcal{N}(\Gamma(\mathcal{V}, \delta))$, we show that $\mathcal{V}' = \mathcal{V}$ and $\delta' = \delta$, using a

case distinction over all facts in $\mathcal{V}$ and δ and all possible shapes in $\Gamma(\mathcal{V}, \delta)$. We refer to our technical report [9, Appendix B.1] for the complete proof.

Second, we establish that the constraint rewriting Γ for direct mapping $\mathcal{M}$ is not semantics preserving. For that, we first recall that mapping $\mathcal{M}$ does not generate: (1) an IRI from a null value, (2) distinct IRIs for the repeated tuples of the relation schema. These facts can be used to construct a counterexample to show that mapping $\mathcal{M}$ generates a consistent RDF graph w.r.t. the generated SHACL constraints when the primary keys of input database are violated. Observe that in Example 4, an obstacle to obtain a semantic-preserving constraint rewriting Γ for $\mathcal{M}$ is the semantics of direct mapping $\mathcal{M}$ w.r.t. the PKs of relation schemas.

Example 4. *Consider a relation schema "create table User (id integer primary key);" with three tuples:* t_1.`id` $= 1$, t_2.`id` $= 1$ *and* t_3.`id` $=$ null, *respectively, violating the primary key constraint of the schema definition. It is straightforward to see that the directly mapped RDF triples of the tuples of schema '*`User`*' (on the right) validate against the SHACL shape generated from the schema definition of '*`User`*' (on the left), which leads to a contradiction w.r.t. the Definition 6.*

Shape(*:User*)	*:User/id=1 rdf:type :User.*
CrdData(*:User*, *:User#id*, *xsd:integer*)	*:User/id=1 :User#id 1.*
UnqTuple$_1$(*:User*, *:User#id*).	

Proposition 1. *The constraint rewriting Γ is not semantics preserving.*

Finally, we study the weak semantics preservation property of Γ.

Example 5. *Consider a database instance such as:* `Emp`*(011, Ida, PhD),* `Prj`*(021, PeTWIN, 034),* `Acc`*(034, NFR) and* `Asg`*(012, 022), of the relational schema given in Example 1, violating the foreign keys* `ToEmp` *and* `ToPrj` *of the relation schema* `Asg`*. Then, $\mathcal{M}$ generates the following RDF triples:*

```
:Emp/E_id=011 rdf:type :Emp;   :Prj/P_id=021 rdf:type :Prj;              :Acc/A_id=034 rdf:type :Acc;
  :Emp#E_id  011;                :Prj#P_id  021; :Prj#Name "PeTWIN";       :Acc#A_id  034;
  :Emp#Name  "Ida";              :Prj#ToAcc  034;                          :Acc#Name  "NFR".
  :Emp#Post  "PhD".              :Prj,Acc#ToAcc,A_id :Acc/A_id=034.
```

satisfying the SHACL shapes given by the SHACL predicates produced by the rewriting Γ in Example 3.

Observe that in Example 5, the SHACL shapes resulted by the Γ in Example 3 fail to detect the violation of foreign keys of the relation schema `Asg`, essentially because the rewriting Γ for $\mathcal{M}$ does not generate SHACL constraints for the binary schema. The main reason behind this flaw is that the direct mapping $\mathcal{M}$ does not generate a class for the binary schema, and hence, the rewriting Γ does not produce a SHACL shape for the schema `Asg` to capture the violations of `ToEmp` and `ToPrj` foreign keys. Likewise, observe that if we change the `Prj` tuple in Example 5 to `Prj`(021, PeTWIN, 031), violating the foreign key `ToAcc`, and we remove the NN constraint on `ToAcc` from the schema `Prj` then the

node '*:Prj/P_id=021*' validates the generated SHAPE(*:Prj*). That means that only the 'not null' SQL data constraints on FKs is sufficient for the constraint rewriting Γ to detect the violation of FKs on the relational schema.

In summary, we observe: (1) The direct mapping $\mathcal{M}$ as defined by Sequeda et al., including the W3C recommendation [2], generates one resource per tuple of a database instance (for non-binary relation schemata) and the IRI of this tuple is generated from the relation's primary key. This approach breaks down if the PK constraint is violated, which explains why semantics preservation does not hold as stated. (2) The binary relations 'BINREL' rule as defined by the direct mapping $\mathcal{M}$ of Sequeda et al., but not in the W3C recommendation, are not suitable for the SHACL constraint rewriting Γ since the mapping $\mathcal{M}$ does not generate a class for these binary schemas in the relational database. (3) Not all data constraints on FKs are strong enough to guarantee the semantics of FKs on relation schema in the SHACL constraint rewriting Γ. However, if we restrict our attention to the database instances $\mathcal{D}_\sigma$ that satisfy their PKs and FKs constraints, then the semantics preservation is restored.

Theorem 2. *The constraint rewriting Γ is weakly semantics preserving.*

Proof Outline: First, we show that the direct mapping $\mathcal{G}$ of a legal instance $\mathcal{D}$ of the relational schema with constraints satisfies all shapes generated by Γ, by a case distinction over the possible shapes. This involves a detailed analysis of the directly mapped triples and the semantics of the SHACL shapes. For the other direction, we show that every possible violation of a data constraint in a database instance $\mathcal{D}$ that satisfies the key constraints entails that $\mathcal{G}$ fails to validate at least one of the SHACL constraints generated by Γ. This requires a case distinction over the data constraints δ and their role for the direct mapping. We refer to our technical report [9, Appendix B.1] for the complete proof.

Finally, in summary, the constraint rewriting Γ defined in Sect. 4 is both constraint preserving and weakly semantics preserving.

