Systematical Representation of RDF-to-Relational Mappings for Ontology-Based Data Access

Lars Runge, Sebastian Schrage and Wolfgang May

Georg-August University of Göttingen, Institute of Computer Science

Abstract. Ontology-Based Data Access (OBDA) systems establish a connection between RDF/OWL application ontologies and relational databases. For this, they use mappings between the two data models. Several systems that aim to generate such mappings automatically have been evaluated in a recent benchmark, that showed that the coverage of the generated mappings is still rather low, and that there is much potential left. The mappings are mostly stored internally in the systems and are not accessible to the user.
As a step towards better handling, this paper presents a framework that stores the OBDA mapping information in an easily understandable format in dedicated metadata tables in the relational database. This makes the metadata accessible for the user, and it can also be used for other programs.

1 Introduction

Ontology-based Data Access (OBDA) systems aim to allow users to state queries based on the vocabulary of an ontology against data stored in a relational database, which combines the advantages of both approaches: ontologies are based on the intuitive, user-friendly conceptual model of classes with a class hierarchy, and properties, which include literal-valued attributes and relationships (the technical formulation of this in OWL is less user-friendly, but has only to be done once by the ontology engineer). On the other hand, relational databases allow for efficient storage and query evaluation, and they are still the most common type of databases.

There are three main settings for OBDA: (1) the "traditional" setting for OBDA is to have an already running application with an existing database, and to design an ontology for it and to connect them. (2) the other direction is to start with an ontology and to design a relational schema for it. (3) in the third setting, both an ontology and an independent, thematically related existing relational database are given, and both must be connected. In either case, the core of the OBDA tool is a mapping between both formats. These mappings are used to translate queries from one data model to the other, and also e.g. to insert or update data.

While (1) can usually be done rather straightforwardly, the outcome is not yet a "good" ontology, but rather a re-engineering-style ontologization of several design decisions (and necessities, like reification) when designing a relational

schema. For a good, user-friendly ontology, further changes (or systematically spoken, transformations) must be appplied. For (2) the same applies in the opposite direction: the straighforward database design might be revised – again this means to apply transformations to the mappings. For (3), the appropriate mappings must be created. This can be done manually, or by (semi)automatic alignment tools. In all cases, the way how to express, store, and transform the mappings is crucial.

When evaluating such (semi)automatic database alignment approaches for (3) through the evaluation of query translation, it was found that the generated alignments are often incomplete or partially incorrect for real-world application complexity [PBJR+15,PBJR+17] (see below for more details). Furthermore, most systems do not allow the user to check the mapping and to adjust it accordingly, because it is only stored internally in the system. Making the mappings available to the user by storing them in a well-defined and accessible format in metadata tables inside the relational database can help to overcome this limitation. Such metadata tables can also be used as a starting point for other programs providing different functionalities that are based on those mappings.

This paper introduces the "RDF2SQL" approach, which defines a framework for storing RDF-to-relational mappings in a *Semantical Data Dictionary* (SDD) consisting of generic metadata tables.

Related Work and Background. The automatical generation of mappings between ontologies and relational databases to grant OBDA to the stored data has been investigated and implemented in several approaches. Some prominent tools that create such mappings include BootOX [JRKZ+15], MIRROR [dMPC15], ontop [CCK+15] (fully automatic) and IncMap [PBKH13], COMA++ [ADMR05], Karma [KSA+12] (semi-automatic). These six tools were recently evaluated by the RODI benchmark [PBJR+15,PBJR+17]. The tools were tasked to generate mappings for provided relational databases and target ontologies. For evaluation, for each such scenario a series of SPARQL queries was given. The tools had to translate the queries into SQL based on the generated mappings, and the results were compared to the provided reference solutions. The tests were conducted on data sets from three different application domains such as conferences (taken from the OAI case studies [EMS+11]), geodata (mondial [Mondial]) and oil & gas exploration. The difficulty of the tests increased from the easier queries of the conference data sets asking only for atomic properties to the more difficult queries of the geodata and oil & gas domains that also select combinations of properties and additional constraints mimicking more realistic queries. Regardless of the approach of the tools, the general results of the benchmark were quite disappointing. While the tests for the atomic queries resulted in nearly acceptable outcomes, the authors concluded however that "all tested tools perform poorly on most of the more advanced challenges that come close to actual real-world problems" [PBJR+15]. Especially for the Mondial and oil & gas ontologies this is due to more advanced mapping challenges, for instance the introduction of a class hierarchy.

In the following, we use the MONDIAL [Mondial] database which contains data about various geographical objects like countries, cities, international organizations, and several geographical features. MONDIAL contains examples for many modeling and design issues as realistic test cases for the mappings.

Structure of the paper. Section 2 explains, what metadata information about an OBDA mapping to a relational database is needed. The section is illustrated by a "well-designed" relational schema obtained from a canonical mapping of an OWL ontology. Next, the intermediate Section 3 illustrates how the mapping metadata is used to translate SPARQL queries into SQL queries. Section 4 illustrates how the SDD structure seamlessly also covers mappings to other, non-canonical schemata. Section 5 describes the current prototype and gives an outlook to future perspectives. Section 6 concludes the paper.

2 Generic Metadata Information

This section introduces the structure and design of the above-mentioned *Semantical Data Dictionary* (SDD) metadata tables. Roughly, these tables store the same information about classes and relationships in an application domain, as it is used when deriving a relational schema from an ER model. In the OBDA case, this information is extracted from the OWL ontology. The analysis of the ontology is done by a series of SPARQL queries and definition of auxiliary class definitions against the OWL-DL ontology as described in [HM13].

