

Conceptually-grounded Mapping Patterns for Virtual Knowledge Graphs

Diego Calvanese^{1,2}, Avigdor Gal³, Davide Lanti¹, Marco Montali¹, Alessandro Mosca¹ and Roei Shraga⁴

¹Free-University of Bozen-Bolzano, Bolzano, Italy

²Umeå University, Umeå, Sweden

³Technion – Israel Institute of Technology, Haifa, Israel

⁴Khoury College of Computer Science, Northeastern University, Boston, Massachusetts

Abstract

Knowledge Graphs (KGs) have been gaining momentum recently in both academia and industry, due to the flexibility of their data model, allowing one to access and integrate collections of data of different forms. Virtual Knowledge Graphs (VKGs), a variant of KGs originating from the field of Ontology-based Data Access (OBDA), are a promising paradigm for integrating and accessing legacy data sources. The main idea of VKGs is that the KG remains virtual: the end-user interacts with a KG, but queries are reformulated on-the-fly as queries over the data source(s). To enable the paradigm, one needs to define declarative mappings specifying the link between the data sources and the elements in the VKG. In this work, we try to investigate common patterns that arise when specifying such mappings, building on well-established methodologies from the area of conceptual modeling and database design.

Keywords

Virtual Knowledge Graphs, Ontology-based Data Access, Mapping patterns, Data Integration

1. Introduction

Data integration and access to legacy data sources are key challenges for contemporary organizations. In the whole spectrum of data integration and access solutions, the approach based on *Virtual Knowledge Graphs (VKGs)* is gaining momentum, especially when the underlying data sources to be integrated come in the form of relational databases (DBs) [1]. VKGs replace the rigid structure of tables with the flexibility of a graph that incorporates domain knowledge and is kept virtual, eliminating redundancies. 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; and (iii) a set of *mappings*, linking the data sources to the ontology. A critical bottleneck in this setting lies in the definition and management of mappings. In this work, we focus on this issue by proposing a comprehensive catalog of *mapping*

SEBD 2022: The 30th Italian Symposium on Advanced Database Systems, June 19-22, 2022, Tirrenia (PI), Italy

✉ calvanese@inf.unibz.it (D. Calvanese); avigal@technion.ac.il (A. Gal); lanti@unibz.it (D. Lanti);


montali@unibz.it (M. Montali); mosca@unibz.it (A. Mosca); r.shraga@northeastern.edu (R. Shraga)

🆔 0000-0001-5174-9693 (D. Calvanese); 0000-0002-7028-661X (A. Gal); 0000-0003-1097-2965 (D. Lanti);

0000-0002-8021-3430 (M. Montali); 0000-0003-2323-3344 (A. Mosca); 0000-0001-8803-8481 (R. Shraga)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

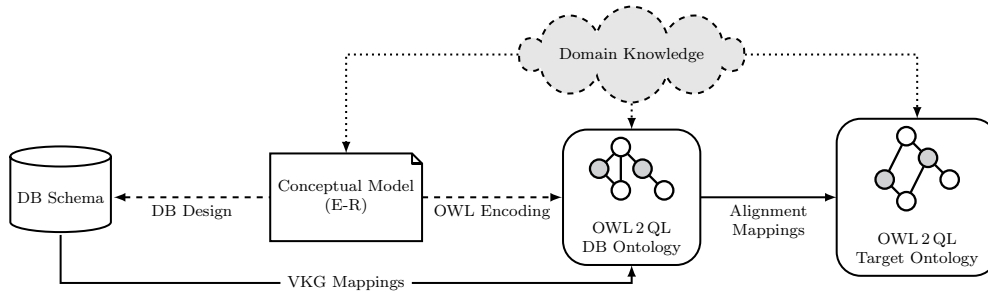


Figure 1: The database and the ontology both stem from common domain knowledge.

patterns that emerge when linking data to ontologies. Our catalog is based on the (somehow reasonable) assumption that *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 may lead to an explicit representation in the form of a structural conceptual model, which can be represented using well-established notations such as UML, ORM, or E-R. On the one hand, this conceptual model provides the basis for creating a corresponding domain ontology through a series of semantic-preserving transformation steps. On the other hand, it can trigger the design process that finally leads to the deployment of an actual DB. The whole view is depicted in Figure 1.

