BOOTOX: Practical Mapping of RDBs to OWL 2

Ernesto Jiménez-Ruiz[1]*, Evgeny Kharlamov[1], Dmitriy Zheleznyakov[1], Ian Horrocks[1], Christoph Pinkel[2], Martin G. Skjæveland[3], Evgenij Thorstensen[3], and Jose Mora[4]

[1] Department of Computer Science, University of Oxford, UK
[2] fluid Operations AG, Walldorf, Germany
[3] Department of Informatics, University of Oslo, Norway
[4] Sapienza Università di Roma, Italy

Abstract. Ontologies have recently became a popular mechanism to expose relational database (RDBs) due to their ability to describe the domain of data in terms of classes and properties that are clear to domain experts. Ontological terms are related to the schema of the underlying databases with the help of mappings, i.e., declarative specifications associating SQL queries to ontological terms. Developing appropriate ontologies and mappings for given RDBs is a challenging and time consuming task. In this work we present BOOTOX, a system that aims at facilitating ontology and mapping development by their automatic extraction (i.e., bootstrapping) from RDBs, and our experience with the use of BOOTOX in industrial and research contexts. BOOTOX has a number of advantages: it allows to control the OWL 2 profile of the output ontologies, bootstrap complex and provenance mappings, which are beyond the W3C direct mapping specification. Moreover, BOOTOX allows to import pre-existing ontologies via alignment.

1 Introduction

The Semantic Web community has actively investigated the problem of bridging the gap between relational databases and ontologies. One of the main targets behind this effort is to enable the access to the data stored in databases via Semantic Web technologies. An advantage of this approach is that ontologies provide a formal specification of the application domain that is close to the end-users' view of the domain, while databases are oriented towards an efficient data storage and retrieval and thus represent the data using structures that often not intuitive to end-users.

Manually building an ontology and connecting it to the data sources via mappings is, however, a costly process, especially for large and complex databases. To aid this process, tools that can extract a preliminary ontology and mappings from database schemata play a critical role. In the literature one can find a broad range of approaches to bootstrap an ontology and mappings from a relational database schema. The interested reader may have a look at the following surveys [30, 34]. These approaches can be classified with respect to different aspects such as:*(i)* level of automation (i.e., manual, semi-automatic, automatic), *(ii)* type of mappings (i.e., complex or direct mappings), *(iii)* language of the bootstrapped mappings and the ontology, *(iv)* reuse of external vocabularies (e.g., domain ontologies or thesauri), and *(v)* purpose (e.g., OBDA, constraint validation, database integration, ontology learning).

In this paper we present BOOTOX[5] (not) yet another ontology and mapping bootstrapper. Our main motivation to implement a new bootstrapper is to give more flexi-

* Corresponding author: ernesto@cs.ox.ac.uk, ernesto.jimenez.ruiz@gmail.com

[5] `http://www.cs.ox.ac.uk/isg/tools/BootOX/`

bility with respect to the classification aspects described above. Most of the existing approaches commit to concrete purposes or to a concrete ontology expressiveness. BOOTOX allows to define different "profiles" depending on the application scenario and the required Semantic Web technologies. For example, if the bootstrapped ontology is to be used in a so-called Ontology Based Data Access (OBDA) scenario where the ontology provides a virtual access layer to the data, OWL 2 QL will be chosen as the ontology language as it is required by the query rewriting engine.[6] Nevertheless, if the data is materialised, one could opt for other OWL 2 profiles depending on the used query answering engine. BOOTOX also allows to import domain ontologies, which will be integrated to the bootstrapped one via alignment [14] or directly mapped to the database. BOOTOX follows the W3C direct mappings directives to connect the ontological vocabulary to the relational database; moreover, it offers a suit of advanced techniques for bootstrapping of mapping that are beyond the direct ones. Furthermore, it extends the bootstrapped mappings with provenance information.

We have tested BOOTOX and compared it to several bootstrapping systems over a number of databases and use cases from industrial and research contexts, including the relational-to-ontology benchmark suite RODI [25].

The paper is organised as follows. Section 2 introduces the concepts behind semantic access to databases. Section 3 presents the problem of bootstrapping. Section 4 describes the techniques implemented in BOOTOX. Section 5 presents the scenarios where we deployed BOOTOX and the conducted experiments to evaluate the quality of the bootstrapping. In Section 6 we provide a summary of relevant related work, and we conclude in Section 7.

2 Exposing Relational Data Through Ontologies

The main idea behind exposing relational data via an ontology is to provide the user with access to the data store via the use of a domain specific vocabulary of classes, i.e., unary predicates, and properties, i.e., binary predicates, that the user is familiar with. This vocabulary is related to the database schema via view definitions, called *mappings*; thus, technical details of the database schema are hidden from end-users. Using an example from the oil industry domain, we now intuitively introduce the main concepts that are needed to understand the approach and refer the reader to [26] for details.

Relational Databases. In the relational data model, a database consists of a schema and instance, where the schema is a set of table names with corresponding attribute names and constraints, e.g., primary and foreign keys. The instance 'populates' tables of the schema with tuples by assigning values to the tables' attributes. Figure 1 shows our running example, a fragment of a relational database based on the oil industry domain. For example, a Wellbore is given a name that is unique, has a content (e.g., GAS, OIL, or DRY), has a depth (e.g., 12,500 feet), belongs to a Well, and is located in a Field.

Ontologies. An ontology is usually referred to as a 'conceptual model' of (some aspect of) the world. It introduces the vocabulary of classes and properties that describe various aspects of the modelled domain. It also provides an explicit specification of the intended meaning of the vocabulary by describing the relationships between different vocabulary terms. These relationships are specified with special first-order formulae, also called axioms, over the vocabulary. An ontology can be thought of simply as a set of such axioms—i.e., a logical theory. Besides axioms, an ontology can contain

[6] The language underlying OWL 2 QL has the first-order (FO) rewritability property [6].

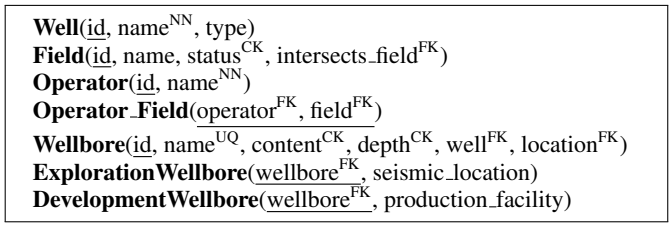


Fig. 1: Relational Database Schema. Primary keys are underlined. FK stands for Foreign Key (the foreign key name indicates the target table), NN stands for Not Null, UQ stand for Unique and CK stands for Check

ontological facts, which can be specified as first-order atoms with constants, but not variables. These constants are usually interpreted as (representations of) objects from the domain. Viewing an ontology as a logical theory opens up the possibility of using automated reasoning for different tasks, including checking an ontology's internal consistency (checking whether the ontology does not entail 'false', i.e., if it is logically consistent), query answering, and other (non-)entailments. OWL 2 is a W3C standard ontology language and it is heavily used in the Semantic Web community. OWL 2 has profiles:[7] EL, QL, and RL, that have different favourable computational properties.

