Data & Knowledge Engineering 145 (2023) 102157

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Conceptually-grounded mapping patterns for Virtual Knowledge Graphs

Diego Calvanese [a,b], Avigdor Gal [c], Davide Lanti [a,*], Marco Montali [a], Alessandro Mosca [a], Roee Shraga [d]

[a] Free-University of Bozen-Bolzano, Bolzano, Italy
[b] Umeå University, Umeå, Sweden
[c] Technion – Israel Institute of Technology, Haifa, Israel
[d] Khoury College of Computer Science, Northeastern University, Boston, MA, United States of America

ARTICLE INFO

Keywords:
Virtual knowledge graphs
Ontology-based data access
Mapping patterns
Data integration

ABSTRACT

Virtual Knowledge Graphs (VKGs) constitute one of the most promising paradigms for integrating and accessing legacy data sources. A critical bottleneck in the integration process involves the definition, validation, and maintenance of mapping assertions that link data sources to a domain ontology. To support the management of mappings throughout their entire lifecycle, we identify a comprehensive catalog of sophisticated mapping patterns that emerge when linking databases to ontologies. To do so, we build on well-established methodologies and patterns studied in data management, data analysis, and conceptual modeling. These are extended and refined through the analysis of concrete VKG benchmarks and real-world use cases, and considering the inherent impedance mismatch between data sources and ontologies. We validate our catalog on the considered VKG scenarios, showing that it covers the vast majority of mappings present therein.

1. Introduction

Data integration and access to legacy data sources using end-user oriented languages are increasingly challenging contemporary organizations. In the whole spectrum of data integration and access solutions, the approach based on *Virtual Knowledge Graphs (VKGs)* is gaining momentum [1], especially when the underlying data sources to be integrated come in the form of relational databases (DBs) [2]. VKGs replace the rigid structure of tables with the flexibility of a graph that incorporates domain knowledge and is kept virtual, eliminating the need of making a copy of the data as in a typical ETL-based (Extract, Transform, Load) approach, thus avoiding duplication of data and guaranteeing the freshness of the information being accessed. A VKG specification consists of three main components: *(i) data sources* (in the context of this paper, constituted by relational DBs) where the actual data are stored; *(ii)* a domain *ontology*, capturing the relevant concepts, relations, and constraints of the domain of interest; *(iii)* a set of *mappings* linking the data sources to the ontology. One of the most critical bottlenecks towards the adoption of the VKG approach, especially in complex, enterprise scenarios, is the definition and management of mappings.

Mappings play a central role in a variety of data management tasks, within both the Semantic Web and the DB communities, and come in different forms. In *schema matching*, for example, mappings (typically referred to as "matches") aim at expressing correspondences between atomic, constitutive elements of two different relational schemata, such as attributes and relation names [3]. In this context, very sophisticated (semi-)automatic techniques are being developed to bootstrap this simple type of mappings,

* Corresponding author.
E-mail address: lanti@inf.unibz.it (D. Lanti).

https://doi.org/10.1016/j.datak.2023.102157
Received 24 March 2022; Received in revised form 7 October 2022; Accepted 25 February 2023
Available online 4 March 2023

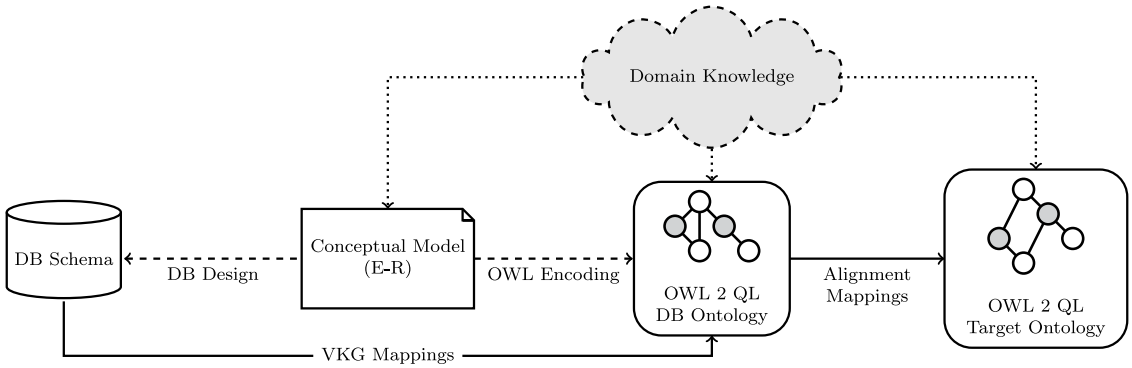


Fig. 1. The database and the ontology both stem from common domain knowledge.

without prior knowledge on the two schemata [4–6]. A similar setting arises in the context of *ontology matching* (also referred to as *ontology alignment*), where the atomic elements to be put into correspondence are concepts and properties [7]. Just like with schema matching, a huge body of applied research has led to effective (semi-)automatic techniques for establishing mappings [8,9]. In *data exchange*, instead, more complex mapping specifications (like the well-known formalism of TGDs [10–12]) are needed to express how data extracted from a source DB schema should be used to populate a target DB schema [13]. Due to the complex nature of these mappings, research in this field has been mainly foundational, with few notable exceptions [14,15].

The VKG approach appears to be the one that poses the most advanced challenges when it comes to mapping specification, debugging, and maintenance. On the one hand, VKG mappings are inherently more sophisticated than those used in schema and ontology matching. On the other hand, while they appear to resemble those typically used in data exchange, they need to overcome the abstraction mismatch between the relational schema of the underlying data storage, and the target ontology; consequently, they are required to explicitly handle how (tuples of) data values extracted from the DB lead to the creation of corresponding objects in the ontology.

It is not surprising, then, that management of VKG mappings throughout their entire lifecycle is currently a labor-intensive and mostly manual effort, which requires highly-skilled professionals [16] that, at once: *(i)* have in-depth knowledge of the domain of discourse and how it can be represented using structural conceptual models (such as UML class diagrams) and ontologies; *(ii)* possess the ability to understand and query the logical and physical structure of the DB; and *(iii)* master languages, methodologies, and technologies for representing the ontology and the mappings using standard frameworks from Semantic Web (such as the OWL 2 profiles and R2RML). Even in the presence of all these skills, writing mappings is demanding and poses a number of challenges related to semantics, correctness, and performance. More concretely, no comprehensive approach currently exists to support ontology engineers and knowledge scientists [17] in the creation of VKG mappings, exploiting all the involved information artifacts to their full potential: the **relational schema** with its constraints and the extensional information stored in the DB, the **ontology axioms**, and a **conceptual model** that lies, explicitly or implicitly, at the basis of the relational schema.

Bootstrapping techniques [18,19] have been developed to relieve the ontology engineer from the "blank paper syndrome". However, they are typically adopted in scenarios where neither the ontology nor the mappings are initially available, and various assumptions are posed over the schema of the DB (e.g., in terms of normalization). Hence, they essentially bootstrap at once the ontology as a "query-preserving" [20] mirrored image of the DB, and the corresponding one-to-one mappings. These approaches typically work at the level of DB schemata, ignoring the data, and therefore they might fail in those cases where the DB schema is either poorly structured or the applicable constraints are not fully specified.

Most of research so-far has been focused on bootstrapping, and a common trait of all these works is that they gloss over, or altogether ignore, the actual conceptual model underlying the given database instance, arguing that such conceptual model is in many cases not available to the bootstrapper. This choice, however, leads to ambiguities and arbitrary decisions: the same relational schema might, in fact, correspond to several different conceptual representations. We here take a completely different approach: we single out mapping patterns by fully accounting for the intended conceptual representation and the database schema corresponding to such representation. This allows us to define each pattern in a non-ambiguous way, and to precisely specify the constraints to be imported into the VKG.[1]

The justification for our approach is depicted in Fig. 1, and foresees a scenario where both the ontology and the DB schema are derived from a conceptual analysis of the domain of interest. The resulting knowledge may stay implicit, or lead to an explicit representation in the form of a structured conceptual model [21], usually represented using well-established notations such as UML, ORM 2, or E-R. On the one hand, this conceptual model provides the basis for creating a corresponding domain ontology through a series of transformation steps, where these steps should ideally preserve the semantics of the original model, modulo the expressivity of the considered ontology language [22–24]. On the other hand, it can trigger the design process that finally leads to the deployment of an actual DB. This is done via a series of restructuring and adaptation steps, considering a number of aspects that go beyond

[1] Modulo the expressive power of the ontology language being considered.

pure conceptualization, such as query load, performance, volume, and the abstraction gap that exists between the conceptual and logical/physical layers. It is precisely the reconciliation of these two transformation chains (resp., from the conceptual model to the ontology, and from the conceptual model to the DB) that is reflected in the VKG mappings.

From this key observation, we derive a catalog of *mapping patterns* that emerge when linking DBs to ontologies. To do so, we build on well-established methodologies and patterns studied in data management (such as W3C Direct Mapping – W3C-DM [25] – and its extensions), data analysis (such as algorithms for discovering dependencies), and conceptual modeling (such as well-known *lossless* transformations from E-R diagrams to DB schemata [26–28]). These are suitably extended and refined, by considering the inherent impedance mismatch between data sources and ontologies, which requires to handle the creation of objects starting from DB values.

The idea of mapping patterns is not new, and was first introduced in [29], later refined in [30]. However, there are substantial differences between the patterns discussed in this line of works and the patterns we introduce here. Specifically, those patterns are usually informally specified and quite permissive, not grounding the DB instance to any particular conceptual representation. This allows the KG practitioner to map KGs that are potentially very different from the intended DB conceptualization. On the contrary, each of our patterns explicitly and non-ambiguously specifies the link between the conceptualization and the DB instance, which is the one arising from applying well-known and *semantics-preserving* transformations studied in the area of DB design. Our choice is motivated by the observation that the conceptual structures "encoded" in the DB instance are not arbitrary, but derive from the design phase of the DB and reflect the actual domain knowledge, and from the fact that the expressive power of the ontology language commonly adopted for VKGs is comparable to that of E-R diagrams.

This foundational grounding, which clearly distinguishes our work from related literature, paves the way towards a variety of VKG design scenarios, depending on which information artifacts are available, and which ones must be produced. For example, our patterns could be used to validate existing mappings, to generate mappings and an ontology if only the DB is available, or even also as a basis for reconstructing the conceptual model when it is left implicit or inaccessible.

Another major contribution of this work, which to the best of our knowledge distinguishes it from all the related literature, is an evaluation of our approach. Concretely, we analyze six concrete VKG scenarios and benchmarks, and report on the coverage of mappings appearing therein in terms of our patterns, as well as on how many times the same pattern recurs. This also gives an interesting indication on which patterns are more pervasively used in practice.

As a final remark, the patterns we introduce here were developed while satisfying the restrictions imposed by the VKG setting (e.g., at the level of expressiveness of the ontology language). However, they can be directly applied in all those contexts where such restrictions are satisfied, regardless of whether the knowledge graph is virtual.

The remainder of this paper is structured as follows: Section 2 introduces the notation and basic notions on VKGs; Section 3 contains our catalog of mapping patterns, the main contribution of this work; Section 4 discusses possible applications of the catalog, depending on the information artifacts that are available to the VKG designer; Section 5 presents an evaluation of the catalog over six different scenarios of different complexities; Section 6 discusses related work, and Section 7 concludes the paper.

2. Preliminaries

In this work, we use the **bold** font to denote tuples, e.g., $\mathbf{x}$, $\mathbf{y}$, are tuples. When convenient and non-ambiguous, we treat tuples as sets and use set operators on them.

We assume that the reader is familiar with standard notions and languages from the relational databases world, such as SQL or E-R diagrams. Other readers might want to refer to the abundant literature on the subject, with [10] as an excellent primer.

We rely on the VKG framework as previously introduced in [31]. A VKG specification is a triple $(\mathcal{T}, \mathcal{M}, \mathcal{S})$ where $\mathcal{T}$ is an *ontology TBox*, $\mathcal{M}$ is a set of *VKG mappings* (or, simply, *mappings*), and $\mathcal{S}$ is a DB schema. We next introduce these elements and their semantics.

The schema $\mathcal{S}$ is a pair (Σ, Γ). Σ is the *signature* of $\mathcal{S}$, that is, a set of *table schemata* of the form $T(A_1, \ldots, A_n)$, where T is a *table name* and $A_1, \ldots, A_n$ are *attributes*, each associated to a SQL *datatype*. Γ is a set of database constraints. In this work, Γ consists of *key* and *foreign key* constraints, as well as *inclusion dependencies*. In this work, foreign keys are inclusion dependencies where the *referred* attributes form a (not necessarily *primary*) key.[2] A *database instance* $\mathcal{D}$ for $\mathcal{S}$ is a first-order interpretation mapping each table name to a relation over the interpretation domain, and that *satisfies*[3] the constraints in Γ.

The ontology $\mathcal{T}$ is formulated in OWL 2 QL [33], but for conciseness we here use its logical underpinning, *DL-Lite*$_\mathcal{R}$ [34], slightly enriched to handle datatypes.

Syntax. We fix four enumerable, pairwise-disjoint sets: **NI** of *individuals*, **NC** of *class names*, **NP** of *object property names*, and **ND** of *data property names*. An OWL 2 QL *TBox* $\mathcal{T}$ is a finite set of *axioms* of the following form:

$B \sqsubseteq C$	$q \sqsubseteq r$
$\rho(d) \sqsubseteq f$	$d \sqsubseteq v$

[2] Although some systems, like MySQL, do not impose particular restrictions over the referred attributes, the common assumption in the literature is that the referred attributes form a *primary key* (e.g., see [32]).

[3] For details, refer to classic literature such as [10].

Table 1
Semantics for OWL 2 QL constructs.

Construct	Syntax element	Example	Semantics
Top class	$\top_C$		Δ_O^I
Top domain	$\top_V$		Δ_V^I
Concept name	$A \in \mathbf{NC}$	Person	$A^I \subseteq \Delta_O^I$
Object property name	$p \in \mathbf{NP}$	hasSpouse	$p^I \subseteq \Delta_O^I \times \Delta_O^I$
Data property name	$d \in \mathbf{ND}$	hasName	$d^I \subseteq \Delta_O^I \times \Delta_V^I$
Datatype	T_i	xsd:int	$T_i^I \subseteq \Delta_V^I$
Existential restriction	$\exists r$	$\exists$hasSpouse	$\{o \in \Delta_O^I \mid \exists o' \in \Delta_O^I : (o, o') \in r^I\}$
Data property domain	$\delta(d)$	δ(hasName)	$\{o \in \Delta_O^I \mid \exists v \in \Delta_V^I : (o, v) \in d^I\}$
Data property range	$\rho(d)$	ρ(salary)	$\{v \in \Delta_V^I \mid \exists o \in \Delta_O^I : (o, v) \in d^I\}$
Concept negation	$\neg A$	¬Human	$\Delta_O^I \setminus A^I$
Inverse object property	p^-	hasSpouse$^-$	$\{(o', o) \mid (o, o') \in p^I\}$
Object property negation	$\neg r$	¬hasSpouse	$\Delta_O^I \times \Delta_O^I \setminus r^I$
Data property negation	$\neg d$	¬hasName	$\Delta_O^I \times \Delta_V^I \setminus d^I$
Individual	$a \in \mathbf{NI}$	george	$a^I \in \Delta_O^I$
Literal	$\ell \in \mathbf{ND}$	"george"	$\ell^I \in \Delta_V^I$

where B, C are *classes*, q, r are *object properties*, d is a *data property*, f is a *datatype expression*, $\rho(d)$ is a *data property range expression*, and v is a *data property expression*. The elements above are defined according to the following grammar, where $A \in \mathbf{NC}$, $d \in \mathbf{ND}$, $p \in \mathbf{NP}$, $\delta(d)$ is a *data property domain expression*, and $T_1, \ldots, T_n$ are the RDF datatypes[4]:

$$\begin{array}{lll} B \to A \mid \exists r \mid \delta(d) & & C \to \top_C \mid B \mid \neg B \\ q \to p \mid p^- & & r \to q \mid \neg q \\ f \to \top_D \mid T_1 \mid \cdots \mid T_n & & v \to d \mid \neg d \end{array}$$

In the rules above, $\top_C$ and $\top_D$ denote the "top" concepts for concepts and data values (called *literals* in the RDF terminology), respectively.

There are a few differences between the (*DL-Lite*$_R$-like) language introduced here and OWL 2 QL. Specifically, OWL 2 QL also allows one to express special features of binary relations, such as reflexivity or transitivity. Since these additional constructs do not affect our patterns, they are not part of the ontology language considered here.