2.1 Basic Ontology to Relational Model Mapping

For mapping the classes and properties of the OWL ontology to a relational model, the standard ER-to-Relational transformation [EN04, Ch. 7] is applied:

For each class, a *ClassTable* is created, where all functional properties are contained as columns. The identifier of an individual is generally its URI, thus each ClassTable has a column uri acting as the primary key. Relationships that are not functional in any direction, called n:m-properties, are basically mapped to binary n:m tables that refer to ClassTables via two foreign keys (here, the URIs). Analogously, literal-valued multivalued attributes are also mapped to binary tables with a single foreign key. Further aspects of the ER model include n-ary ($n > 2$) relationships and relationships with attributes; both are commonly dealt with by reification. Note that in an RDF setting, the RDF modeling and the corresponding OWL ontology must already represent the modeling *after* reification. Additionally, ontologies often use class hierarchies, which are possible, but rarely used in the Extended ER model.

In the following, the mapping and the SDD contents are illustrated by the fragment of the MONDIAL database shown in Figure 1. The fragment has been chosen to contain examples for typical patterns in conceptual modeling:

Countries have functional literal-valued properties name, code, population, and area, and an 1:n relationship hasCity to one or more cities, and a 1:1-reference

relationship capital to a city. Cities also have a name and a population. Organizations have a name, an abbreviation, and an optional n:1 relationship hasHq to the city where the organization's headquarter is located.

Some classes represent geographical features: rivers, lakes and seas, which are different kinds (subclasses) of waters; mountains, and islands.

There is a simple n:m relationship locatedAt between cities and waters. The locatedOn relationship is polymorphic: between cities and islands it is also n:m, while between mountains and islands, it is n:1. Note that this polymorphism does not effect the ER model (where it just occurs as two relationship types of the same name) and the canonical relational model, but leads to a complex description in the OWL ontology.

Another specialty is the attributed isMember relationship between countries and organizations that includes a type attribute. This leads to reification, which in the relational model means an n:m table with an aditional type column. In RDF (or UML) modeling, there is an artificial reified Membership class with reference properties ofCountry and inOrganization as shown in Figure 2 as UML diagram (indicating also the names of the inverse relationships).

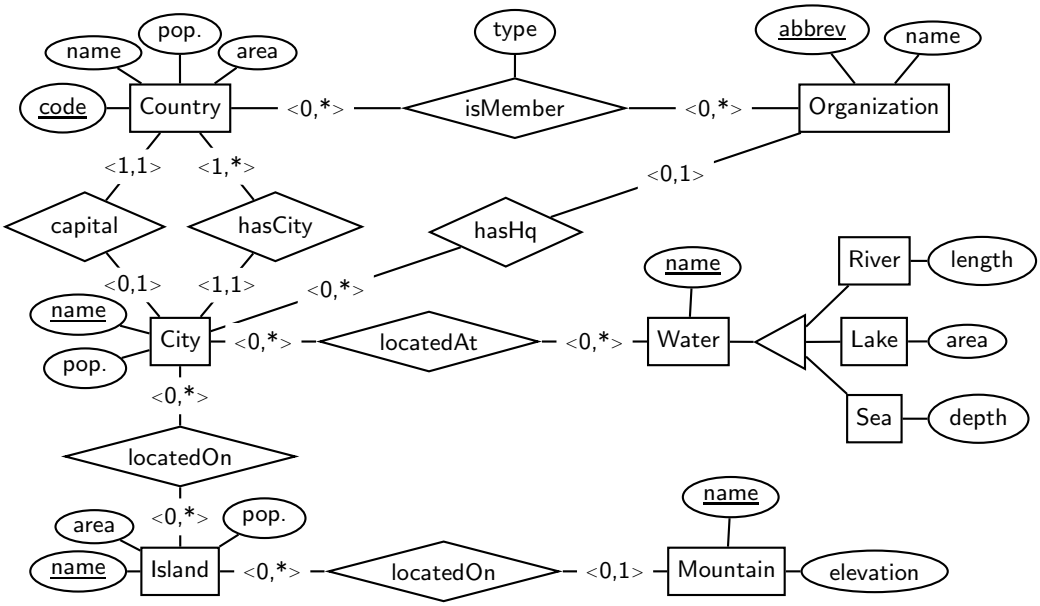

Fig. 1. Excerpt from the Mondial ER model

In an OBDA setting, the OWL ontology of the application contains the same information; the (full) MONDIAL ontology can be found at [Mondial]. The polymorphic locatedOn property and the attributed isMember relationship (with reification, according to Fig. 2) are modeled as follows:

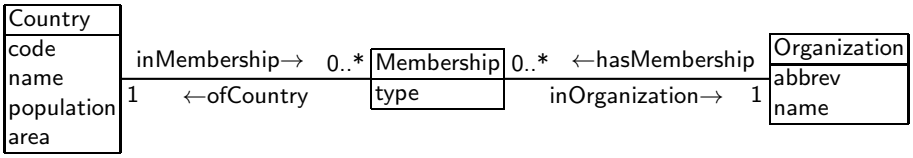

Fig. 2. Reified Modeling of attributed "isMember" relationship as "Memberships" class (in UML)

```
:locatedOn a owl:ObjectProperty;
  rdfs:domain [ owl:unionOf ( :City :Mountain ) ]; rdfs:range :Island.
:Mountain rdfs:subClassOf [a owl:Restriction;
  owl:onProperty :locatedOn; owl:maxCardinality 1].

:Membership a owl:Class.
:ofMember a owl:ObjectProperty; a owl:FunctionalProperty;
  rdfs:domain :Membership; owl:inverseOf :inMembership; rdfs:range :Country.
:inOrganization a owl:ObjectProperty; a owl:FunctionalProperty;
  rdfs:domain :Membership; owl:inverseOf :hasMembership; rdfs:range :Organization.
```