Mappings. Mappings declaratively define how ontological terms are related to terms occurring in the relational schema and are essentially view definitions of the following form that declare how to populate classes with objects—in OWL 2 objects are represented with Uniform Resource Identifiers (URIs)—and to populate properties with object-object and object-value pairs:

$$\begin{aligned} \textit{Class}(f_o(x)) &\leftsquigarrow \texttt{SQL}(x), \\ \textit{objectProperty}(f_o(x), f_o(y)) &\leftsquigarrow \texttt{SQL}(x, y), \\ \textit{dataProperty}(f_o(x), f_v(y)) &\leftsquigarrow \texttt{SQL}(x, y), \end{aligned}$$

where $\texttt{SQL}(x)$ and $\texttt{SQL}(x, y)$ are SQL queries with respectively one and two output variables, and f_o, f_v are functions that 'cast' values returned by the SQL queries into respectively objects, i.e, URIs, and values.[8] Classes are populated with URIs $f_o(x)$ computed from the values x returned by $\texttt{SQL}(x)$. Properties can relate two objects, e.g., by stating that a wellbore has a particular location, or assigning a value to an object, e.g., stating that a field has a particular name (a string); and they are respectively populated with pairs of objects $f_o(x)$, $f_o(y)$ or pairs of an object $f_o(x)$ and value $f_v(y)$ computed from the values x and y returned by the SQL query. For our running example, we may create the following mappings:

$$\begin{aligned} \mathit{Wellbore}(f_o(\texttt{id})) &\leftsquigarrow \texttt{SELECT id FROM Wellbore}, \\ \mathit{hasLocation}(f_o(\texttt{id}), f_o(\texttt{location})) &\leftsquigarrow \texttt{SELECT id, location FROM Wellbore}, \\ \mathit{hasContent}(f_o(\texttt{id}), f_v(\texttt{content})) &\leftsquigarrow \texttt{SELECT id, content FROM Wellbore}. \end{aligned}$$

Query Answering. Consider a data access instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$, where $\mathcal{D}$ is an RDB, $\mathcal{V}$ is an ontological vocabulary, $\mathcal{O}$ is a set of ontological axioms over $\mathcal{V}$, and $\mathcal{M}$ is a set of mappings between $\mathcal{V}$ and $\mathcal{D}$. There are two approaches to answer a query Q over $\mathcal{V}$: *(i) materialisation*: ontological facts are materialised (i.e., classes and properties participating in mappings are populated with individuals by evaluating SQL queries participating in mappings) and this gives a set of ontological facts $\mathcal{A}$ and then Q is evaluated

[7] OWL 2 profiles: `http://www.w3.org/TR/owl2-profiles/`

[8] These functions ensure coherent generation of URIs that respects primary and foreign keys.

over $\mathcal{O}$ and $\mathcal{A}$ with standard query-answering engines for ontologies, or *(ii) virtual*: Q should be first rewritten into an SQL query SQL using $\mathcal{O}$ and $\mathcal{M}$ and then SQL should be executed over $\mathcal{D}$.

In either case, the ontology and mappings are the backbone of query processing in this data access approach; thus, the problem of obtaining them has to be addressed in any implementation of such approach. The exact kind of ontology and mappings required for data access depends on the application scenario. For example, in a scenario where virtual query answering is required, the ontology should be in OWL 2 QL. In the case of materialisation the ontology should fall in the profile supported by the underlying systems, e.g., *EOLO* [35] can do query answering over OWL 2 EL ontologies, and thus the bootstrapping should output an EL ontology; a semantic faceted search system SEMFACET [2] relies on RDFOX [23] that can do query answering over OWL 2 RL ontologies and thus the bootstrapper facilitating SEMFACET should produce RL ontologies; PAGOdA [40] does query answering over the whole OWL 2 and can work with any bootstrapper that produces OWL 2 ontologies.

3 Bootstrapping: Problem Definition

In this section we define the tasks for bootstrapping ontologies and mappings, or *assets*, from RDBs and quality metrics for these tasks. The tasks are the following:

(i) *Semantic access instance generation*: Given a relational database $\mathcal{D}$, generate an instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$. This task can be naturally divided into two sub-tasks.
- *Vocabulary and Ontology generation*: Given $\mathcal{D}$, create a vocabulary $\mathcal{V}$ and an ontology $\mathcal{O}$ over $\mathcal{V}$.
- *Mapping generation*: Given $\mathcal{D}$, $\mathcal{V}$, and $\mathcal{O}$ create a set of mappings $\mathcal{M}$ relating $\mathcal{D}$ with $\mathcal{V}$.

(ii) *Importing*: Given an instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}_1, \mathcal{M})$ and an ontology $\mathcal{O}_2$, return an instance $(\mathcal{D}, \mathcal{V}, \mathcal{O}, \mathcal{M})$, where $\mathcal{O}$ is the alignment of $\mathcal{O}_1$ and $\mathcal{O}_2$.

Task (ii) is important in applications where ontologies (partially) describing the domain of interest have been already created and users want to incorporate them into their semantic data access system.

The bootstrapping of the ontologies and the mappings enables a (preliminary) deployment of semantic data access system. However, the bootstrapped assets should meet some minimal requirements so that they can be usable in practice. We have identified the following *metrics* to measure the quality of generated assets:

(1) *Ontology language*: compliance with standard ontology languages with well-defined semantics like OWL 2 and its profiles to enable the use of a wide-range of Semantic Web technologies. Note that the choice of the ontology language (e.g., one of OWL 2 profiles) also affects suitability of the different query answering engines.

(2) *Mapping language*: compliance with standard directives like the W3C direct mapping specification[9] and standard W3C mapping languages like R2RML.[10]

(3) *Query coverage*: suitability of the ontology vocabulary to formulate the queries that the user is interested in.

(4) *Query results*: accuracy of the query results obtained with the bootstrapped instance in the sense that the query answer should satisfy the user's expectations.

[9] Direct mappings: http://www.w3.org/TR/rdb-direct-mapping/

[10] R2RML language: http://www.w3.org/TR/r2rml/

4 Bootstrapping Techniques in BOOTOX

In this section we present our bootstrapping techniques implemented in BOOTOX. Section 4.1 focuses on the techniques to generate the ontological vocabulary and the axioms over this vocabulary, where we give special attention to the concrete language of the generated ontology. In Section 4.2 we describe how the database terms are linked to the ontological vocabulary. Furthermore, we also present how mappings are enhanced with provenance information. Section 4.3 gives an overview of the incorporated ontology alignment techniques to import pre-existing ontologies. Finally, Section 4.4 summarises the conformance with the requirements presented in Section 3.