An OWL 2 QL *ABox* $\mathcal{A}$ is a finite set of *assertions* of the form $A(a)$, $p(a, b)$, or $d(a, \ell)$, where $A \in \mathbf{NC}$, $p \in \mathbf{NP}$, $d \in \mathbf{ND}$, a and b are individuals in $\mathbf{NI}$, and ℓ is a literal value. We call the pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ an OWL 2 QL *ontology*.

Semantics. Similarly to first-order logic, the semantics of OWL 2 QL ontologies is given through Tarski-style *interpretations* of the form $\mathcal{I} = (\Delta_O^I, \Delta_V^I, \cdot^I)$, where Δ_O^I is a non-empty *domain of objects*, Δ_V^I is a non-empty *domain of values*, and $\cdot^I$ is an *interpretation function* defined according to the inductive definition of Table 1.

Let $\mathcal{I}$ be an interpretation. $\mathcal{I}$ *satisfies an inclusion axiom* $\phi \sqsubseteq \psi$, denoted as $\mathcal{I} \vDash \phi \sqsubseteq \psi$, if $\phi^I \subseteq \psi^I$. $\mathcal{I}$ *satisfies a TBox* $\mathcal{T}$, denoted as $\mathcal{I} \vDash \mathcal{T}$, if it satisfies all axioms in $\mathcal{T}$. $\mathcal{I}$ *satisfies a class assertion* $C(a)$, denoted as $\mathcal{I} \vDash C(a)$, if $a^I \in C^I$. $\mathcal{I}$ *satisfies an object property assertion* $R(a, b)$ *(resp., a data property assertion* $d(a, \ell)$*)*, denoted as $\mathcal{I} \vDash r(a, b)$ (resp., $\mathcal{I} \vDash d(a, \ell)$), if $(a^I, b^I) \in r^I$ (resp., $(a^I, \ell^I) \in d^I$). $\mathcal{I}$ *satisfies an ABox* $\mathcal{A}$, denoted as $\mathcal{I} \vDash \mathcal{A}$, if it satisfies all assertions in $\mathcal{A}$. Finally, $\mathcal{I}$ *satisfies an ontology* $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, denoted as $\mathcal{I} \vDash \mathcal{O}$, if $\mathcal{I} \vDash \mathcal{T}$ and $\mathcal{I} \vDash \mathcal{A}$.

Mappings. In the VKG literature, mappings specify how to populate classes and properties of the ontology with individuals and values constructed from the data in the underlying DB. In other words, mappings provide the ABox that, together with a given TBox, realizes an ontology. In VKGs, the adopted language for mappings in real-world systems is R2RML [35], but for conciseness we use here a more convenient abstract notation inspired by the literature [31]: a *mapping* m is a pair of the form

$$s : Q(\mathbf{x}) \qquad t : \mathbf{L}(\mathfrak{t}(\mathbf{x}))$$

where $Q(\mathbf{x})$ is a SQL query over the DB schema S, called *source query*, and $\mathbf{L}(\mathfrak{t}(\mathbf{x}))$ is a list of *target atoms* of the form $C(\mathfrak{t}_1(\mathbf{x_1}))$, $p(\mathfrak{t}_1(\mathbf{x_1}), \mathfrak{t}_2(\mathbf{x_2}))$, or $d(\mathfrak{t}_1(\mathbf{x_1}), \mathfrak{t}_2(\mathbf{x_2}))$, where $C \in \mathbf{NC}$, $p \in \mathbf{NP}$, $d \in \mathbf{ND}$, and $\mathfrak{t}_1(\mathbf{x_1})$ and $\mathfrak{t}_2(\mathbf{x_2})$ are terms that we call *templates*. In this work we express source queries using the notation of *relational algebra*, omitting answer variables under the assumption that they coincide with the variables used in the target atoms. Intuitively, a template $\mathfrak{t}(\mathbf{x})$ in the target atom of a mapping corresponds to an R2RML *string template*,[5] and is used to generate object *IRIs* (Internationalized Resource Identifiers) or (RDF) *literals*, starting from DB values retrieved by the source query in that mapping. Note that R2RML constants can be rendered in our syntax by using templates of zero arity, and that data values (e.g., using the rr:column predicate) can be simulated by introducing a dedicated special template symbol.

[4] The RDF and OWL Recommendations use the simple types from XML Schema (https://www.w3.org/TR/xmlschema-2/).
[5] https://www.w3.org/TR/r2rml/#dfn-string-template.

Given a set $\mathcal{M}$ of mappings and a database instance $\mathcal{D}$, the *virtual ABox* $\mathcal{A}_{\mathcal{M}(\mathcal{D})}$ *exposed by* $\mathcal{D}$ *through* $\mathcal{M}$ is the set of ABox assertions:

$$\left\{\mathbf{L}(\mathrm{at}(\mathrm{t}(\mathbf{x}),(\mathbf{x}\mapsto\mathbf{o}))) \mid (\mathbf{x}\mapsto\mathbf{o})\in Q(\mathbf{x})^{\mathcal{D}}, (s : Q(\mathbf{x}), t : \mathbf{L}(\mathrm{t}(\mathbf{x})))\in\mathcal{M}\right\}$$

where $(\mathbf{x}\mapsto\mathbf{o})$ is a *solution mapping* belonging to the evaluation $Q(\mathbf{x})^{\mathcal{D}}$ of the source query $Q(\mathbf{x})$ over the DB instance $\mathcal{D}$, and $\mathbf{L}(\mathrm{at}(\mathrm{t}(\mathbf{x}),(\mathbf{x}\mapsto\mathbf{o})))$ is a set of ABox assertions deriving from the *application of the template* $\mathrm{t}(\mathbf{x})$ over the solution mapping $(\mathbf{x}\mapsto\mathbf{o})$. Such template application is usually defined by replacing $\mathbf{x}$ with $\mathbf{o}$ in $\mathbf{t}(\mathbf{x})$, through *string concatenation* operations.

In the following, we provide an example of mapping and virtual ABox derived through it. For the examples, we will use the concrete syntax adopted by the Ontop VKG system [36], in which the source query is expressed in SQL and each target atom is expressed as an RDF *triple pattern with templates*. The answer variables of the source query occurring in the target atoms are distinguished by enclosing them in curly brackets `{⋯}`. The following is an example mapping expressed in such syntax:

```
@prefix ex: <http://www.example.com/> .
source    SELECT ssn FROM person
target    ex:person/{ssn} a ex:Person .
```

The first line is a prefix declaration, used to abbreviate URIs. For instance, `ex:Person` is an abbreviation for the URI http://www.example.com/Person. The effect of such mapping, when applied to a DB instance $\mathcal{D}$ for Σ, is to populate the class `ex:Person` with IRIs constructed by replacing the answer variable `ssn` occurring in the target atom with the corresponding assignments for that variable in the solution mappings to the source query evaluated over $\mathcal{D}$. For instance, if the SQL query in the source retrieves the solution mappings (`ssn` ↦ `000-00-0000`) and (`ssn` ↦ `000-00-0001`), then the mapping above produces the following RDF graph (expressed in the Turtle [37] syntax), stating that individuals `ex:person/000-00-0000` and `ex:person/000-00-0001` are both instances of class ex:Person:

```
@prefix ex: <http://www.example.com/> .
ex:person/000-00-0000 a ex:Person .
ex:person/000-00-0001 a ex:Person .
```

Given a VKG specification $(\mathcal{T},\mathcal{M},\mathcal{S})$ and a database instance $\mathcal{D}$ of $\mathcal{S}$, the ontology $\mathcal{O}=(\mathcal{T},\mathcal{A}_{\mathcal{M}(\mathcal{D})})$ is called *Virtual Knowledge Graph of* $(\mathcal{T},\mathcal{M},\mathcal{S})$ *through* $\mathcal{D}$. The term "virtual" in the name derives from the fact that the virtual ABox $\mathcal{A}_{\mathcal{M}(\mathcal{D})}$ in a VKG setting is not materialized and stored somewhere. Query answering in VKGs, in fact, is carried out through *query rewriting and query unfolding techniques* [31,36]: SPARQL queries get translated on-the-fly into equivalent SQL queries, which are directly evaluated against the DB, transparently to the end-user.

2.1. R2RML mappings vs. VKG mappings

In this work, we adopt an abstract syntax for mappings inspired by the well-established scientific literature on VKGs, e.g., [31]. On the other hand, R2RML was inspired by the works on RDB2RDF systems,[6] and does not enforce some of the conventions that are common (and, sometimes, required) in a VKG setting. Usually these low-level differences are never addressed explicitly in the scientific literature, but in our experience of maintainers of the Ontop VKG system [36] we have witnessed that these can be a major source of confusion for VKG practitioners. Since this work is also targeting potential users of a VKG system that will have to write concrete R2RML mappings, we here explicitly clarify what these conventions are.

We recognize two main differences between R2RML mappings and VKG mappings: the first is that R2RML mappings support what we here call *intensional mappings*, and the second one is that R2RML also allows for templates that are not *injective*.

Intensional mappings. R2RML allows for mappings at the *intensional level*, i.e., for mappings defining TBox axioms rather than ABox assertions, like the following one (expressed in the Ontop [36] concrete mapping syntax):

```
@prefix ex: <http://www.example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
source    SELECT star_id, star_type FROM star
target    ex:star/{star_type} rdfs:subClassOf :Star .
```

Mappings like the one above go beyond those considered in the scientific literature for VKGs, and are often a source of confusion among VKG practitioners: they are perfectly valid R2RML mappings, hence virtually all R2RML-based VKG systems accept them, however the semantics of query answering in the presence of such mappings is system-dependent, often leading to "surprising"

[6] https://www.w3.org/2001/sw/rdb2rdf/.

results that are due to the well-studied computational complexity boundaries of the VKG approach [31,38]. Here, we ignore such kind of mappings, and focus instead on traditional VKG mappings.

Injective templates. In classic VKG literature [31], templates are always *injective*, that is, the application of different templates (i.e., templates differing in either function symbol or arity) can never generate the same *ground term*.[7] This detail is *usually*[8] never made explicit, because it is a trivial consequence of the fact that, in those settings, equality between terms is realized through syntactic *unification*. R2RML deals with URIs, not with logical terms. Hence, it is inherently more flexible, and different URI templates do not guarantee that two different URIs will be generated. As an example, consider the following pair of mappings:

```
@prefix : <http://www.example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
source    SELECT star_id, star_type FROM star
target    :star-{star_id} a :Star .

@prefix : <http://www.example.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
source    SELECT star_id1, star_id2 FROM binary_systems
target    :star-{star_id1}-{star_id2} a :BinaryStarSystem .
```

Assume a database instance such that the SQL query in the first mapping retrieves (`star_id` $\mapsto$ `001-100`), and the SQL query in the second mapping retrieves (`star_id1` $\mapsto$ `001`, `star_id2` $\mapsto$ `100`). Then, these solutions together with the mapping assertions above would construct the following RDF graph:

```
@prefix : <http://www.example.com/> .
:star-001-100 a :Star .
:star-001-100 a :BinaryStarSystem .
```

That is, object `:star-001-100` is both recognized as a single star and as a binary system composed of two stars. Note that this does not correspond to the information that was contained in the original DB instance: it might be the result of a careful analysis of the domain (hence, the original DB was either incomplete, or not in 1st Normal Form), or it could be just a mistake induced by the fact of not having used injective templates. If the latter, to guarantee injectivity it would have sufficed to use a *safe URI separator* in place of the dash "`-`" in the target of the second mapping. As a matter of fact, the R2RML recommendation states that designers *should* use safe URI separators[9]: if this recommendation is observed, then the injectivity condition is trivially satisfied.

Contrarily to the issue of mappings at the intensional level discussed above, the VKG specification can naturally be extended to the scenario of non-injective templates. However, adopting non-injective templates usually worsens the performance of query reformulation, since joins over individuals built out of different templates are not anymore guaranteed to be empty. Non-injective templates can become useful in particular applications where the URIs to be produced for object identifiers should conform to some global vocabulary [40]. In this work we adopt the standard assumption of injective templates, as we are interested in preserving the semantics of the original data instance.

3. Mapping patterns

We now enter into the core contribution of this paper, namely the catalog of *mapping patterns*. In our vision, (ontology) mapping patterns can be used to unravel the high-level conceptualization behind the database design, and to exploit this conceptualization to better link the content of the database to a domain ontology, according to the vision displayed in Fig. 1. To justify our formalization of patterns, we make the following two fundamental observations:

- *(i)* a conceptual model may have more than one admissible relational representation, according to the applied methodology, as well as to considerations about efficiency, performance optimization, and space consumption on the final information system;
- *(ii)* given the logical schema of a relational database, regardless of its normal form, multiple conceptual models can provide (admissible) alternative representations of its domain.

By *(i)* and *(ii)*, and differently from most other approaches in the literature, our patterns explicitly specify the target conceptual model, in order to disambiguate among the various admissible conceptualizations for the same database schema.

Our patterns are tailored towards the VKG setting, however they can be directly applied also in those, possibly non-virtualized, contexts requiring a lightweight ontology language. The only strict requirements are that the target ontology language is not more

[7] In DL literature, logical first-order terms are used in place of URIs or literals from the RDF world.

[8] An exception is the notion of "OBDA-complete mapping" introduced in [39].

[9] https://www.w3.org/TR/r2rml/#dfn-template-valued-term-map.

Table 2
R2RML natural mapping.

SQL datatype	RDF datatype
String-like	xsd:string
BINARY, BINARY VARYING, BINARY LARGE OBJECT	xsd:hexBinary
NUMERIC, DECIMAL	xsd:decimal
SMALLINT, INTEGER, BIGINT	xsd:integer
FLOAT, REAL, DOUBLE PRECISION	xsd:double
BOOLEAN	xsd:boolean
DATE	xsd:date
TIME	xsd:time
TIMESTAMP	xsd:dateTime

expressive than OWL 2 QL, and that the mapping language conforms to the restrictions set in place in Section 2. Studying patterns under different assumptions, e.g., more expressive ontology languages, goes beyond the scope of this work.

In its basic form, a mapping pattern is a quadruple $(\mathcal{C}, \mathcal{S}, \mathcal{M}, \mathcal{O})$, where $\mathcal{C}$ is a conceptual model, $\mathcal{S}$ a database schema, $\mathcal{M}$ a set of mappings, and $\mathcal{O}$ an (OWL 2 QL) ontology. In such pattern, the pair $(\mathcal{C}, \mathcal{S})$ puts into correspondence a conceptual representation to one of its (many) admissible (i.e., formally sound [41,42]) database schemata, like those prescribed by *well-established database modeling methodologies*. The pair $(\mathcal{M}, \mathcal{O})$, instead, is formed by the *DB ontology* $\mathcal{O}$, which is the OWL 2 QL encoding[10] of the conceptual model $\mathcal{C}$, and the set $\mathcal{M}$ of mappings, providing the link between $\mathcal{S}$ and $\mathcal{O}$. The term "DB ontology" refers to an ontology whose concepts and properties reflect the constructs of the conceptual model, mirroring the structure of the relational database, as displayed in Fig. 1.

As pointed out in Section 1, we do not fix which of these information artifacts are given, and which are produced as output, but we simply describe how they relate to each other, on a per-pattern basis. Inputs and outputs will depend on the specific application scenario, as we will discuss in Section 5.

Some of the more advanced patterns have a more complex structure, where pairs of conceptual models and/or pairs of database schemata are respectively used in place of $\mathcal{C}$ and $\mathcal{S}$ (e.g., Patterns **SRR** or **SHa**, which we will introduce in Section 3.2, fall under this category). These patterns prescribe specific *transformations* to be applied on an *input* conceptual (resp., DB) schema, in order to obtain an *output* conceptual (resp., DB) schema. These output artifacts make explicit the presence of specific structures that are revealed through the application of the pattern itself. We will see that these structures can in turn enable further applications of patterns.

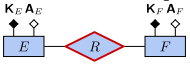
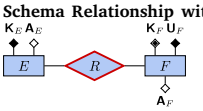
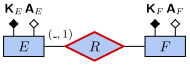
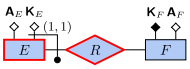
3.1. Pattern organization and presentation conventions

Pattern organization. We organize patterns in two major groups: *schema-driven patterns*, shaped by the structure of the DB schema and its explicit constraints, and *data-driven patterns*, which in addition consider constraints emerging from specific configurations of the data in the DB. Observe that, for each schema-driven pattern, we actually identify a corresponding data-driven version in which the constraints over the schema are not explicitly specified, but *hold in the data*. We denote such pattern as its schema-driven counterpart, but with a leading "**D**" in place of "**S**" (e.g., Pattern **DE** is the data-driven version of Pattern **SE** shown in Table 3).