Our catalog is built on well-established methodologies and patterns studied in data management (e.g., W3C direct mappings (W3C-DM)¹ and extensions), data analysis (e.g., algorithms for discovering dependencies), and conceptual modeling (e.g., relational mapping techniques).

The idea of mapping patterns is not new. For instance, work in [2] is closely related to ours, as it also introduces a catalog of mapping patterns. However, there are some key differences with our approach. One difference is that we consider KGs (with ontologies), whereas that work focuses on *property graphs* without an ontology. More importantly, in [2] and in the related literature, patterns are not formalized or grounded to a specific conceptual representation, but are rather informally specified and discussed in a “by-example” fashion. 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 *semantics-preserving* transformations studied in the area of DB design.

We argue that this foundational grounding paves the way for 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, or to automatically generate (i.e., bootstrap) ontology and mappings when only the DB is available. In fact, specific patterns have been proposed also in relation to ontology and mapping *bootstrapping*, for which a variety of tools and approaches have been developed in the last two decades [3, 4, 5, 6, 7]. The approaches in the literature differ in terms of the overall *purposes* of bootstrapping (e.g., OBDA, data integration, ontology learning, checking of DB schema constraints using ontology reasoning), the adopted *ontology and mapping languages* (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

¹<http://www.w3.org/TR/rdb-direct-mapping/>

Table 1
Semantics of the $DL-Lite_{\mathcal{R}}$ constructs that involve datatypes.

Construct	Syntax Element	Example	Semantics
Top domain	\top_V		$\Delta_V^{\mathcal{I}}$
Literal	$\ell \in \mathbf{NL}$	“george”	$\ell^{\mathcal{I}} \in \Delta_V^{\mathcal{I}}$
Datatype	T_i	xsd:int	$T_i^{\mathcal{I}} \subseteq \Delta_V^{\mathcal{I}}$
Data property name	$d \in \mathbf{ND}$	hasName	$d^{\mathcal{I}} \subseteq \Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}}$
Data property domain	$\delta(d)$	$\delta(\text{hasName})$	$\{x \in \Delta_V^{\mathcal{I}} \mid \exists v \in \Delta_V^{\mathcal{I}} : (x, v) \in d^{\mathcal{I}}\}$
Data property range	$\rho(d)$	$\rho(\text{hasName})$	$\{v \in \Delta_V^{\mathcal{I}} \mid \exists o \in \Delta_O^{\mathcal{I}} : (o, v) \in d^{\mathcal{I}}\}$
Data property negation	$\neg d$	$\neg \text{hasName}$	$\Delta_O^{\mathcal{I}} \times \Delta_V^{\mathcal{I}} \setminus d^{\mathcal{I}}$

closely follow W3C-DM, deriving ontologies that mirror the structure of the input DB.

The remainder of the paper is structured as follows: Section 2 introduces the notation and basic notions on VKGs, Section 3 presents (an extract of) our catalog of mapping patterns, and Section 4 concludes the paper.

2. Preliminaries

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 familiarity with standard notions and languages from DBs [8], such as SQL or E-R diagrams.

A VKG specification is a triple $\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ where \mathcal{T} is an *ontology* (or *TBox*), \mathcal{M} a set of *mappings*, and \mathcal{S} the schema of a DB (with constraints, e.g., primary and foreign keys). In VKGs, the ontology is formulated in OWL 2 QL², but for conciseness we use its Description Logic (DL) counterpart, $DL-Lite_{\mathcal{R}}$ [9], here slightly enriched to handle datatypes.

We fix the following enumerable, pairwise-disjoint sets: \mathbf{NI} of *individuals*, \mathbf{NL} of *literal values*, \mathbf{NC} of *class names*, \mathbf{NP} of *object property names*, and \mathbf{ND} of *data property names*.