The corresponding relational schema looks as follows:

```
Country(uri, name, code, population, area, capital).
Organization(uri, abbrev, name, hasHq).
Membership(uri, ofCountry, inOrganization, type).
City(uri, name, population, hasCity_inv).
River(uri, name, length).
Lake(uri, name, area).
Sea(uri, name, depth).
locatedAt(city, water).
Mountain(uri, name, locatedOn).
Island(uri, name, area).
locatedOn(city, island).
```

The mapping of most of the functional properties is obvious. The 1:n-property hasCity is stored in its functional direction from City to Country, here called hasCity_inv (note that the corresponding OWL ontology could assign a name via :inCountry owl:inverseOf :hasCity). The mapping of the reified Membership type is canonical since ofCountry and inOrganization are functional. Note that in our setting, there is no Water table (i.e., the Water class is abstract; other modeling alternatives can be configured for RDF2SQL), but the waters are stored in the River, Lake, and Sea tables. LocatedAt is a plain n:m table whose columns are usually named after the ranges of the references keys; since Water is abstract, the locatedAt.water column is not a true foreign key, but references to River, Lake, and Sea. Note that for the locatedOn property, it is stored for mountains in the Mountain table (on this part of its domain it is functional), and for cities it is stored in the n:m locatedOn table.

2.2 The Semantical Data Dictionary

In the following, the metadata tables of the SDD are described. Their use will immediately be illustrated by showing how SPARQL queries are mapped to the

relational schema. For the SPARQL queries, only the WHERE pattern is shown; the projection (usually to everything except the URIs) is implicit. The renaming of the SQL output to the variable names is also omitted. When translating a query, in a first step, the set of potential classes of its variables is determined using domain and range information.

Inverses. The first, rather minor metadata table is the one for storing the inverses. This is important insofar as ontologies can explicitly name both directions, and 1:n properties are stored in the functional direction, like the hasCity relationship between countries and cities, which is stored as City.hasCity_inv. Table 1 shows the *Inverses (INV)* table for the above example.

INV	
Property	**Inverse**
hasCity	hasCity_inv
capital	capital_inv
locatedAt	locatedAt_inv
locatedOn	locatedOn_inv

$\cup$

INV	
Property	**Inverse**
hasHq	hasHq_inv
ofCountry	inMembership
inOrganization	hasMembership

Table 1. Sample Inverses Table contents

The central table: The Mapping Dictionary. The *Mapping Dictionary (MD)* represents where a property p of a class c is stored, i.e., in which table, and in which column of it. To be able to connect queries along references, also the range *class* of every property is stored (which looks redundant in the simple cases, but will be motivated in Examples 2 and 5). The MD is thus a mapping of the form MD : (Class $\times$ Property) $\rightarrow$ (Class $\times$ Table $\times$ Column).

For functional properties, the entries refer to the ClassTables where a property is stored; obviously, for functional properties of non-abstract classes, the tablename equals the classname. For n:m properties, the entries refer to the n:m tables, and there to the column that points to the target object.

Example 1. Consider the following query that uses the abstract class Water:

```
{ ?W a :Water; :name ?N }
```

Water is also mentioned in the MD, which tells that the names of waters can be found in the River, Lake, and Sea tables. Multiple entries are translated into a union:

```
(SELECT uri, name FROM River) union (SELECT uri, name FROM Lake)
  union (SELECT uri, name FROM Sea)
```

The MD contains also the inverses, e.g., hasCity_inv of City. On the other hand, for the property hasCity of Countries, there is no direct entry, which denotes that its inverse direction (which is functional) is stored explicitly, and must serve for lookup.

For the n:m tables, here locatedAt as a plain example, the entries tell that the (URIs of the) waters where a city is located can be found in the Water column

