# Ontology-Based Integration of Cross-Linked Datasets

Diego Calvanese[1], Martin Giese[2], Dag Hovland[2], and Martin Rezk[1](✉)

[1] Free University of Bozen-Bolzano, Bolzano, Italy
mrezk@inf.unibz.it
[2] University of Oslo, Oslo, Norway

**Abstract.** In this paper we tackle the problem of answering SPARQL queries over *virtually integrated* databases. We assume that the entity resolution problem has already been solved and explicit information is available about which records in the different databases refer to the same real world entity. Surprisingly, to the best of our knowledge, there has been no attempt to extend the standard *Ontology-Based Data Access* (OBDA) setting to take into account these DB links for SPARQL query-answering and consistency checking. This is partly because the OWL built-in `owl:sameAs` property, the most natural representation of links between data sets, is not included in OWL 2 QL, the *de facto* ontology language for OBDA. We formally treat several fundamental questions in this context: how links over database identifiers can be represented in terms of `owl:sameAs` statements, how to recover rewritability of SPARQL into SQL (lost because of `owl:sameAs` statements), and how to check consistency. Moreover, we investigate how our solution can be made to scale up to large enterprise datasets. We have implemented the approach, and carried out an extensive set of experiments showing its scalability.

## 1 Introduction

Since the mid 2000s, *Ontology-Based Data Access* (OBDA) [9,14,15] has become a popular approach for *virtual* data integration [6]. In (virtual) OBDA, a conceptual layer is given in the form of (the intensional part of) an ontology (usually in OWL 2 QL) that defines a shared vocabulary, models the domain, hides the structure of the data sources, and can enrich incomplete data with background knowledge. The ontology is connected to the data sources through a declarative specification given in terms of mappings [4] that relate symbols in the ontology (classes and properties) to (SQL) views over data. The ontology and mappings together expose a virtual RDF graph, which can be queried using SPARQL queries, that are then translated into SQL queries over the data sources. In this setting, users no longer need an understanding of the data sources, the relation between them, or the encoding of the data.

One aspect of OBDA for data integration is less well studied however, namely the fact that in many cases, complementary information about the same entity is distributed over several data sources, and this entity is represented using

M. Arenas et al. (Eds.): ISWC 2015, Part I, LNCS 9366, pp. 199–216, 2015.
DOI: 10.1007/978-3-319-25007-6_12

different identifiers. The first important issue that comes up is that of *entity resolution*, which requires to understand which records actually represent the same real world entity. We do not deal with this problem here, and assume that this information is already available.

Traditional relational data integration techniques use extract, transform, load (ETL) processes to address this problem [6]. These techniques usually choose a single representation of the entity, merge the information available in all data sources, and then answer queries on the merged data. However, this approach of physically merging the data is not possible in many real world scenarios where one has no complete control over the data sources, so that they cannot be modified, and where the data cannot be moved due to freshness, privacy, or legal issues (see, e.g., Section 3).

An alternative that can be pursued in OBDA is to make use of mappings to *virtually merge* the data, by consistently generating only one URI per real world entity. Unfortunately, also this approach is not viable in general: 1. it does not scale well for several datasets, since it requires a central authority for defining URI schemas, which may have to be revised along with all mappings whenever a new source is added, and 2. it is crucial for the efficiency of OBDA that URIs be generated from the primary keys of the data sources, which will typically differ from source to source.

The approach we propose in this paper is based on the natural idea of representing the links between database records resulting from entity resolution in the form of *linking tables*, which are binary tables in *dedicated* data sources that simply maintain the information about pairs of records representing the same entity. This bring about several problems that need to be addressed: 1. links over database identifiers should be represented in terms of OWL `owl:sameAs` statements, which is the standard approach in semantic technologies for connecting entity identifiers; 2. the presence of `owl:sameAs` statements, which are inherently transitive, breaks rewritability of SPARQL queries into SQL queries over the sources, and one needs to understand whether rewritability can be recovered by imposing suitable restrictions on the linking mechanism; 3. a similar problem arises for checking consistency of the data sources with respect to the ontology, which is traditionally addressed through query answering; 4. since performance can be prohibitively affected by the presence of `owl:sameAs`, it becomes one of the key issues to address, so as to make the proposed approach scalable over large enterprise datasets.