An OWL 2 QL *TBox* \mathcal{T} is a finite set of *inclusion axioms* of the form $B \sqsubseteq C$, $q \sqsubseteq r$, $\rho(d) \sqsubseteq f$, or $d \sqsubseteq v$, 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*. These 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, \dots, T_n are the *RDF datatypes*:

$B \rightarrow A \mid \exists r \mid \delta(d)$	$q \rightarrow p \mid p^-$	$f \rightarrow \top_D \mid T_1 \mid \dots \mid T_n$
$C \rightarrow \top_C \mid B \mid \neg B$	$r \rightarrow q \mid \neg q$	$v \rightarrow d \mid \neg d$

In the rules above, \top_C denotes the “top” element for concepts and \top_D the one for data values (called *literals* in the RDF terminology). 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 $\ell \in \mathbf{NL}$. We call the pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ an OWL 2 QL *Knowledge Graph* (KG).

²<http://www.w3.org/TR/owl2-overview/>

Similarly to first-order logic, the semantics of *DL-Lite_R* KGs is given through Tarski-style *interpretations* $\mathcal{I} = \langle \Delta_O^{\mathcal{I}}, \Delta_V^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta_O^{\mathcal{I}}$ is a non-empty *domain of objects*, $\Delta_V^{\mathcal{I}}$ is a non-empty *domain of values*, and $\cdot^{\mathcal{I}}$ is an interpretation function. Table 1 reports the semantics for the constructs involving datatypes. The other constructs are defined as in standard *DL-Lite_R* [9]. As usual [10], we say that an interpretation \mathcal{I} *satisfies a KG* \mathcal{K} , denoted by $\mathcal{I} \models \mathcal{K}$, if \mathcal{I} satisfies the ABox assertions and the inclusion axioms in \mathcal{K} .

Mappings. 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 a KG. In VKGs, the adopted language for mappings in real-world systems is R2RML³, but for conciseness we use here a more convenient abstract notation inspired by the literature [11]: a *mapping* m is a pair of the form $\langle s: Q(\mathbf{x}), t: \mathbf{L}(t(\mathbf{x})) \rangle$, where $Q(\mathbf{x})$ is a SQL query with answer variables \mathbf{x} over the DB schema \mathcal{S} , called *source query*, and $\mathbf{L}(t(\mathbf{x}))$ is a list of *target atoms* of the form $C(t_1(\mathbf{x}_1)), p(t_1(\mathbf{x}_1), t_2(\mathbf{x}_2))$, or $d(t_1(\mathbf{x}_1), t_2(\mathbf{x}_2))$, where $C \in \mathbf{NC}$, $p \in \mathbf{NP}$, $d \in \mathbf{ND}$, and $t_1(\mathbf{x}_1)$ and $t_2(\mathbf{x}_2)$ are terms that we call *templates*. We express source queries in *relational algebra*, omitting answer variables under the assumption that they coincide with the variables used in the target atoms.

Intuitively, a template $t(\mathbf{x})$ in the target atom of a mapping corresponds to an R2RML *string template*⁴, and is used to generate an *IRI* (hence, an object identifier) or an RDF *literal*, starting from DB values retrieved by the source query in that mapping. For the examples, we use the concrete syntax from the Ontop VKG system [6], 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 $\{ \dots \}$. The following is an example mapping expressed in such syntax:

source	SELECT ssn FROM person
target	ex:pers/{ssn} a ex:Person .

