B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017

IncMap: A Journey towards Ontology-based Data Integration

Christoph Pinkel[1] Carsten Binnig[2] Ernesto Jimenez-Ruiz[3] Evgeny Kharlamov[3]
Andriy Nikolov[1] Andreas Schwarte[1] Christian Heupel[1] Tim Kraska [2]

Abstract: Ontology-based data integration (OBDI) allows users to federate over heterogeneous data sources using a semantic rich conceptual data model. An important challenge in ODBI is the curation of mappings between the data sources and the global ontology. In the last years, we have built *IncMap*, a system to semi-automatically create mappings between relational data sources and a global ontology. *IncMap* has since been put into practice, both for academic and in industrial applications. Based on the experience of the last years, we have extended the original version of *IncMap* in several dimensions to enhance the mapping quality: (1) *IncMap* can detect and leverage semantic-rich patterns in the relational data sources such as inheritance for the mapping creation. (2) *IncMap* is able to leverage reasoning rules in the ontology to overcome structural differences from the relational data sources. (3) *IncMap* now includes a fully automatic mode that is often necessary to bootstrap mappings for a new data source. Our experimental evaluation shows that the new version of *IncMap* outperforms its previous version as well as other state-of-the-art systems.

1 Introduction

As large volumes of data are being constantly created in a variety of domains, the challenge of data integration becomes more and more important to get a holistic view on existing knowledge. Example use cases include web data analysis, the reconciliation of enterprise internal data, as well as applications in science or medicine. Integrating such data into a common model allows correlating information and hence discovering new and interesting patterns [Bh15, DS13].

One recent approach to this problem is ontology-based data integration (OBDI), where data sources are integrated via a global ontology. The advantage of using an ontology as a global view is its semantic richness, allowing domain experts to model their information needs in a conceptual high-level model. However, data in many of today's applications is still commonly stored in relational databases. To achieve an integration of these data sources, the relational database schemata of the sources have to be mapped into the unified global ontology (called *RDB2RDF* mappings further on).

Creating such *RDB2RDF* mappings manually is a time consuming task, which requires considerable effort and expertise. To minimize the required costs, several different systems (e.g., [Ji15, Pi13, Kn12, TSM13, Au05]) have recently been proposed that assist users in

[1] fluid Operations AG, Walldorf, Germany
[2] Brown University, Providence, RI, USA
[3] University of Oxford, Oxford, UK

creating these mappings in a semi-automated or even fully automated way. To provide such assistance, we have previously built semi-automatic system called *IncMap* [Pi13], which we have put to use in practice over the last few years in different application domains.

Contributions: The mapping generation in the first version of *IncMap* [Pi13] was mainly based on lexical as well as structural similarities between the schema elements of the relational data source and the target ontology. For example, if an entity *Author* has a relation to an entity *Paper* there should exist similar entities in both the ontology and the relational schema as well as a path that links both entities.

However, based on the use of *IncMap* over the last years we have seen that *IncMap* was not able to find more complex mappings in many real-world scenarios which had two main reasons: First, ontologies follow an open-world approach (i.e., not everything must be explicitly modeled if it can be derived by reasoning) whereas relational schemata follow a closed-world approach. Second, many of the high-level concepts such as inheritance can not be directly modeled in a relational schema and are often implemented using different modeling patterns.

Based on this experience, we have developed a new version of *IncMap* that tackles these problems. The main features of the new version that provided most benefits are: (1) As a first major extension, *IncMap* is now able to leverage knowledge derived from reasoning over the input ontology, and (2) at the same time utilizes information about typical design patterns in relational databases. To the best of our knowledge, *IncMap* is the only direct system that combines these two approaches into a mapping system. Both these extensions have shown to be extremely fruitful and are the main reason why *IncMap* currently outperforms other state-of-the-art systems. (3) In comparison with its predecessor, *IncMap* now also supports fully automatic mapping generation to bootstrap the mapping for larger data sources. Moreover, we have also added minor extensions such as a better lexical matching and a number of engineering improvements.

The contributions of this paper are three-fold: (1) We present all these new features of *IncMap* in detail. (2) In order to show the effectiveness of all our extensions, we compare *IncMap* against other state-of-the-art systems using our recent benchmark suite for benchmarking ODBI integration systems, called RODI [Pi15].[4] Our results show that *IncMap* improves the quality of the generated mappings significantly over its predecessor and typically outperforms other state-of-the-art systems. (3) In addition, we also present the results of a user-study. We have built the study on a small, real-world industry mapping problem to demonstrate the utility of *IncMap* in practice, independent from any synthetically designed benchmark.

Outline: The remainder of this paper is organized as follows. First, in Section 2 we start with an overview of our extended system *IncMap* that leverages the novel *IncGraph+* model to support the before-mentioned extensions. In Section 3, we discuss in detail

[4] `https://github.com/chrpin/rodi`

the construction of the *IncGraph*+ model and then elaborate on the role of reasoning in Section 3.2 and the use typical design patterns in Section 3.3. Afterwards, in Section 4 we explain how the this information is leveraged to generate mappings from the extended *IncGraph*+ model. Finally, we present our experimental evaluation in Section 5 and discuss the results of our user study in Section 6. We conclude with related work in Section 7 and a summary in Section 8.

2 *IncMap* Overview

IncMap generates *RDB2RDF* mappings between a given relational data source and a target ontology based on the following procedure that operates in five steps: (1) creating source and target *schema graphs* from the relational schema and the ontology, (2) use reasoning and heuristic pattern *annotation* to infer additional information, (3) apply initial lexical matching to build a *matching score* between relation schema and the ontology, (4) refine matching scores using a *fixpoint computation*, and (5) generate *RDB2RDF mappings*.

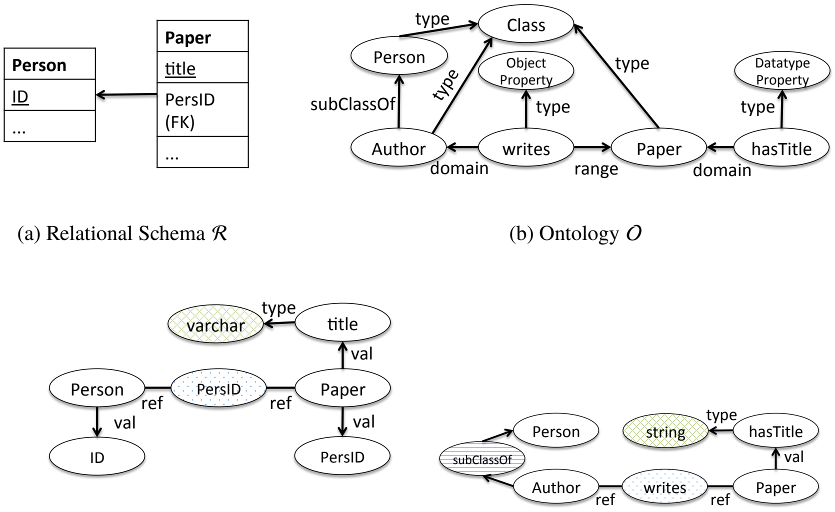
This procedure can be repeated after the user has verified or rejected individual mappings of the previous iteration. This allows *IncMap* to integrate user feedback and create mappings in an incremental manner.