MappingDict					
Class	**Property**	**RangeClass**	**Table**	**LookupAttr**	comment
Country	code	xsd:string	Country	code	
Country	name	xsd:string	Country	name	
Country	population	xsd:int	Country	population	
Country	area	xsd:decimal	Country	area	
Country	capital	City	Country	capital	
Country	hasCity	City	–null–	–null–	(see inv)
Country	inMembership	Membership	–null–	–null–	(see inv)
City	name	xsd:string	City	name	
City	population	xsd:int	City	population	
City	hasCity_inv	Country	City	hasCity_inv	(inv)
City	locatedAt	Water	locatedAt	Water	(n:m)
City	locatedOn	Island	locatedOn	Island	(n:m)
City	capital_inv	Country	–null–	–null–	(see inv)
Organization	abbrev	xsd:string	Organization	abbrev	
Organization	hasHq	City	Organization	hasHq	
Organization	hasMembership	Membership	–null–	–null–	(see inv)
Membership	inOrganization	Organization	Membership	inOrganization	
Membership	ofCountry	Country	Membership	ofCountry	
Membership	type	xsd:string	Membership	type	
Water	name	xsd:string	River	name	(abstract)
Water	name	xsd:string	Lake	name	(abstract)
Water	name	xsd:string	Sea	name	(abstract)
Water	locatedAt_inv	City	locatedAt	city	(abstract)
River	name	xsd:string	River	name	
River	length	xsd:decimal	River	length	
River	locatedAt_inv	City	locatedAt	city	
Lake	name	xsd:string	Lake	name	
Lake	area	xsd:decimal	Lake	area	
Lake	locatedAt_inv	City	locatedAt	city	
Sea	name	xsd:string	Sea	name	
Sea	depth	xsd:int	Sea	depth	
Sea	locatedAt_inv	City	locatedAt	city	
Mountain	name	xsd:string	Mountain	name	
Mountain	elevation	xsd:decimal	Mountain	elevation	
Mountain	locatedOn	Island	Mountain	locatedOn	
Island	name	xsd:string	Island	name	
Island	area	xsd:decimal	Island	area	
Island	population	xsd:decimal	Island	population	
Island	locatedOn_inv	City	locatedOn	city	
Island	locatedOn_inv	Mountain	–null–	–null–	(see inv)

Table 2. Sample Mapping Dictionary contents

of the locatedAt table. Note the different handling of locatedOn for Cities (n:m) and mountains (functional). Analogously the inverse of locatedOn can be found via the n:m table locatedOn (cities) or by using the locatedOn column of the Mountain table in the inverse direction.

So far, the basic lookup information is covered. In case of relationships, the lookup yields the URI of an object, which then must be joined appropriately; in case of n:m tables, also the "backwards" join to the domain side is needed.

Further Tables. Theoretically, the MD is sufficient for translating SPARQL queries to SQL, and for other applications, if the domains and ranges of properties are known from the ontology, and if the relational schema is canonically derived from the ontology. The goal of the SDD is to make its information self-contained, accessible by the (relational) user, and adaptable/updatable for non-canonical relational schemata, e.g. in the general OBDA case, or even for denormalized storage schemata. The query translation algorithm uses only the information stored in the SDD, without any time-consuming (DL and/or SPARQL) queries against the original OWL ontology.

Thus, the SDD is required to contain enough information to also generate the join conditions: join the uris obtained for variables in object position with the class tables, and join ClassTables with NMtables.

Range Tables Table. The MD contains a column containing the range *class* of object-valued properties. This class is often an abstract one, or even an unnamed class specified by an owl:unionOf. In such cases, the concrete objects are stored in one or more ClassTables.

The (basic) *Range Tables Table* (RTTab) gives for each pair (table, column) the ClassTables where the referenced URIs can be found; the sample RTTab is given in Table 3.

RTTab (basic)			
Table	**LookupAttr**	**RangeTable**	comment
Country	capital	City	
City	hasCity_inv	Country	
Organization	hasHq	City	
Membership	inOrganization	Organization	
Membership	ofCountry	Country	
locatedAt	City	City	
locatedAt	Water	River	(abstract)
locatedAt	Water	Lake	(abstract)
locatedAt	Water	Sea	(abstract)
locatedOn	City	City	
locatedOn	Island	Island	
Mountain	locatedOn	Island	

Table 3. Sample Range Tables Table contents

The RTTab contains one or more entries for each object-valued property stored in a ClassTable (i.e., the functional ones), and for *each* of the columns of an n:m table. If it contains several entries for a (table, column) pair, they again denote a union. This may be the case if an abstract class (like Water) is stored in several subclass tables, or if the range of some property naturally includes several concrete classes.

Example 2. Consider a lookup along a functional relationship:

{ ?C a Country; :name ?CN; :capital [:name ?XN] }

The MD tells that the capital of a country can be looked up in Country.capital, whose range class is City (whose names can be found in the City table) and its range table. The RTTab indicates that the range table of Country.capital is also the City table, which yields the join condition:

SELECT country.name, city.name
FROM country, city
WHERE country.capital = city.uri

Example 3. Consider a query that uses an inverse-functional relationship:

{ ?C a Country; :name ?CN; :hasCity [:name ?XN] }

Here, the MD tells that the hasCity property of Country is not stored (directly). Instead, its inverse, the hasCity_inv property of its range class, City, is listed to be looked up in City.hasCity_inv whose range is Country and the range table is country:

SELECT country.name, city.name
FROM country, city
WHERE country.uri = city.hasCity_inv

NM Join Table. The RTTab covers the generation of joins over the range of a property. For n:m properties, the domain side of the property needs also to be "back"-joined with one or more ClassTables (it is possible that the domain and range for n:m properties consist of multiple concrete classes). The *NM Join Table (NMJ)* table yields the column name to be used for the join. It is a mapping

$$\mathsf{NMJoinTab}: (\mathsf{Class} \times \mathsf{Table} \times \mathsf{(LookUp)Attribute}) \to \mathsf{(FKJoin)Attribute}$$

where an entry $(c, t, l) \mapsto f$ indicates that, given an instance a class c, for which the attribute l should be looked up in the (n:m-)table t, the attribute f of t must be matched with the URI column of the ClassTable of c. Note that a lookup in the NMJ always follows a lookup for (c, p) yielding (t, l) in the MD. The NMJ for the running example is given in Table 4. (As long, as only pure "typical" n:m tables are considered, the NMJ is just "give me the other column name in t").