In this paper we tackle the above issues in the setting where we are given an OWL 2 QL ontology that is mapped to a set of data sources, which are then extended with linking tables. Specifically, we provide the following contributions:

– We propose a mapping-based framework that carefully virtually constructs `owl:sameAs` statements from the linking tables, and deals with transitivity and symmetry, in such a way that performance is not compromised.
– We define a suitable set of restrictions on the linking mechanisms that ensures rewritability of SPARQL query answering, despite the presence of `owl:sameAs` statements.

– We develop a sound and complete SPARQL query translation technique, and show how to apply it also for consistency checking.
– We show how to optimize the translation so as to critically reduce the size of the produced SQL query.
– To empirically demonstrate scalability of our solution, we carry out an extensive set of experiments, both over a real enterprise cross-linked data set from the oil&gas industry, and in a controlled environment; this demonstrates the feasibility of our approach.

The structure of the paper is as follows: Section 2 briefly introduces the necessary background needed to understand this paper, and Section 3 describes our enterprise scenario. Section 4 provides a sound and complete SPARQL query translation technique for cross-linked datasets. Section 5 presents the main contribution of the paper, showing how to construct an OBDA setting over cross-linked datasets, and Section 6 presents our optimization technique. Section 7 presents an extensive experimental evaluation. Section 8 surveys related work, and Section 9 concludes the paper.

## 2 Preliminaries

**Ontology Based Data Access.** In the traditional OBDA setting $(\mathcal{T}, \mathcal{M}, D)$, the three main components are a set $\mathcal{T}$ of OWL 2 QL [12] axioms (called the TBox), a relational database $D$, and a set $\mathcal{M}$ of mappings. The OWL 2 QL profile of OWL 2 guarantees that queries formulated over $\mathcal{T}$ can be *rewritten* into SQL [2]. The mappings allow one to define how classes and properties in $\mathcal{T}$ should be populated with objects constructed from the data retrieved from $D$ by means of SQL queries. Each mapping has one of the forms:

$$\text{Class(subject)} \leftarrow \texttt{sql}_{class} \qquad \text{Property(subject,object)} \leftarrow \texttt{sql}_{prop},$$

where $\texttt{sql}_{class}$ and $\texttt{sql}_{prop}$ respectively are a unary and binary SQL query over $D$. For both types of mappings we also use the equivalent notation $(s\ p\ o) \leftarrow \texttt{sql}$. Subjects and objects in RDF triples are resources (individuals or values) represented by URIs or literals. They are generated using templates in the mappings. For example, the URI template for the subject can take the form `<http://www.statoil.com/{id}>` where `{id}` is an attribute in some DB table, and it generates the URI `<http://www.statoil.com/25>` when `{id}` is instantiated as `"25"`. From $\mathcal{M}$ and $D$, one can derive a (*virtual*) RDF graph $G_{\mathcal{M},D}$, obtained by applying all mappings. Any RDF graph can be seen as a set of logical assertions. Thus, the Tbox together with $G_{\mathcal{M},D}$ constitutes an *ontology* $\mathcal{O} = (\mathcal{T}, G_{\mathcal{M},D})$.

To handle ontology-based integration of cross-linked datasets, we extend here the traditional OBDA setting with a fourth component $\mathcal{A}_S$ containing a set of statements of the form `owl:sameAs` $(o_1,o_2)$. Thus, in this paper, an OBDA setting is a tuple $(\mathcal{T}, \mathcal{M}, D, \mathcal{A}_S)$, and its corresponding *ontology* is the tuple $\mathcal{O} = (\mathcal{T}, G_{\mathcal{M},D} \cup \mathcal{A}_S)$. Unless stated differently, in the following we work with OBDA settings of this form.

**Semantics:** To interpret ontologies, we use the standard notions of first order interpretation, model, and satisfaction. That is, $\mathcal{O} \models A(\boldsymbol{v})$ iff for every model $\mathcal{I}$ of $\mathcal{O}$, we have that $\mathcal{I} \models A(\boldsymbol{v})$. Intuitively, adding an ontology $\mathcal{T}$ on top of an RDF graph $G$, *extends* $G$ with extra triples inferred by $\mathcal{T}$. Formally, the RDF graph (*virtually*) exposed by the OBDA setting $((\mathcal{T}, \mathcal{M}, D, \mathcal{A}_S)$ is $G^{(\mathcal{T},\mathcal{M},D,\mathcal{A}_S)} = \{A(\boldsymbol{v}) \mid (\mathcal{T}, G_{\mathcal{M},D} \cup \mathcal{A}_S) \models A(\boldsymbol{v})\}$.