In the following, we give an overview of all these steps. Details and a more formal explanation are given afterwards in Section 3 and Section 4.

1. Creating Schema Graphs: As a first step, the source and target schema graphs are created using our *IncGraph*+ model. In order to construct the source and target schema, we iterate over all schema elements in the relational schemata and the ontology and create the two graphs.

Figure 1 depicts a small example of the schema graph construction from the conference domain. The relational schema (Figure 1a) and the ontology (Figure 1b) both capture the same information about persons and papers. In a first step, the relational schema and the ontology are mapped into the source and target schema graph as shown in Figure 1c and Figure 1d that consists of typed nodes and labeled edges. A formal definition of the schema graph will be given in the following section.

While we can see already some clear correspondences between the elements in both schema graphs that result from lexical and structural similarities (e.g., *Paper*s and their *title*s), some other schema elements are harder to match. For example, while *Author* is a dedicated concept in the ontology it does not have a direct correspondence in the relational schema. The reason is that in the relational schema, *Author*s are modeled as *Person*s who have authored at least one paper. This results from a typical modeling pattern of how inheritance is implemented in a relational schema.



(a) Relational Schema $\mathcal{R}$ (b) Ontology O

(c) Basic *IncGraph+* ($\mathcal{R}$) (d) Basic *IncGraph+* (O)

Fig. 1: Example: schema and ontology, with basic *IncGraph+* representations.

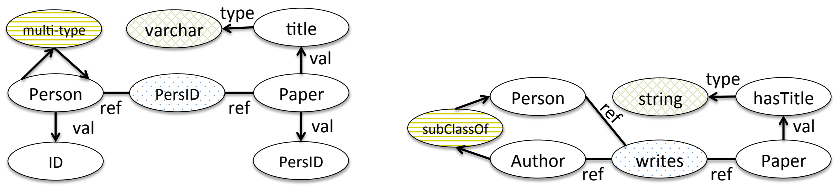
Moreover, there are many more subtle differences: e.g., while on the database side foreign keys (i.e., referential constraints) model a clear direction and allow to deduce cardinality constraints, object properties in an ontology are do not yield any cardinality information.

In order to simplify the subsequent matching step (3), we also annotate each node in a schema graph with a type indicated by the color coding in both figures. For example, the green-striped nodes in a schema graph represent the *data type* information. We explain the different types that we support in *IncMap* later. Important is that for the matching step (3), only nodes with the same color need to be considered as potential correspondences.

2. Reasoning and Patterns: As discussed before, there exist several possible distortions between the schema graph representations of a relational schemata and an ontology. These distortions have two main causes: First, ontologies typically follow an open-world approach (i.e., not everything must be explicitly modeled if it can be derived by reasoning) whereas relational schemata follow a closed-world approach. Second, many of the high-level concepts such as inheritance can not be directly modeled in a relational schema and are often implemented using different modeling patterns.

These problems also materialize in Figure 1. Intuitively, the *Person* class is the most accurate match for the *Person* table while the *Author* class has most probably no match candidate. This is because the *subClassOf* connection between *Author* and *Person* is not explicit anymore neither in the schema graph of the relational schema (since it was never modeled) nor the schema graph of the ontology.

For this reason, as a second step we apply reasoning techniques on the input ontology and use heuristics to annotate patterns on the source database. Figure 2 depicts derived knowledge from reasoning and annotated patterns for the example in Figure 1.



(a) Full *IncGraph+* ($\mathcal{R}$) (b) Full *IncGraph+* (O)

Fig. 2: Example (ctd.): advanced *IncGraph*+s.

In Figure 2a, the relational schema graph now annotates the *Person* node with a pattern, which *heuristically* states that this table is very likely to contain *individuals* of several types (e.g., using sub classes or sibling classes). This information can now be used to derive a new correspondences between *Author* node in the schema graph of the ontology and the *Person* node in the schema graph of the relational data source.

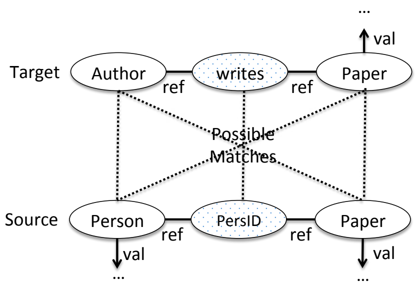
Additionally, a new reference edge is added to the ontology schema graph which directly connects *Person* and *Writes*. This knowledge is derived through reasoning and basically states that some persons in the ontology write papers. In our example this encourages correspondences between the *Person* nodes in both schema graphs since they are now not only lexically similar but also structurally.

3. Matching Step: Based on a source and target schema graph, we next calculate the initial matching. Figure 3 illustrates a simplified initial matching for the two schema graphs of the previous example.

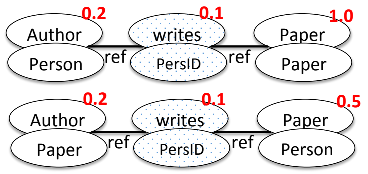
For creating the initial matching, each node in the source graph is paired with each node in the target graph that has the same color (or type) into possible matches. Figure 3a shows the possible pairs for an excerpt of the schema graphs in Figure 2. Afterwards, only those pairs in the set of possible matches are kept that have the same set of labeled edges (in the source and target graph).

Based on the remaining possible matches a so called pairwise connectivity graph (PCG) is created where each subgraph represents a possible alignment of nodes in the source and target schema graph. Figure 3b shows the PCG for our example where the upper node in each subgraph is always from the target schema graph (i.e., the ontology) and the lower node from the source schema graph (i.e., the relational schema). For each pair of nodes in the PCG an initial score is calculated using a lexical matcher (e.g., using Jaccard similarity).

4. Fixpoint Computation: In subsequent next steps, *IncMap* then computes a fixpoint computation using the PCG. The fixpoint computation serves to refine match scores based



(a) Source and Target *IncGraph*+s



(b) Pairwise Connectivity Graph

Fig. 3: Matching *IncGraph*+s (Simplified Excerpt)

on the graph structure. The fixpoint computation in *IncMap* follows the idea of the similarity flooding algorithm described in [MGMR02]. Intuitively, the process favors nodes in larger subgraphs over smaller ones and increases the scores of strongly connected nodes.