NMJoinTab			
Class	**Table**	**LookupAttr**	**FKJoinAttr**
City	locatedAt	Water	City
Water	locatedAt	City	Water
River	locatedAt	City	Water
Lake	locatedAt	City	Water
Sea	locatedAt	City	Water
Island	locatedOn	City	Island
City	locatedOn	Island	City

Table 4. Sample NM Join Table contents

Example 4. Consider the following SPARQL query pattern:

{ ?C a :City; :name ?CN; :locatedAt ?L . ?L a :Lake; :name ?LN }

The MD tells to look up the locatedAt property of cities in locatedAt.Water whose range tables (for ?L) are according to the RTTab River, Lake, and Sea. Since ?L is restricted in the query to lakes whose names are looked up in Lake.name, the union is not needed here. The NMJ tells to join City.URI with locatedAt.City:

```
SELECT city.name, lake.name
FROM country, locatedAt, lake
WHERE locatedAt.Water = lake.uri AND city.uri = locatedAt.city
```

Class Hierarchy. To be self-contained, the SDD also contains information about the class hierarchy of the ontology. Straightforwardly, there are two tables:

- AllCl $\subset$ (sub)class $\times$ (super)class: contains the whole class hierarchy (i.e., concrete and abstract classes).
- SubCl $\subset$ (sub)class $\times$ (non-abstract-super)class: contains for each class its finest non-abstract superclass.

Inverse-Functional Properties. As illustrated in Example 3, for properties like Country.hasCity that are inverse-functional, the MD contains null entries that mean that a lookup in the inverse direction is required. For more easier use, the MD is extended with an additional column "Inv" which is false by default, and true if the property is stored in the inverse direction, and then directly contains the table/column where it is found. The extended MD is shown in Table 5:

MappingDict					
Class	**Property**	**RangeClass**	**Table**	**LookupAttr**	**inv**
... all entries where tablename is not null from Table 2 ...					false
Country	hasCity	City	City	hasCity_inv	true
Country	inMembership	Membership	Membership	ofCountry	true
City	capital_inv	Country	Country	capital	true
Organization	hasMembership	Membership	Membership	inOrganization	true
Island	locatedOn_inv	Mountain	Mountain	locatedOn	true

Table 5. Sample extended Mapping Dictionary contents for inverse-functional properties

Primary Keys in Relational Databases. The above modeling and discussion uses the URIs as primary keys and as targets of references. The relational databases that should be made accessible by OBDA do not necessarily have this feature. For that, additional SDD tables for keys and foreign keys (similar to those of relational data dictionaries) are used to store the information about keys; covering also n-ary keys. With this, the generation of joins consists of an additional lookup for the actual names of the key/foreign key columns.

Redundancy vs Flexibility. When the relational schema is derived canonically from the OWL ontology, the RTTab is (so far) mostly redundant because the information can be obtained from the RangeClass column of the MD and another

lookup in the MD for that/these class/classes. The NMJ is in such cases also redundant (i.e., can be defined as a view that just yields the other column name of the n:m table). If an OBDA mapping to another relational schema that deviates from the canonical translation in its structure or in the naming is required, these tables and entries deviate. In Section 4.3 it will be shown how the NMJ can be used to define the "original" binary relationship underlying an attributed reified property. In Section 4.4 it will be shown that actually the inherent "NMJoin" semantics is actually a more general "back-join" that links tables with a ClassTable, e.g., for representing vertical partitioning.

3 Usage for Transformation of Queries

The transformation of a SPARQL queries to SQL queries has already been used above for illustrating the contents of the SDD tables.

- functional properties in the MD; for object-valued properties lookup the range table(s) in the RTTab;
- n:m relationships also in the MD, get name/column in an n:m-table, lookup the range table for this column (RTTAB), and lookup the column name of the column which has to be joined with the ClassTable of the subject (NMJ);
- multivalued literal-valued properties: analogously.

Non-Typed Variables in Queries. If the class of one or more variables is not explicitly specified in the query (e.g. {?X :name ?N; :area ?A}), the MD allows to generate the query as an SQL union based on all classes that provide the required properties (here, countries and lakes):

```
(SELECT name, area FROM country) UNION (SELECT name, area FROM lake)
```

Variables in Property Position. If a variable in the SPARQL query occurs in property position, all properties must be considered.

Example 5. Consider the query

```
{ ?X :name ?N; :area ?A; ?P ?Y . ?Y :name ?Z }
```

that asks for all names of objects ?X that have an area and some relationship ?P to another object ?Y that has a name ?Z.

The query transformation can again restrict the set of bindings for ?X to countries and lakes (only these have an area property). The transformation then must consider all properties (e.g., country.area, country.population, country.capital, ..., lake.area, ... lake.locatedAt_inv) for binding ?P. For these, it must check the range class information in the MD, and check whether this class provides a name property. Only those contribute to the union, whose constituents are then constructed by appropriate joining (note again that country.capital if functional, while country.hasCity is inverse-functional, and lake.locatedAt_inv is n:m):

```
(SELECT x.name, x.area, 'capital', y.name FROM country x, city y
 WHERE x.capital = y.uri) UNION
(SELECT x.name, x.area, 'hasCity', y.name FROM country x, city y
 WHERE x.uri = y.hasCity_inv) UNION
(SELECT x.name, x.area, 'locatedAt_inv', y.name FROM lake xl, locatedAt p, city y
 WHERE x.uri = p.water AND p.city = y.uri)
```

Usage for inserting data. In an obvious way, the mapping information can also be used for inserting new RDF triples into the database. By this, an initially empty schema can be populated from an RDF data source, or e.g. dynamically by harvesting data from the Web.