**SPARQL.** SPARQL is a W3C standard language designed to query RDF graphs. Its vocabulary contains four pairwise disjoint and countably infinite sets of symbols: $\mathbf{I}$ for *IRIs*, $\mathbf{B}$ for *blank nodes*, $\mathbf{L}$ for *RDF literals*, and $\mathbf{V}$ for *variables*. The elements of $\mathbf{T} = \mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$ are called *RDF terms*. A *triple pattern* is an element of $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$. A *basic graph pattern* (*BGP*) is a finite set of triple patterns. Finally, a *graph pattern*, $Q$, is an expression defined by the grammar

$$Q \ ::= \ \mathrm{BGP} \ \mid \ \textsc{Filter}(P, F) \mid \ \textsc{Union}(P_1, P_2) \ \mid \textsc{Join}(P_1, P_2) \ \mid \ \textsc{Opt}(P_1, P_2, F),$$

where $F$, is a *filter expression*. More details can be found in [3].

A *SPARQL query* $(Q, V)$ is a graph pattern $Q$ with a set of variables $V$ which specifies the *answer variables*—the set of variables in $Q$ whose values we are interested in. The values to variables are given by *solution mappings*, which are *partial* maps $s\colon \mathbf{V} \to \mathbf{T}$ with (possibly empty) domain $dom(s)$. Here, following [9,15], we use the set-based semantics for SPARQL (rather than the bag-based one, as in the specification).

The SPARQL algebra operators are used to evaluate the different fragments of the SPARQL query. Given an RDF graph $G$, the *answer to a graph pattern $Q$ over $G$* is the set $[\![Q]\!]_G$ of solution mappings defined by induction using the SPARQL algebra operators and starting from the base case: triple patterns. Due to space limitation, and since the entailment regime only modifies the SPARQL semantics for triple patterns, here we only show the definition of for this basic case. We provide the complete definition in our technical report [3].

For a triple pattern $B$, $[\![B]\!]_G = \{s\colon var(B) \to \mathbf{T} \mid s(B) \subseteq G\}$ where $s(B)$ is the result of substituting each variable $u$ in $B$ by $s(u)$. This semantics is known as *simple entailment*. Given a set $V$ of variables, the *answer to* $(Q, V)$ *over* $G$ is the restriction $[\![Q]\!]_{G|V}$ of the solution mappings in $[\![Q]\!]_G$ to the variables in $V$.

**SPARQL Entailment Regime.** We present now the standard W3C semantics for SPARQL queries over OWL 2 ontologies under different entailment regimes. We use here the entailment regimes only to reason about individuals and, unlike [9], we do not allow for variables in triple patterns ranging over class and property names. We leave the problem of extending our results to handle also this case for future work, but we do not expect this to present any major challenge.

We work with TBoxes expressed in the OWL 2 QL profile, which however may contain also `owl:sameAs` statements. Therefore, we consider two Direct Semantics entailment regimes for SPARQL queries, which differ in how they interpret `owl:sameAs`: the *DL entailment regime* (which defines $\models_{DL}$) interprets `owl:sameAs` internally, implicitly adding to the ontology $\mathcal{O}$ the axioms to handle equality, i.e., transitivity, symmetry, and reflexivity. Instead, the *QL entailment regime* (which defines $\models_{QL}$) interprets `owl:sameAs` as a standard object property, hence does not assign to it any special semantics.

Observe that a basic property of logical equality is that if $a$ and $b$ are equal, everything that holds for $a$ should hold also for $b$, and viceversa. In the context of SPARQL, informally it means that given the answer $[\![B]\!]_{\mathcal{T},G\cup\mathcal{A}_S}$ to a triple pattern $B$, if the answer contains the solution mapping $s : v \mapsto o$ and $\mathcal{T} \models$ `owl:sameAs`$(o, o')$, then $[\![B]\!]_{\mathcal{T},G\cup\mathcal{A}_S}$ must also contain a solution mapping $s'$ that coincides with $s$ but $s' : v \mapsto o'$. Formally, the answer $[\![B]\!]^{R}_{\mathcal{T},G\cup\mathcal{A}_S}$ to a BGP $B$ over an ontology $\mathcal{O}$ under entailment regime $R$ is defined as follows:

$$[\![B]\!]^{R}_{\mathcal{O}} \;=\; \{s\colon \mathit{var}(B) \rightarrow \mathbf{T} \mid (\mathcal{O}) \models_R s(B)\},$$

Starting from the $[\![B]\!]^{R}_{\mathcal{O}}$ and applying the SPARQL operators in $Q$, we compute the set $[\![Q]\!]^{R}_{\mathcal{O}}$ of *solution mappings.*

## 3 Use Case and Motivating Example

In this section we briefly describe the real-world scenario we have examined at Statoil, and we illustrate the challenges it presents for OBDA with an example.