In the mapping above, the string `ex:` denotes a URI prefix, e.g., `ex:Person` is an abbreviation for the URI `http://www.example.com/Person`. Such mapping, when applied to a DB instance \mathcal{D} of \mathcal{S} , populates the class `ex:Person` with IRIs constructed by replacing the answer variable `ssn` occurring in the target atom with the corresponding values assigned to that variable by the answers to the SQL source query evaluated over \mathcal{D} . For instance, if the source query returns two answers that assign to the answer variable `ssn` respectively the values 1234 and 5678, then the mapping above produces the following RDF graph (expressed in the Turtle syntax⁵), stating that individuals `ex:pers/1234` and `ex:pers/5678` are both instances of class `ex:Person`:

<code>ex:pers/1234 a ex:Person .</code>	<code>ex:pers/5678 a ex:Person .</code>
---	---

We denote by $\mathcal{A}_{\mathcal{M}(\mathcal{D})}$ the virtual ABox constructed through mappings \mathcal{M} from a DB \mathcal{D} . Given a VKG specification $\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ and a database instance \mathcal{D} of \mathcal{S} , the KG $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}_{\mathcal{M}(\mathcal{D})} \rangle$ is called the *Virtual Knowledge Graph of $\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ through \mathcal{D}* . The qualifier “virtual” in the

³<http://www.w3.org/TR/r2rml/>

⁴<https://www.w3.org/TR/r2rml/#dfn-string-template>

⁵<http://www.w3.org/TR/turtle/>

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* [11, 6]: user queries, expressed in SPARQL⁶, get translated on-the-fly into equivalent SQL queries, which then are directly evaluated against the DB.

3. Mapping Patterns

In its basic form, a mapping pattern is a quadruple $\langle \mathcal{C}, \mathcal{S}, \mathcal{M}, \mathcal{T} \rangle$, where \mathcal{C} is a conceptual model, \mathcal{S} a database schema, \mathcal{M} a set of mappings, and \mathcal{T} an (OWL 2 QL) ontology. In such pattern, the pair $\langle \mathcal{C}, \mathcal{S} \rangle$ puts into correspondence a conceptual representation with one of its (many) admissible (i.e., formally sound [12, 13]) database schemata, like those prescribed by *well-established database modeling methodologies*. The pair $\langle \mathcal{M}, \mathcal{T} \rangle$, instead, is formed by the *DB ontology* \mathcal{T} , which is the OWL 2 QL encoding⁷ of the conceptual model \mathcal{C} , and the set \mathcal{M} of mappings, providing the link between \mathcal{S} and \mathcal{T} . 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 Figure 1.

Some of the more advanced patterns have a more complex structure, where pairs of conceptual models and/or pairs of database schemata are used in place of \mathcal{C} and \mathcal{S} , respectively (e.g., the pattern “**SHa**” falls in this category). These patterns prescribe specific *transformations* to be applied to 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. These structures can in turn enable further applications of patterns.

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 [14]). To compactly represent sets of attributes, we use a small diamond in place of the small circle used for single attributes in Chen’s 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 $\text{key}_T(\mathbf{U})$ the fact that \mathbf{U} form a *key* for T . Referential integrity constraints (like, e.g., foreign keys) are denoted with arcs, pointing from the referencing attribute(s) to the referenced one(s). For conciseness, we denote sets of the form $\{o \mid \text{condition}\}$ as $\{o\}_{\text{condition}}$. 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*⁸ correspondence provided by the R2RML recommendation. 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,


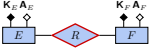
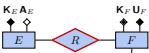
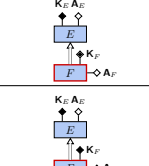
⁶<http://www.w3.org/TR/sparql11-query>

⁷Modulo the expressivity of the OWL 2 QL language.

⁸<https://www.w3.org/TR/r2rml/#natural-mapping>

Table 2

An extract of our catalog of mapping patterns.