Presentation conventions. We show the fragment of the conceptual model that is affected by the pattern in E-R notation (adopting the original notation by Chen [43]) — but any structural conceptual modeling language, such as UML or ORM 2 [24,44], would work as well. To compactly represent sets of attributes, we use a small diamond in place of the small circle used for single attributes in Chen notation. For cardinality constraints, we follow the "*look-here*" convention, that is, the cardinality constraint for a *role* is placed next to the entity participating in that role. In the DB schema, we use $T(\underline{\mathbf{K}}, \mathbf{A})$ to denote a *table* with name T, *primary key* consisting of the attributes $\mathbf{K}$, and additional attributes $\mathbf{A}$. Given a set $\mathbf{U}$ of attributes in T, we denote by $\mathsf{key}_T(\mathbf{U})$ the fact that $\mathbf{U}$ forms a *key* for T. Referential integrity constraints (like, e.g., inclusion dependencies and foreign keys) are depicted with edges, pointing from the referencing attribute(s) to the referenced one(s). For conciseness, we denote sets of the form $\{o \mid condition\}$ as $\{o\}_{condition}$. In order to express datatypes for data properties, we introduce two auxiliary functions: a function τ that, given a DB attribute A, returns the DB datatype of A, and a function μ that associates, to each DB datatype, a corresponding RDF datatype. For the definition of μ, we re-use the *Natural Mapping*[11] correspondence provided by the R2RML recommendation, and displayed in Table 2. As a final note, following the E-R-diagrams convention, we assume a default $(1, 1)$ cardinality on attributes. For such a reason, in the DB schema we assume all attributes to be *not nullable* by default (using the SQL convention, declared as "`NOT NULL`"). If attribute A is instead optional, we denote this fact in the DB schema through the notation $opt(A)$. Such notation extends in the natural way to the case of a set $\mathbf{A}$ of attributes.

[10] Modulo the expressivity of the OWL 2 QL language.
[11] https://www.w3.org/TR/r2rml/#natural-mapping.

Table 3
Schema-driven patterns: Entities and binary relationships without attributes.

Conceptual model	DB schema	Mappings	Ontology
Schema Entity (SE) 	$T_E(\underline{\mathbf{K}}, \mathbf{A})$	s: T_E t: $C_E(\mathtt{t}_E(\mathbf{K}))$, $\{d_A(\mathtt{t}_E(\mathbf{K}), A)\}_{A \in \mathbf{K} \cup \mathbf{A}}$	$\left\{ \begin{array}{l} \delta(d_A) \sqsubseteq C_E, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_E \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathbf{K} \cup \mathbf{A}}$
In case of optional attributes, for each optional attribute A' add an $opt(A')$ constraint to the DB schema and drop the corresponding axiom $C_E \sqsubseteq \delta(d_{A'})$ from the ontology.			
Schema Relationship (SR) 	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E)$ $T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_R(\underline{\mathbf{K}_{RE}, \mathbf{K}_{RF}})$	s: T_R t: $p_R(\mathtt{t}_{C_E}(\mathbf{K}_{RE}), \mathtt{t}_{C_F}(\mathbf{K}_{RF}))$	$\exists p_R \sqsubseteq C_E$ $\exists p_R^- \sqsubseteq C_F$
• In case of $(\_, 1)$ cardinality on role R_E (resp., R_F), the primary key for T_R is restricted to the attributes $\mathbf{K}_{RE}$ (resp., $\mathbf{K}_{RF}$). In case both roles have $(\_, 1)$ cardinality, either choice for the primary key is made, and the remaining attributes form a non-primary key in the logical schema. • In case of $(1, N)$ cardinality on role R_E (resp., R_F), the inclusion dependency $\mathbf{K}_E \subseteq \mathbf{K}_{RE}$ (resp., $\mathbf{K}_F \subseteq \mathbf{K}_{RF}$) holds in the schema, and the first (resp., second) inclusion axiom in the ontology holds in both directions. • In case of $(1, 1)$ cardinality on role R_E (resp., R_F), the same considerations as for $(\_, 1)$ cardinality apply; in addition, the foreign key $\mathbf{K}_E \subseteq \mathbf{K}_{RE}$ (resp., $\mathbf{K}_F \subseteq \mathbf{K}_{RF}$) holds in the schema, and the first (resp., the second) inclusion axiom in the ontology holds in both directions.			
Schema Relationship with Identifier Alignment (SRa) 	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E)$ $T_F(\underline{\mathbf{K}_F}, \mathbf{U}_F, \mathbf{A}_F)$ $T_R(\underline{\mathbf{K}_{RE}, \mathbf{U}_{RF}})$ $\mathsf{key}_{T_F}(\mathbf{U}_F)$	s: $T_R \bowtie_{\mathbf{U}_{RF} = \mathbf{U}_F} T_F$ t: $p_R(\mathtt{t}_{C_E}(\mathbf{K}_{RE}), \mathtt{t}_{C_F}(\mathbf{K}_F))$	$\exists p_R \sqsubseteq C_E$ $\exists p_R^- \sqsubseteq C_F$
Cardinality constraints are handled similarly as for Pattern **SR**, with the difference that now the constraints involve U_{RF} and U_F. The case where both sets of attributes in T_R require alignment is treated similarly.			
Schema Relationship with Merging (SRm) 	$T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_E(\underline{\mathbf{K}_E}, \mathbf{K}_{EF}, \mathbf{A}_E)$	s: T_E t: $p_{EF}(\mathtt{t}_{C_E}(\mathbf{K}_E), \mathtt{t}_{C_F}(\mathbf{K}_{EF}))$	$\exists p_{EF} \sqsubseteq C_E$ $\exists p_{EF}^- \sqsubseteq C_F$
Cardinality constraints are handled similarly as for Pattern **SR**, with the catch that in case of $(0, 1)$ cardinality on role R_E, we have that $\mathbf{K}_{EF}$ is nullable. The alignment variant **SRma/DRma**, where the foreign key references a non-primary identifier, is defined in the straightforward way.			
Schema Weak-Entity (SEw) 	$T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_E(\underline{\mathbf{K}_E, \mathbf{K}_{EF}}, \mathbf{A}_E)$	s: T_E t: $C_E(\mathtt{t}_E(\mathbf{K}_E, \mathbf{K}_{EF}))$, $\{d_A(\mathtt{t}_E(\mathbf{K}_E, \mathbf{K}_{EF}), A)\}_{A \in \mathbf{K}_E \cup \mathbf{A}_E}$ $p_{EF}(\mathtt{t}_E(\mathbf{K}_E, \mathbf{K}_{EF}), \mathtt{t}_{C_F}(\mathbf{K}_{EF}))$	$\left\{ \begin{array}{l} \delta(d_A) \sqsubseteq C_E, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_E \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathbf{K}_E \cup \mathbf{A}_E}$ $\exists p_{EF} \equiv C_E$ $\exists p_{EF}^- \sqsubseteq C_F$
Cardinality constraints are handled similarly as for Pattern **SR**. Optional attributes are handled similarly as for Pattern **SE**. The alignment variant **SEwa/DEwa**, where the foreign key references a non-primary identifier, is defined in the straightforward way.			

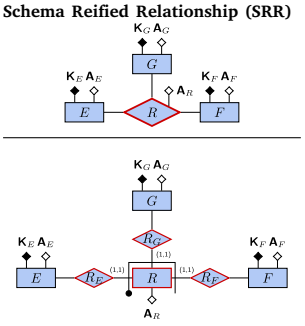
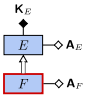
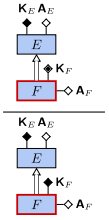
3.2. Schema-driven patterns

Next we comment on schema-driven patterns, shown in Tables 3 and 4. For each pattern, we provide an example and references to the relevant related literature. We point out that the reference to the literature is *only with respect to the pattern being considered*. Hence, patterns that have been proposed or that can be identified in the related literature, but that do not find a correspondence with our patterns, are out of the scope of this section. These and other considerations will instead be discussed in Section 6.

Schema Entity (SE). This fundamental pattern describes the correspondence between an entity with a primary identifier and attributes to a class and data properties in the ontology. The entity is expressed in the DB schema through a single table T_E with primary key $\mathbf{K}$ and other attributes $\mathbf{A}$, as it is the norm in sound DB design practices. The mappings column explains how T_E is mapped into a corresponding class C_E. The primary key of T_E is employed to construct the objects that are instances of C_E, using a template $\mathtt{t}_E$ specific for that entity. Each *relevant* attribute of T_E is mapped to a data property of C_E, with suitable domain and range axioms. A mandatory participation constraint is added to the each data property corresponding to a mandatory attribute.

Example: A client registry table containing *social security numbers* (SSNs) of clients, together with their name as an additional attribute, is mapped to a Client class using the SSN to construct its objects. In addition, the SSN and name are mapped to two corresponding data properties.

Table 4
Schema-driven patterns: Relationships with attributes, n-ary relationships, and hierarchies. For patterns yielding views, we show the views together with the DB schema, separating them from the original tables using a thick horizontal bar. We use a similar notation for changes in the conceptual model.

Conceptual model	DB schema	Mappings	Ontology
Schema Reified Relationship (SRR) 	$\mathsf{K}_R := \mathsf{K}_{RE}, \mathsf{K}_{RF}, \mathsf{K}_{RG}$ $T_G(\underline{\mathsf{K}_G}, \mathsf{A}_G)$ $T_R(\underline{\mathsf{K}_{RE}, \mathsf{K}_{RF}, \mathsf{K}_{RG}}, \mathsf{A}_R)$ $T_E(\underline{\mathsf{K}_E}, \mathsf{A}_E) \quad T_F(\underline{\mathsf{K}_F}, \mathsf{A}_F)$	s: T_R t: $C_R(\mathsf{t}_R(\mathsf{K}_R))$, $\{d_A(\mathsf{t}_R(\mathsf{K}_R), A)\}_{A \in \mathsf{A}_R}$, $p_{RE}(\mathsf{t}_R(\mathsf{K}_R), \mathsf{t}_{C_E}(\mathsf{K}_{RE}))$, $p_{RF}(\mathsf{t}_R(\mathsf{K}_R), \mathsf{t}_{C_F}(\mathsf{K}_{RF}))$, $p_{RG}(\mathsf{t}_R(\mathsf{K}_R), \mathsf{t}_{C_G}(\mathsf{K}_{RG}))$	$\exists p_{RE} \equiv C_R$ $\exists p_{RE}^- \sqsubseteq C_E$ $\exists p_{RF} \equiv C_R$ $\exists p_{RF}^- \sqsubseteq C_F$ $\exists p_{RG} \equiv C_R$ $\exists p_{RG}^- \sqsubseteq C_G$ $\left\{ \begin{array}{l} \delta(d_A) \sqsubseteq C_R, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_R \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathsf{A}_R}$
Pattern **SRR** applies whenever there are three or more participating roles, or when the relationship has attributes. Given the nature of RDF graphs, in order to handle these cases we need reification, hence this pattern requires a change in the conceptual model (see ER-diagram below the line). After reification, we apply the patterns discussed for binary relationships (cardinality constraints, weak entities, and optional attributes are handled as discussed). Observe that, in the conversion to OWL 2 QL, the identification constraint on R is lost (similarly to other identifiers).			
Schema Hierarchy (SH) 	$T_E(\underline{\mathsf{K}_E}, \mathsf{A}_E)$ $T_F(\underline{\mathsf{K}_{FE}}, \mathsf{A}_F)$	s: T_F t: $C_F(\mathsf{t}_{C_E}(\mathsf{K}_{FE}))$, $\{d_A(\mathsf{t}_{C_E}(\mathsf{K}_{FE}), A)\}_{A \in \mathsf{A}_F}$	$C_F \sqsubseteq C_E$ $\left\{ \begin{array}{l} \delta(d_A) \sqsubseteq C_F, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_F \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathsf{A}_F}$
Optional attributes are handled as in Pattern **SE**.			
Schema Hierarchy with Identifier Alignment (SHa) 	$T_E(\underline{\mathsf{K}_E}, \mathsf{A}_E) \quad \mathsf{key}_{T_F}(\mathsf{K}_{FE})$ $T_F(\underline{\mathsf{K}_F}, \mathsf{K}_{FE}, \mathsf{A}_F)$ $T_E(\underline{\mathsf{K}_E}, \mathsf{A}_E) \quad \mathsf{key}_{V_F}(\mathsf{K}_F)$ $V_F(\mathsf{K}_F, \underline{\mathsf{K}_{FE}}, \mathsf{A}_F) = T_F$	s: V_F t: $C_F(\mathsf{t}_{C_E}(\mathsf{K}_{FE}))$, $\{d_A(\mathsf{t}_{C_E}(\mathsf{K}_{FE}), A)\}_{A \in \mathsf{K}_F \cup \mathsf{A}_F}$	$C_F \sqsubseteq C_E$ $\left\{ \begin{array}{l} \delta(d_A) \sqsubseteq C_F, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_F \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathsf{K}_F \cup \mathsf{A}_F}$
In this pattern, the "alignment" is meant to align the primary identifier used in the child entity to the primary identifier used in the parent entity. The other two possibilities for the application of the pattern are: • the foreign key in the child entity is the primary key of that entity, and references a non-primary key of the parent entity; • the foreign key in the child entity is a non-primary key of that entity, and references a non-primary key of the parent entity. We here depict the most common scenario, where the foreign key points to the primary key of the parent entity. Observe that this pattern requires a change in the conceptual model (essentially keeping track the attributes used for identifying the objects of the subclass). Optional attributes are handled as in Pattern **SH**.			

Conceptual model	DB schema
ssn:int — Client — name:string	client(ssn:int, name:string)

Mappings	Ontology	
	:Client a owl:Class .	
	:client#ssn a owl:DatatypeProperty .	:client#name a owl:DatatypeProperty .
s: SELECT * FROM client	:client#ssn rdfs:domain :Client .	:client#ssn rdfs:range xsd:integer .
t: :client/{ssn} a :Client .	:client#name rdfs:domain :Client .	:client#name rdfs:range xsd:string .
:client/{ssn} :client#ssn {ssn} .	:Client rdfs:subClassOf _:r1 .	_:r1 a owl:Restriction .
:client/{ssn} :client#name {name} .	_:r1 owl:onProperty :client#ssn .	_:r1 owl:someValuesFrom rdfs:Literal .
	:Client rdfs:subClassOf _:r2 .	_:r2 a owl:Restriction .
	_:r2 owl:onProperty :client#name .	_:r2 owl:someValuesFrom rdfs:Literal .

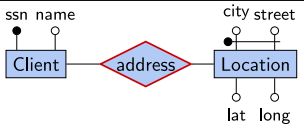
References: This is the most basic pattern, and variants or portions of it are widespread in other approaches.

- W3C-DM states that each table with a primary key must be transformed into a class similarly to Pattern **SE**. This approach does not fix the conceptual model, nor does it consider ontology axioms.
- The extension of W3C-DM with OWL 2 discussed in [20] includes domain and range axioms for properties, but not mandatory participation of data properties. It includes, however, OWL 2 *key* axioms (`owl:hasKey`), which we ignore here because they fall outside of the OWL 2 QL profile. As for W3C-DM, also this approach does not fix the conceptual model underlying the DB schema.

- Pattern **SE** resembles the combination of the cases *non-binary relation* and *data attribute* discussed in BootOX [18], however there are also substantial differences: the catalog of BootOX does not fix the underlying conceptual representation, and as a result ambiguities can arise. For instance, **SE** adds a mandatory participation constraint for a data property only if such property is mapped to a mandatory attribute, whereas in BootOX the choice of having or not each constraint is left to the user.
- The algorithm of MIRROR [45] applies a variant of this pattern where the datatype information is encoded into the R2RML mappings, and not added as ontology axioms. Mandatory participation for properties relative to *not nullable* attributes cannot be encoded through this approach.
- Among the mapping patterns from [30], our **SE** resembles the union of *Direct Concept* and *Direct Concept Attribute*. Ontology axioms are not included.

Schema Relationship (SR). This pattern describes the correspondence between a binary relationship without attributes to an OWL 2 QL object property, for the case where such relationship is represented in the DB as a separate (usually, "many-to-many") table. This pattern considers three tables T_R, T_E, and T_F, in which the set of columns in T_R is partitioned into two parts K_{RE} and K_{RF} that are foreign keys to T_E and T_F, respectively. The identifier of T_R depends on the role cardinalities in the E-R model. The pattern captures how T_R is mapped to an object property p_R, using the two partitions K_{RE} and K_{RF} to construct respectively the subject and the object of the triples in p_R. The templates t_{C_E} and t_{C_F} must be those used for building instances of classes C_E and C_F, respectively.