At Statoil, users access several databases on a daily basis, some of them are the Exploration and Production Data Store (EPDS), the Norwegian Petroleum Directorate (NPD) FactPages, and several OpenWorks databases. EPDS is a large Statoil-internal legacy SQL (Oracle 10g) database comprising over 1500 tables (some of them with up to 10 million tuples), 1600 views and 700 Gb of data. The NPD FactPages[1] is a dataset provided by the Norwegian government, and it contains information regarding the petroleum activities on the Norwegian continental shelf. OpenWorks Databases contain projects data produced by geoscientists at Statoil. The information in these databases overlap, and often they refer to the same entities (companies, wells, licenses) with different identifiers. In this use case the entity resolution problem has been solved since the links between records are available.

The users at Statoil need to query (and get an answer in reasonable time) the information about these objects without worrying about what is the particular identifier in each database. Thus, we assume that the SPARQL queries provided by the users *will not* contain `owl:sameAs` statements. The equality between identifiers should be handled *internally* by the OBDA system. To illustrate this we provide the following simplified example:

*Example 1.* Suppose we have the three datasets (from now on $D_1, D_2, D_3$) with wellbore[2] information, and a dataset $D_4$ with information about companies and

[1] http://factpages.npd.no/

[2] A wellbore is a hole drilled for the purpose of exploration or extraction of natural resources.

$D_1$

| id1 | Name |
|---|---|
| a1 | 'A' |
| a2 | 'B' |
| a3 | 'H' |

$D_2$

| id2 | Name | Well |
|---|---|---|
| b1 | null | 1 |
| b2 | 'C' | 2 |
| b6 | 'B' | 3 |

$D_3$

| id3 | AName |
|---|---|
| c3 | 'U1' |
| c4 | 'U2' |
| c5 | 'U6' |

$D_4$

| id4 | LName |
|---|---|
| 9 | 'Z1' |
| 8 | 'Z2' |
| 7 | 'Z3' |

**Fig. 1.** Wellbore datasets $D_1$, $D_2$, $D_3$, and company dataset $D_4$

licenses, as illustrated in Figure 1. The wellbores in $D_1$, $D_2$, $D_3$ are linked, but companies in $D_4$ are not linked with the other datasets. These four datasources are integrated virtually by topping them with an ontology. The ontology contains the concept `Wellbore` and the properties `hasName`, `hasAlternativeName` and `hasLicense`.

The terms `Wellbore` and `hasName` are defined using $D_1$ and $D_2$. The property `hasAlternativeName` is defined using $D_3$. The property `hasLicense` is defined over the isolated dataset $D_4$. We assume that mappings for wellbores from $D_i$ use URI templates $uri_i$. In addition, we know that the wellbores are cross-linked between datasets as follows: wellbores $a1, a2$ in $D_1$ are equal to $b2, b1$ in $D_2$ and $c3, c4$ in $D_3$, respectively. In addition, $a3$ is equal to $c5$. These links are represented at the ontology level by `owl:sameAs` statements of the form: `owl:sameAs` (`uri1(a1)`,`uri2(b2)`), `owl:sameAs` (`uri2(b2)`,`uri3(c3)`), etc.

Consider now a user looking for all the wellbores and their names. According to the SPARQL entailment regime, the system should return all the 12 combinations of equivalent ids and names (`(uri1(a1),A)`, `(uri2(b2),A)`, `(uri3(c3),A)`, `(uri1(a2),B)`, `(uri2(b1),B)`, etc.) since all this tuples are entailed by the ontology and the data (c.f. Section 2). Note that no wellbores from $D_4$ are returned. □

The first issue in the context of OBDA is how to translate the user query into a query over the databases. Recall that `owl:sameAs` is not included in OWL QL, thus it is not handled by the current query translation and optimization techniques. If we solve the first issue by applying suitable constraints, we get into a second issue, how to minimize the negative impact on the query execution time when reasoning over cross-linked datasets.A third issue is how to check, for instance, whether `hasName` is a functional property considering the linked entities. A fourth issue is how to handle the multiplicity of equivalent answers required by the standard. For instance, in our example, in principle, it could be enough to pick individuals with template $uri_1$ as class representative, and return only those triples. In the next sections we will tackle all these issues in turn.