Different from the original similarity flooding algorithms, however, we introduce modifications for features such as weighted edges and selectively activating edges during the fixpoint computation (e.g., based on the annotated patterns). Moreover, in order to additionally support incremental mapping scenarios where user feedback is available, we can accommodate information on partial mappings. However, *IncMap* is also able to generate mappings completely without any user feedback.

5. Mapping Generation: Finally, mappings are generated from the correspondences resulting from the PCG. Correspondences are selected based on the highest matching scores of the fixpoint computation to form a consistent alignment interpretation; i.e., each node in the original target schema graph has maximally one correspondence in the source schema graph. To enable the mapping generation, we encode provenance information with all nodes of the PCG to refer back the nodes in the source and target schema graphs.

3 *IncGraph+*

In this section, we formally define the schema graphs that we create as a first step of *IncMap*. The data model used for a schema graph in our recent version of *IncMap* is called *IncGraph*+.

3.1 Basic *IncGraph+*

An *IncGraph*+ is defined as a labeled graph $\mathcal{G} = (\mathcal{V}, \mathtt{Lbl}_{\mathcal{V}}, \mathcal{E}, \mathtt{Lbl}_{\mathcal{E}}, \mathcal{W}_{\mathcal{E}})$. It can be used as input by the matching algorithm of *IncMap*. $\mathcal{V}$ is a set of vertices, $\mathcal{E} = \mathcal{E}_d \cup \mathcal{E}_u$ is a set of directed $\mathcal{E}_d$ and undirected $\mathcal{E}_u$ edges, $\mathtt{Lbl}_{\mathcal{V}}$ and $\mathtt{Lbl}_{\mathcal{E}}$ are the labeling relations for vertices (i.e., relating one label to each vertex) and edges (i.e., one label for each edge) respectively, and $\mathcal{W}_{\mathcal{E}} \subset E \times [-w_{max}; w_{max}]$ is a weight assignment relation for edges.[5] Label l_v is the label of $v \in \mathcal{V}$ if $(v, l_v) \in \mathtt{Lbl}_{\mathcal{V}}$, and represents a name of a schema element. Similarly $l_e \in \{\text{“}ref'', \text{“}value'', \text{“}type'', \text{“}pattern''\}$ is a label of edge $e \in \mathcal{E}$ if $(e, l_e) \in \mathtt{Lbl}_{\mathcal{E}}$, and describes the function of the edge.

3.1.1 Basic Graph Construction:

Let $\mathcal{R}$ be a relational schema, O an ontology.

Basic nodes (vertices) and edges for *IncGraph*+ are based on input schema elements, i.e., tables and attributes for *IncGraph*+ ($\mathcal{R}$), or classes and properties for *IncGraph*+ (O).

Relational Schemata (*IncGraph*+ ($\mathcal{R}$)): Let T the set of tables (relations) in the schema, A_t the set of attribute of table $t \in T$, $P \subset \{(t_1, a_1, t_2, a_2) | t_1, t_2 \in T, a_1 \in A_{t_1}, a_2 \in A_{t_2}\}$ the set of non-compound referential constraints between tables in $\mathcal{R}$. Then:

(Table Nodes) $t \in T \longrightarrow v_t \in \mathcal{V} \wedge v_t.type = \text{"}T\text{"} \wedge (v_t, name(t)) \in \mathtt{Lbl}_{\mathcal{V}}$

(Attribute Nodes) $a \in A_t \wedge t \in T \longrightarrow v_a \in \mathcal{V} \wedge v_a.type = \text{"}A\text{"}$
$\wedge (v_a, name(a)) \in \mathtt{Lbl}_{\mathcal{V}} \wedge e_a = (v_t, v_a) \in \mathcal{E}_d \wedge (e_a, \text{"}val\text{"}) \in \mathtt{Lbl}_{\mathcal{E}}$

(Datatype Nodes) $v_a \in \mathcal{V} \wedge v_a.type = \text{"}A\text{"} \longrightarrow v_{dt} \in \mathcal{V} \wedge v_{dt}.type = \text{"}D\text{"}$
$\wedge (v_{dt}, name(dt(v_a))) \in \mathtt{Lbl}_{\mathcal{V}} \wedge e_{dt} = (v_a, v_{dt}) \in \mathcal{E}_d \wedge (e_{dt}, \text{"}type\text{"}) \in \mathtt{Lbl}_{\mathcal{E}}$

(Reference Nodes) $(t_1, a_1, t_2, a_2) \in P \longrightarrow v_p \in \mathcal{V} v_p.type = \text{"}P\text{"}$
$\wedge (v_p, name(a_1)) \in \mathtt{Lbl}_{\mathcal{V}} \wedge e_{p_1} = (v_{t_1}, v_p) \in \mathcal{E}_u \wedge (e_{p_1}, \text{"}ref\text{"}) \in \mathtt{Lbl}_{\mathcal{E}}$
$\wedge e_{p_2} = (v_{t_2}, v_p) \in \mathcal{E}_u \wedge (e_{p_2}, \text{"}ref\text{"}) \in \mathtt{Lbl}_{\mathcal{E}}$

Ontologies (*IncGraph*+ (O)): Let C, D_P, O_P the set of class axioms, datatype property axioms and object property axioms in O, respectively, and X be the set of OWL datatypes. Then:

(Class Nodes) $c \in C \longrightarrow v_c \in \mathcal{V} \wedge v_c.type = \text{"}T\text{"} \wedge (v_c, name(c)) \in \mathtt{Lbl}_{\mathcal{V}}$

(Datatype Property Nodes) $d \in D \wedge c = domain(d) \in D_P$
$\longrightarrow v_d \in \mathcal{V}, v_d.type = \text{"}A\text{"} \wedge (v_d, name(d)) \in \mathtt{Lbl}_{\mathcal{V}} \wedge e_d = (v_c, v_d) \in \mathcal{E}_d$
$\wedge (e_d, \text{"}val\text{"}) \in \mathtt{Lbl}_{\mathcal{E}}$

(Datatype Range Nodes) $v_d \in \mathcal{V} \wedge v_d.type = \text{"}A\text{"} \wedge r = range(v_d) \in X$
$\longrightarrow v_x \in \mathcal{V} \wedge v_x.type = \text{"}D\text{"} \wedge (v_x, name(r)) \in \mathtt{Lbl}_{\mathcal{V}} \wedge e_x = (v_d, v_x) \in \mathcal{E}_d$
$\wedge (e_x, \text{"}type\text{"}) \in \mathtt{Lbl}_{\mathcal{E}}$

[5] We denote optional edges with negative weights; they can be activated during matching by multiplying their weight with -1.