4.1 Vocabulary and Ontology Generation

The general rule to create ontological vocabulary from a relational schema would translate *(i)* each (non-binary) table into an OWL class; *(ii)* each attribute not involved in a foreign key into an OWL datatype property; *(iii)* each foreign key into an OWL object property. Generation of axioms, however, is usually a more involved process. BOOTOX follows Table 1 to automatically create the vocabulary and a set of OWL 2 axioms from the listed database features.[11] One could opt for adding all the axioms associated with a feature or only a selection of them depending on the intended purpose. In the remainder of the section we will discuss the main considerations regarding bootstrapping that should be taken into account.

Closed-world vs Open-world Semantics. Modelling database features in OWL 2 inevitably leads to different interpretations due to the Closed-world (CWA) and Open-world assumptions (OWA). In a relational database, every fact that occurs in the database is true, and all other facts are false (CWA). In an ontology, the truth value of some facts may be unknown, i.e., they are neither provably true nor provably false (OWA). Moreover, constraints have a very different meaning in databases and ontologies [22, 30]. In a database, constraints define valid database states, and updates that would violate these constraints are simply rejected. In an ontology, constraints are treated as material implications, and may lead to the derivation of "new" facts that are not explicitly stated in the data. Such derivations may produce unintended consequences,[12] as we discuss next.

Table 1 presents a possible encoding of relational database features as OWL 2 axioms. The encoding, in general, does not lead to only one option, but several possible OWL 2 axioms. For example, considering the running example of Figure 1, for the data attribute *name* in table *Field* we apply encoding pattern (3) in Table 1, which proposes to add five different axioms. The general rule translates the data attribute *name* as a data property declaration axiom. In addition one may opt to add global restrictions (e.g., '$name$ Domain: $Field$')[13] and/or local restrictions (e.g., '$Field$ SubClassOf: $name$ some $xsd{:}string$'). The use of global and/or local restrictions will lead to different interpretations and potentially to different logical consequences when combined with data facts. For example, the fact '$operator987$ Facts: $name$ Statoil' in combination with the domain axiom for *Field* and *name* will lead to the undesired consequence '$operator987$ Types: $Field$'. In this case, if the property *name* is to be used in different contexts, using only local restrictions seems to be more appropriate.

[11] When not stated the contrary in Table 1, a class C_T, an object property P_f, a data property R_a and a datatype d_t represent the ontology encoding of a table T, a foreign key fk, a data attribute a, and an SQL type t, respectively.

[12] Note that the use of disjointness axioms may help in the detection of such "errors".

[13] For writing OWL 2 axioms we use the Manchester OWL Syntax [13].

Table 1: Encoding of relational database features as OWL 2 axioms. OWL 2 axioms are expressed in the Manchester OWL Syntax [13]. * Enumeration with only one literal

#	RDB feature	Ontology feature	OWL 2 axiom	OWL 2 profile		
				QL	RL	EL
(1)	**Non-binary Relation / Table** T	A class C_T for the non-binary table	Class: C_T	✓	✓	✓
(2)	**Binary Relation or Many-to-Many Table** referencing tables T_1 and T_2	A property P (and its inverse Q) associated to the classes C_{T_1} and C_{T_2} with local and/or global constraints	ObjectProperty: P	✓	✓	✓
			Q InverseOf: P	✓	✓	-
			P Domain: C_{T_1}	✓	✓	✓
			P Range: C_{T_2}	✓	✓	✓
			C_{T_1} SubClassOf: P some C_{T_2}	✓	-	✓
			C_{T_1} SubClassOf: P only C_{T_2}	-	✓	-
(3)	**Data attribute** in table T of (sql) type t	A property R_a associated to the class C_T and datatype d_t with local and/or global constraints.	DataProperty: R_a	✓	✓	✓
			R_a Domain: C_T	✓	✓	✓
			R_a Range: d_t	✓	✓	✓
			C_T SubClassOf: R_a some d_t	✓	-	✓
			C_T SubClassOf: R_a only d_t	-	✓	-
(4)	**Foreign Key** in table T_1, referencing T_2, **no intersection with** or **strict subset of** primary key	A property P_f associated to the classes C_{T_1} and C_{T_2} with local and/or global constraints	ObjectProperty: P_f	✓	✓	✓
			P_f Domain: C_{T_1}	✓	✓	✓
			P_f Range: C_{T_2}	✓	✓	✓
			C_{T_1} SubClassOf: P_f some C_{T_2}	✓	-	✓
			C_{T_1} SubClassOf: P_f only C_{T_2}	-	✓	-
(5)	**Foreign Key is the primary key** in T_1, ref. T_2	Class C_{T_1} is subsumed by class C_{T_2}	C_{T_1} SubClassOf: C_{T_2}	✓	✓	✓
(6)	**Foreign Key** in table T **referencing the same table**	A property P_f associated to class C_T with a self-restriction. The property P_f may also be declared with several characteristics	C_T SubClassOf: P_f some C_T	✓	-	✓
			C_T SubClassOf: P_f some Self	-	-	✓
			P_f Characteristics: Transitive	-	✓	✓
			P_f Characteristics: Symmetric	✓	✓	-
			P_f Characteristics: Reflexive	✓	-	✓
(7)	**Primary Key** or **Unique constraint** in table T_1 on a **foreign key** fk referencing T_2	Key axiom for class C_{T_1} and property P_f. P_f is associated to (local and/or global) cardinality constraints	C_{T_1} HasKey: P_f	-	✓	✓
			P_f Characteristics: Functional	-	✓	-
			P_f Characteristics: InverseFunctional	-	✓	-
			C_{T_1} SubClassOf: P_f exactly 1 C_{T_2}	-	-	-
			C_{T_1} SubClassOf: P_f max 1 C_{T_2}	-	✓	-
			C_{T_1} SubClassOf: P_f some C_{T_2}	✓	-	✓
(8)	**Primary Key** or **Unique constraint** on a **data attribute** a of (sql) type t in table T	Key axiom for class C_T and data property R_a. R_a is associated to (local and/or global) cardinality constraints.	C_T HasKey: R_a	-	✓	✓
			R_a Characteristics: Functional	-	✓	✓
			C_T SubClassOf: R_a exactly 1 d_t	-	-	-
			C_T SubClassOf: R_a max 1 d_t	-	✓	-
			C_T SubClassOf: R_a some d_t	✓	-	✓
(9)	**Composed Primary Key** in table T	Key axiom for the class C_T and the data and object properties involved in primary key	C_T HasKey: $R_1 \dots R_n$ $P_1 \dots P_n$	-	✓	✓
(10)	**Not Null Constraint** on a **data attribute** in T, type t	Existential or cardinality restriction over R_a in C_T	C_T SubClassOf: R_a min 1 d_t	-	-	-
			C_T SubClassOf: R_a some d_t	✓	-	✓
(11)	**Not Null Constraint** on a **foreign key** in T_1, ref. T_2	Existential or cardinality restriction over P_f in C_T	C_{T_1} SubClassOf: P_f min 1 C_{T_2}	-	-	-
			C_{T_1} SubClassOf: P_f some C_{T_2}	✓	-	✓
(12)	**Check Constraint** on data attribute a of type t in table T **listing posible values** $v_1 \dots v_n$ (n ≥ 1)	Enumeration of literals: in a restriction in class C_T and/or as a range of Ra. Alternatively, one could create subclasses for each of the values	C_T SubClassOf: R_a some $\{v_1 \dots\}$	-	-	✓*
			C_T SubClassOf: R_a only $\{v_1 \dots\}$	-	-	-
			C_T SubClassOf: R_a value v_1	-	-	✓
			R_a Range: $\{v_1 \dots\}$	-	-	✓*
			C_{Tv_i} SubClassOf: C_T	✓	✓	✓
(13)	**Check Constraint** on attrib. a in table T **restricting numerical range** of t	Datatype restriction: in a class restriction in C_T and/or as a range of Ra	C_T SubClassOf: R_a some $d_t[> x]$	-	-	-
			C_T SubClassOf: R_a only $d_t[> x]$	-	-	-
			R_a Range: $d_t[> x]$	-	-	-
(14)	**Several data attributes** $a_1 \dots a_n$ **in different tables** $T_1 \dots T_n$ **with the same name and type** t	Group properties $R_1 \dots R_n$ under new superproperty R_a or merge $R_1 \dots R_n$ into new property R_a	R_i SubPropertyOf: R_a	✓	✓	✓
			R_a Domain: C_{T_1} *or* $\dots$ *or* C_{T_n}	-	-	-
			C_{T_i} SubClassOf: R_a some d_t	✓	-	✓
			C_{T_i} SubClassOf: R_a only d_t	-	✓	-
(15)	T_1 **and** T_2 **not involved in a inheritance relationship**	Class C_{T_1} is disjoint with class C_{T_2}	C_{T_1} DisjointWith: C_{T_2}	✓	✓	✓