## 4 Handling `owl:sameAs` by SPARQL Query Rewriting

In this section we present the theoretical foundations for query answer over ontology-based integrated datasets. We also discuss how to perform consistency checking using this approach. We assume for now that the links are given in the form of `owl:sameAs` statements, and address later, in Section 5, the proper OBDA scenario, where links are not given between URIs, but between database records. Recall that `owl:sameAs` is not in the OWL 2 QL profile, and moreover, by adding the unrestricted use of `owl:sameAs` we lose first order rewritability [1], since one can encode reachability in undirected graphs. This implies that, if we allow for the unrestricted use of `owl:sameAs`, we cannot offer a sound and complete translation of SPARQL queries into SQL.[3]

We present here an approach, based on partial materialization of inference, that in principle allows us to exploit a relational engine for query answering in the presence of `owl:sameAs` statements. This approach, however, is not feasible in practice, and we will then show in Section 5 how to develop it into a practical solution. Our approach is based on the simple observation that we can expand the set $\mathcal{A}_S$ of `owl:sameAs` facts into the set $\mathcal{A}_S^*$ obtained from $\mathcal{A}_S$ by closing it under reflexivity, symmetry, and transitivity. Unlike other approaches based on (partial) materialization [8], we do not expand here also data triples (specifically, those in $G_{\mathcal{M},D}$), but instead rewrite the input SPARQL query to guarantee completeness of query answering. We assume that user queries in general will not contain `owl:sameAs` statements, and therefore, for simplicity of presentation, here we do not consider the case where they are present as input. However, our approach can be easily extended to deal also with `owl:sameAs` statements in user queries. Given a SPARQL query $(Q, V)$ over $(\mathcal{T}, G \cup \mathcal{A}_S)$, we generate a new SPARQL query $(\varphi(Q), V)$ over $(\mathcal{T}, G \cup \mathcal{A}_S^*)$ that returns the same answers as $(Q, V)$ over $(\mathcal{T}, G \cup \mathcal{A}_S)$. This approach is very similar to the singularisation technique in [11]. The translation $\varphi(\cdot)$ is defined as follows.

**Definition 1.** *Given a query $(Q, V)$, the query $(\varphi(Q), V)$ is obtained by replacing every triple pattern $t$ in $Q$ with $\varphi(t)$, where:*[4]

- $\varphi(\{$`?v :P ?w`$\}) = \{$`?v owl:sameAs _:a . _:a :P _:b . _:b owl:sameAs ?w .`$\}$
- $\varphi(\{$`?v rdf:type :C`$\}) = \{$`?v owl:sameAs _:a . _:a rdf:type :C .`$\}$

The following proposition states that answering SPARQL queries over a TBox $\mathcal{T}$ under the DL entailment regime can be reduced to answering SPARQL queries under the QL entailment regime (where `owl:sameAs` has no built-in semantics).

**Proposition 1.** *Given OBDA setting $(\mathcal{T}, \mathcal{M}, D, \mathcal{A}_S)$ and a query $(Q, V)$, we have that $[\![Q]\!]^{DL}_{\mathcal{T}, G_{\mathcal{M},D} \cup \mathcal{A}_S}|_V = [\![\varphi(Q)]\!]^{QL}_{\mathcal{T}, G_{\mathcal{M},D} \cup \mathcal{A}_S^*}|_V$.*

*Consistency Check:* Ontology languages, such as OWL 2 QL, allow for the specification of constraints on the data. If the data exposed by the database through the mappings does not satisfy these constraints, then we say that the ontology is *inconsistent with respect to the mappings and the data.* OBDA allows one to

[3] Using the linear recursion mechanism of SQL-99, a translation would be possible, but with a severe performance penalty for evaluating queries involving transitive closure.