Example: An additional table in the client registry stores the addresses of each client, and has a foreign key to a table with locations. The former table is mapped to an address object property, for which the ontology asserts that the domain is the class Person and the range an additional class Location, which corresponds to the latter table.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	client(ssn, name) location(city, street, lat, long) address(client, ccity, cstreet)

Mappings	Ontology
s: SELECT * FROM address t: :client/{ssn} :address :location/{ccity}/{cstreet} .	:address a owl:ObjectProperty . :address rdfs:domain :Client . :address rdfs:range :Location .

References: The pattern scenario foresees a separate table for the relationship. When this is the case, other approaches often prefer to apply a *relationship reification* instead, where the relationship is reified into a class.

- It is not present in W3C-DM, where the scenario captured by Pattern **SR** is instead handled through relationship reification (see Pattern **SRR**).
- Pattern **SR** slightly corresponds to the handling of binary relations in [20], however that approach is substantially more limited than Pattern **SR**:
  - it only applies to tables with exactly two attributes;
  - the primary key must comprise both attributes;
  - incoming foreign keys are not allowed.

  On the other hand, the approach in [20] for binary relations applies also in those cases where the foreign keys do not refer the primary keys of the tables participating in the relationship. In our methodology, we decided to render this variant explicit through the dedicated Pattern **SRa**. Finally, their definition of foreign-key ignores the standard assumption that the referred attributes must form a key.
- Case 2 in BootOX [18] is comparable to Pattern **SR**, however there are some ambiguities arising from the lack of a fixed conceptual representation. For instance, the treatment of BootOX is incomplete w.r.t. mandatory participation: constraints are handled correctly for role R_E, but they are ignored for role R_F. Other details of BootOX are omitted, and so it is unclear how they are handled: for instance, it is stated that mappings are generated according to W3C-DM, however W3C-DM does not handle the case of binary relationships (as discussed above), which is an apparent contradiction. BootOX does not impose any constraint on the attributes referred by the foreign keys, which could in principle not form a key.
- In the algorithm of MIRROR [45], ontology axioms over object properties are never included, since that algorithm does not consider OWL 2 and only works at the level of R2RML. Modulo this major difference, our Pattern **SR** is *partially* covered by different cases there, which differentiate from each other depending on the cardinality constraints in the logical schema:
  - Cases 5b and 8 resemble Pattern **SR** with a $(1,1)$ or a $(1,N)$ cardinality on one of the two roles, and $(0,N)$ on the other. A notable difference, though, is that in Pattern **SR** the identification constraint entailed by the $(1,1)$ cardinality is correctly translated into a key constraint in the logical schema, whereas it is ignored by MIRROR.
  - Case 7 resembles Pattern **SR** with mandatory participation on both roles. In our case, however, we can exploit OWL 2 QL to partially encode the mandatory participation, using suitable class equivalence axioms.

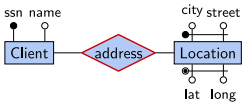
  - The case where both roles of a relationship participate with cardinality $(0, N)$ is pruned on purpose from their catalogue, with the argumentation that "primary keys must be defined not null and unique". Clearly, the fact that primary keys must be not null and unique is completely independent from the fact of having $(0, N)$ relationships in a conceptual model. Hence, we are not able to actually reconstruct what the authors' intention there was.

- Pattern **SR** strictly includes *Pattern 12: Many to Many Table* in [29]. In particular, Pattern **SR** covers all possible cardinality constraints, whereas Pattern 12 should cover only the many-to-many case (according to its name). However, such pattern is not formally specified, and the choice on where to apply it is totally left open to the domain expert. One could in fact apply it even to tables representing binary relationships with arbitrary cardinality constraints, simply by ignoring such cardinalities. Pattern 12 does not discuss ontology axioms, and it does not rule out the case where the relationship has attributes (which cannot be correctly handled through Pattern **SR**, nor through Pattern 12, because it requires reification, as we will describe for Pattern **SRR**).
- Pattern **SR** strictly includes pattern *Relationship: Many to Many* in [30], since such pattern is just a renaming of *Pattern 12: Many to Many Table* from [29]. This pattern does not rule out the case where the relationship has attributes, which in the scope of that work is perfectly reasonable since the KGs they consider allow for specifying attributes on properties. This is common, for instance, in *property graphs* [1]. Recently, an extension of RDF called *RDF-star*[12] is being proposed that also incorporates this feature.

Schema Relationship with Identifier Alignment (SRa). This pattern is similar to Pattern **SR**, but it comes with a *modifier* **a** indicating that the pattern can be applied after the identifiers involved in the relationship have been *aligned*. The alignment is necessary because the foreign key in T_R does not refer to the primary key $\mathbf{K}_F$ of T_F, but to an alternative key $\mathbf{U}_F$. Since the instances of the class C_F corresponding to T_F are constructed using the primary key $\mathbf{K}_F$ of T_F (cf. Pattern **SE**), also the pairs that populate p_R should refer in their object position to that primary key, which can only be retrieved via a join between T_R and T_F on the non-primary key $\mathbf{U}_F$.

Note that alignment variants can be defined in a straightforward way for other patterns involving relationships. For conciseness, we omit these variants from our catalog.

Example: The primary key of the table with locations is not given by the city and street, which are used in the table that relates clients to their addresses, but is given by the latitude and longitude of locations.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	$\text{key}_{\text{location}}$(city,street) client(ssn, name) location(city, street, lat, long) address(client, ccity, cstreet)
Mappings	**Ontology**
s: SELECT A.client, L.lat, L.long FROM address A, location L WHERE L.city=A.ccity AND L.street=A.cstreet t: :client/{ssn} :address :location/{lat}/{long} .	:address a owl:ObjectProperty . :address rdfs:domain :Client . :address rdfs:range :Location .

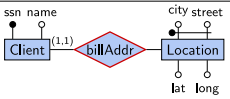
References: This pattern is original, as either it has never been considered by other approaches or it is mixed with the general strategy for handling binary relationships.

- It is not present in W3C-DM, where the scenario captured by Pattern **SRa** is instead handled through relationship reification.
- Pattern **SRa** covers the handling of binary relations in [20] and in BootOX [18] for the case when the foreign keys refer to non-primary keys of the participating entities. The issues of these approaches highlighted in comparison to Pattern **SR** apply for Pattern **SRa** as well.
- Pattern **SRa** does not correspond to any of the cases treated in MIRROR [45] or to any of the patterns presented in [30].

Schema Relationship with Merging (SRm). This pattern handles the case where the binary relationship is not rendered in the DB as a separate table, but rather merged into the table representing one of the participating entities, which can be done without introducing redundancy whenever such participation is with cardinality $(\_, 1)$. It considers a table T_E in which the foreign key $\mathbf{K}_{EF}$ referring a table T_F is disjoint from its primary key $\mathbf{K}_E$. The table T_E is mapped to an object property, whose subject and object are derived respectively from $\mathbf{K}_E$ and $\mathbf{K}_{EF}$.

Example: The relationship between a client and its *unique* billing address is merged into the client table. In the ontology, a billingAddress object property relates the Client class to the Location class, and is populated via a mapping from the client table.

[12] https://www.w3.org/2021/12/rdf-star.html.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	client(ssn, name, ccity, cstreet) location(city, street, lat, long)

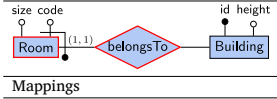
Mappings	Ontology	
	:billingAddress a owl:ObjectProperty .	
s: SELECT ssn, ccity, cstreet FROM client	:billingAddress rdfs:domain :Client .	:billingAddress rdfs:range :Location .
t: :client/{ssn} :billingAddress :location/{ccity}/{cstreet} .	:Client rdfs:subClassOf _:r1 .	_:r1 a owl:Restriction .
	_:r1 owl:onProperty :billingAddress .	_:r1 owl:someValuesFrom owl:Thing .

References: This pattern, as its alignment variant **SRma**, slightly overlaps with the common strategy of transforming each foreign key into an object property.

- It is partially covered by W3C-DM, where each foreign key is translated to an object property. However, there are substantial differences, going beyond the fact that OWL 2 QL constraints are not considered in W3C-DM. For instance, in case of cardinality $(1, 1)$, where the foreign-key holds in both directions, W3C-DM prescribes the creation of two different object properties, whereas Pattern **SRm** would create only one property (the one focused on the relationship), encoding the additional foreign-key as an additional OWL 2 QL inclusion.
- Since [20] adopts an approach close to W3C-DM, similar considerations apply.
- It roughly corresponds to *Case 4* in BootOX [18]. However, since the catalog of BootOX does not fix the underlying conceptual representation, ambiguities can arise: for instance, Pattern **SRm** adds a mandatory participation constraint only if such constraint derives from a mandatory participation in the ER-diagram, whereas constraints in BootOX are explicitly handled by the user (without an indication on how this should be done). Similarly to what we observed when discussing Pattern **SR**, BootOX Case 4 seems to be incomplete, for example, cardinality constraints on the role for the entity F are ignored.
- In the algorithm of MIRROR [45], a substantial portion of Pattern **SRm** corresponds to the combination of Cases 2b, 4, and 5a. The only difference is that Pattern **SRm** also covers the case when F participates with cardinality $(1, 1)$ to R, and that MIRROR does not produce ontology axioms by design.
- In [30], the only pattern resembling Pattern **SRm** is *Direct Relationship*, which is the same approach used in W3C-DM. Hence, all considerations discussed for W3C-DM apply to *Direct Relationship* as well.

Schema Weak-Entity (SEw). This pattern considers a *weak entity* E identified through a relationship R. The table T_E, corresponding to E, is mapped to a class C_E, whose instances are built through the primary identifier of T_E, and whose data properties correspond to the attributes of E. The relationship R is captured through an object property p_{EF}, as in Pattern **SRm**.

Example: A room in a university is identified by a code and the building it belongs to. Since a room must belong to exactly one building, the relationship belongsTo has been merged into entity Room, and Room is a weak entity identified through belongsTo. In the ontology, a class Room is created, populated via a mapping that builds Room individuals by combining the code of the room together with the identifier of the building. Room individuals are put in correspondence to their respective Building individuals through an appropriate belongsTo object property.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	building(id, height) room(code, building, size)

Mappings	Ontology	
	:Room a owl:Class .	:room#code a owl:DatatypeProperty .
	:room#size a owl:DatatypeProperty .	:belongsTo a owl:ObjectProperty .
	:room#code rdfs:domain :Client .	:room#size rdfs:domain :Client .
s: SELECT * FROM room	:belongsTo rdfs:domain :Room .	:belongsTo rdfs:range :Client .
t: :room/{code}/{building} a :Room .	:Room rdfs:subClassOf _:r1 .	_:r1 a owl:Restriction .
:room/{code}/{building} :room#code {code} .	_:r1 owl:onProperty :room#code .	_:r1 owl:someValuesFrom rdfs:Literal .
:room/{code}/{building} :room#size {size} .	:Client rdfs:subClassOf _:r2 .	_:r2 a owl:Restriction .
:room/{code}/{building} :belongsTo :Building/{building} .	_:r2 owl:onProperty :room#size .	_:r2 owl:someValuesFrom rdfs:Literal .
	:Client rdfs:subClassOf _:r3 .	_:r3 a owl:Restriction .
	_:r3 owl:onProperty :belongsTo .	_:r3 owl:someValuesFrom owl:Thing .

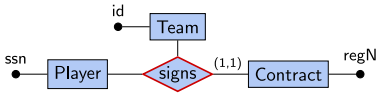
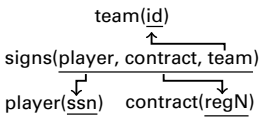
References: This pattern, and its alignment variant **SEwa**, slightly overlaps with the common strategy of transforming each foreign key into an object property.

- It is partially covered by W3C-DM, where each foreign key is translated to an object property, and object identifiers are always built from primary keys. However, the differences already mentioned for Patterns **SE** and **SRm** apply to this pattern as well.
- Since [20] adopts an approach close to W3C-DM, similar considerations apply.
- From a DB schema as the one in Pattern **SEw**, the combination of Rules 1, 2, 3, 4 and 7 from BootOX [18] would generate mappings and ontology conforming to Pattern **SEw**.

- Pattern **SEw** is partially captured by Case 6b of MIRROR [45]. MIRROR only captures the case of cardinality $(0, N)$ for role R_F, whereas Pattern **SEw** captures all possible cases. Another difference is that, in MIRROR, no alignment variant **SEwa** is discussed. Finally, MIRROR does not produce ontology axioms by design.
- To obtain mapping assertions of Pattern **SEw** from [30], one can apply the *Direct* patterns described therein, which correspond to W3C-DM. Such work does not consider ontology axioms.

Schema Reified Relationship (SRR). This pattern deals with n-ary relationships and/or relationships with attributes. For both cases, it is necessary to first *reify* the relationship into a class. This is because RDF can only encode unary predicates (through classes and assertions on them) and binary predicates (through object properties and assertions on them), and it does not allow for attributes over properties (recall that this is instead possible in property graphs [1]). The pattern considers a table T_R whose primary key is partitioned in at least three parts K_{RE}, K_{RF}, and K_{RG}, that are foreign keys to three additional tables; or when the primary key is partitioned in at least two such parts, but there are additional attributes in T_R. Such a table naturally corresponds to an n-ary relationship R with $n > 2$ (or with attributes), and to represent it at the ontology level we require a class C_R, which reifies R, whose instances are built from the primary key of T_R. The mapping accounts for the fact that the components of the n-ary relationship have to be represented by suitable object properties, one for each such component, and that the tuples that instantiate these object properties can all be derived from T_R alone.

Example: A table containing information about signed contracts, which involve a player, a team, and the contracts themselves. This information is represented by a relationship that is inherently ternary. The ontology should contain a class corresponding to the reification of such relationship, e.g., a class Signs.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	

Mappings	Ontology	
	:Signs a owl:Class .	:hasContract a owl:ObjectProperty .
	:hasPlayer a owl:ObjectProperty .	:hasTeam a owl:ObjectProperty .
	:hasContract rdfs:domain :Signs .	:hasContract rdfs:range :Contract .
s: SELECT * FROM signs	:hasPlayer rdfs:domain :Signs .	:hasPlayer rdfs:range :Player .
t: :signs/{player}/{contract}/{team} a :Signs .	:hasTeam rdfs:domain :Signs .	:hasTeam rdfs:range :Team .
:signs/{player}/{contract}/{team}	:Signs rdfs:subClassOf _:r1 .	_:r1 a owl:Restriction .
:hasContract :Contract/{contract} .	_:r1 owl:onProperty :hasContract .	_:r1 owl:someValuesFrom owl:Thing .
:signs/{player}/{contract}/{team}	:Signs rdfs:subClassOf _:r2 .	_:r2 a owl:Restriction .
:hasPlayer :Player/{player} .	_:r2 owl:onProperty :hasPlayer .	_:r2 owl:someValuesFrom owl:Thing .
:signs/{player}/{contract}/{team}	:Signs rdfs:subClassOf _:r3 .	_:r3 a owl:Restriction .
:hasTeam :Team/{team} .	_:r3 owl:onProperty :hasTeam .	_:r3 owl:someValuesFrom owl:Thing .
	:Contract rdfs:subClassOf _:r4 .	_:r4 a owl:Restriction .
	_:r4 owl:onProperty _:r5 .	_:r5 owl:inverseOf :hasContract .
	_:r4 owl:someValuesFrom owl:Thing .	

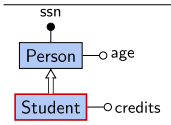
References: This pattern, which corresponds to *reification* in ontological and conceptual modeling [22,23,46], is commonly used to handle the case of "many-to-many" tables. A main difference with respect to other approaches is that we do not create data properties for the attributes identifying the reified relationship (i.e., entity R), because such attributes are already being represented as data properties of the classes encoding the entities participating to the relationship.

- In W3C-DM, every table encoding a relationship is handled according to a strategy similar to Pattern **SRR**, devoid of ontology axioms.
- Also [20] adopts an approach very similar to W3C-DM, so the same considerations discussed for W3C-DM apply as well, apart from the fact that domain and range axioms (but not mandatory participations) are added to the ontology.
- Case 2 of BootOX [18] is used to handle both the case of binary relationships and general relationships needing reification. For these reasons, the same considerations already discussed for Pattern **SR** apply to Pattern **SRR** as well.
- None of the cases in MIRROR [45] handles the situation of a table encoding an n-ary relationship or a relationship with attributes.
- The notion of reification is also present in the *Direct* Patterns of [30], given that such patterns encode W3C-DM. A notable difference, though, is that the focus there is on property graphs, which as mentioned allow one to avoid reification for the case of binary relationship with attributes.