Profiling the Ontology. BOOTOX takes into account the concrete language (i.e., one of OWL 2 profiles) that is generated from the relational database features, which will enable the use of different Semantic Web technologies (e.g., query answering engines). As mentioned above, database features can be encoded using different axioms which may lead to different OWL 2 profiles. For example, primary keys and unique constraints (patterns (7) and (8) in Table 1) can be modelled with the OWL 2 construct *HasKey* (e.g., '$Well$ HasKey: id'), which is supported in OWL 2 RL and OWL 2 EL, but must be avoided if the target ontology language is OWL 2 QL. Alternatively (or in addition) one could use global and local cardinality restrictions (e.g., 'id Characteristics: Functional' and '$Well$ SubClassOf: id exactly 1 $xsd{:}int$', respectively), which leads to similar issues since not all profiles support them. For example, none of the OWL 2 profiles support exact cardinalities. If a profile does not support a type of axiom, an approximation is used instead (e.g., '$Well$ SubClassOf: id max 1 $xsd{:}int$' that is supported in OWL 2 RL, or '$Well$ SubClassOf: id some $xsd{:}int$' that is allowed in both OWL 2 QL and EL profiles). However, a representative approximation or alternative may not be always available for a database feature, as for the *datatype restrictions* (pattern (13) in Table 1), which is not supported by any of the profiles. BOOTOX, when the required ontology output is one of the OWL 2 profiles, keeps this knowledge as OWL 2 annotation axioms (e.g., '$depth$ Annotations: $ann{:}max\_value$ "$12,500$"'). These axioms have no logical impact on the ontology, but they can potentially be used in user interfaces to guide the formulation of queries (e.g., [33]).

Selection of Ontology Axioms. BOOTOX, in the default setting, encodes the database features with all axioms suitable for a required OWL 2 profile. The user can optionally select the kind of preferred axioms (i.e., local or global restrictions). Additionally, the suggested OWL 2 axioms may require an extra validation step to accurately represent a database feature. This is the case of pattern (6) in Table 1 where the generated object property can be declared with different characteristics which may not necessarily hold in the database. For example, in our running example from Figure 1, the foreign key *intersects_field* in table *Field* represents a self-reference to the same table. In this case, the foreign key can be encoded as a *reflexive* (a field intersects with itself) and *symmetric* object property; transitivity, however, does not necessarily apply for this property.

Mapping SQL Datatypes to OWL 2. There is a clear mapping between SQL datatypes to XML schema datatypes. However, there are some XML schema datatypes that are not built-in datatypes in OWL 2. For example, $xsd{:}date$ is not supported in OWL 2. Furthermore, OWL 2 profiles also include specific restrictions on the supported datatypes. For example, $xsd{:}boolean$ or $xsd{:}double$ are not supported in OWL 2 QL nor OWL 2 EL. BOOTOX addresses this issue by mapping SQL datatypes, whose XML counterparts are not supported in OWL 2 (resp., OWL 2 profile), to the OWL 2 built-in datatype $rdfs{:}Literal$. This datatype denotes the union of all datatypes (i.e., top datatype). Note that a more intuitive approximation of unsupported datatypes (e.g., $xsd{:}double$ to $xsd{:}decimal$) is not possible since the value spaces of primitive datatypes are disjoint.[14]

4.2 Mapping Generation

Generation of Direct Mappings. BOOTOX follows the W3C direct mapping guidelines to generate mappings from a relational database to an ontological vocabulary. BOOTOX relies on the R2RML language to produce direct mappings, which are particular cases

[14] `http://www.w3.org/TR/swbp-xsch-datatypes/`

of the mappings that can be expressed in R2RML. Intuitively, an R2RML mapping allows to map any valid SQL query or view (i.e., logical table) into a target vocabulary.

The W3C direct mapping specification does not introduce specific restrictions on the used target ontological vocabulary, that is, it only requires to reference the ontological vocabulary via its URI. Hence, the generated mappings may reference the vocabulary of any given ontology. BOOTOX allows this behaviour and it can produce mappings for a vocabulary generated from the relational database (see Section 4.1) or for the vocabulary of an external domain ontology.

Generation of Mappings beyond Direct Ones. Besides direct mappings described above, that involve only projection operator in the SQL part, BOOTOX can help user to build *complex* R2RML mappings, that may also include selection and/or join operators. There are three general approaches that we use to find candidate mappings:

(i) In the direct mapping approach, tables are mapped to classes. To generalise this, consider a table T and a table T' that has a foreign key constraint referencing T or that can be joined. By taking the left join of T and T' we obtain a subset of the tuples in T. If this subset of tuples is sufficiently big and sufficiently different from T itself, we obtain a subclass of the class T was mapped to. Furthermore, by letting this process to continue recursively, we obtain rich class hierarchies. Here, as in direct mappings, the many-to-many exception applies.

(ii) Another way to discover subclasses in the database is to look for clusters of tuples in a table. To this end, we can consider a tuple to be a vector of numerical values (assigning distinct numbers to each value), and look for sets of tuples whose distance from each other is sufficiently small.

(iii) Finally, one can also consider subsets of the attributes of a table, particularly when tables are not known to in standard normal forms, such as BCNF. In such tables, there will usually be tuples with repeating values in attributes. Such sets of tuples can, again, be considered subclasses of the class the table itself was mapped to.