(Object Property Nodes) $p \in P \wedge d = domain(p) \wedge r = range(p)$
$\rightarrow v_p \in \mathcal{V} v_p.type = "P" \wedge (v_p, name(p)) \in \texttt{Lbl}_{\mathcal{V}} \wedge e_{p_1} = (v_d, v_p) \in \mathcal{E}_u$
$\wedge (e_{p_1}, "ref") \in \texttt{Lbl}_{\mathcal{E}} \wedge e_{p_2} = (v_r, v_p) \in \mathcal{E}_u \wedge (e_{p_2}, "ref") \in \texttt{Lbl}_{\mathcal{E}}$

3.2 Reasoning

We extend *IncGraph+* using reasoning on $\mathcal{O}$ in two ways. First, we assume that a reasoner infers and materializes relevant axioms prior to *IncGraph+* generation (basic reasoning). Second, we derive implicit information using custom, non-standard reasoning rules and directly encode consequences into *IncGraph+* ($\mathcal{O}$). This concerns aspects that may result in additional relevant matches with *IncGraph+* ($\mathcal{R}$) of a database schema: domains and ranges of properties other than the ones explicitly asserted, and inverse properties.

3.2.1 Sub Classes/Super Classes:

Domains and ranges are often modeled in a database at a granularity other than the one expressed in a corresponding ontology. In one case, this may be one or more specific sub class(es) of the actual domain or range, if the database is designed to accept only information about those specific sub classes. In another case, however, this can even be a *super* class of the domain or range, following the reasoning that *some* of the individuals of the super class can have values of that property. Although somewhat less frequent, databases occasionally happen to model properties in such an overly generic way.

In order to solve this, we add reference edges to the graph to encode all alternative connections. To express the fact that some cases are less expected, we assign weight factors to the additional edges:

(Sub Classes) $v \in \mathcal{V} \wedge v_d.type \in \{"A", "P"\} \wedge e = (v, v') \in \mathcal{E}_u \wedge v'.type = "C"$:
$\forall s \in C.subClass(class(v), s) \rightarrow e' = (v, s) \in \mathcal{E}_u \wedge (e', "ref") \in \texttt{Lbl}_{\mathcal{E}}$

(Super Classes) $v \in \mathcal{V} \wedge v_d.type \in \{"A", "P"\} \wedge e = (v, v') \in \mathcal{E}_u \wedge v'.type = "C"$:
$\forall s \in C : superClass(class(v), s) \rightarrow e' = (v, s) \in \mathcal{E}_u \wedge (e', "ref") \in \texttt{Lbl}_{\mathcal{E}} \wedge (e', 0.5) \in \mathcal{W}_{\mathcal{E}}$

3.2.2 Pseudo Equivalence:

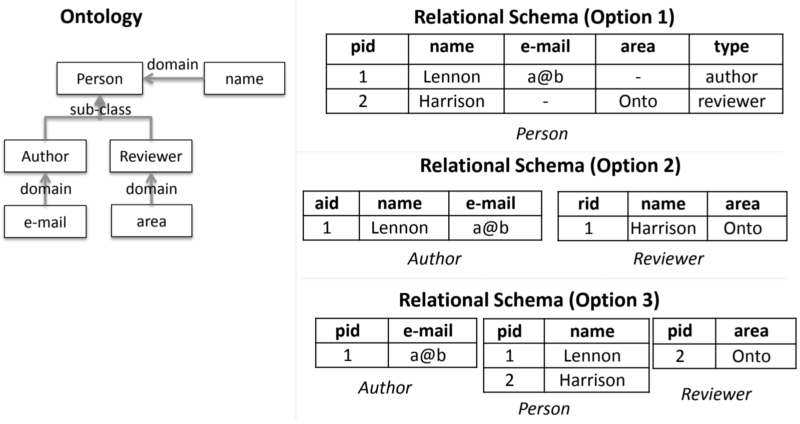
While real equivalences can be made explicit by a reasoner in pre-processing, another more subtle notion of equivalence may apply on *IncGraph+* on axioms that are not actually equivalent in the ontology: pseudo-equivalence. As relations and referential constraints have no semantic direction and because in *IncGraph+* we only model aspects that can find correspondences, properties also lose their direction in *IncGraph+*. This means, that inverse properties become effectively equivalent w.r.t. *IncGraph+*. However, they are not in the underlying ontology and thus are modeled twice during graph construction. Consequently, they compete for matches and distract structural alignment.

We solve this issue by unifying pseudo-equivalent property axioms into a single node during *IncGraph+* construction, but maintain both labels for alternative lexical matching.

3.3 Patterns and Meta Knowledge

Due to different data modeling approaches, the same information is structured differently in relational databases and ontologies, and direct correspondences between modeling constructs are often difficult to establish: e.g., relational database schemata do not model concept and property hierarchies explicitly, many-to-many relations are expressed using intermediate tables, etc.: for example, typically three different ways are used in which an ontological *subClassOf* relation can be implemented in a relational model (se Fig. 4). In the database community, research has identified common design patterns for modeling interrelated data [GMUW08], and the Semantic Web research in turn noted various common ways for matching these database schema structures with the semantically equivalent ontology constructs [Se12, HM13].

Such common correspondence patterns provide valuable background knowledge helping to generate mappings between ontologies and relational databases: a set of initial mappings can be reinforced if they appear to satisfy a common pattern.



Relational Schema (Option 1)

pid	name	e-mail	area	type
1	Lennon	a@b	-	author
2	Harrison	-	Onto	reviewer

Person

Relational Schema (Option 2)

aid	name	e-mail
1	Lennon	a@b

Author

rid	name	area
1	Harrison	Onto

Reviewer

Relational Schema (Option 3)

pid	e-mail
1	a@b

Author

pid	name
1	Lennon
2	Harrison

Person

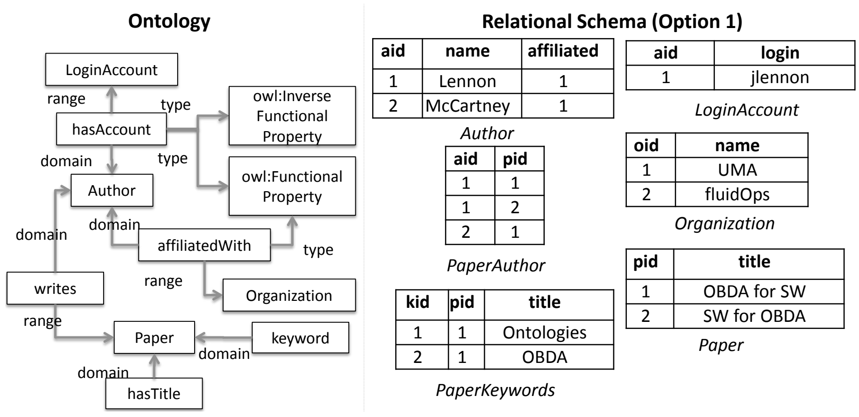
pid	area
2	Onto

Reviewer

Fig. 4: Correspondence patterns: class-subclass hierarchy

Based on these common structures, we selected the following patterns, which can directly be used :