[4] Recall that terms of the form _:x are blank nodes that, when occuring in a query, correspond to existential variables.

check two types of constraints: (i) *functionality* of properties (although it cannot be expressed in OWL 2 QL), which imposes that an individual is connected to at most one element; (ii) *disjointness* of classes/properties, which cannot have (pairs of) individuals in common. In OBDA, consistency checking can be reduced to query-answering [2]. This does not hold anymore in general, when considering cross-linked datasets (where UNA does not hold). For instance, suppose we want to check if the property `:hasName` in Example 1 is functional. Clearly without considering equality between datasets the property is functional, however, when we integrate the datasets, it is not anymore since we have in the graph (`url1(a1) :hasName 'A'`) and (`url2(b2) :hasName 'C'`) and (`url1(a1) owl:sameAs url2(b2)`). This implies that the wellbore `url1(a1)` has two names. Using the translation above we can extend the results in [2] for checking violations of class disjointness and of functionality of data and object properties, to account for `owl:sameAs` statements. For disjointness and functionality of data properties this is accomplished straightforwardly by the translation. Instead, for functionality of object properties, we need to modify the query used in [2] and explicitly incorporate the negation of `owl:sameAs`. For instance, to check if functionality of the object property `:isRelatedTo` *might be violated*, we can check if the following query returns a non-empty answer over $(\mathcal{T}, G \cup \mathcal{A}_S^*)$:

```
SELECT ?x ?y1 ?y2 ?y3 WHERE {
    ?x :isRelatedTo ?y1 .  ?x :isRelatedTo ?y2 .
    FILTER(?y1 != ?y2 AND NOT EXISTS {?y1 owl:sameAs ?y2} ) }
```

If the answer is non-empty, the returned elements might witness the violation of functionality. Notice that, because of the OWA if two elements are *not known* to be equal, in general we cannot infer that they are not equal, and hence functionality might still hold in some models. We refer to [3] for more details.

## 5 Handling Cross-Linked Datasets in Practice

We now deal with the proper case of querying cross-linked datasets, where we are given: *(a)* an OWL 2 QL TBox, *(b)* a collection of datasets, *(c)* a set of mappings, and *(d)* a set of *linking tables*[5] stating equality between records in different datasets that represent the same entity. For simplicity, we can think of each dataset as corresponding to a different data source, but datasets could be decoupled from the actual physical data sources. In

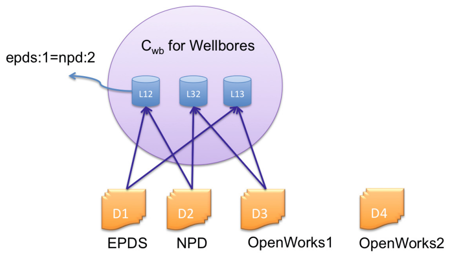


**Fig. 2.** Linking tables for the wellbores category

[5] Note that these tables could be available virtually, and hence retrieved through queries.

general, in different datasets, the same identifiers might be used to denote different objects, and the same objects might be denoted by different identifiers. Moreover, each dataset may contain data records belonging to different *pairwise disjoint* categories $C_1, \ldots, C_m$, for example wellbores, or company names. A category corresponds to a set of records that can be mapped to individuals in the ontology belonging to the same TBox class (different from owl:Thing), and that could, in principle, be joined. For instance, cats and men belong to the same class mammal, but a cat can never be joined with a man, hence cat and men constitute two different categories. We assume that in addition to the datasets $D_1, \ldots, D_n$, for each category $C$ there is a database $D^C$ containing the linking tables for the records in $C$. Specifically, we denote a linking table for datasets $D_i$, $D_j$ and category $C$ with $L^C_{ij}(id_i, id_j)$. A tuple $r_1, r_2$ in $L^C_{ij}$ means that the record $r_1$ in $D_i$ represents the same object as the record $r_2$ in $D_j$. Notice that, we do not assume that there is a linking table for each pair of datasets $D_i$, $D_j$ for each category $C$. The concepts above are illustrated in Figure 2. Our aim is to efficiently answer user SPARQL queries in this setting.

The approach presented in the previous section is theoretical, and cannot be effectively applied in practice because: *(1)* it assumes that the links are given in the form of `owl:sameAs` statements whereas in practice, in an cross-linked setting, they will be given as tables (with the results of the entity resolution process); and *(2)* it requires pre-computing a large number of triples (namely $\mathcal{A}^*_S$) and materializing them into the ontology. Since these triples are not stored in the database, they cannot be efficiently retrieved using SQL. This negatively impacts the performance of query execution.