4 Extensions and Modeling Variants

The previous sections dealt with the canonical transformation. In this section, some more insight is given for handling special situations. The section also illustrates the flexibility of the chosen structure for the SDD.

4.1 Abstract Subclasses

The above example already showed how abstract superclasses (like Water for River, Lake, and Sea) are dealt with in the canonical transformation. Often, a subclass is discriminated against its superclass only by some special, extending properties; e.g., Volcano as a special subclass of Mountain, with an additional attribute lastEruption. In such cases, the schema designer may choose to keep the subclass *abstract*: there is no separate table, and the instances are completely stored in the table of the superclass. Such a schema is e.g. advantageous, when most queries use also attributes of the superclass.

In this case, the functional attributes are canonically added to ClassTable that contains the functional properties of the superclass, and there are two possibilities to distinguish the elements of the subclass:

- a boolean column with the name of the subclass (especially, if there are multiple, maybe non-disjoint subclasses of the superclass), or
- a column, e.g. called "type" (chosen by the schema/transformation designer), that contains a marker value (e.g., the name of the subclass). This variant is only possible if the subclasses are disjoint.

The relevant metadata information is a mapping of the form SubClass $\rightarrow$ (Table$\times$ Column $\times$ Value). This extends the "SubCl" table which contains for each class its most specific concrete superclass.

Example 6. Consider that Mountain has (disjoint) abstract subclasses Volcano with an additional attribute lastEruption, and Monolith; and Island has abstract subclasses VolcanicIsland and Atoll, which are modeled according to the first variant.

Sample entries of the "SubCl" table are shown in Table 6. Additionally, MD entries for the subclasses have to be added, including (Volcano, lastEruption, xsd:string, Mountain, lastEruption, false). By this, queries like

{ ?X a :Volcano; :name ?N; :lastEruption ?D }

can be easily translated into

SELECT name, lastEruption FROM Mountain WHERE type='Volcano';

In the canonical modeling, the entries in the "Superclass" column are always the same as in the "Table" column – this might be different when the relational schema (structure and naming) differs from the canonical one.

Subcl				
Subclass	**Superclass**	**Table**	**ColumnName**	**MarkerValue**
Volcano	Mountain	Mountain	type	Volcano
Monolith	Mountain	Mountain	type	Monolith
VolcanicIsland	Island	Island	volcanic	true
Atoll	Island	Island	atoll	true

Table 6. Sample entries of the SubClasses table

4.2 Multiple Inheritance

If multiple inheritance (i.e., a subclass might have *multiple* finest superclasses) should be supported, appropriate entries have to be added to the SubCl table, and the MD entries for the inherited properties of the subclass indicate in which "parent class" table each property has to be looked up.

4.3 Making Reified Properties Accessible as Properties

Attributed binary relationships, like the isMember relationship between countries and organizations have to be modeled by artificial (reified) entity types; here Membership (cf. Figures 1 and 2). The respective table, here Membership(uri, ofCountry, inOrganization, type), has two foreign keys that reference the involved instances of the entity types, and additional columns for the attributes. By this, such tables can be seen as generalizations of the binary n:m tables. The *projection* to the foreign keys, here π[ofCountry, inOrganization](Membership), represents the original binary relationship.

Thus, this relationship can be added to the OBDA mapping by appropriate entries in the MD (to define it as a lookup-able property) and NMJ (to handle the reified class table as an NM table) without having to be stored explicitly as facts.

Example 7. Consider the above-mentioned isMember relationship. The additional entries in the MD (to define it as a lookup-able property) and NMJ are shown in Table 7. Then, queries using isMember can seamlessly be transformed:

{ ?C a :Country; :name ?CN; :isMember [:name ?Org]. }

```
SELECT c.name, o.name
FROM Country c, Membership m, Organization o
WHERE c.uri = m.country AND m.organization = o.uri;
```

Note that in this case, the (before seemingly redundant) NMJ information is necessary to guide how to join "through" the 4-ary table.

MappingDict				
Class	**Property**	**Range**	**Table**	**LookupAttr**
Country	isMember	Organization	Membership	Organization
Organization	hasMember	Country	Membership	Country

NMJoinTab			
Class	**Table**	**LookupAttr**	**FKJoinAttr**
Country	Membership	Organization	Country
Organization	Membership	Country	Organization

Table 7. MD & NMJ entries that allow the usage of the original *isMember* property

4.4 Multi-Range (object-valued) Functional Properties

For functional relationships where the range consists of multiple classes (or an abstract superclass of such), even for the "canonical" transformation from ER to the relational model there are different variants. The SDD approach allows to express the OBDA mapping in all these cases; mainly by having the appropriate entries in the MD – the RTTab must just be completed accordingly, and the NMJ is not effected. Below, an extended RTTab is introduced to be more efficient to "choose" the range table depending on a specific range class.