Schema Hierarchy (SH). This pattern captures the most common case of ISA (i.e., the parent–child E-R relation) between entities. It considers a table T_F whose primary key is a foreign key referring the primary key of a table T_E. Then, T_F is mapped to a class C_F in the ontology that is a sub-class of the class C_E to which T_E is mapped. Hence, C_F "inherits" the template t_E of C_E, so that

the instances of the two classes are "compatible". Note that here we discuss the case where both the child and the parent tables are maintained in the database schema. Other strategies for handling ISAs between entities might be considered as well [24].

Example: An entity Student in an ISA relation with an entity Person.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	person(ssn, age) student(sssn, credits)

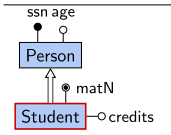
Mappings	Ontology	
s: SELECT * FROM student	:Student a owl:Class .	:Student rdfs:subClassOf :Person .
t: :person/{sssn} a :Student .	:student#credits a owl:DatatypeProperty .	:student#credits rdfs:domain :Student .
:person/{sssn}	:Student rdfs:subClassOf _:r1 .	_:r1 a owl:Restriction .
:student#credits {credits} .	_:r1 owl:onProperty :student#credits .	_:r1 owl:someValuesFrom rdfs:Literal .

References: This is an advanced pattern, and it is present only in few approaches.

- It is not present in W3C-DM, where each foreign-key is translated into an object property instead.
- It is also not present in the extension of W3C-DM discussed in [20].
- Class subsumption is considered in Case 5 of BootOX [18]. However, it is unclear how the mapping assertions are actually built. In the text, it is stated that mappings are generated according to W3C-DM, however such a strategy would not produce the desired results, since the templates used for generating objects of the subclass differ from those used for generating objects of the superclass.
- Case 2b of MIRROR [45] should handle ISAs between two entities, however it differs from Pattern **SH** and BootOX [18] since, in the DB schema, the foreign key is not required to refer to a key. This seems to be a glitch.
- Class hierarchies are not discussed in any of the mapping patterns from [30].

Schema Hierarchy with Identifier Alignment (SHa). Such pattern is like Pattern **SH**, apart from the foreign-key constraint that can come in three different variants. In the depicted one, the foreign key in T_F is over a non-primary key $\mathbf{K}_{FE}$. The objects for C_F have to be built out of $\mathbf{K}_{FE}$, rather than out of its primary key. For this purpose, the pattern creates a view V_F in which $\mathbf{K}_{FE}$ is the primary key, and the foreign key relations are preserved. Such view might enable further applications of patterns (see Example 1).

Example: An ISA relation between entities Student and Person. Students are identified by their matriculation number, whereas persons are identified by their SSN.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	$\text{key}_{\text{student}}$(sssn) person(ssn, age) student(sssn, matN, credits)

Mappings	Ontology	
s: SELECT * FROM student	:Student a owl:Class .	:Student rdfs:subClassOf :Person .
t: :person/{sssn} a :Student .	:student#matN a owl:DatatypeProperty .	:student#credits a owl:DatatypeProperty .
:person/{sssn}	:student#credits rdfs:domain :Student .	:student#matN rdfs:domain :Student .
:student#matN {matN} .	:Student rdfs:subClassOf _:r1 .	_:r1 a owl:Restriction .
:person/{sssn}	_:r1 owl:onProperty :student#credits .	_:r1 owl:someValuesFrom rdfs:Literal .
:student#credits {credits} .	:Student rdfs:subClassOf _:r2 .	_:r2 a owl:Restriction .
	_:r2 owl:onProperty :student#matN .	_:r2 owl:someValuesFrom rdfs:Literal .

References: This is an advanced pattern, and it is present only in few approaches.

- It is not present in W3C-DM, where each foreign-key is translated into an object property instead.
- It is not present in [20].
- Case 5 of BootOX [18] also deals also with the situation in which the foreign-key points to a non-primary identifier of the parent entity. However, it does not deal with the two other variants admitted by Pattern **SHa**. Regarding the mapping assertions, the same considerations discussed for Pattern **SH** apply here as well.
- Regarding MIRROR [45], the same considerations we had for Pattern **SH** apply.
- Class hierarchies are not discussed in any of the mapping patterns from [30].

We now provide an example showing one of the possible usages of schema-driven mapping patterns, specifically, to derive an ontology and mappings starting from a conceptualization and a DB schema.

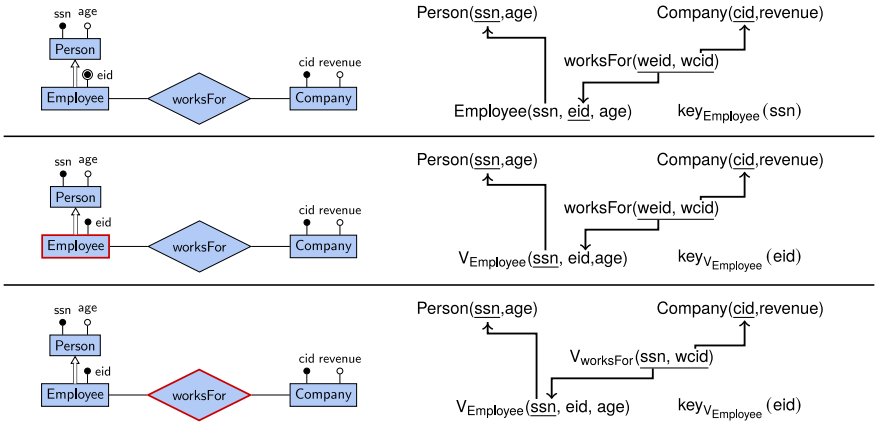


Fig. 2. The application of patterns can induce a refactoring of the DB schema and conceptual model.

Example 1. Consider the situation depicted in the top row of Fig. 2. For conciseness, we omit datatypes. We use schema-driven patterns to derive an ontology and mappings.

Under such a configuration of conceptual model/DB schema, we can only apply Pattern **SE** on Company or Person. As IRI template functions, we adopt here the W3C-DM convention, assuming the *base IRI* http://www.example.com/.[13]

We start with the entity Company. The application of Pattern **SE** yields the following mapping assertion and ontology axioms:

Mappings (Datatypes and mandatory participations omitted)	Ontology	
s: SELECT * FROM Company t: :Company/cid={cid} a :Company ; :Company#cid {cid} ; :Company#revenue {revenue} .	:Company a owl:Class . :Company#cid a owl:DatatypeProperty . :Company#revenue a owl:DatatypeProperty .	:Company#cid rdfs:domain :Company . :Company#revenue rdfs:domain :Company .

We proceed similarly for entity Person:

Mappings (Datatypes and mandatory participations omitted)	Ontology	
s: SELECT * FROM Person t: :Person/ssn={ssn} a :Person ; :Person#ssn {ssn} ; :Person#age {age} .	:Person a owl:Class . :Person#ssn a owl:DatatypeProperty . :Person#age a owl:DatatypeProperty .	:Person#ssn rdfs:domain :Person . :Person#age rdfs:domain :Person .

Since the IRI template for the superclass Person has been established, Pattern **SHa** becomes now applicable over entity Employee:

Mappings (Datatypes and mandatory participations omitted)	Ontology
s: SELECT * FROM Employee t: :Person/ssn={ssn} a :Employee ; :Employee#eid {eid} .	:Employee a owl:Class . :Employee#eid a owl:DatatypeProperty . :Employee#eid rdfs:domain :Employee .

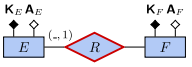
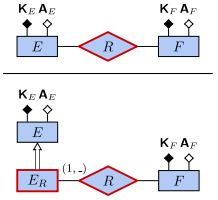
As by-product of the application of such pattern, we also obtain the updated conceptual model and DB schema depicted in the middle row of Fig. 2. Such by-product enables the application of Pattern **SRa** over relationship worksFor, leading to the following mapping assertion and ontology axioms:

Mappings (Datatypes and mandatory participations omitted)	Ontology
s: SELECT ssn, wcid FROM Employee JOIN worksFor ON weid=eid t: :Person/ssn={ssn} :worksFor :Company/cid={wcid} .	:worksFor a owl:ObjectProperty . :worksFor rdfs:domain :Employee . :worksFor rdfs:range :Company .

No further pattern is applicable, and the obtained ontology is indeed a DB ontology because it represents all the entities and relationships in the conceptual model.

[13] Hence, the prefix ":" is associated to the IRI http://www.example.com/.

Table 5
Data-driven patterns.

Conceptual model	DB schema	Mappings	Ontology
Data Entity with Merged 1-N Relationship and Entity (DR1Nm)			
	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E, \mathbf{K}_F, \mathbf{A}_F)$ $\mathsf{fd}(T_E\colon \mathbf{K}_F \to \mathbf{A}_F)$ $V_E(\underline{\mathbf{K}_E}, \mathbf{K}_{EF}, \mathbf{A}_E) = \pi_{\mathbf{K}_E, \mathbf{K}_F, \mathbf{A}_E}(T_E)$ $V_F(\underline{\mathbf{K}_F}, \mathbf{A}_F) = \pi_{\mathbf{K}_F, \mathbf{A}_F}(T_E)$	$s\colon V_E$ $t\colon p_R(\mathtt{t}_E(\mathbf{K}_E), \mathtt{t}_F(\mathbf{K}_{EF}))$	$\exists p_R \sqsubseteq C_E$ $\exists p_R^- \sqsubseteq C_F$
Mappings and ontology axioms for classes C_E and C_F, not shown here, conform to Pattern **SE/DE** on the newly introduced views V_E and V_F. All considerations on cardinality constraints and optional attributes described for Pattern **SRm** extend to this pattern in the natural way.			
Data Entity with Optional Participation in a Relationship (DH01)			
	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E)$ $T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_R(\underline{\mathbf{K}_{RE}, \mathbf{K}_{RF}})$ $T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E)$ $T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_R(\underline{\mathbf{K}_{RE}, \mathbf{K}_{RF}})$ $V_{E_R}(\underline{\mathbf{K}_E}) = \pi_{\mathbf{K}_{RE}}(T_R)$	$s_1\colon V_{ER}$ $t_1\colon C_{E_R}(\mathtt{t}_{C_E}(\mathbf{K}_E))$ $s_2\colon T_R$ $t_2\colon p_R(\mathtt{t}_{C_E}(\mathbf{K}_{RE}), \mathtt{t}_{C_F}(\mathbf{K}_{RF}))$	$C_{E_R} \sqsubseteq C_E$ $\exists p_R \equiv C_{E_R}$ $\exists p_R^- \sqsubseteq C_F$
This pattern extends in the natural way to the variants with identifier alignment (see Pattern **SRa**) and reified relationship (see Pattern **SRR**). All considerations on cardinality constraints and optional attributes described for Patterns **SE** and **SRm** extend to this pattern in the natural way.			
Clustering Entity to Class (CE2C)			
 $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $part(\mathbf{B}\ E)$ partitions to classes	$T_E(\underline{\mathbf{K}}, \mathbf{A})$ $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $part(\mathbf{B}, E)$	$\left\{ \begin{array}{l} s\colon \sigma_{\mathtt{p}}(T_E) \\ t\colon C_E^{\mathtt{p}}(\mathtt{t}_{C_E}(\mathbf{K})) \end{array} \right\}_{\mathtt{p} \in part(\mathbf{B},E)}$	$\{C_E^{\mathtt{p}} \sqsubseteq C_E\}_{\mathtt{p} \in part(\mathbf{B},E)}$
Clustering Entity to Object (CE2O)			
 $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $part(\mathbf{B}, E)$ partitions to objects	$T_E(\underline{\mathbf{K}}, \mathbf{A})$ $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $part(\mathbf{B}, E)$	$\left\{ \begin{array}{l} s\colon \sigma_{\mathtt{p}}(T_E) \\ t\colon p_{\mathbf{B}}(\mathtt{t}_{C_E}(\mathbf{K}), \gamma_{\mathtt{p}}) \end{array} \right\}_{\mathtt{p} \in part(\mathbf{B},E)}$	$\exists p_{\mathbf{B}} \sqsubseteq C_E$
Clustering Entity to Data Value (CE2D)			
 $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $part(\mathbf{B}, E)$ partitions to values	$T_E(\underline{\mathbf{K}}, \mathbf{A})$ $\mathbf{B} \subseteq \mathbf{K} \cup \mathbf{A}$ $part(\mathbf{B}, E)$	$\left\{ \begin{array}{l} s\colon \sigma_{\mathtt{p}}(T_E) \\ t\colon d_{\mathbf{B}}(\mathtt{t}_{C_E}(\mathbf{K}), \xi_{\mathtt{p}}) \end{array} \right\}_{\mathtt{p} \in part(\mathbf{B},E)}$	$\delta(d_{\mathbf{B}}) \sqsubseteq C_E$

3.3. Data driven mapping patterns

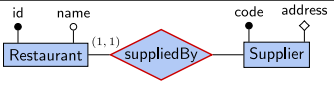
Data-driven patterns are mapping patterns that depends both on the schema and on the actual data in the DB. They are not limited to the variants corresponding to the schema-driven patterns, but they also comprehend specific patterns that do not have a corresponding schema version, e.g., due to *denormalized tables*. Such patterns, for which we provide a detailed description below, are shown in Table 5. Similarly as we did for schema-driven patterns, we provide an example and references to related literature for each data-driven pattern.

Data Entity with Merged 1-N Relationship and Entity (DR1Nm). This pattern describes the situation where both the relationship and the participating entity have been merged into a table. It considers a table T_E that has, besides its primary key $\mathbf{K}_E$, also attributes $\mathbf{K}_F$ which functionally determine attributes $\mathbf{A}_F$. Observe that the latter condition is not possible if the DB schema is in 3rd normal form[14] [26]. When this pattern is applied, the key $\mathbf{K}_F$ and the attributes $\mathbf{A}_F$ that go along with it, can be projected out from T_F,

[14] A straightforward variant of this pattern, violating the 2nd normal form, could be added to our list.

resulting in a view V_F to which further patterns can be applied, for instance Pattern **SE**. An additional view V_E is also created, representing the entity E.

Example: A table restaurant containing information about restaurants, their unique supplier (identified by a code), and the address of the supplier. The supplier identifier, which is not a key for restaurant, uniquely determines the address of the supplier. Table restaurant will be vertically partitioned into two views, restaurant and suppliers, that will later be linked to their respective ontology classes Restaurant and Supplier through the applications of Pattern **SE** on the fresh views. An object property suppliedBy is created connecting restaurants to their suppliers.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	restaurant(id, name, supC, supA) fd(restaurant: supC → code)

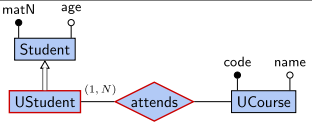
Mappings	Ontology	
s: SELECT id, supC FROM V-restaurant t: :restaurant/{id} :suppliedBy :supplier/{supC} . V-restaurant := SELECT id, name, supC FROM restaurant	:suppliedBy a owl:ObjectProperty . :Restaurant rdfs:subClassOf _:r1 . _:r1 a owl:Restriction . _:r1 owl:onProperty :suppliedBy . _:r1 owl:someValuesFrom rdfs:Literal .	:suppliedBy rdfs:domain :Restaurant . :suppliedBy rdfs:range :Supplier .

References: Slight variants for this pattern can be found in the literature:

- BootOX [18] reports a mappings generation strategy for situations similar to the one of Pattern **DR1Nm**. However, details of how this is actually carried out are not provided.
- Pattern *Relationship: One to Many with Duplicates* from [30] handles the same situation as Pattern **DR1Nm**, proposing a similar solution. Such work also proposes an alternative solution in Pattern *Relationship: One to Many without Duplicates*, where the primary identifier of table T_E is used to create both IRIs for C_E and C_F.

Data Entity with Optional Participation in a Relationship (DH01). This pattern describes the situation where a non-mandatory relationship is transformed into a mandatory one through the introduction of a subconcept on one of its participating roles. It is characterized by a table T_E that represents the merge of a child entity E_R into a father entity E, and E_R has a mandatory participation in a relationship R. The join between the tables T_R and T_E identifies the objects in E that are instances of E_R, and is used in a mapping to create instances of the concept C_{R_E}, as well as the object property R connecting E_R to F. This pattern produces a view V_{E_R}, to which further patterns can be applied.

Example: A table student and a table attends relating students to undergraduate courses. Each student participating in such relationship is an undergraduate student.