- *rdfs:subClassOf* relations between classes:
  - *One common table for all.* An example is shown as Option 1 in Fig. 4. Here one table stores information about all people: both authors and reviewers. The



aid	name	affiliated
1	Lennon	1
2	McCartney	1

Author

aid	login
1	jlennon

LoginAccount

aid	pid
1	1
1	2
2	1

PaperAuthor

oid	name
1	UMA
2	fluidOps

Organization

kid	pid	title
1	1	Ontologies
2	1	OBDA

PaperKeywords

pid	title
1	OBDA for SW
2	SW for OBDA

Paper

Fig. 5: Correspondence patterns: properties

type column stores the type of the record, and the type-specific fields, such as *e-mail* and *area* receive NULL values if they are not relevant for the record.

– *Separate unrelated tables*. Option 2 in Fig. 4 provides an illustration: authors and reviewers are stored in separate tables containing only fields relevant for the specific type. Common fields such as *name* are contained in both tables.

– *Separate tables linked via a 1:1 foreign key relation*. See Option 3 in Fig. 4: common fields are defined in the Person table which has 1:1 foreign key relations to both category-specific ones.

- *owl:ObjectProperty* links:

  – *1:1 relation*: two tables are connected via their unique keys, similarly to the relation between the tables Author and LoginAccount in Fig. 5.

  – *1:n relation*: two tables are connected via a foreign key, which is unique in one of the tables, as is the Organization table in Fig. 5. The unique key *oid* of the Organization table is referenced in the Author table.

  – *n:m relation*: two tables are connected via an intermediate table containing two foreign key columns. The PaperAuthor table in Fig. 5 is used as such an intermediary between the Author and Paper tables.

- *owl:DatatypeProperty* links:

  – *1:1 relation*: a column in the table directly contains the value of a datatype functional property (e.g., *hasTitle*).

  – *1:n relation*: a separate table is linked via a foreign key to the main table and contains an additional column for the data values. The example is the PaperKeywords table in Fig. 5.

IncMap exploits these patterns in heuristic rules that enrich the schema graphs. When building a schema graph, these rules add to the graphs special *pattern nodes*. Such pattern node represents a specific pattern type (e.g., “class-subClass”) and is connected by edges to the relevant content nodes (tables and fields for the database schema graph, classes and properties for the ontology graph). Apparently, while patterns in the database are merely structural and may or may not represent the assumed semantics, their correspondences on the ontology side are factual axioms. In addition, some patterns are also ambiguous (e.g., 1:1 relations vs. the third subClass pattern). Also, a number of patterns cannot even be identified with certainty, but heuristics apply that result in varying confidence scores. Thus, on the database side, we employ weighted edges to connect pattern nodes, representing their detected confidence score. The role of the pattern nodes is to reinforce the connection between involved nodes on both sides and thus make the fixpoint computation algorithm more likely to lead to higher similarity scores for pairs of nodes in the neighborhood of corresponding or aligned pattern node.

Formally, for each supported pattern on relational schema $\mathcal{R}$, there is a pattern qualifier heuristic H that assigns each subset of schema elements (i.e., each $E \in \mathcal{P}(\mathcal{R})$) a score, denoting the confidence they might form the specified pattern.

Then: $H(E) > 0.0 \rightarrow \forall el \in E : v_x \in \mathcal{V} \wedge v_x.type = "X" \wedge e = (v_{el}, v_x) \in \mathcal{E}_d \wedge e = (v_{el}, v_x) \in \mathtt{Lbl}_{\mathcal{V}}$

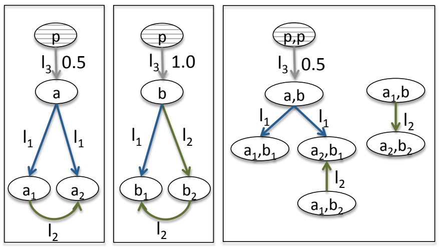
4 Matching & Mapping

IncMap produces mappings in two main stages: (1) matching, i.e., finding correspondences between nodes in schema graphs of the relational source and the target ontology, and (2) mapping generation based on matched correspondences.

4.1 Matching Process

Matching a source and target *IncGraph+* starts with calculating the cartesian products between nodes in the source and target, for each respective node type. Each pair, i.e., each potential correspondence, is then evaluated by an initial match operator, which calculates an initial lexical similarity between the nodes based on their labels. The preferred initial match operator in *IncMap* is word-bag Jaccard similarity on stemmed, stop-word filtered tokenizations of the labels. For nodes with no lexical labels, such as pattern nodes, this operator assigns an initial similarity of 0.5.

Next, paired nodes get reassembled into a new graph, based on common edges that both paired nodes shared in their respective *IncGraph+*. This is in preparation of a following phase, where a fixpoint computation based on the Similarity Flooding algorithm [MGMR02] refines the initial scores based on structural similarities. In [MGMR02], the reassembled graph used as input for Similarity Flooding is called a Pairwise Connectivity Graph, (PCG). Our graph is based on PCG but extended in several ways, primarily to accommodate



(a) Source (b) Target (c) PCG+

Fig. 6: Construction of Extended Pairwise Connectivity Graph (Simplified)

undirected edges, weighted edges and optional edges, which are not supported in the original PCG. We thus refer to this reassembled graph as Extended Pairwise Connectivity Graph (PCG+). Figure 6 shows a generic and simplified example of two input graphs and the resulting PCG+ (Figure 6c). The PCG+ has a node for every pair in the cross products of nodes from the input graphs, where nodes of some type (or color) in one graph will be paired only with nodes of the same color in the other. Therefore, in the figure, all combinations of a, a_1, a_2, b, b_1 and b_2 are in the graph, but only one node pairing p_1 with p_2. Edges are added to the PCG+ wherever both constituent nodes of a pair had a shared type of edge in the same direction. For instance, a and b have both an outgoing edge labeled l_1, which lead to a_1 and b_1, respectively. Therefore, the pair (a, b) in the PCG+ also has an outgoing edge l_1 leading to the pair (a_1, b_1). In addition, if either edge was weighted, their weights get multiplied for the PCG+. Unweighted edges are assumed to carry a weight of 1.0. If either edge is optional, the common edge becomes optional.

The PCG+ is then transformed into the final input for Similarity Flooding by adding inverse edges to avoid black holes in the fixpoint computation and by (re-)assigning weights to all edges to balance the score distribution.