CONCEPTUAL MODEL	DB SCHEMA	MAPPINGS	ONTOLOGY
Schema Entity (SE)			
	$T_E(\underline{\mathbf{K}}, \mathbf{A})$	$s: T_E$ $t: C_E(\mathfrak{t}_E(\mathbf{K})),$ $\{d_A(\mathfrak{t}_E(\mathbf{K}), A)\}_{A \in \mathbf{K} \cup \mathbf{A}}$	$\left\{ \begin{array}{l} \delta(d_A) \sqsubseteq C_E, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_E \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathbf{K} \cup \mathbf{A}}$
In case of optional attributes, for each optional attribute A' of E , add an $opt(A')$ constraint to the DB schema and drop the corresponding inclusion axiom $C_E \sqsubseteq \delta(d_{A'})$ from the ontology.			
Schema Relationship (SR)			
	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E) \quad T_F(\underline{\mathbf{K}_F}, \mathbf{A}_F)$ $T_R(\overleftarrow{\mathbf{K}_{RE}}, \overrightarrow{\mathbf{K}_{RF}})$	$s: T_R$ $t: p_R(\mathfrak{t}_{C_E}(\mathbf{K}_{RE}), \mathfrak{t}_{C_F}(\mathbf{K}_{RF}))$	$\exists p_R \sqsubseteq C_E$ $\exists \overline{p_R} \sqsubseteq C_F$
<ul style="list-style-type: none"> In case of cardinality $(_, 1)$ on role R_E (resp., R_F), the primary key of T_R is restricted to the attributes \mathbf{K}_{RE} (resp., \mathbf{K}_{RF}). In case both roles have cardinality $(_, 1)$, either choice for the primary key is made, and the remaining attributes form a non-primary key in the logical schema. In case of cardinality $(1, _)$ 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. Note that when the maximum cardinality on role R_E (resp., R_F) is 1, the corresponding inclusion dependency is actually a foreign key. 			
Schema Relationship with Identifier Alignment (SRa)			
	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E) \quad T_F(\underline{\mathbf{K}_F}, \mathbf{U}_F, \mathbf{A}_F)$ $T_R(\overleftarrow{\mathbf{K}_{RE}}, \overrightarrow{\mathbf{U}_{RF}}) \quad \text{key}_{T_F}(\mathbf{U}_F)$	$s: T_R \bowtie_{\mathbf{U}_{RF}=\mathbf{U}_F} T_F$ $t: p_R(\mathfrak{t}_{C_E}(\mathbf{K}_{RE}), \mathfrak{t}_{C_F}(\mathbf{K}_F))$	$\exists p_R \sqsubseteq C_E$ $\exists \overline{p_R} \sqsubseteq C_F$
Schema Hierarchy with Identifier Alignment (SHa)			
	$T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E) \quad \text{key}_{T_F}(\mathbf{K}_{FE})$ $T_F(\underline{\mathbf{K}_F}, \overleftarrow{\mathbf{K}_{FE}}, \mathbf{A}_F)$ <hr/> $T_E(\underline{\mathbf{K}_E}, \mathbf{A}_E) \quad \text{key}_{V_F}(\mathbf{K}_F)$ $V_F(\mathbf{K}_F, \overleftarrow{\mathbf{K}_{FE}}, \mathbf{A}_F) = T_F$	$s: V_F$ $t: C_F(\mathfrak{t}_{C_E}(\mathbf{K}_{FE})),$ $\{d_A(\mathfrak{t}_{C_E}(\mathbf{K}_{FE}), A)\}_{A \in \mathbf{K}_F \cup \mathbf{A}_F}$	$\left\{ \begin{array}{l} C_F \sqsubseteq C_E \\ \delta(d_A) \sqsubseteq C_F, \\ \rho(d_A) \sqsubseteq \mu(\tau(A)), \\ C_F \sqsubseteq \delta(d_A) \end{array} \right\}_{A \in \mathbf{K}_F \cup \mathbf{A}_F}$
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 applying the pattern are:			
<ul style="list-style-type: none"> 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 depict here 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 of the attributes used for identifying the objects of the subclass).			

declared as “NOT NULL”). An optional attribute A is instead denoted by adding $opt(A)$ to the DB schema. Such notation extends in the natural way to a set \mathbf{A} of attributes.

Pattern Catalog. Table 2 shows an excerpt of our patterns, which we discuss in detail here.

Schema Entity (SE). This fundamental pattern describes the correspondence between an entity with a primary identifier and attributes in the DB schema, and 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 IRIs of the objects that are instances of C_E , using a template \mathfrak{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 each data property corresponding to a mandatory attribute.