To tackle these problems, in this section we show how to: *(a)* expose, using mapping assertions that are *optimization-friendly*, the information in the tables expressing equality between DB records, as a set $\mathcal{A}_S$ of `owl:sameAs` statements; *(b)* extend the mappings so as to encode also transitivity and symmetry (but not reflexivity), and hence expose the symmetric transitive closure $\mathcal{A}^+_S$ of $\mathcal{A}_S$; *(c)* modify the query-rewriting algorithm (cf. Definition 1) so as to return sound and complete answers over the (virtual) ontology extended with $\mathcal{A}^+_S$. We detail now the above steps.

**(a) Generating $\mathcal{A}_S$:** We now present a set of constraints on the structure of the linking tables that are fully compatible with real-world requirements, and that allow us to process queries efficiently, as we will show below:

1. All the information about which objects of category $C$ are linked in datasets $D_i$ and $D_j$ is contained in $L^C_{ij}$. Formally: If there are tables $L^C_{ij}$, $L^C_{ik}$ and $L^C_{kj}$, then $L^C_{ij}$ contains all the tuples in $\pi_{id_i,id_j}(L^C_{ik} \bowtie L^C_{kj})$, when evaluated over $D^C$.
2. Linking tables cannot state equality between different elements in the same dataset[6]. Formally: There is no join of the form $L^C_{ik} \bowtie \cdots \bowtie L^C_{ni}$ such that

[6] Observe that this amounts to making the Unique Name Assumption for the objects retrieved by the mappings from one dataset

$L_{1,2}$

| id1 | id2 |
|---|---|
| a1 | b2 |
| a2 | b1 |

$L_{2,3}$

| id2 | id3 |
|---|---|
| b1 | c4 |
| b2 | c3 |

$L_{1,3}$

| id1 | id3 |
|---|---|
| a1 | c3 |
| a2 | c4 |
| a3 | c5 |

**Fig. 3.** Linking Tables

$(o, o')$, with $o \neq o'$, occurs in $\pi_{L^C_{ik}.id_i, L^C_{ni}.id_i}(L^C_{ik} \bowtie \cdots \bowtie L^C_{ni})$, when evaluated over $D^C$.

*Example 2 (Categories).* Consider Example 1. Here we consider only wellbores, therefore we have a single category $C_{wb}$ with three linking tables $L^{C_{wb}}_{12}$, $L^{C_{wb}}_{23}$, and $L^{C_{wb}}_{13}$ as shown in Figure 3. From the constraints above we know that $\pi_{id_1,id_3}(L^{C_{wb}}_{12} \bowtie L^{C_{wb}}_{23})$ is contained in $L^{C_{wb}}_{13}$, when both are evaluated over $D^{C_{wb}}$. $\square$

A key factor that affects performance of the overall OBDA system, is the form of the mappings, which includes the structure of the URI templates used to generate the URIs. Here, we discuss how the part of the mappings (including URI templates) that deal with linking tables should be designed, so this approach scales up. The SPARQL-to-SQL translation must add *all* the SQL queries defining `owl:sameAs`. However, as shown in Section 6, we exploit our URI design to (intuitively) remove as many `owl:sameAs` SQL definitions as possible before query execution.

We propose here to use a different URI template $\texttt{uri}_{C,D}$ for each pair constituted by a category $C$ and a dataset $D$.[7] Observe that this design decision is quite natural, since objects belonging to different categories should not join, even if in some dataset they are identified in the same way. For example, wellbore n. 25 should not be confused with the employee whose id is 25.

Next we generate the set of equalities $\mathcal{A}_S$ extending the set of mappings $\mathcal{M}$, using a different URI template for each tuple (category $C$,dataset $D$). More precisely, to generate $\mathcal{A}_S$ out of the categories $C_1 \ldots C_n$, $\mathcal{M}$ is extended with mappings as follows. For each category $C$, and each linking table $L^C_{ij}$ we extend $\mathcal{M}$ with:

$$\texttt{uri}_{C,D_i}(\{id_i\})\ \texttt{owl:sameAs}\ \texttt{uri}_{C,D_j}(\{id_j\}) \leftarrow \mathit{select} * \mathit{from}\ L^C_{ij} \qquad (1)$$

When the category $C$ is clear from the context we write $\texttt{uri}_i$ to denote $\texttt{uri}_{C,D_i}$

*Example 3 (Mappings).* To generate the `owl:sameAs` statements from the tables in Example 2, we extend our set of mappings $\mathcal{M}$ with the following mappings (fragment):