Finally, the fixpoint computation will repeatedly distribute scores of matches to neighboring nodes depending on edge weights, thus refining the score of correspondences structurally. After each iteration, we check for optional edges supported by a pattern node with a top score, in which case the edge would be activated for the next iteration. The fixpoint computation halts if maximum score changes drop below a configurable maximal delta during one iteration or if a maximum number of iterations has been reached.

4.2 Mapping Generation

IncMap exports mappings based on most likely correspondences calculated during the fixpoint computation.

Correspondences are selected based on final scores. *IncMap* supports the interpretation of several correspondences as 1:1 mappings or n:1 mappings, but not n:1 or n:m mappings. Thus, intuitively, for each target side node (i.e., each node in the ontology *IncGraph*+), one correspondence should be selected. We refer to this set of correspondences as the *target top-1* set of correspondences.

However, target top-1 correspondences may lead to significant inconsistencies, lowering the quality of the resulting mappings. For instance, the best match for a property will in some cases match its range class to a different table key then the one chosen as best match for the class node of the range. While this might even be correct on occasion – either, because the range class match is the one, which is incorrect, or because both matched keys define identical individuals – our general experience is that accepting those inconsistencies lowers mapping quality on average. Therefore, we do not select target top-1 correspondences but first choose target top-1 correspondences for classes only and then choose for properties from a restricted set that interprets domains and ranges consistently with the chosen class matches.

Finally, for each correspondence, one mapping rule is being generated using R2RML (RDB to RDF Mapping Language)[6] as mapping language. Technically, this is enabled by provenance information encoded within every node.

5 Experimental Evaluation

While all the extensions of *IncMap* discussed in this paper were motivated by putting our system into practice over the last few years in different application domains, we did run experiments to systematically analyze the impact of each extensions, also in comparison to other systems.

5.1 Benchmark Suite

We ran experiments using our mapping generation benchmark suite called RODI [Pi15]. RODI was designed to test *RDB2RDF* mappings in end-to-end mapping scenarios: it provides an input database and a target ontology and requests complete mappings that enable to execute queries over the target ontology while retrieving the results from the underlying source data. The effectiveness of mappings is then judged by a score that mainly represents the number of query tests that return the expected results on mapped data sources.

While RODI is extensible and can run scenarios in different application domains, it ships with a set of *default scenarios* that are designed to test a wide range of fundamental *RDB2RDF* mapping challenges in a controlled fashion.

5.2 Benchmark Results

In the following, we discuss the results of evaluating our system *IncMap*. We test two versions of *IncMap*, a basic setup with only basic reasoning (no custom reasoning rules)

[6] `http://www.w3.org/TR/r2rml/`

and without patterns (*IncMap basic*), and a complete version with all features combined (*IncMap*). We directly compare *IncMap* to its predecessor (called IncMap 1.0 [Pi13]), as well as to the best-performing systems from the original RODI benchmark experiments [Pi15], BootOX [Ji15], and -ontop- [Ro15]. We have also added experiments in a setup with COMA++ [Au05] as a well-known baseline for generic schema matching.

RODI Scenarios: As a first series of experiments we followed the RODI default scenarios. Most default scenarios are set in the conference domain, with two additional scenarios; one in the domain of geographic data as well as another in the oil & gas domain.

The default benchmark scenarios therefore provide nine different relational databases that are to be mapped to their respective corresponding target ontologies, which are provided as T-Boxes (i.e., only schema information with no data). Each benchmark scenario (i.e., each pair of a source database and a target ontology) is based on an existing pair of an ontology/database schema (CMT, CONFERENCE, and SIGKDD) as well as variants where the ontology/database schema was mutated to test a different set of integration challenges.

For example, "adjusted naming" scenarios in the conference domain are closely modeled after the original ontology, only with adjusted identifier names while the database schema was normalized to fourth normal form ($4NF$). "Restructured" scenarios are remodeled from ground up to follow widely used relational database design patterns, rather than following closely the patterns used in ontologies. Most significantly, this includes class hierarchies, resulting in cases where abstract parent classes have no corresponding table in the database, several sibling classes being jointly represented in a single table, etc. In addition, a selection of scenarios with advanced modeling patterns also forms part of the default scenarios: "missing FKs" represents the case of a database with no explicit referential constraints at all, and "denormalized" scenarios contain some denormalized tables. Finally, in the "Combined Cases" different mutations are mixed together (e.g., "adjusted naming" and "missing FKs"). Moreover, all non-conference scenarios use complex real-world relational schemata and ontologies of significant size.

Each scenario has different test categories that allow to break down results to, e.g., class mappings vs. property mappings, mappings based on simple 1:1 correspondences vs. more complex compositions, etc. RODI calculates scores between 0.0 and 1.0, based on the averages of per-test F-measures for each scenario and test category.

Results: Table 1 shows RODI scores for each default scenario on every tested system. RODI scores basically indicate the percentage of successfully passed query tests, although technically defined on the basis of averages of per-query F-measures.

For most scenarios, *IncMap* outperforms all other systems with varying margins. In particular, *IncMap* outperforms its predecessor (IncMap 1.0) by far, improving in all case and even rising by a factor of 10 and more in some of the more complex cases (SIGKDD combined, CONFERENCE missing FKs). Numbers also indicate that *IncMap* has successfully overcome architectural issues in graph construction that had caused its

Tab. 1: Average results of all tests per scenarios (scores based on F-measure). Best numbers per scenario in bold print.

Scenario	**IncMap 1.0**	**-ontop-**	**BootOX**	**COMA++**	***IncMap* basic**	***IncMap***
	Conference adjusted naming					
CMT	0.45	0.28	**0.76**	0.48	0.45	0.66
CONFERENCE	0.26	0.26	0.51	0.36	0.56	**0.64**
SIGKDD	0.21	0.38	0.86	0.66	0.79	**0.90**
	Conference restructured					
CMT	0.38	0.14	0.41	0.38	0.45	**0.64**
CONFERENCE	0.16	0.13	0.41	0.31	0.46	**0.56**
SIGKDD	0.11	0.21	0.52	0.41	0.45	**0.69**
	Conference combined case					
SIGKDD	0.05	0.21	0.48	0.28	0.45	**0.55**
	Conference missing FKs					
CONFERENCE	0.03	-	0.33	0.21	**0.41**	**0.41**
	Conference denormalized					
CMT	0.22	0.20	0.44	-	0.52	**0.71**
	Geodata					
Geodata	0.00	-	**0.13**	-	0.08	0.08
	Oil & Gas					
Oil & Gas	0.06	0.10	0.14	0.02	0.12	**0.17**

predecessor to fall way behind the competition in different cases. This mostly affects scenarios involving ontologies that are rather expressive (CONFERENCE and SIGKDD, as opposed to CMT).