Example 8. Consider that a river flows (finally) into at most one water, which is *either* another river, a lake or a sea. As a functional property, it is canonically stored in the River table. There are two alternatives for this:

a) The property is stored as a single column River.flowsInto whose values refer to the IDs of rivers, lakes and seas. The MD entries (including the inverse) and the RTTab entries are shown in Table 8.
   For a SPARQL query of the form
   { ?R a :River; :name ?RN; :flowsInto ?L . ?L a :Lake; :name ?LN }
   the query translation can exploit that for Lake as intended range class, the extended RTTab tells that only a join with the Lake table must be done (and the SQL inner join does the rest to list only rivers that flow into a lake):
   SELECT r.name, l.name FROM River r, Lake l WHERE r.flowsInto = l.uri
b) The property is stored as multiple columns flowsInto_River, flowsInto_Lake and flowsInto_Sea whose values refer to the IDs of their respective class. The MD entries (including the inverse) are shown in Table 9.
   The same query as above is now translated into
   SELECT r.name, l.name FROM River r, Lake l WHERE r.flowsInto_Lake = l.uri
   Here, the MD information tells that for range Lake only one column of these must be considered. Without the query restriction to lakes, in both cases, a union would be generated.

Vertical Partitioning. Vertical partitioning of a database means that a table, usually one that corresponds to a class, is partitioned into several tables with fewer columns. Without loss of generality, one of these tables can be seen as the main ClassTable for that class, while the others are "outsourced" tables.

MappingDict					
Class	**Property**	**RangeClass**	**Table**	**LookupAttr**	**inv**
River	flowsInto	Water (still, abstract)	River	flowsInto	false
Sea	flowsInto_inv	River	River	flowsInto	true
Lake	flowsInto_inv	River	River	flowsInto	true
River	flowsInto_inv	River	River	flowsInto	true

RTTab			
Table	**LookupAttr**	**RangeClass**	**RangeTable**
River	flowsInto	River	River
River	flowsInto	Lake	Lake
River	flowsInto	Sea	Sea

Table 8. MD and (extended) RTTab entries for the "flowsInto" property as a single column

MappingDict					
Class	**Property**	**RangeClass**	**Table**	**LookupAttr**	**inv**
River	flowsInto	River	River	flowsInto_River	false
River	flowsInto	Lake	River	flowsInto_Lake	false
River	flowsInto	Sea	River	flowsInto_Sea	false
Sea	flowsInto_inv	River	River	flowsInto_Sea	true
Lake	flowsInto_inv	River	River	flowsInto_Lake	true
River	flowsInto_inv	River	River	flowsInto_River	true

RTTab			
Table	**LookupAttr**	**RangeClass**	**RangeTable**
River	flowsInto	River	River
River	flowsInto	Lake	Lake
River	flowsInto	Sea	Sea

Table 9. MD and (extended) RTTab entries for the "flowsInto" property split into multiple columns

Actually, the semantics of the back-join of these tables with the ClassTable is analogous to that for n:m tables and is covered by the NMJ (which could thus more generally be called "back-join-table").

Example 9. Consider economical attributes of countries like inflation, unemployment, and gross domestic product (gdp), that are stored in a separate table economy(country, inflation, unemployment, gdp) where the country column references to the uri column of the Country table. The MD and NMJ entries are given in Table 10.

For the SPARQL query

{ ?C a :Country; :name ?N; :inflation ?I ,}

the MD tells that name is looked up in the Country ClassTable, while inflation is looked up in the economy table. The NMJ tells that for looking up the inflation column in the economy table for an instance of class Country, the join must be via economy.country:

```
SELECT Country.name, economy. inflation
FROM Country, economy
WHERE economy.Country = Country.uri
```

MappingDict					
Class	**Property**	**Range**	**Table**	**LookupAttr**	**inv**
Country	inflation	economy	xsd:decimal	inflation	false
Country	unemployment	economy	xsd:decimal	unemployment	false
Country	gdp	economy	xsd:decimal	gdp	false

NMJoinTab			
Class	**Table**	**LookupAttr**	**FKJoinAttr**
Country	economy	inflation	Country
Country	economy	unemployment	Country
Country	economy	gdp	Country

Table 10. MD and NMJ entries for vertical partitioning

4.5 Symmetric Non-Functional Properties

Amongst recursive properties, symmetric ones (i.e. where $p \equiv p^-$ holds) play a special role. Consider for example a mergesWith relationship between seas. It is mapped to an n:m table mergesWith(sea1,sea2). The instances can then be stored in two alternative ways: Either storing only one direction of each instance of the relationship explicitly ("antisymmetric"), or (redundantly) storing both directions, which makes query formulation easier (no "union") and slightly faster. The SDD mapping described above can be configured for both cases:

- in the the symmetric/redundant case, there is just an MD entry $(c, p, c, tab(p), col_1, \mathsf{false})$ where $tab(p)$ is the n:m table where p is stored, for each class c in the domain of p.
- in the the antisymmetric case, there are two MD entries $(c, p, c, tab(p), col_1, \mathsf{false})$ and $(c, p, c, tab(p), col_2, \mathsf{false})$ for each such c (again, multiple entries mean a union).

Note that the handling of symmetric properties that are functional (like e.g., a "marriedWith" relationship between persons) is integrated within the class tables where the storage is inherently symmetric and redundant. Here, an antisymmetric storage would be counterintuitive.

5 Current Functionality of RDF2SQL and Perspectives

The current prototype of RDF2SQL can be found at [RDF17]. The central functionality is the transformation of an OWL ontology (optionally with additional annotations about concrete and abstract classes, reification etc.) into a relational schema and the generation of the SDD tables. RDF data can be inserted either from a file, or triplewise (e.g. when harvesting data from Web sources). SPARQL queries can be translated into SQL queries and can be evaluated.