Note that the generation of these complex mappings heavily relies on interaction with the users since these techniques allow to generate only the SQL queries for mappings, while they do not offer the names of classes and properties defined by these queries; the names should be introduced by the users.

Generation of Provenance Mappings. BOOTOX automatically extends direct mappings with metainformation about provenance.[15] The provenance metainformation is modelled in the mapping assertion, adding, for instance, the source database from which the information is extracted, and more granular information, like table and column identifiers. Currently, provenance in BOOTOX comes into three different granularity levels, whose convenience will depend on the intended use of this information:

(i) *URI level*: each generated URI is associated with provenance information. This is especially interesting when different database elements are merged into the same URI (e.g., see pattern (14) in Table 1).

(ii) *Triple level*: triples are annotated with provenance via RDF reification (i.e., three new triples are added for each triple).

(iii) *Graph level*: in this granularity, provenance is attached to RDF named graphs that group triples with similar provenance characteristics. For example, one could group triples generated by automatically generated mappings or triples derived from the same data source.

[15] Based on the W3C recommendation PROV-O `http://www.w3.org/TR/prov-o/`.

This metainformation can be later used for a wide range of purposes. For example it can be used to provide a richer query answering interface, to help in the debugging of a data access instance (i.e., identifying faulty ontology axioms or mappings), or to discard some sources based on a variety of criteria, e.g., licenses, access privileges, costs, etc.

4.3 Importing

It is increasingly often the case that a high quality ontology of (parts of) the domain already exists and captures the domain experts vocabulary better than the directly mapped ontology. When such an ontology is available, BOOTOX allows importing it to extend the automatically generated ontology.[16] To this end, BOOTOX integrates the ontology alignment system LogMap [14], that aligns two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ by deriving OWL 2 equivalence and sub-class(property) axioms between the entities from $\mathcal{O}_1$'s and $\mathcal{O}_2$'s vocabularies using the lexical characteristics of the terms and the structure of the ontologies. BOOTOX gives special care to avoid introducing unwanted consequences that may lead to unexpected answers. Thus, LogMap will discard alignment axioms that would lead to inconsistencies, or faulty consequences like '$Well$ SubClassOf: $WellBore$'. This is based on novel techniques to avoid violations of the so called consistency and conservativity principles (the interested reader please refer to [32]).

4.4 Conformance with Problem Requirements

BOOTOX covers the two bootstrapping tasks defined in Section 3 and fully meets the Metrics (1) and (2) regarding the ontology and mapping language.

Furthermore, as shown in Section 5, it provides reasonable results in practice with respect to Metrics (3) and (4). *Query coverage*, i.e., Metric (3), is enhanced thanks to the importing facility. Regarding *query results*, i.e., Metric (4), note that we supply the system with *provenance* capabilities, which are useful for analysis. In the case when query results contain unexpected answers, provenance will help the user to "trace" the source of these answers, thus helping to gain a better understanding of possible issues with the ontology or mappings.

5 BOOTOX at Work

BOOTOX has been tested with a number of databases and scenarios (e.g., virtual and materialised data access). In this section we present results with respect to the quality of the bootstrapped assets (i.e., vocabulary, ontology, and mappings). We have evaluated *(i)* the ability of formulating queries with the bootstrapped vocabulary in an industrial scenario (Section 5.1), and *(ii)* the ability of (enabling the) answering of queries with the bootstrapped ontology and mappings in a controlled scenario (Section 5.2).

We compared BOOTOX[17] to three other bootstrapping systems: IncMap [24], MIRROR [21], and -ontop- [27]. IncMap is designed to directly map a relational database to a target ontology, but is focused on a semi-automatic, incremental/interactive approach rather than direct automated bootstrappings. Both -ontop- and MIRROR follow an approach that is similar to the one employed in BOOTOX, with respect to the generation of a semantic data access instance (i.e., vocabulary, ontology, and mappings).[18]

[16] If the imported ontology is outside the desired target profile it should first be approximated using off-the-shelf semantic or syntactic approximation techniques (e.g., [9]).

[17] Note that we have evaluated BOOTOX with its automatic setting, that is, with the functionality to generate complex mappings turned off.

[18] -ontop- generates only vocabulary, that is, an ontology containing only declaration axioms.

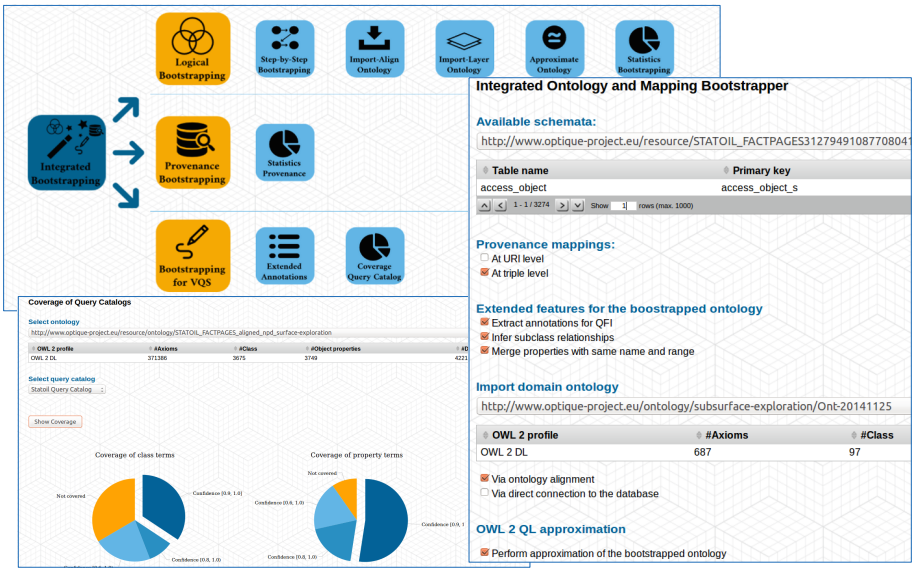


Fig. 2: BOOTOX interfaces in the Optique platform, from left to right, top to bottom: the main BOOTOX menu, the bootstrapping form, and the query catalog coverage

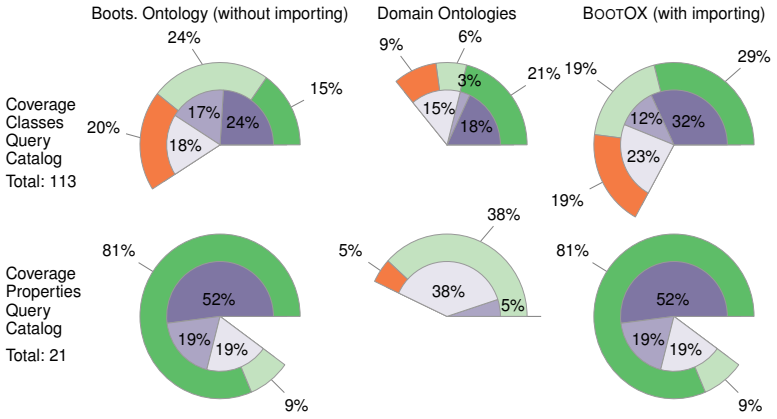
5.1 Query Coverage in the EU Optique Project

In this section we assess the quality of the bootstrapped ontology vocabulary to enable the formulation of queries. To this end we used the terms from the query catalogs available in the industrial scenario provided by the EU Optique project. We looked at how many terms from the catalogs were *covered* by the bootstrapped ontological vocabulary and then did manual verification of the quality of the coverage.