Between the two versions of *IncMap*, the *basic* version and complete *IncMap*, a positive impact of the advanced patterns and reasoning in the full version can generally be observed. The only exception is "CONFERENCE Missing FKs", which is natural as both reasoning and patterns rely on referential constraints to produce additional correspondences with the database.

In general, it can however be observed that *IncMap*, even while outperforming other systems, still significantly struggles with the more complex scenarios as well as the two real-world cases (Geodata and Oil & Gas). The further the database schema deviates from its corresponding ontology by using traditional relational database design patterns (restructured, advanced scenarios), the more mapping result quality drops.

6 User Study

Next to systematic experiments with our benchmark suite RODI, we have conducted a user study to evaluate the merits of *IncMap* in practice. We have therefore asked a number of *RDB2RDF* data integration professionals to perform a small, but real-world, industry mapping problem with and without automatically bootstrapped mappings, did ask for their impressions and observed how long the task took them.

6.1 Study Design

We have set the study problem in a commercial application domain that we frequently see in our everyday consulting practice with customers at fluid Operations: data center management. The domain comprises information about hardware, software and services in large data centers, monitoring information etc. As relational data source for the study, we have used one realistic customer database of advanced complexity, containing 61 tables with a combined 776 columns. The target ontology was a subset of the enterprise datacenter/cloud vocabulary that we normally use for commercial data integration projects in this domain. The full vocabulary is publicly available[7]. We did limit the mapping task to mapping information regarding *hosts*. This limits the actually relevant part of the target ontology to 3 classes, 8 datatype properties and 4 object properties. While scaled down, this setup closely resembles our typical real-world data integration projects insofar as it requires to map parts of a holistic ontology from only a small subset of information in a large source database. To further specify the task we have provided a set of dashboard queries for which the participants needed to curate the mapping.

We have asked 7 randomly selected data integration experts to create the required mapping, either starting with an intial, automatically bootstrapped mapping using *IncMap*, or from scratch. We then measured the time taken to complete the mapping to each expert's satisfaction in both cases. We asked all users to first perform the task using *IncMap* mappings. Note, that this puts *IncMap* at a disadvantage over the from-scratch case, as any time taken to get basically familiarized with the source schema would count against the first task.

Additionally, we did ask users before starting their first task to judge the automatically generated mappings by their perceived utility without any corrections. We also asked after the experiments about the preferred approach for future tasks. Specifically, we did ask the following two questions:

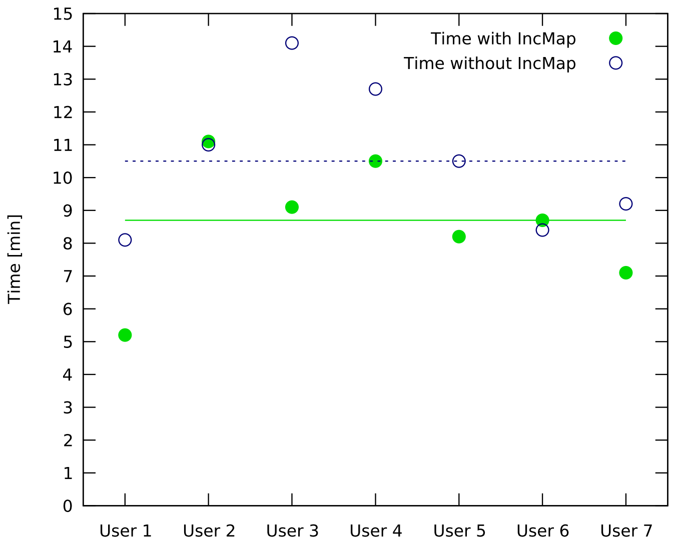
Question 1 (before first task): "When inspecting the initially bootstrapped IncMap mappings, how useful do you think they are on a scale from 'not useful at all' to 'highly useful'?"

Question 2 (after last task): "When performing a similar task again in the future, would you prefer to work with bootstrapped IncMap mappings or start from scratch?"

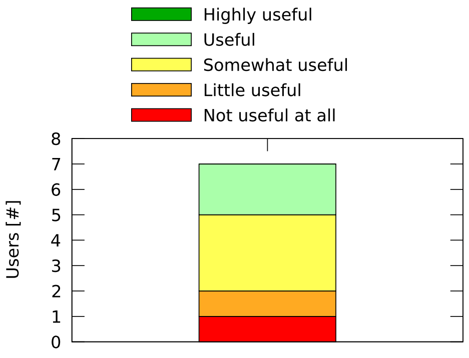
6.2 Results and Observations

Figure 7 shows our observations from the user study. First, Figure 7a depicts the times taken to complete the mapping with and without *IncMap* bootstrapping for each of the users. The green line marks the median time taken with *IncMap* bootstrapping, while the dashed blue

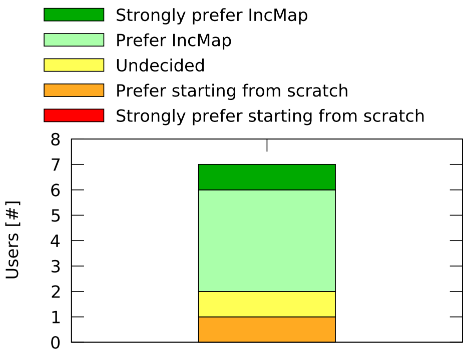
[7] `http://www.fluidops.com/en/company/training/open_source`



(a) Times taken to complete



(b) Question 1 (before task)



(c) Question 2 (after task)

Fig. 7: User Study Results

line marks the median time taken without *IncMap*. It shows that, although all users did first perform the task on *IncMap* bootstrapped mappings and thus were already familiar with the source schema when trying again from scratch, it took all but two of them less time to complete the mapping when working with *IncMap*, with the two exceptions reporting very small differences. On average, despite this disadvantage for *IncMap* in the experiment, users did save 121 seconds (roughly 20%), with a difference of 108 seconds between median times.

Figure 7b shows the impressions of users regarding the *initial* automatic mappings: although some users did acknowledge them to be somewhat useful, results are poor on average. This is in line with expectations, as fully automatic mappings on relatively large schemata are known to be generally difficult to use without prior corrections, due to both incomplete matches and a high number of false positives (e.g., [HQ07]).

Hence, the more relevant question is whether the mappings are useful as a starting point for manual editing. Besides the obvious indication in this direction from task completion times, we also did ask users about their *subjective* judgment, namely which method of building methods they would prefer next time for a similar task. The responses are depicted in Figure 7c: a clear majority of 5 out of 7 users would prefer automatically bootstrapped mappings from *IncMap*, with another one neutral and only one user leaning slightly against.