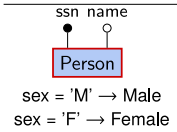
Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	student(matN, age) ucourse(code, name) attends(smatN, ccode)

Mappings	Ontology	
s1: SELECT smatN FROM v-UStudent t1: :student/{smatN} a :UStudent . s2: SELECT smatN, ccode FROM attends t2: :student/{smatN} :attends :course/{ccode} . v-UStudent := SELECT smatN FROM attends	:UStudent a owl:Class . :attends rdfs:domain :UStudent . :UStudent rdfs:subClassOf _:r1 . _:r1 owl:onProperty :attends .	:attends a owl:ObjectProperty . :attends rdfs:range :UCourse . _:r1 a owl:Restriction . _:r1 owl:someValuesFrom owl:Thing .

References: To the best of our knowledge, only BootOX [18] reports a mapping generation strategy handling a scenario that resembles the one of Pattern **DH01**. However, details of how this is actually carried out are not provided.

Clustering Entity to Concept/Data Property/Object Property (CE2C/CE2D/CE2O). Such patterns are characterized by an entity E and a *derivation rule* defining sub-entities of E according to the values for attributes **B** in E. Instances in these sub-entities can be mapped to objects in the subclasses $C_E^{\mathfrak{p}}$ of the ontology (Pattern **CE2C**), to objects connected through a data property to some literal constructed through a *value invention* function ξ applied on a partition $\mathfrak{p}$ (Pattern **CE2D**), or to objects (i.e., IRIs) constructed through an *object invention* function γ applied on $\mathfrak{p}$ (Pattern **CE2O**).

Example: A table person containing people with an attribute defining their sex and ranging over 'M' and 'F'. The ontology defines two sub-classes Male and Female of the class Person (corresponding to the entity Person). Then, Pattern **CE2C** clusters the table according to the sex attribute, so as to obtain objects to become instances of either of the two classes.

Conceptual model (datatypes omitted)	DB schema (datatypes omitted)
	person(ssn, name, sex)

Mappings	Ontology	
s1: SELECT ssn FROM person WHERE sex='M'		
t1: :person/{ssn} a :Male .	:Male a owl:Class .	:Male owl:subClassOf :Person
s2: SELECT ssn FROM person WHERE sex='F'	:Female a owl:Class .	:Female owl:subClassOf :Person .
t2: :person/{ssn} a :Female .		

Alternatively, the ontology could define a data property hasSex, ranging over the two RDF literals "Male"and "Female". Then, Pattern **CE2D** clusters the table according to the sex attribute, so as to obtain objects to be linked to either of the two RDF literals.

References: For what concerns the **CE2C** variant, the clustering pattern is related to a number of works:

- In BootOX [18], an automated approach that mentions "clustering" is reported. However, the clusters in their approach are sets of "similar" tuples. This is different from our Pattern **CE2C**, which instead requires a set of columns whose values explicitly determine the different clusters.
- In [30], Patterns *Complex Concept: Conditions* and *Complex Concept: Data as Concept* share the idea of imposing a condition in order to identify subsets of a table and creating concepts out of them. However, it has to be noted that ontology axioms are not in the scope of that work. Pattern *Complex Concept Attribute: Constant Value* is similar to our Pattern **CE2D**, since it associates objects satisfying a certain equality filter condition to a constant data value.

Example 2. Consider again the conceptual model, DB schema, mappings, and ontology derived in Example 1. Assume a derivation rule on entity Person identifying two sub-entities: an entity representing those whose age is greater than or equal to 18, which we call OfAge, and another one representing the others, which we call UnderAge. Under these assumptions, we can apply Pattern **CE2C** and obtain:

Mappings (Datatypes and mandatory participations omitted)	Ontology	
s1: SELECT ssn FROM person where age >= 18		
t1: :Person/ssn={ssn} a :OfAge .	:OfAge a owl:Class .	:UnderAge a owl:Class .
s2: SELECT ssn FROM person where age < 18	:OfAge owl:subClassOf :Person .	:UnderAge owl:subClassOf :Person .
t2: :Person/ssn={ssn} a :Underage .		

3.4. Variations and combinations

More complex patterns arise from the combination of the patterns described so far. For instance, recall the example we discussed for Pattern **DH01**. Graduate students, which are a by-product of the application of such pattern, might be in relationship with an entity Graduation. The object property capturing the relationship might be created by applying Pattern **DR**. In our analysis, we have observed that combinations are quite common in those VKG specifications where the DB has been created independently from the ontology.

Another important variation is the one introduced by modifiers, such as *value invention or combination*, in which DB values are used and combined to get RDF literals, typically by relying on R2RML *templates*. We have already encountered an instance of value invention, specifically when we introduced the **CE2D** pattern.

3.5. Automatic discovery of data-driven patterns

When it comes to discovering data-driven patterns, our methodology may benefit from techniques that were developed in the research discipline of schema matching [3]. Over the years, the proposed methods were shown to serve as a solid basis to handle small-scale schemata, typically encountered as a part of a mapping process [6]. Schema matching becomes handy when a pattern involves two (or more) under-specified schemata. By way of motivation, consider the case of an implicit relationship (Pattern **DR**). In such a case, the mapping may consider several relation pair candidates that may be semantically interpreted as representing a missing relationship. Let T_E and T_F with primary keys $\mathbf{K}_E = K_{E_1}, \ldots, K_{E_n}$ and $\mathbf{K}_F = K_{F_1}, \ldots, K_{F_m}$, respectively, be a relation candidate pair. A matching process between $\mathbf{K}_E$ and $\mathbf{K}_F$ aligns their attributes using *matchers* that utilize matching cues such as attribute names, instance data, schema structure, etc. Accordingly, the matching process yields similarity values (typically a real number in $[0, 1]$) between $K_{E_i} \in \mathbf{K}_E$ and $K_{F_j} \in \mathbf{K}_F$. These values are then used to deduce a *match* $\sigma(\mathbf{K}_E, \mathbf{K}_F)$. Such a match may comply with different constraints as set by the environment, e.g., a one-to-one matching. For example, the similarity values may be assigned using a string similarity matcher like [47]

$$\frac{len(K_{E_i}.name \cap K_{F_j}.name)}{max(len(K_{E_i}.name), len(K_{F_j}.name))}$$

and a match may be inferred using a threshold selection rule [4]. Once a match is obtained, it may serve as a realization of a schema relationship mapping pattern. Obviously, not all tables have relationships. Thus, one should decide which of the generated relationships should be included in the final mapping. To do so, we should assess the quality of the matching outcome, allowing to rank among them [48] (e.g., selecting matches with high similarity values to be included in the final mapping). Match quality may also be learned as a domain specific input introducing other quality measures to the usefulness of a match.

The schema matching literature offers a mechanism to map multiple schemata to a global schema, which bears similarity to Pattern **SR1Nm**. *Schema cover* [49] matches parts of schemata (subschemata) with concepts, using schema matching techniques, aiming at covering all attributes of the global schema with minimum number of overlaps between the subschemata. Using a similar approach, one can "cover" a schema by using multiple ontology concepts to generate an instance of Pattern **SR1Nm**. Recalling the example we discussed for Pattern **SR1Nm**, using the schema cover methodology, we can cover the restaurants table using the properties relative to restaurants defined in the ontology, and the data property address from the concept Supplier.

4. Usage scenarios for VKG patterns

We now comment on how having a catalog of patterns for VKG specifications is instrumental in a number of usage scenarios. The provided list is by no means complete, but gives a fair account on the usefulness of our approach.

Debugging of a VKG Specification. This scenario arises when a full VKG specification is already in place and must be debugged. If the conceptual model is available, for instance in the form of knowledge by the data curators, each component of the specification can be checked for compliance against the patterns. Such checks are more effective on ontologies that are closer to the actual conceptual representation of the DB, and can answer questions such as whether the relationships are being correctly represented in terms of their domains, range, and mandatory participations, or whether subclasses are defined using the correct templates. Ontologies whose structure is far from the DB ontology require more effort. A possibility is to adopt a two-step process of deriving the DB ontology first, conforming to the patterns, and then checking whether there exists a "lossless" *alignment* between the DB ontology and the target one (in line with the intuition of Fig. 1).

Conceptual Model Reverse Engineering. Another relevant scenario arising when a full VKG specification is given, is that of inferring a conceptual model of the DB that represents the domain of interest by reflecting the content of the VKG specification. Here the ontology provides the main source to reconstruct entities, attributes, and relationships, while the DB and the mappings provide the basis to ground the conceptual model in the actual DB, and to infer additional constraints that are not captured by the ontology (e.g., due to limited expressivity of OWL 2 QL). As for the debugging case, also this approach works best if the ontology is semantically close to the conceptual model.

Mapping Bootstrapping. In this scenario, the DB and the ontology are given, but mappings relating them are not. We envision this as a two-step process: in the first step, we use our patterns to derive a well-structured DB ontology; in the second step, we rely on techniques coming from the field of *ontology matching or alignment* [50] to produce so-called *alignment mappings*, which relate the DB ontology to the actual target ontology. Schema patterns are the most suitable ones to automatically guide the bootstrapping process. When patterns contain tables that merge multiple entities/relationships, the presence of a conceptual model becomes crucial to disambiguate the mappings to be bootstrapped. This is, e.g., the case for Pattern **DR1Nm** and the patterns based on clustering. If the conceptual model is not available in this tricky case, bootstrapping can still be attempted by relying on schema matching techniques [3], as done in BootOX [18]. Specifically, schema matching comes handy when a pattern involves two (or more) under-specified schemata. For instance, in the case of Pattern **DR**, *pair candidates* between primary keys can be *matched* in order to make implicit relationships explicit. This can be done through matchers (such as string similarity matchers [47]) that employ attribute names, instance data, schema structure, etc. To separate genuine relationships from false positives generated by poor matchers, ranking techniques have to be employed [48].

Ontology+Mapping Bootstrapping. Here, neither the ontology nor the mappings are given as input, and have to be synthesized. This scenario can be tackled as the **Mapping Bootstrapping** one, by omitting the second step. As already discussed, best results are to be expected when a conceptual model is available, since the obtained ontology will likely be closer to the level of abstraction expected by the domain experts.

VKG Bootstrapping. In this scenario, we just have a conceptual model of the domain, and the goal is to set up a VKG specification. The conceptual model can be then transformed into a normalized DB schema using well-established *relational mapping* techniques (e.g., [24]). At the same time, as pointed out above, a direct encoding into ontology axioms can be applied to bootstrap the ontology. The generation of mappings becomes then a quite trivial task, considering that the induced DB and ontology are very close in terms of abstraction. This setting resembles, in spirit, that of *object-relational mapping*, used in software engineering to instrument a DB and corresponding access mechanisms starting from classes written in object-oriented code.

5. Analysis of scenarios

In this section we look at a number of VKG scenarios in order to understand how patterns occur in practice, and with which frequency. To this purpose, we have gathered 6 different scenarios,[15] coming either from the literature on VKGs, or from actual real-world applications. Table 6 shows the results of our analysis, and for each cell pattern/scenario, it reports the number of applications

[15] Available here: https://github.com/ontop/ontop-examples/tree/master/dke-2022-mapping-patterns/.

Table 6
Occurrences of mapping patterns over the considered scenarios.

	BSBM		NPD		UOBM		ODH		ST-OD		Cordis		Total	
SE	8	52	34	406	8	16	10	43	8	37	13	60	81	614
SR	–	–	–	–	2	2	–	–	–	–	3	3	5	5
SRm	8	8	74	74	5	5	–	–	7	7	10	10	104	104
SEw/DEw	–	–	30	266	1	1	–	–	–	–	–	–	31	267
SRR	–	–	1	12	–	–	–	–	–	–	1	16	2	28
SH	–	–	3	132	–	–	–	–	–	–	–	–	3	132
DE	–	–	–	–	–	–	–	–	3	7	4	9	7	16
DRm	5	5	17	17	36	36	2	2	1	1	2	2	63	63
DH	–	–	–	–	5	9	–	–	–	–	–	–	5	9
DRR	2	2	–	–	–	–	–	–	–	–	–	–	2	2
DR1Nm	4	4	19	54	–	–	6	78	14	29	1	1	44	166
DH01	–	–	–	–	–	–	–	–	–	–	1	2	1	2
CE2C	–	–	11	82	6	19	5	23	–	–	1	12	23	136
CE2D	–	–	23	49	–	–	–	–	–	–	–	–	23	49
CE2O	–	–	13	148	–	–	–	–	–	–	–	–	13	148
UNKNOWN	–	–	3	6	2	13	1	4	1	12	4	9	11	44

of that pattern over that scenario (leftmost number in the cell) and the number of mappings involved (rightmost number in the cell). The last column in the table reports total numbers. We have manually classified a total of 1559 mapping assertions, falling in 407 applications of the described patterns. Of these applications, about 52.8% are of schema-driven patterns, 44.7% of data-driven patterns, and 2.5% are of patterns falling outside of our categorization. In the remainder of this section we describe the detailed results for each scenario. In [51] we present a similar evaluation, but restricted to schema-driven patterns and based on the results of an automated tool able to discover patterns starting from DB schemas.

Berlin Sparql Benchmark (BSBM) [52]. This scenario is built around an e-commerce use case in which products are offered by vendors and consumers review them. Such benchmark does not natively come with mappings, but these have been created in different works belonging to the VKG literature. We analyzed those in [53]. The ontology in BSBM reflects quite precisely the actual organization of data in the DB. Due to this, each mapping falls into one of the patterns we identified. Notably, in the DB foreign key constraints are not specified. Therefore, we notice a number of applications of data-driven patterns, which cannot be captured by simple approaches based on W3C-DM.

NPD Benchmark (NPD) [54]. This scenario is built around the domain of oil and gas extraction. It presents the highest number of mappings (>1k). The majority of these were automatically generated, and fall under W3C-DM or schema-driven patterns. There are, however, numerous exceptions. Mainly, there are a few denormalized tables that require the use of Pattern **DR1Nm**, such as for the following mapping:

```
target  npd:quadrant/{wlbNamePart1} a npdv:Quadrant .
source  SELECT "wlbNamePart1" FROM "wellbore_development_all"
```

A quadrant is not an entity in the DB schema (because wlbNamePart1 is not a key of wellbore_development_all), but it is represented as a class in the ontology. Moreover, quadrants have themselves their own data (resp., object) properties, triggering the application of other patterns in composition with Pattern **DR1Nm**.

University Ontology Benchmark (UOBM) [55]. This scenario is built around the academic domain. Such benchmark provides a tool to automatically generate OWL ontologies, but does not include mappings nor a DB instance. These two have been manually crafted in [56], by reverse-engineering the ontology. The mappings in this setting are quite interesting, and are mostly data-driven, as witnessed by the many applications of the clustering patterns. One critical aspect about these mappings is the use of a sophisticated version of the identifier alignment pattern modifier. Specifically, the table People has the following primary key:

```
PRIMARY KEY (ID,deptID,uniID,role)
```

Table GraduateStudent, which at the conceptual level corresponds to a subclass of the class People, has the following key, which is incompatible with the one of the superclass:

```
PRIMARY KEY (studentID,deptID,uniID)
```

The subclass relation between People and GraduateStudents requires the two keys to be aligned. This is done "artificially", in the sense that the missing field role is created on-the-fly by the mapping:

```
source  SELECT deptID, univID, studID, 'GraduateStudent' as role FROM GraduateStudents
target  <http://www.Dept{deptID}.Univ{univID}.edu/{role}{studID}> a :GraduateStudent .
```

Suedtirol OpenData (ST-OD).[16] This is an application scenario coming from the turism domain. The ontology has been created independently from the DB. Moreover, the DB is itself highly de-normalized, since it is essentially a relational rendering of a

[16] https://github.com/dinglinfang/suedTirolOpenDataOBDA.

JSON file. These aspects have a direct impact on the patterns we observed. In particular, we identified several applications of Pattern **DR11m**, which, as we discussed, poses a huge challenge to automatic generation of mappings. Further complications arise from a number of applications of the value invention pattern modifier, which appears quite often in the form, for instance, of language tags:

```
source  SELECT istat_code, name_i, name_d FROM municipalities
target  :mun/mun={istat_code} a :Municipality ; rdfs:label {name_i}@it, {name_d}@de .
```

Open Data Hub VKG (ODH).[17] This setting is the one behind the SPARQL endpoint located at the Open Data Hub portal from the Province of Bozen-Bolzano (Italy). This setting is also a denormalized one, and the same considerations we made for ST-OD apply to this setting as well.

Cordis.[18] This setting is provided by SIRIS Academic S.L., a consultancy company specialized in higher education and research, and is designed around the domain of competitive research projects. As opposed to the previous two scenarios, this one comes with a well-structured relational schema, which reflects in a number of applications of schema patterns. Although in this scenario we have DB views, such views have explicit constraints defined on them (such as, `UNIQUE` constraints in SQL) that allow for the application of schema patterns.