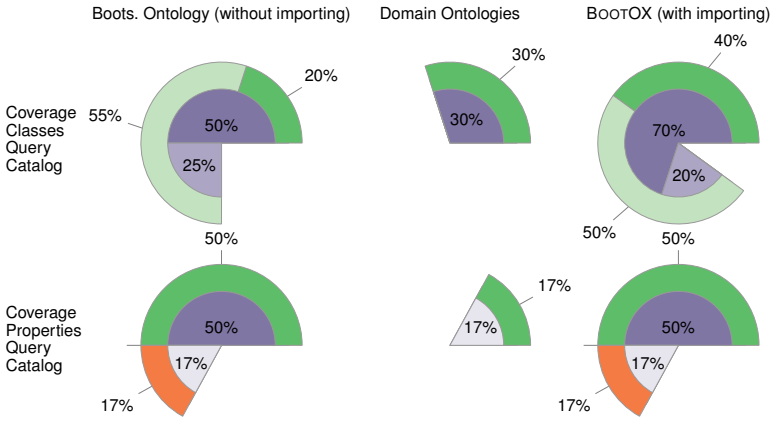
Industrial Scenario. The EU Optique project aims at facilitating scalable end-user access to big data in the oil and gas industry [12]. Optique advocates for an OBDA approach where the ontology provides a virtual access to the data and the mappings connect the ontology with the data source. The project is focused around two demanding use cases provided by Optique industry partners Siemens and Statoil. BOOTOX has been deployed in Siemens [17] and Statoil [15] as part of the "Ontology and mapping management module" provided by Optique's platform [16]. The Optique scenario requires BOOTOX to bootstrap an OWL 2 QL as it is required by the query rewriting engine in order to rewrite the queries formulated over the ontology into queries on the database using reasoning [6].

BOOTOX *in the Optique Platform.* Figure 2 shows an overview of the BOOTOX related interfaces. The main menu presents to the user the available options currently supported in BOOTOX: automatic bootstrapper, guided bootstrapper, ontology alignment, provenance bootstrapping, bootstrapping related statistics, etc. The screenshot on the right shows the integrated bootstrapping form in BOOTOX where the user can select from a number of options including the database schema, imported ontology, provenance, OWL 2 QL approximation. The bottom screenshot shows the coverage of a selected query catalog by the vocabulary of a bootstrapped ontology.

Coverage of Statoil Query Catalog. The query catalog at Statoil currently includes 60 queries that contain representative information needs from Statoil geologists. Most of



(a) Statoil use case.



(b) Siemens use case.

Fig. 3: Coverage of terms from the Optique query catalog with terms from ontologies. Inner pie charts show coverage by lexical confidence using I-SUB [36]: ■ in $[0.9, 1.0]$, ■ in $[0.8, 0.9)$, □ in $[0.6, 0.8)$. Outer pie charts represent the manual verification of the quality of the terms with a coverage above 0.6: ■ *true positive*, ■ *semi-true positive*, ■ *false positive*. Note that *semi-true positives* are not clear-cut cases where the ontology term has a broader or narrower meaning with respect to the query term

the data needed by Statoil geologists is stored in the Exploration and Production Data Store (EPDS), Statoil's corporate data store for exploration and production data and interpretations. The *NPD FactPages* [31] ontology is relevant to the domain of EPDS.

We bootstrapped ontological vocabulary from the relevant parts of EPDS using BOOTOX, MIRROR and -ontop-. The generated vocabulary, on average, contained more than 3,000 classes, 3,500 object properties, and 42,000 datatype properties. Note that IncMap relies on the vocabulary of the available domain ontology. BOOTOX, unlike -ontop- and MIRROR, includes a built-in ontology alignment system which allows to import the vocabulary of the domain ontologies into the bootstrapped ontology.

The results of the query catalog coverage are summarised in Figure 3(a). The first column represent the coverage of the bootstrapped ontologies computed by BOOTOX (without importing the domain ontologies), -ontop- and MIRROR. Since all three sys-

tems rely on the direct mapping directives, the bootstrapped vocabulary is, apart from minor differences, basically the same. The middle columns represent the coverage of the vocabulary of the domain ontology, which is equal to the coverage of IncMap. The third columns shows the coverage results achieved by the ontology bootstrapped by BOOTOX including importing. For example, 44% of the classes in the query catalog has a good lexical intersection (greater or equal 0.8) with terms of the ontology bootstrapped by BOOTOX; furthermore, 29% of the classes are fully covered (i.e., true positives).

Coverage of Siemens Query Catalog. The data in the Siemens use-case is stored in several databases with different schemata. Although the schemata are not specially large, the size of the data is in the order of hundreds of terabytes, e.g., there is about 15 GB of data associated to a single turbine, and it currently grows with the average rate of 30 GB per day [17]. As for the Statoil use case, we extracted the relevant terms of the query catalog, bootstrapped an ontology from one of the Siemens databases using BOOTOX, MIRROR and -ontop-. Additionally, BOOTOX performed alignment with two Siemens ontologies about diagnostic procedures and turbines. Finally, IncMap relied on the vocabulary of these ontologies. The results of the coverage are summarised in Figure 3(b).

Quality Assessment. The experiments show that the bootstrapped ontologies without importing had a higher coverage than the domain ontologies in isolation, e.g., 39% of query class coverage against 27% in the Statoil use case. These results suggest that there is an adequate number of table and column names with potentially adequate semantic relations with the terms that domain experts at Statoil and Siemens have in mind when they access data, and thus, the ontology vocabulary computed by the ontology bootstarppers is indeed relevant to query formulation. Nevertheless, the domain ontologies naturally complement the vocabulary obtained from the database and hence BOOTOX is able to bootstrap and ontology with better coverage over the query catalog than the ones generated by -ontop- and MIRROR. For example, 48% of the classes (resp., 90%) in the Statoil (resp., Siemens) catalog are fully or partially covered in the bootstrapped ontology computed by BOOTOX.

5.2 Query Answering in a Controlled Scenario

In this section we assess the quality of the bootstrapped ontology and mappings to enable the answering of queries in a controlled scenario.[19] To this end we ran experiments with a recently released relational-to-ontology benchmark suite, RODI [25], comparing BOOTOX to IncMap, -ontop- and MIRROR. RODI is designed to test relational-to-ontology mappings end-to-end: it provides an input database and a target ontology and requests complete mappings or mapped data to query. RODI is based on *scenarios*, with each scenario comprising several query tests. While RODI is extensible and can run scenarios in different application domains, it ships with a set of *default scenarios* in the conference domain that are designed to test a wide range of fundamental relational-to-ontology mapping challenges in a controlled fashion. The effectiveness of mappings is then judged by a score that mainly represents the number of query tests that return expected results on mapped data.