6 Discussion

We have extended the direct mapping $\mathcal{M}$ from relational data to RDF, proposed in [8], with SHACL constraints by using the SQL data constraints, including data types which were missing from both previous extensions of direct mappings [3,8]. All of the good properties of the original extension of direct mapping [8] apply to our extension. Contrary to previous work, our extension describes the mapped data using SHACL constraints instead of OWL axioms. This is what makes our mapping (weakly) semantic preserving.

We note that our constraint rewriting Γ, specified in Sect. 4.2, is not semantics preserving if: (1) relation schemas without PKs are considered. This is because $\mathcal{M}$ produces blank nodes for the tuples of relation schemas without PKs, which are problematic for core SHACL but could be handled by extensions to the core;[5] and (2) relational databases violating the PKs and FKs constraints

[5] e.g., using SPARQL-based Target Types in https://www.w3.org/TR/shacl-af/.

are considered for the constraint rewriting Γ, because the mapping $\mathcal{M}$ often produces an RDF graph that is consistent w.r.t. the generated SHACL shapes even if the PKs and FKs are violated in the source database. This cannot easily be fixed, since $\mathcal{M}$ relies on the uniqueness of PKs to generate distinct RDF resources.

The interest of database instances that might violate their primary or foreign key constraints lies purely in the formulation of the 'completeness' direction (right to left) of semantics preservation. Stating that every database constraint violation entails a violation of a shape on the RDF graph means that the shapes are 'strong enough,' they give the strongest possible guarantee on the shape of RDF graphs produced by $\mathcal{M}$. We have seen in the discussion following Example 4 and 5 that the 'semantics preservation' approach does not work well with key constraints unless the database instance is explicitly included in the constraint rewriting, similar to the work of Sequeda et al., to trigger the unsatisfiability of the directly mapped graph whenever keys are violated in the source database. However, Sequeda et al. [8, Theorem 3] established that the desirable condition of direct mapping being monotone is an obstacle to obtain a semantics preserving even if the database instance is explicitly included in the constraint rewriting. Therefore, we believe that, instead of relying on the database instance, a more useful formulation of the completeness of constraint rewriting for direct mapping can be found, such as maximally implied SHACL constraints, i.e., completeness meaning that any other SHACL constraints are either not implied by the source constraints, or subsumed by the maximally implied SHACL constraints.

Further, we observe that our constraint rewriting could be extended for relation schemas without PKs in combination with OWL axioms, in a similar manner as shown for the combination of DL-Lite$_{\text{RDFS}}$ axioms and tree-based identification constraints in [3], where the relation schemas without PKs could be used to generate OWL axioms if there exists no foreign key referential integrity constraint between the schemas with and without PKs. However, the presence of referential integrity constraints between schemas with and without PKs might be an obstacle to generate a semantic-preserving constraint rewriting in this setting, therefore, we leave this transformation as an open question.

7 Conclusion

In this paper, we have proposed an extension of direct mapping with the constraint rewriting. The constraint rewriting transfers the semantic information of SQL constraints from the relational database to the RDF graph while keeping intact all the good properties of the direct mapping [8], i.e., information preserving, query preserving and monotonic. In contrast to previous work, we have studied the extension of direct mapping with SHACL constraints instead of the OWL axioms. Finally, we have shown that our constraint rewriting extends the original form of direct mapping of relational databases to an RDF graph while guaranteeing constraint and weak semantics preservation.

The SHACL descriptions of a directly mapped RDF graph could be useful for the semantic optimization of SPARQL queries, analogous to the database

constraints that can be used for efficient query answering in an *Ontology-Based Data Access* platform [11]. Further, any ontology alignments that follow the W3C direct mapping directives to connect the ontological vocabulary to the relational database, such as BootOX [6], could be improved by extending bootstrapping with the SHACL description of source data that fits more closely with RDF/OWL representation.

In future work, we would like to concentrate on the more intuitive interpretation of constraint rewriting for the direct mapping specified in denotational semantics [2]. We also aim to extend our constraint rewriting from direct mapping to the interrelated and complementary W3C standard: R2RML [5].

Acknowledgements. This work was supported by the Norwegian Research Council via the SIRIUS Centre for Research Based Innovation, Grant Nr. 237898. We thank Evgeny Kharlamov and Egor Kostylev for many fruitful discussions, Holger Knublauch for help with SHACL-SPARQL, and Roman Kontchakov for invaluable assistance in preparing the final version.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases, vol. 8. Addison-Wesley, Reading (1995)
2. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A direct mapping of relational data to RDF. W3C recommendation, W3C, September 2012
3. Calvanese, D., Fischl, W., Pichler, R., Sallinger, E., Simkus, M.: Capturing relational schemas and functional dependencies in RDFS. In: Twenty-Eighth AAAI Conference on Artificial Intelligence (2014)
4. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014
5. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language, W3C recommendation, W3C (2012)
6. Jiménez-Ruiz, E., et al.: BootOX: practical mapping of RDBs to OWL 2. In: Arena, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 113–132. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_7
7. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C recommendation, W3C, July 2017
8. Sequeda, J.F., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: Proceedings of 21st International Conference on World Wide Web, pp. 649–658. ACM (2012)
9. Thapa, R.B., Giese, M.: A source-to-target constraint rewriting for direct mapping. Technical report Nr 498, Department of Informatics, University of Oslo, July 2021
10. Tirmizi, S.H., Sequeda, J., Miranker, D.: Translating SQL applications to the semantic web. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2008. LNCS, vol. 5181, pp. 450–464. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85654-2_40
11. Xiao, G., Calvanese, D., Kontchakov, R., et al.: Ontology-based data access: a survey. In: Proceedings of International Joint Conference on Artificial Intelligence. Survey Track, pp. 5511–5519. IJCAI Organization (2018)