6. Related work

In the last two decades a plethora of tools and approaches have been developed to bootstrap an ontology and mappings from a DB. The approaches in the literature differ in terms of the overall *purposes* of the bootstrapping (e.g., VKGs, data integration, ontology learning, check of DB schema constraints using ontology reasoning), the *ontology and mapping languages* in place (e.g., OWL 2 profiles or RDFS, as ontology languages, and R2RML or custom languages, for the specification of mappings), the different focus on *direct and/or complex mappings*, and the assumed *level of automation*. The majority of the most recent approaches closely follow W3C-DM, deriving ontologies that mirror the structure of the input DB, and are equipped with further ontology-to-ontology mapping techniques and custom mapping definition interfaces and languages in order to support the user in aligning the extracted ontology with domain-specific ontologies whose concepts and relationships pertain to a given domain of interest.

A notable example in this category is represented by the D2RQ system [57]. Once in its automated mode, D2RQ relies on the implementation of W3C-DM, and of additional reverse engineering methods for the discovery of many-to-many relationships and the translation of foreign keys into object properties. Manual and semi-automatic modes are also allowed in the system. In these cases, the user has available an RDF-compatible mapping language that can be used to map subset only of a relation, specify the mechanism for the generation of the individual IRIs in the ontology and schemes for the translation of database values. Another tool that complements the automatic extraction of database-to-ontology mapping is Ultrawrap Mapper [58]. The main aim of this system is to hide the complexity of the R2RML language to the user by offering a bootstrapping of the mapping process which is based on an enhanced version of W3C-DM. The user can interact with the system in several ways: by choosing which tables and attributes of the database are to be mapped, refining the automatically extracted ontology, suggesting that specific data values have to be treated as ontology classes, specifying domain-driven SQL views to represent new concepts and, finally, by uploading a target domain-ontology for which specific mapping recommendations will be provided based on ontology matching techniques.

A special category of bootstrapping tools that is worth to mention is the one including those systems that implement extensions of W3C-DM, while keeping a semi-automatic or fully automatic approach to mapping generation. A system that is representative of this category is BootOX [18], which relies on the R2RML language to produce direct mappings. BootOX implements a (fully-automatic) schema-driven bootstrapping. It aims at facilitating creation of an ontology by automatic extraction through bootstrapping from relational databases. Differently from other systems, BootOX covers all the OWL 2 profiles of the output ontologies and suggest how to implement the ontology axioms resulting from a given mapping according to them (e.g., "functionality" or "inverse functionality" of a given data property are covered by the OWL 2 RL and OWL 2 EL profiles only, as well as "key axiom" for a given class). The user can select the preferred OWL 2 profile, as well as special characteristics of the extracted properties (e.g., symmetry, transitivity, or reflexivity) which may not be explicitly present in the source database. Besides W3C-DM, BootOX supports the user in building complex mappings based on selection and/or join operators. The proposed extensions to W3C-DM are about mapping patterns that resolve into sub-class relationships and class hierarchies. To this aim, clusters of tuples in a table are detected by means of numerical vectors distance metrics, as well as subsets of the attributes of a table (especially when tables are not in BCNF normal form) with repeated values, and special evaluation of the application of outer join operations among tables. Still, the generation of these complex mappings heavily relies on the user intervention, e.g., in the naming of the newly discovered classes and properties. The generation of *provenance mappings* (at different levels of granularity) is also an extension to W3C-DM provided by BootOX: provenance information, such as the source database from which the information is extracted or the table and column identifiers, are modeled in the mapping assertions. BootOX is able to perform ad-hoc alignment with a target ontology using the LogMap system [59–61].

MIRROR [45] is also a tool for the automatic generation of R2RML mappings that extends W3C-DM. Taking a DB as input (which is assumed to be, at least, in 3NF), MIRROR generates two groups of mappings: a first one strictly derived from the application of

[17] https://sparql.opendatahub.bz.it/.

[18] https://www.sirisacademic.com/wb/.

W3C-DM, and a second one complementing the first with additional transformations that are extracted from information, such as class hierarchies and many-to-many relationships, that are usually not directly derivable from a database schema. From the point of view of the present contribution, the MIRROR categorization of the relationships between tables (two or more) that can be observed in the physical implementation of a relational database is especially interesting. Most of the categories identified in MIRROR can be linked with the mapping patterns introduced in the present paper, as we have seen in Section 3.2.

Rather than providing a bootstrapping algorithm, some works have studied the foundational framework underlying W3C-DM and proved a number of results [20,62]. Specifically, [20] presents a *Direct Mapping* (different from W3C-DM) that enjoys *query preservation*, that is, every query that can be posed over the original DB schema can be expressed as a query over the mapped KG. It also shows that Direct Mapping is not *lossless*, due to the impossibility of rendering constraints such as foreign and primary keys in the OWL 2 language. Similar conclusions are reached by [62], which instead of OWL 2 uses SHACL *constraints* [63], in order to better translate DB (closed-world) constraints into their KG counterparts.

Bootstrapping approaches based on learning techniques and similarity measures are also present in the literature. In [64–66], for instance, ontology learning techniques are used to mine the source data and discover patterns that suggest how to specify the target ontology. The core of RDBToOnto [66] is based on the learning algorithm RTAXON [65], which performs hierarchy mining in the data to identify categorization patterns from which class hierarchies can be generated. RDBToOnto (which is currently part of the commercial portfolio of software by the RedcoolMedia company[19]) also allows users to iteratively refine the results provided by the automatic generation of the ontology, by suggesting a number of predefined constraints to be added (e.g., the selection of relevant categorization attributes and user-defined instance naming). The tool also includes a database normalization step driven by the identification of specific inclusion dependencies by the user, whose main aim is to try to eliminate data duplication and, more in general, redundancy due to bad design in the source tables. Unfortunately, we have not been able to find a description of the mappings generated by the tool, and this prevents us from a deeper comparison with our approach.

In schema matching literature, simple rule-based mappings are used to create a uniform representation of the data sources to be matched, may they be schemata or ontologies [67–69]. For example, in COMA++ [67], concept hierarchies, attributes and relationship types are mapped into generic model representation based on directed acyclic graphs. Using such mappings, schema matchers are applied to the uniform representation to create a matching result. Similarly, IncMap [69] relies on a graph structure called IncGraph to represent schema elements from both ontologies and relational schemata in a unified way. Therefore, the main algorithmic task is to convert the ontology as well as the relational schema into the unified IncGraph representation. IncMap then computes ranked correspondences between elements of the two graphs using lexical and structural similarities, based on the Similarity Flooding algorithm, and converts the correspondences into direct mappings between the ontology and schema. The entire process is semi-automatic and each suggestion needs to be accepted by the user (human inputs are used to rerank correspondences after each round of interaction).

The Karma system [70] provides support for extracting data from a variety of sources (relational databases, CSV files, JSON, and XML), for cleaning and normalizing data, for mapping them to a target vocabulary, for integrating multiple data sources, and for publishing in a variety of formats (CSV, KML, and RDF). Karma does not support a fully automatic mapping generation: it supports the users with a sophisticated user interface where an OWL-specified vocabulary, one of more data sources and a database of so-called semantic types are assumed as input. The algorithmic core of the system, which computes the relationships among the schema elements of a source, is based on conditional random fields (CRF) [71] and a Steiner tree algorithm. The idea is that Karma automatically infers the semantic types (i.e., pairs made of a concept and/or a data property of the domain ontology, such as, <Person,name> or <Artwork,title>) that it has been trained to identify in the source tables, whereas it asks the user to explicitly specify them whenever this process fails. In a second step, those semantic labeled source attributes are related to each other in terms of the properties of the target ontology in order to reconstruct the overall semantic model of the data source (for instance, in the case above, painter is then suggested to be the right relationship between Person and Artwork, instead of owner or sculptor). More details about the way Karma exploits the knowledge from a domain ontology and the semantic models of previously modeled sources to automatically learn a rich semantic model for a new source can be found in [72,73]. These papers focus on the characteristics of the algorithms that are responsible for the generation of the semantic models of the sources, but the final mappings are never explicitly specified according to a standard mapping language. Nonetheless, the CRF-based approach is declared to encompass other existing approaches base on schema matching techniques that have been used to identify the semantic types of source attributes by comparing them with already labeled ones, such as [74], or [75], which is based on learning regular expression-like rules for data in the source columns.

Although the work in [76] starts from a setting that is different from the one assumed in this paper, it is worth mentioning it. The paper focuses on automatically compiling R2RML mappings, once a set of algebraic correspondences between a relational source and a target ontology have been manually specified by the user. The so-called "Correspondence Assertions" are a set of algebraic assertions that are meant to express data-metadata mapping, mappings containing custom value functions (e.g., transformation functions to change the data formats), and union, intersection, and difference between tables. SQL views are then created on the basis of the specified user assertions and used as an intermediate step to automatically compile the final R2RML mappings. The design and implementation of an authoring tool supporting the user in the specification of the algebraic correspondence assertions was also foreseen.

As for a systematic categorization of mappings, the proposals in [29,30] are very closely related to ours, as they also introduce a catalog of mapping patterns. However, there are major differences to our proposal, since in those works:

[19] https://www.redcoolmedia.net/

- patterns are not formalized, and are presented in a "by-example" fashion following the R2RML syntax;
- patterns are derived from "commonly-occurring mapping problems" based on the experience of the authors, whereas in our work patterns are derived from conceptual modeling and database design principles;
- patterns are not evaluated against a number of different real-world, and complex scenarios over heterogeneous domains and design practices, as has been done here.

Other works provide a systematic categorization of mappings in VKGs [18,45], however they do so while focussing on supporting mapping bootstrapping. Other contributions restrict their attention to the algorithms behind the generation of mappings, notably [36,58,69] for the R2RML language. We mention also some surveys and comparative analyses [16,77–80], where the interested reader can further explore the tools and techniques mentioned here. We finally notice that, in our review, we did not find any study introducing an in depth analysis of existing real scenarios of DB-to-ontology mapping, as we do in the present paper, aimed at showing that the identified categories actually reflect the real design choices and methodologies in use by the mapping designers.

7. Conclusions and future work

In this work we identified and formally specified a number of mapping patterns emerging when linking DBs to ontologies in a typical VKG setting. Our patterns are grounded to well-established practices of DB design, and render explicit the connection between the conceptual model, the DB schema, and the ontology. We argue that the organization in patterns can enable a number of relevant tasks, apart from the classic one of bootstrapping mappings in an incomplete VKG scenario. Through a systematic analysis of various VKG scenarios, ranging from benchmarks to real world and denormalized ones, we observed that the patterns we formalized occur in practice, and capture most cases.

This work is only a first step, with respect to both categorization of patterns, and their actual use. Regarding the former, our plan is to extend this initial catalog with more advanced data-driven patterns, such as those deriving from other kinds of transformations for the hierarchies in the conceptual model (e.g., the case where all children and their attributes are fully merged in the parent entity, or the inverse case), and to better explore the interaction between patterns and pattern modifiers, such as value invention or identifier alignment. Regarding the latter, in this paper we have used patterns to investigate, and highlight, the specific problems to address when setting-up a VKG scenario. We plan to investigate solutions to these problems, by exploiting approaches from other fields, e.g., schema matching.

One point we did not formally investigate in this work is which properties are guaranteed to hold when applying our patterns (such as losslessness or query preservation). Since our patterns are grounded in textbook methodologies from DB design, such as standard translations of ER-diagrams into the relational model, we believe that query preservation might work without any substantial modification. We plan to investigate these aspects in future work, in the same spirit as the literature in [20] or [62].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A link to supplemental material is provided in Section 5.

Acknowledgments

This research has been partially supported by the EU H2020 project INODE (grant agreement No. 863410), by the Italian Ministry of University and Research (MUR) under PRIN projects HOPE (Prot. 2017MMJJRE) and PINPOINT (Prot. 2020FNEB27), and by the Free University of Bozen-Bolzano, Italy through the projects MP4OBDA, ADAPTERS, and QUEST. Diego Calvanese also acknowledges the support of the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, Sweden, Avigdor Gal the support of the Benjamin and Florence Free Chair, and Roee Shraga the support of the National Science Foundation (NSF) under award number IIS-1956096.

References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, S. Kirrane, J.E.L. Gayo, R. Navigli, S. Neumaier, A.N. Ngomo, A. Polleres, S.M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge Graphs, Synthesis Lectures on Data, Semantics, and Knowledge, Morgan & Claypool Publishers, 2021.

[2] G. Xiao, L. Ding, B. Cogrel, D. Calvanese, Virtual Knowledge Graphs: An overview of systems and use cases, Data Intell. 1 (3) (2019) 201–223, http://dx.doi.org/10.1162/dint_a_00011.

[3] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, Very Large Database J. 10 (4) (2001) 334–350.

[4] H.-H. Do, E. Rahm, COMA++ system for flexible combination of schema matching approaches, in: Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB), Elsevier, 2002, pp. 610–621.

[5] C. Chen, B. Golshan, A.Y. Halevy, W.-C. Tan, A. Doan, BigGorilla: An open-source ecosystem for data preparation and integration, IEEE Data Eng. Bull. 41 (2) (2018) 10–22.

[6] R. Shraga, A. Gal, H. Roitman, ADnEV: Cross-domain schema matching using deep similarity matrix adjustment and evaluation, in: Proc. of the 46th Int. Conf. on Very Large Data Bases (VLDB) 13 (9), 2020, pp. 1401–1415.

[7] J. Euzenat, P. Shvaiko, Ontology Matching, Springer, 2007.

[8] V. Ivanova, B. Bach, E. Pietriga, P. Lambrix, Alignment Cubes: Towards interactive visual exploration and evaluation of multiple ontology alignments, in: Proc. of the 16th Int. Semantic Web Conf. (ISWC), in: LNCS, vol. 10587, Springer, 2017, pp. 400–417.

[9] P. Kolyvakis, A. Kalousis, D. Kiritsis, Deepalignment: Unsupervised ontology matching with refined word vectors, in: Proc. of NAACL, Association for Computational Linguistics, 2018, pp. 787–798.

[10] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison Wesley Publ. Co, 1995.

[11] G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, On reconciling data exchange, data integration, and peer data management, in: Proc. of the 26th ACM Symp. on Principles of Database Systems (PODS), 2007, pp. 133–142.

[12] P.G. Kolaitis, Schema mappings, data exchange, and metadata management, in: Proc. of the 24th ACM Symp. on Principles of Database Systems (PODS), 2005, pp. 61–75.

[13] M. Lenzerini, Data integration: A theoretical perspective, in: Proc. of the 21st ACM Symp. on Principles of Database Systems (PODS), 2002, pp. 233–246, http://dx.doi.org/10.1145/543613.543644.

[14] R. Fagin, L.M. Haas, M.A. Hernández, R.J. Miller, L. Popa, Y. Velegrakis, Clio: Schema mapping creation and data exchange, in: Conceptual Modeling: Foundations and Applications – Essays in Honor of John Mylopoulos, in: LNCS, vol. 5600, 2009, pp. 198–236, http://dx.doi.org/10.1007/978-3-642-02463-4_12.

[15] B. ten Cate, P.G. Kolaitis, K. Qian, W. Tan, Active learning of GAV schema mappings, in: Proc. of the 37th ACM Symp. on Principles of Database Systems (PODS), ACM, 2018, pp. 355–368.

[16] D.-E. Spanos, P. Stavrou, N. Mitrou, Bringing relational databases into the semantic web: A survey, Semantic Web J. 3 (2) (2012) 169–209.

[17] G. Fletcher, P. Groth, J.F. Sequeda, Knowledge scientists: Unlocking the data-driven organization, 2020, CoRR abs/2004.07917 arXiv:2004.07917 URL https://arxiv.org/abs/2004.07917.

[18] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M.G. Skjæveland, E. Thorstensen, J. Mora, BootOX: Practical mapping of RDBs to OWL 2, in: Proc. of the 14th Int. Semantic Web Conf. (ISWC), in: LNCS, vol. 9367, Springer, 2015, pp. 113–132, http://dx.doi.org/10.1007/978-3-319-25010-6_7.

[19] E. Kharlamov, D. Hovland, M.G. Skjæveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao, A. Soylu, D. Lanti, M. Rezk, D. Zheleznyakov, M. Giese, H. Lie, Y.E. Ioannidis, Y. Kotidis, M. Koubarakis, A. Waaler, Ontology based data access in Statoil, J. Web Semant. 44 (2017) 3–36, http://dx.doi.org/10.1016/j.websem.2017.05.005.

[20] J.F. Sequeda, M. Arenas, D.P. Miranker, On directly mapping relational databases to RDF and OWL, in: Proc. of the 21st Int. World Wide Web Conf. (WWW), ACM, 2012, pp. 649–658.