IncMap is designed to automatically map the target ontology directly to the input database, while BOOTOX approached this task in two steps: first, it bootstrapped

[19] Note that assessing the quality of the bootstrapped ontology and mapping in an open scenario like Optique requires a huge involvement of domain experts, thus we leave it in Optique for future work.

Table 2: RODI results of all tests per scenario. Values in cells are the success scores

Scenario	IncMap	MIRROR	-ontop-	BOOTOX
	Adjusted naming			
CMT	**0.5**	0.28	0.39	0.39
CONFERENCE	0.26	0.27	**0.37**	**0.37**
SIGKDD	0.21	0.3	**0.45**	**0.45**
	Cleaned hierarchies			
CMT	**0.44**	0.17	0.28	0.28
CONFERENCE	0.16	0.23	**0.3**	**0.3**
SIGKDD	0.11	0.11	**0.16**	**0.16**
	Combined case			
SIGKDD	0.05	0.11	**0.16**	**0.16**
	Missing FKs			
CONFERENCE	0.03	**0.17**	-	**0.17**
	Denormalised			
CMT	0.22	0.22	**0.28**	**0.28**

an intermediate ontology and mappings from the database. Then, it aligned this intermediate, bootstrapped ontology to the target ontology as provided by the benchmark. As mentioned in Section 5.1, neither -ontop- nor MIRROR include a built-in ontology alignment system to support the importing of the target ontology provided by the benchmark. In order to be able to evaluate these systems with RODI, we aligned the generated ontologies by -ontop- and MIRROR with the target ontology using the LogMap system in a similar setup to the one used in BOOTOX.

Scenarios. RODI default scenarios are a selection of benchmark scenarios set in the conference domain, based on three different conference ontologies: CMT, CONFERENCE, and SIGKDD[20]. For each ontology, the benchmark provides a set of database instances that are to be mapped to the ontology vocabulary. While all databases are modelled to contain the same data that the ontologies might contain, they deviate largely in how close they are to their corresponding ontologies in terms of structure and modelling patterns. For each ontology, there is a number of mapping challenges that can be tested: *(i) Adjusted naming*. Here the identifier names in the databases are syntactically changed in comparison with the names in the ontology. *(ii) Cleaned hierarchies*. In this scenarios, the databases are remodelled from ground to follow widely used relational database design patterns. Most significantly, this includes cases where abstract parent classes have no corresponding table in the database, several sibling classes are jointly represented in a single table, etc. *(iii) Combined case* mixes changes made in scenarios with adjusted naming and cleaned hierarchies. *(iv) Missing FKs* represents the case of a database with no explicit referential constraints at all. *(v)* In the *denormalised* case, the database contains a few denormalised tables.

Results.[21] Results show that the BOOTOX comes out in top position for seven out of nine tested scenarios (see Table 2). What is more, in most of those cases where BOOTOX does not rank first, it is not lagging far behind the top contender. This shows that BOOTOX is well suited for meeting the requirements of an end-to-end scenario. On a more generic note, however, results also demonstrate that none of the tested systems to date manages to solve relational-to-ontology mapping challenges with a score above

[20] `http://oaei.ontologymatching.org/2015/conference/`

[21] Note that RODI only provides the percentage of correctly answered queries. RODI is being extended to include a more fine grained evaluation in terms of Precision and Recall in order to take into account partially answered queries.

0.5. This confirms the need for specialised relational-to-ontology bootstrapping systems such as BOOTOX, which build the foundation for better solutions.

Provenance. When we evaluated BOOTOX with RODI without the use of provenance mappings, the results were slightly worse than the ones in Table 2: in three scenarios we detected unexpected answers. Then we made use of the provenance functionality of BOOTOX to analyse the source of these unexpected answers. As an outcome of such an analysis we identified and fixed the faulty mappings, hence improving the results for those scenarios.

6 Related Work

The implementation of BOOTOX has been motivated by the fact that existing ontology and mapping bootstrappers provide limited or no support for the bootstrapping tasks and quality requirements described in Section 3 (see, e.g., [30, 34]). Most of the state of the art bootstrappers fail to conform with the ontology and mapping language standards, or they do not provide profiling capabilities for the outpout ontology. Moreover, to the best of our knowledge they do not provide bootstrapping of complex mappings. For historical reasons (i.e., OWL was not yet defined), former systems used RDFS and F-Logic axioms (e.g., [3, 37]). Other systems have also used DLR-Lite based languages (e.g., [20]) and extensions based on SWRL (e.g., [11, 19]). Regarding mapping generation, before R2RML became a W3C recommendation, system typically relied on their own native language to define mappings (e.g., D2RQ [5], Mastro [8]). To the best of our knowledge, currently only IncMap [24], MIRROR [21], -ontop- [27], and Ultrawrap [28, 29] produce mappings in the R2RML language.

Among the systems using OWL or OWL 2 as the ontology language, only BOOTOX put special attention to the target ontology expressiveness. BOOTOX allows to output different ontology axioms to conform to the required OWL 2 profile as discussed in Section 4.1. Many bootstrapping systems typically use exact or min cardinality restrictions which fall outside the three OWL 2 profiles (e.g., [1, 39]). Furthermore, other systems, like [4], produce an ontology that falls into OWL 2 Full due to the use of the *Inverse-Functional* characteristic in both data and obtect properties. Finally, MIRROR, -ontop-, and Ultrawrap are conformant to the OWL 2 QL, but they do not support profiling to the other sublanguages of OWL 2.

As BOOTOX, systems like Automapper [11], Relational.OWL [18] and ROSEX [10] complement the automatically generated ontology with links to domain ontologies. However, none of these systems apply logic-based techniques to assess the consequences of such links to domain ontologies.

Special mention require the approaches in [7, 31]. These approaches use (semi-automatic) ontology learning techniques to exploit the data and discover interesting patterns that can be included to enrich the ontology. Currenly, BOOTOX only relies on a (fully-automatic) schema-driven generation of the ontology and mappings.

In the literature one can also find several appoaches to overcome the OWA problem when dealing with data-centric applications (e.g., [22, 38]). These approaches typically extend the semantics of OWL 2. The integration of these approaches in a bootstrapping scenario is, however, still an open problem.

7 Conclusions and Future Work

We have presented BOOTOX, an automatic ontology and mapping bootstrapper. To the best of our knowledge, BOOTOX is the only bootstrapper that *(i)* profiles the concrete

language of the output OWL ontology, *(ii)* puts special attention to the datatype conversion, *(iii)* enhances the direct mappings with provenance metainformation, *(iv)* bootstraps a range of complex mappings, and *(v)* includes a bult-in logic-based ontology alignment system. Furthermore we have tested BOOTOX in a number of databases and test cases involving materialised or virtual data access. The evaluation suggests that automatic techniques to bootstrap an initial ontology and mappings are indeed suitable to be used in practice.