Example: A client registry table containing 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.

Schema Relationship (SR). This pattern describes the correspondence between a binary relationship without attributes and 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 , for which the set of columns in T_R is partitioned into two parts \mathbf{K}_{RE} and \mathbf{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 parts \mathbf{K}_{RE} and \mathbf{K}_{RF} of the partition 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 respectively used for building instances of classes C_E corresponding to T_E and C_F corresponding to T_F .

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.

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 key \mathbf{U}_F .

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.

Schema Hierarchy with Identifier Alignment (SHa). This patterns handles the case where a hierarchy is specified and the child entity uses a primary identifier different from the one in the parent entity. In this situation, the foreign-key constraint 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 the primary key of T_F . For this purpose, the pattern creates a view V_F identical to T_F , except that \mathbf{K}_{FE} is the primary key. Also the foreign key relations are preserved. Such view might enable further applications of patterns.

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

4. Conclusions and Future Work

In this work, we have identified and formally specified a number of mapping patterns emerging when linking DBs to ontologies in a typical VKG setting. Our patterns are grounded in well-established practices of DB design, and render explicit the connection between the conceptual model, the DB schema, and the ontology. We envision that the organization in patterns can enable a number of relevant tasks, notably mapping bootstrapping for incomplete VKGs.

This work is only a first step, with respect to both categorization of patterns, and their actual use. Regarding the former, we are currently extending this initial catalog with more advanced “data-driven” patterns, which are patterns where the data component needs to be taken into account. Regarding the latter, we are investigating solutions to specific problems that need to be addressed when setting-up a VKG scenario, like the problem of mapping bootstrapping.

Acknowledgments

This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, by the Italian Basic Research (PRIN) project HOPE, by the EU H2020 project INODE (grant agreement 863410), and by the project MENS, funded through the 4th Call for Research of the Autonomous Province of Bolzano (IN2219).

References

- [1] G. Xiao, L. Ding, B. Cogrel, D. Calvanese, Virtual Knowledge Graphs: An overview of systems and use cases, *Data Intelligence* 1 (2019) 201–223.
- [2] J. Sequeda, O. Lassila, *Designing and Building Enterprise Knowledge Graphs*, Morgan & Claypool Publishers, 2021.
- [3] L. F. de Medeiros, F. Priyatna, Ó. Corcho, MIRROR: Automatic R2RML mapping generation from relational databases, in: *Proc. ICWE*, volume 9114 of *LNCS*, Springer, 2015, pp. 326–343.
- [4] 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. ISWC*, volume 9367 of *LNCS*, Springer, 2015, pp. 113–132.
- [5] C. Pinkel, C. Binnig, E. Kharlamov, P. Haase, IncMap: Pay as you go matching of relational schemata to OWL ontologies., in: *Proc. 8th Int. Workshop on Ontology Matching (OM)*, volume 1111 of *CEUR*, CEUR-WS.org, 2013, pp. 37–48.
- [6] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodríguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web J.* 8 (2017) 471–487.
- [7] J. F. Sequeda, D. P. Miranker, Ultrawrap Mapper: A semi-automatic relational database to RDF (RDB2RDF) mapping tool, in: *Proc. ISWC Posters & Demonstrations Track*, volume 1486 of *CEUR*, CEUR-WS.org, 2015.
- [8] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison Wesley, 1995.
- [9] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family, *JAR* 39 (2007) 385–429.
- [10] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation and Applications*, 2nd ed., Cambridge University Press, 2007.
- [11] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, *J. on Data Semantics* 10 (2008) 133–173.

- [12] R. Hull, Relative information capacity of simple relational database schemas, *SIAM J. on Computing* 15 (1986) 856–886.
- [13] R. J. Miller, Y. E. Ioannidis, R. Ramakrishnan, Schema equivalence in heterogeneous systems: Bridging theory and practice, *Information Systems* 19 (1994) 3–31.
- [14] P. P. Chen, The Entity-Relationship model: Toward a unified view of data, *ACM TODS* 1 (1976) 9–36.