[21] D. Calvanese, T.E. Kalayci, M. Montali, A. Santoso, W. van der Aalst, Conceptual schema transformation in ontology-based data access, in: Proc. of the 21st Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW), in: LNCS, vol. 11313, Springer, 2018, pp. 50–67, http://dx.doi.org/10.1007/978-3-030-03667-6_4.

[22] D. Calvanese, M. Lenzerini, D. Nardi, Description logics for conceptual data modeling, in: J. Chomicki, G. Saake (Eds.), Logics for Databases and Information Systems, Kluwer Academic Publishers, 1998, pp. 229–264.

[23] D. Berardi, D. Calvanese, G. De Giacomo, Reasoning on UML class diagrams, Artificial Intelligence 168 (1–2) (2005) 70–118, http://dx.doi.org/10.1016/j.artint.2005.05.003.

[24] T. Halpin, T. Morgan, Information Modeling and Relational Databases, Morgan Kaufmann, 2010.

[25] M. Arenas, A. Bertails, E. Prud'hommeaux, J. Sequeda, A Direct Mapping of Relational Data To RDF, W3C Recommendation, W3C, 2012, available at http://www.w3.org/TR/rdb-direct-mapping/.

[26] E.F. Codd, Further Normalization of the Data Base Relational Model, Research Report / RJ / IBM /, San Jose, California RJ909, 1971.

[27] T. Halpin, T. Morgan, Information Modeling and Relational Databases, Morgan Kaufmann, 2010, (Chapter 11).

[28] D.W. Embley, B. Thalheim, Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges, Springer, 2011, (Chapter 5).

[29] J. Sequeda, F. Priyatna, B. Villazón-Terrazas, Relational database to RDF mapping patterns, in: Proc. of the 3rd Int. Conf. on Ontology Patterns, in: CEUR Workshop Proceedings, vol. 929, CEUR-WS.org, 2012, pp. 97–108.

[30] J. Sequeda, O. Lassila, Designing and Building Enterprise Knowledge Graphs, Morgan & Claypool, 2021, http://dx.doi.org/10.2200/S01105ED1V01Y202105DSK020.

[31] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semantics 10 (2008) 133–173, http://dx.doi.org/10.1007/978-3-540-77688-8_5.

[32] A. Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts, seventh ed., McGraw-Hill Book Company, 2020, URL https://www.db-book.com/db7/index.html.

[33] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL 2 Web Ontology Language Profiles, 2nd, W3C Recommendation, W3C, 2012, available at http://www.w3.org/TR/owl2-profiles/.

[34] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The DL-Lite family, J. of Automated Reasoning 39 (3) (2007) 385–429, http://dx.doi.org/10.1007/s10817-007-9078-x.

[35] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation, W3C, 2012, available at http://www.w3.org/TR/r2rml/.

[36] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, Semantic Web J. 8 (3) (2017) 471–487, http://dx.doi.org/10.3233/SW-160217.

[37] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, G. Carothers, RDF 1.1 Turtle – Terse RDF Triple Language, W3C Recommendation, W3C, 2014, available at http://www.w3.org/TR/turtle/.

[38] A. Artale, D. Calvanese, R. Kontchakov, M. Zakharyaschev, The DL-Lite family and relations, J. of Artificial Intelligence Research 36 (2009) 1–69, http://dx.doi.org/10.1613/jair.2820.

[39] D. Lanti, Benchmarking and Optimization of OBDA Systems (Ph.D. thesis), Free University of Bozen-Bolzano, 2018, URL http://hdl.handle.net/10863/6081.

[40] D. Calvanese, D. Lanti, T.M. de Farias, A. Mosca, G. Xiao, Accessing scientific data through knowledge graphs with Ontop, Patterns 2 (10) (2021) http://dx.doi.org/10.1016/j.patter.2021.100346.

[41] R. Hull, Relative information capacity of simple relational database schemas, SIAM J. Comput. 15 (3) (1986) 856–886.

[42] R.J. Miller, Y.E. Ioannidis, R. Ramakrishnan, Schema equivalence in heterogeneous systems: Bridging theory and practice, Inf. Syst. 19 (1) (1994) 3–31.

[43] P.P. Chen, The Entity-Relationship model: Toward a unified view of data, ACM Trans. on Database Systems 1 (1) (1976) 9–36.

[44] Unified Modeling Language (UML) specification Version 2.5.1, Object Management Group, 2017, Available at https://www.omg.org/spec/UML/.

[45] L.F. de Medeiros, F. Priyatna, O. Corcho, MIRROR: Automatic R2RML mapping generation from relational databases, in: Proc. of the 15th Int. Conf. on Web Engineering (ICWE), in: LNCS, vol. 9114, Springer, 2015, pp. 326–343.

[46] D. Calvanese, M. Lenzerini, D. Nardi, Unifying class-based representation formalisms, J. of Artificial Intelligence Research 11 (1999) 199–240, http://dx.doi.org/10.1613/jair.548.

[47] A. Gal, Uncertain schema matching, Synthesis Lectures in Data Management 3 (1) (2011) 1–97.

[48] A. Gal, H. Roitman, R. Shraga, Learning to rerank schema matches, IEEE Trans. on Knowledge and Data Engineering (2019).

[49] A. Gal, M. Katz, T. Sagi, M. Weidlich, K. Aberer, H.Q.V. Nguyen, Z. Miklós, E. Levy, V. Shafran, Completeness and ambiguity of schema cover, in: Proc. of Confederated Int. Conf. on the Move To Meaningful Internet Systems (OTM), in: LNCS, vol. 8185, Springer, 2013, pp. 241–258.
[50] É. Thiéblin, O. Haemmerlé, N. Hernandez, C. Trojahn, Survey on complex ontology matching, Semantic Web J. 11 (4) (2020) 689–727, http://dx.doi.org/10.3233/SW-190366.
[51] D. Calvanese, A. Gal, N. Haba, D. Lanti, M. Montali, A. Mosca, R. Shraga, [Adamap]: Automatic alignment of relational data sources using mapping patterns, in: Proc. of the 33rd Int. Conf. on Advanced Information Systems Engineering (CAiSE), in: LNCS, vol. 12751, Springer, 2021, pp. 193–209, http://dx.doi.org/10.1007/978-3-030-79382-1_12.
[52] C. Bizer, A. Schultz, The Berlin SPARQL benchmark, Int. J. Semantic Web Information Systems 5 (2) (2009) 1–24, http://dx.doi.org/10.4018/jswis.2009040101.
[53] D. Lanti, G. Xiao, D. Calvanese, VIG: Data scaling for OBDA benchmarks, Semantic Web J. 10 (2) (2019) 413–433, http://dx.doi.org/10.3233/SW-180336.
[54] D. Lanti, M. Rezk, G. Xiao, D. Calvanese, The NPD benchmark: Reality check for OBDA systems, in: Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT), 2015, pp. 617–628, http://dx.doi.org/10.5441/002/edbt.2015.62.
[55] Y. Zhou, B.C. Grau, I. Horrocks, Z. Wu, J. Banerjee, Making the most of your triple store: query answering in OWL 2 using an RL reasoner, in: Proc. of the 22nd Int. World Wide Web Conf. (WWW), 2013, pp. 1569–1580.
[56] E. Botoeva, D. Calvanese, V. Santarelli, D.F. Savo, A. Solimando, G. Xiao, Beyond OWL 2 QL in OBDA: Rewritings and approximations, in: Proc. of the 30th AAAI Conf. on Artificial Intelligence (AAAI), 2016, pp. 921–928, URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12238.
[57] C. Bizer, A. Seaborne, D2RQ – Treating non-RDF databases as virtual RDF graphs, in: Proc. of the ISWC 2004 Posters Track, 2004.
[58] J.F. Sequeda, D.P. Miranker, Ultrawrap Mapper: A semi-automatic relational database to RDF (RDB2RDF) mapping tool, in: Proc. of the ISWC 2015 Posters & Demonstrations Track, in: CEUR Workshop Proceedings, vol. 1486, CEUR-WS.org, 2015, URL http://ceur-ws.org/Vol-1486/paper_105.pdf.
[59] A. Solimando, E. Jiménez-Ruiz, G. Guerrini, Detecting and correcting conservativity principle violations in ontology-to-ontology mappings, in: Proc. of the 13th Int. Semantic Web Conf. (ISWC), in: LNCS, vol. 8797, Springer, 2014, pp. 1–16.
[60] E. Jiménez-Ruiz, B.C. Grau, Y. Zhou, I. Horrocks, Large-scale interactive ontology matching: Algorithms and implementation, in: Proc. of the 20th Eur. Conf. on Artificial Intelligence (ECAI), 2012, pp. 444–449.
[61] E. Jiménez-Ruiz, B.C. Grau, LogMap: Logic-based and scalable ontology matching, in: Proc. of the 10th Int. Semantic Web Conf. (ISWC), in: LNCS, vol. 7031, Springer, 2011, pp. 273–288.
[62] R.B. Thapa, M. Giese, A source-to-target constraint rewriting for direct mapping, in: Proc. of the 21st Int. Semantic Web Conf. (ISWC), ISWC, in: LNCS, vol. 12922, Springer, 2021, pp. 21–38, http://dx.doi.org/10.1007/978-3-030-88361-4_2.
[63] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), W3C Recommendation, W3C, 2017, available at https://www.w3.org/TR/shacl/.
[64] F. Cerbah, N. Lammari, Ontology learning from databases: Some efficient methods to discover semantic patterns in data, in: Perspectives on Ontology Learning, Vol. 18, IOS Press, 2014, pp. 207–222.
[65] F. Cerbah, Mining the content of relational databases to learn ontologies with deeper taxonomies, in: Proc. of the IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology, Vol. 1, IEEE, 2008, pp. 553–557.
[66] F. Cerbah, Learning highly structured semantic repositories from relational databases, in: Proc. of the 5th European Semantic Web Conf. (ESWC), in: LNCS, vol. 5021, Springer, 2008, pp. 777–781.
[67] D. Aumueller, H.-H. Do, S. Massmann, E. Rahm, Schema and ontology matching with COMA++, in: Proc. of the 2005 ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 906–908.
[68] A. Gal, G. Modica, H. Jamil, OntoBuilder: Fully automatic extraction and consolidation of ontologies from web sources, in: Proc. of the 20th IEEE Int. Conf. on Data Engineering (ICDE), 2004, p. 853.
[69] C. Pinkel, C. Binnig, E. Kharlamov, P. Haase, IncMap: pay as you go matching of relational schemata to OWL ontologies, in: Proc. of the 8th Int. Workshop on Ontology Matching, in: CEUR Workshop Proceedings, vol. 1111, CEUR-WS.org, 2013, pp. 37–48.
[70] S. Gupta, P. Szekely, C.A. Knoblock, A. Goel, M. Taheriyan, M. Muslea, Karma: A system for mapping structured sources into the Semantic Web, in: Proc. of the ESWC Satellite Events, in: LNCS, vol. 7540, Springer, 2012, pp. 430–434.
[71] J.D. Lafferty, A. McCallum, F.C.N. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data, in: Proc. of the 18th Int. Conf. on Machine Learning (ICML), Morgan Kaufmann, 2001, pp. 282–289.
[72] C.A. Knoblock, P. Szekely, J.L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, P. Mallick, Semi-automatically mapping structured sources into the semantic web, in: Proc. of the 9th Extended Semantic Web Conf. (ESWC), in: LNCS, vol. 7295, Springer, 2012, pp. 375–390.
[73] M. Taheriyan, C.A. Knoblock, P. Szekely, J.L. Ambite, Learning the semantics of structured data sources, J. of Web Semantics 37 (2016) 152–169.
[74] A. Doan, P.M. Domingos, A.Y. Levy, Learning source description for data integration, in: WebDB (Informal Proceedings), 2000, pp. 81–86.
[75] K. Lerman, A. Plangprasopchock, C.A. Knoblock, Semantic labeling of online information sources, Int. J. Semantic Web Information Systems (IJSWIS) 3 (3) (2007) 36–56.
[76] V.M. Pequeno, V.M. Vidal, M.A. Casanova, L.E.T. Neto, H. Galhardas, Specifying complex correspondences between relational schemas and rdf models for generating customized r2rml mappings, in: Proc. of the 18th Int. Database Engineering & Applications Symposium, 2014, pp. 96–104.
[77] J.F. Sequeda, S.H. Tirmizi, O. Corcho, D.P. Miranker, Survey of directly mapping SQL databases to the semantic web, Knowledge Engineering Review 26 (4) (2011) 445–486.
[78] K. Mogotlane, J.V. Fonou Dombeu, Automatic conversion of relational databases into ontologies : A comparative analysis of Protègè plug-ins performances, Int. J. Web Semantic Technology 7 (2016) 21–40, http://dx.doi.org/10.5121/ijwest.2016.7403.
[79] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, W. May, A. Nikolov, A. Sasa, M.G. Skjæveland, A. Solimando, M. Taheriyan, C. Heupel, I. Horrocks, Rodi: Benchmarking relational-to-ontology mapping generation quality, Semantic Web J. 9 (2016) 25–52.
[80] S.-C. Haw, J.W. May, S. Subramaniam, Mapping relational databases to ontology representation: A review, in: Proc. of the 1st Int. Conf. on Digital Technology in Education (ICDTE), ACM, 2017, pp. 54–58, http://dx.doi.org/10.1145/3134847.3134852.

Diego Calvanese is a Full Professor at the Research Centre for Knowledge and Data (KRDB) at the Faculty of Engineering of the Free University of Bozen-Bolzano (Italy), and Wallenberg Guest Professor in AI for Data Management at Umeå University (Sweden). His research interests include knowledge representation and reasoning, virtual knowledge graphs, ontology languages, description logics, conceptual data modeling and data integration. He is one of the editors of the Description Logic Handbook. He is a Fellow of EurAI and a Fellow of ACM. He is the originator and a co-founder of Ontopic, a startup whose mission is to bring the VKG technology to industry.

Avigdor Gal is the Benjamin and Florence Free Chaired Professor at the Technion - Israel Institute of Technology, were he established the Data Science & Engineering program. He specializes in various aspects of data management and mining with about 150 publications in leading journals, books, and conference proceedings. He served as a program co-chair and general co-chair of several conferences, including BPM and DEBS. In the past he gave keynotes and tutorials in leading conferences in the areas of data and process management. Avigdor Gal is a recipient of the prestigious Yannai award for excellence in academic education, multiple best paper and test-of-time awards.

Davide Lanti is an Assistant Professor at the Research Centre for Knowledge and Data (KRDB) at the Faculty of Engineering of the Free University of Bozen-Bolzano (Italy), where he carries out research on Virtual Knowledge Graphs, Semantic Web, Databases, and Description Logics. He received his MSc degree in Computational Logic jointly from the Technische Universität Dresden (Germany), and the Free University of Bozen-Bolzano (Italy). He received his PhD at the Faculty of Computer Science at the Free University of Bozen-Bolzano, Italy.

Marco Montali is Full Professor and Vice-Dean of Teaching at the Faculty of Engineering, Free University of Bozen-Bolzano, Italy, where he also coordinates the MSc Program in Computational Data Science. He investigates foundational and applied techniques grounded in artificial intelligence, formal methods, knowledge representation and reasoning, for the model- and data-driven analysis of business processes and multiagent systems. He has served as PC Chair of BPM 2018, RuleML+RR 2019, ICPM 2020, and CBI 2021, as General Chair of ICPM 2022 and EDOC 2022, and is steering committee member of the IEEE task force on process mining. He is co-author of more than 200 papers, many of which in top-tier conferences and journals, and recipient of 8 best paper awards. He received the "Marco Cadoli 2015" award, given by the Italian Association of Artificial Intelligence to the best under 35 Italian researcher who autonomously contributed to advance the state-of-the-art in Artificial Intelligence.

Alessandro Mosca is Assistant Professor at the Research Centre for Knowledge and Data (KRDB) at the Faculty of Engineering, and member of the Smart Data Factory, the technology and knowledge transfer lab for Computer Science, Free University of Bozen-Bolzano (Italy). His research interests include logic-based formalisms for knowledge representation and conceptual modeling for data management. He works on the theoretical and methodological aspects behind the creation of ontology-based data management solutions which, in particular, subsume the design and development of formal ontologies, multi-format data integration and access services, efficient ontology-based query answering.

Roee Shraga is a Postdoctoral fellow at the Khoury College of Computer Science at Northeastern University in Boston. He received his PhD degree from the Technion – Israel Institute of Technology in 2020. Roee has published more than a dozen papers in leading journals and conferences on the topics of data integration, human-in-the-loop, machine learning, process mining, and information retrieval. He is a recipient of several PhD fellowships including the Leonard and Diane Sherman Interdisciplinary Fellowship (2017) and the Miriam and Aaron Gutwirth Memorial Fellowship (2020).