Schema Transformation Operators. As described above, for an ontology, there are usually different variants how to store it in a relational schema. These differences between such variants can be broken down to a set of "atomic" transformations of the schema, e.g. handling a property as functional or as n:m, partitioning according to range or domain (for n:m tables), turning classes concrete or abstract, vertical partitioning of columns, etc. Each such operation corresponds to a transformation of the schema, and of the SDD data. With this, different schema variants for the same ontology have been generated for the use in the RODI experiments [PBJR+17].

"Inverse" Transformation. Obviously, a preliminary ontology can also be created by re-engineering from the relational schema: deriving what can be classes, and what should be properties. The quality of the obtained ontology depends on identifying artifacts from reification and partitioning. The corresponding SDD is also created. The testcase here is to take the traditional relational MONDIAL schema, and to provide OBDA to it from the generated ontology.

Perspectives. One research direction is to use RDF2SQL and especially the SDD for *finding* and describing a mapping between a *given* ontology and a *given* relational database. A combination of ontology re-engineering and ontology alignment with *direct* generation of an SDD describing the mapping has been experimented with in [Sch16] with already encouraging results. Based on this, classical ontology alignment strategies can be applied in combination with atomic transformations of the SDD which guarantee that with every step, full correctness of the mapping is maintained.

6 Conclusion

The presented metadata structure, called *Semantical Data Dictionary (SDD)*, provides a flexible, extensible and efficient technique for storing ODBA mapping metadata in SQL with a human readable representation which is comfortable and intuitive to use. With the RDF2SQL implementation, the SDD is automatically generated and populated from a given OWL ontology.

The approach has been implemented originally for the generation of a relational schema from an ontology (which has been used above as running example), and for translating SPARQL queries into corresponding SQL queries. It already proved useful for generating different modeling variants, either controlled via annotation properties in the ontology (about abstract and concrete classes), or by transformations of the generated schema (e.g., turning properties from functional into n:m and vice versa). Such variants were used in [PBJR+17].

Therefore the SDD can not only be used for query translation in case that the relational schema was created from the ontology, but might also serve as a base for automatically connecting an ontology to a different, given relational database by generating the aligning mapping by transformations of the SDD.

The SDD structure can easily be extended by additional tables for further needs since everything is stored transparently in the SQL database.

The SDD can also be used for mappings from other data models (e.g. XML with elements, subelements, and attributes) to a relational database. Furthermore, it could be used as a virtual language to provide communication between different database languages and sources.

References

[ADMR05] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and Ontology Matching with COMA++. In *SIGMOD*, pp. 906–908. ACM, 2005.

[CCK+15] D. Calvanese, B. Cogrel, E. G. Kalayci, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao. OBDA with the Ontop Framework. In *23rd Italian Symposium on Advanced Database Systems, SEBD*, pp. 296–303, 2015.

[dMPC15] L. F. de Medeiros, F. Priyatna, and O. Corcho. MIRROR: Automatic R2RML Mapping Generation from Relational Databases. In *ICWE*, Springer LNCS 9114, pp. 326–343, 2015.

[EMS+11] J. Euzenat, C. Meilicke, H. Stuckenschmidt, P. Shvaiko, and C. T. dos Santos. Ontology Alignment Evaluation Initiative: Six Years of Experience. *J. Data Semantics*, 15:158–192, 2011.

[EN04] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 4th Edition*. Addison-Wesley-Longman, 2004.

[HM13] T. Hornung and W. May. Experiences from a TBox Reasoning Application: Deriving a Relational Model by OWL Schema Analysis. In *OWLED 2013*, *CEUR Workshop Proceedings* 1080. CEUR-WS.org, 2013.

[JRKZ+15] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skjæveland, E. Thorstensen, and J. Mora. BootOX: Practical Mapping of RDBs to OWL 2. *ISWC*, Springer LNCS 9367, pp. 113–132, 2015.

[KSA+12] C. A. Knoblock, P. Szekely, J. L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick. Semi-automatically Mapping Structured Sources into the Semantic Web. In *ESWC*, Springer LNCS 7295, pp. 375–390, 2012.

[Mondial] W. May. The MONDIAL Database. Available at http://dbis.informatik.uni-goettingen.de/Mondial/.

[PBJR+15] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, W. May, D. Ritze, M. G. Skjæveland, A. Solimando, and E. Kharlamov. RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration. In *ESWC*, Springer LNCS 9088, pp. 21–37, 2015.

[PBJR+17] C. Pinkel, C. Binnig, E. Jiménez-Ruiz, E. Kharlamov, W. May, A. Nikolov, A. S. Bastinos, M. G. Skjæveland, A. Solimando, M. Taheriyan, C. Heupel, and I. Horrocks. RODI: Benchmarking Relational-to-Ontology Mapping Generation Quality. *Semantic Web Journal*, page to appear, 2017.

[PBKH13] C. Pinkel, C. Binnig, E. Kharlamov, and P. Haase. IncMap: Pay As You Go Matching of Relational Schemata to OWL Ontologies. In *Workshop on Ontology Matching*, *CEUR Workshop Proceedings* 1111, 2013.

[RDF17] RDF2SQL - Relational Storage of RDF Data (Demo). Available at http://www.semwebtech.org/rdf2sql/, 2017.

[Sch16] S. Schrage. Transformation-based Ontology Mapping. Master's thesis, Universität Göttingen, Institut für Informatik, 2016. Available at https://www.dbis.informatik.uni-goettingen.de/Teaching/Theses/PDF/MSc-Schrage-SchemaMatch-2016.pdf.