We see bootstrapping as the first step towards the creation of a fully-fledged semantic data access system. Bootstrapped assets are by no means perfect, and thus they should be post-processed, validated, and extended. Our evaluation in both industrial (i.e., Optique) and research contexts has also served to guide the extension of BOOTOX with semi-automatic techniques. For example, while working with Statoil's EPDS database, we found that the discovery of implicit constraints represents a critical feature since EPDS has very few constraints, e.g., many tables are materialised views without specified primary or foreign keys. In the close future we aim at extending our evaluation to understand the limits of semi-automatic ontology and mapping bootstrappers to enable the answering of the queries in open scenarios like Optique.

Acknowledgements. This work was partially funded by the EU project Optique (FP7-ICT-318338), the Research Council of Norway through the project DOIL (RCN project n. 213115), and the EPSRC projects MaSI^3, Score!, and DBOnto.

8 References

[1] N. Alalwan, H. Zedan, et al. Generating OWL Ontology for Database Integration. In: *SEMAPRO*. 2009.
[2] M. Arenas, B. Cuenca Grau, et al. Faceted Search over Ontology-Enhanced RDF Data. In: *CIKM*. 2014.
[3] I. Astrova. Reverse Engineering of Relational Databases to Ontologies. In: *ESWS*. 2004.
[4] I. Astrova. Rules for Mapping SQL Relational Databases to OWL Ontologies. In: *MTSR*. 2007.
[5] C. Bizer and A. Seaborne. D2RQ-Treating non-RDF Databases as Virtual RDF Graphs. In: *ISWC posters*. 2004.
[6] D. Calvanese, G. De Giacomo, et al. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. English. In: *JAR* 39.3 (2007).
[7] F. Cerbah and N. Lammari. Perspectives in Ontology Learning. In: AKA / IOS Press. Serie, 2012. Chap. Ontology Learning from Databases: Some Efficient Methods to Discover Semantic Patterns in Data.
[8] C. Civili, M. Console, et al. MASTRO STUDIO: Managing Ontology-Based Data Access Applications. In: *PVLDB* 6.12 (2013).
[9] M. Console, J. Mora, et al. Effective Computation of Maximal Sound Approximations of Description Logic Ontologies. In: *ISWC*. 2014.
[10] C. Curino, G. Orsi, et al. Accessing and Documenting Relational Databases through OWL Ontologies. In: *FQAS*. 2009.
[11] M. Fisher, M. Dean, et al. Use of OWL and SWRL for Semantic Relational Database Translation. In: *OWLED*. 2008.
[12] M. Giese, A. Soylu, et al. Optique — Zooming In on Big Data Access. In: *Computer* 48.3 (2015).
[13] M. Horridge, N. Drummond, et al. The Manchester OWL Syntax. In: *OWLED*. 2006.
[14] E. Jiménez-Ruiz, B. Cuenca Grau, et al. Large-Scale Interactive Ontology Matching: Algorithms and Implementation. In: *ECAI*. 2012.

[15] E. Kharlamov, D. Hovland, et al. Ontology Based Access to Exploration Data at Statoil. In: *ISWC*. 2015.
[16] E. Kharlamov, E. Jiménez-Ruiz, et al. Optique: Towards OBDA Systems for Industry. In: *Extended Semantic Web Conference (ESWC Selected papers)*. 2013.
[17] E. Kharlamov, N. Solomakhina, et al. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In: *ISWC*. 2014.
[18] C. P. de Laborda and S. Conrad. Database to Semantic Web Mapping Using RDF Query Languages. In: *ER*. 2006.
[19] D. V. Levshin. Mapping Relational Databases to the Semantic Web with Original Meaning. In: *Int. J. Software and Informatics* 4.1 (2010).
[20] L. Lubyte and S. Tessaris. Automatic Extraction of Ontologies Wrapping Relational Data Sources. In: *DEXA*. 2009.
[21] L. F. de Medeiros, F. Priyatna, et al. MIRROR: Automatic R2RML Mapping Generation from Relational Databases. In: *ICWE*. 2015.
[22] B. Motik, I. Horrocks, et al. Bridging the Gap between OWL and Relational Databases. In: *WWW*. 2007.
[23] B. Motik, Y. Nenov, et al. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In: *AAAI*. 2014.
[24] C. Pinkel, C. Binnig, et al. IncMap: Pay as You Go Matching of Relational Schemata to OWL Ontologies. In: *OM*. 2013.
[25] C. Pinkel, C. Binnig, et al. RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration. In: *ESWC*. 2015.
[26] A. Poggi, D. Lembo, et al. Linking Data to Ontologies. In: *J. Data Semantics X*. 2008.
[27] M. Rodriguez-Muro and M. Rezk. Efficient SPARQL-to-SQL with R2RML Mappings. In: *To appear, J. Web Sem.* (2015).
[28] J. Sequeda, M. Arenas, et al. On Directly Mapping Relational Databases to RDF and OWL. In: *WWW*. 2012.
[29] J. Sequeda and D. P. Miranker. Ultrawrap: SPARQL Execution on Relational Data. In: *J. Web Sem.* 22 (2013).
[30] J. Sequeda, S. H. Tirmizi, et al. Survey of Directly Mapping SQL Databases to the Semantic Web. In: *Knowledge Eng. Review* 26.4 (2011).
[31] M. G. Skjæveland, M. Giese, et al. Engineering Ontology-Based Access to Real-World Data Sources. In: *To appear in J. Web Sem.* (2015).
[32] A. Solimando, E. Jiménez-Ruiz, et al. Detecting and Correcting Conservativity Principle Violations in Ontology-to-Ontology Mappings. In: *ISWC*. 2014.
[33] A. Soylu, M. Giese, et al. Experiencing OptiqueVQS: A Multi-Paradigm and Ontology-Based Visual Query System for End Users. In: *Univ. Access in the Inform. Society* (2015).
[34] D.-E. Spanos, P. Stavrou, et al. Bringing Relational Databases into the Semantic Web: A Survey. In: *Semantic Web* 3.2 (2012).
[35] G. Stefanoni and B. Motik. Answering Conjunctive Queries over EL Knowledge Bases with Transitive and Reflexive Roles. In: *AAAI*. 2015.
[36] G. Stoilos, G. B. Stamou, et al. A String Metric for Ontology Alignment. In: *ISWC*. 2005.
[37] L. Stojanovic, N. Stojanovic, et al. Migrating Data-Intensive Web Sites into the Semantic Web. In: *SAC*. 2002.
[38] J. Tao, E. Sirin, et al. Integrity Constraints in OWL. In: *AAAI*. 2010.
[39] S. H. Tirmizi, J. Sequeda, et al. Translating SQL Applications to the Semantic Web. In: *DEXA*. 2008.
[40] Y. Zhou, Y. Nenov, et al. Pay-As-You-Go OWL Query Answering Using a Triple Store. In: *AAAI*. 2014.