7 Related Work

IncMap automatically generates mappings between relational databases and ontologies in R2RML, a recent W3C recommendation. Our approach builds on top of previous research on database patterns [Se12, HM13], lexical (e.g., [CH13]) and structural [MGMR02] ontology matching techniques, and introduces original graph construction algorithms.

A number of other mapping generation systems have been developed or updated to support R2RML. QODI [TSM13] generates mappings with a rather simple mechanism but additionally exploits the query workload to improve mappings. The approach of [Ji15] relies on the ontology alignment tool LogMap [JRG11] after transforming the database schema into an ontology representation in a naive pre-processing step. Similar setups have been built in conjunction with -ontop- [Ro15] and MIRROR [de15]. MIRROR generates specialized mappings that consider relational-to-ontology mapping patterns, but produces a target-agnostic ontology and thus requires ontology alignment to map to a target ontology. Karma [Kn12] employs an interactive, semi-automatic approach to produce R2RML mappings directly from input relational schemata but produces no fully automatic (i.e., initial) mappings. [Ne13] also produces R2RML but requires an extreme level of human interaction and generates only final mappings automatically, after correspondences have been specified.

Other related systems predate current *RDB2RDF* approaches as well as the R2RML language. Most prominently, COMA++ [Au05] was originally designed to match relational schemata but has evolved to also perform inter-model matchings, although it is not its main focus. Other previous efforts towards *RDB2RDF* mapping generation (e.g., [Pa06]) date back to the early days of OBDI, and work has since been discontinued. A comprehensive overview of *RDB2RDF* efforts, including related approaches of automatic mapping generation, can be found in the survey of Spanos et al. [SSM12].

To evaluate and compare *IncMap*, we use RODI [Pi15], a recently released *RDB2RDF* benchmark suite. RODI is designed to measure the quality of mapping generation end-to-end by testing queries on the mapping result, and specifically targets mapping challenges that arise from the inter-model gap. No other benchmark for this setting exists to the best of our knowledge. A number of papers discuss related quality aspects from a more general and theoretic perspective (e.g., [CL14, MC14]). However, theoretical criteria such as consistency and completeness give no indication of whether mapping results reflect the expected semantics and lead to useful query results.

8 Conclusion

We have presented *IncMap*, a system to automatically generate direct mappings between relational databases and given target ontologies. *IncMap* is based on an intermediate internal graph representation that allows the representation of both factual knowledge and heuristically observed patterns from the input. The graph model supports a combination of lexical and structural matching for initial alignment.

To evaluate *IncMap*, we experimentally compared it to other systems using RODI [Pi15], a recent benchmark for automatically generated *RDB2RDF* mappings. Results demonstrate that *IncMap* does not only improve massively over its predecessor *IncMap*, but also outperforms all other tested systems. We have also conducted a user study that demonstrates *IncMap* to be useful in practice.

Future work will include support for n-way joins in mapping object properties (for $n > 3$) and improved pattern heuristics.

References

[Au05] Aumueller, David; Do, Hong-Hai; Massmann, Sabine; Rahm, Erhard: Schema and Ontology Matching with COMA++. In: SIGMOD. 2005.

[Bh15] Bhardwaj, Anant P.; Bhattacherjee, Souvik et al.: DataHub: Collaborative Data Science & Dataset Version Management at Scale. CIDR, 2015.

[CH13] Cheatham, Michelle; Hitzler, Pascal: String Similarity Metrics for Ontology Alignment. In: ISWC. 2013.

[CL14] Console, Marco; Lenzerini, Maurizio: Data Quality in Ontology-Based Data Access: The Case of Consistency. In: AAAI. 2014.

[de15] de Medeiros, LF. et al.: MIRROR: Automatic R2RML Mapping Generation from Relational Databases. In: ICWE. 2015.

[DS13] Dong, Xin Luna; Srivastava, Divesh: Big Data Integration. PVLDB, 6(11):1188–1189, 2013.

[GMUW08] Garcia-Molina, Hector; Ullman, Jeffrey D.; Widom, Jennifer: Database Systems – The Complete Book. Prentice Hall, 2008.

[HM13] Hornung, Thomas; May, Wolfgang: Experiences from a TBox Reasoning Application: Deriving a Relational Model by OWL Schema Analysis. In: OWLED Workshop. 2013.

[HQ07] Hu, Wei; Qu, Yuzhong: Discovering Simple Mappings Between Relational Database Schemas and Ontologies. In: ISWC/ASWC. 2007.

[Ji15] Jiménez-Ruiz, E. et al.: BootOX: Practical Mapping of RDBs to OWL 2. In: ISWC. 2015.

[JRG11] Jiménez-Ruiz, Ernesto; Grau, Bernardo Cuenca: LogMap: Logic-Based and Scalable Ontology Matching. In: International Semantic Web Conference. 2011.

[Kn12] Knoblock, C. et al.: Semi-Automatically Mapping Structured Sources into the Semantic Web. In: ESWC. 2012.

[MC14] Mora, Jose; Corcho, Oscar: Towards a Systematic Benchmarking of Ontology-Based Query Rewriting Systems. In: ISWC. 2014.

[MGMR02] Melnik, Sergey; Garcia-Molina, Hector; Rahm, Erhard: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In: ICDE. 2002.

[Ne13] Neto, L.E.T. et al.: R2RML by Assertion: A Semi-automatic Tool for Generating Customised R2RML Mappings. In: ESWC (Satellite Events). 2013.

[Pa06] Papapanagiotou, P. et al.: Ronto: Relational to Ontology Schema Matching. AIS SIGSEMIS BULLETIN, 2006.

[Pi13] Pinkel, Christoph; Binnig, Carsten; Kharlamov, Evgeny; Haase, Peter: IncMap: Pay-as-you-go Matching of Relational Schemata to OWL Ontologies. In: OM. 2013.

[Pi15] Pinkel, Christoph; Binnig, Carsten; Jimenez-Ruiz, Ernesto et al.: RODI: A Benchmark for Automatic Mapping Generation in Relational-to-Ontology Data Integration. In: ESWC. 2015.

[Ro15] Rodriguez-Muro, M. et al.: Efficient SPARQL-to-SQL with R2RML mappings. Journal of Web Semantics, 2015.

[Se12] Sequeda, J. et al.: Relational Database to RDF Mapping Patterns. In: WOP. 2012.

[SSM12] Spanos, Dimitrios-Emmanuel; Stavrou, Periklis; Mitrou, Nikolas: Bringing Relational Databases into the Semantic Web: A Survey. Semantic Web, 3(2), 2012.

[TSM13] Tian, Aibo; Sequeda, Juan F; Miranker, Daniel P: QODI: Query as Context in Automatic Data Integration. In: ISWC. 2013.