[7] In the special case where there *are* several datasets that can be mapped to use common URIs, there is no need for linking tables or any of the techniques presented in this paper. We address the more general case, where this is not the case.

```
uri1({id1}) owl:sameAs uri2({id2}) ← SELECT * FROM $L^C_{1,2}$
uri2({id2}) owl:sameAs uri3({id3}) ← SELECT * FROM $L^C_{2,3}$
```

Observe that this also implies that to populate the concept `Wellbore` with elements from $D_1$, the mappings in $\mathcal{M}$ will have to use the URI template: `uri1`. □

Considering that the same URIs in different triples of the virtual RDF graph can be generated from different mapping assertions, we observe that the form of the templates in the mappings related to linking tables will affect also those in the remaining mapping assertions in the OBDA system.

**(b) Approximating $\mathcal{A}^+_S$:** To be able to rewrite SPARQL queries into SQL without adding $\mathcal{A}^*_S$ as facts in the ontology, (relying only on the databases), we embed the `owl:sameAs` axioms together with the axioms for symmetry and transitivity into the mappings, that is, extending the notion of $\mathcal{T}$-mappings [14] ($\mathcal{T}$ stands for terminology). Intuitively, $\mathcal{T}$-mappings embed the consequences from a OWL QL ontology into the mappings. This allow us to drop the implicit axioms for symmetry, and transitivity from the Tbox $\mathcal{T}$.

For each category $C$ and for each set of non-empty tables $L^C_{i_1,i_2} L^C_{i_2,i_3} \dots L^C_{i_{n-1},i_n}$, if $L^C_{i_1,i_n}$ *does not exist*, we include the following transitivity mappings in $\mathcal{M}$:

$$t_1(\{id_1\})\ \texttt{owl:sameAs}\ t_n(\{id_n\}) \leftarrow select * from\ L^C_{i_1,i_2} \bowtie \cdots \bowtie L^C_{i_{n-1},i_n} \quad (2)$$

and for each of the `owl:sameAs` mapping described in (1) and (2) we include the following symmetry mappings in $\mathcal{M}$:

$$t_j(\{id_j\})\ \texttt{owl:sameAs}\ t_i(\{id_i\}) \leftarrow select * from\ sql_{ij} \quad (3)$$

We call the resulting set of mappings $\mathcal{M}_S$

**(c) Rewriting the query $Q$:** Encoding reflexivity would be extremely detrimental for performance, not only by the large number of extra mappings we should consider but also because it would render the optimizations explained in the next sections ineffective. Intuitively, the reason for this is that while symmetry and transitivity affect only elements which are linked to other datasets, reflexivity affects *all* the objects in the OBDA setting. Thus, we would not be able to distinguish during the query transformation process, which classes and properties actually deal with linked objects (and should be rewritten) and which ones are not. Therefore, we modify the query-rewriting technique to keep soundness and completeness with respect to the DL entailment regime while evaluating the query under the QL entailment regime over $(\mathcal{T}, \mathcal{M}_S, D)$.

We modify the query translation as follows:

**Definition 2 ($(\varphi(Q), V)$).** *Given a query $(Q, V)$, the query $(\varphi(Q), V)$ is obtained by replacing every triple pattern $t$ in $Q$ with $\varphi(t)$, where: $\varphi(\{$`?v :P ?w`$\})$ is shown in Fig. 4 (A) and $\varphi(\{$`?v rdf:type :C`$\})$ is shown in Fig. 4 (B).*

```
{ ?v :P ?w . } UNION {
?v owl:sameAs _:z1 . _z1 :P ?w .
} UNION {
?v :P _:z2 . _:z2 owl:sameAs ?w .
} UNION {
?v owl:sameAs _:a .
_:b owl:sameAs ?w . _:a :P _:b . }
```

(A)

```
?v rdf:type :C . UNION {
?v owl:sameAs [ rdf:type :C ] .
}
```

(B)

**Fig. 4.** SPARQL translation to handle `owl:sameAs` without Reflexivity

Intuitively, following up our running example, the first BGP in Fig. 4 (A) gets all triples such as (`uri1(a1)`, `:hasName`, `A`) that do not need equality reasoning. The second BGP, will get triples such as (`uri1(a1)`, `:hasName`, `C`), that require `owl:sameAs`(`uri1(a1)`,`uri2(b2)`). The two last BGPs are used *only* for object properties, and it tackles the cases where equality reasoning is needed for the object (?w).

Recall that we do not allow `owl:sameAs` in the user query language. Therefore the user will not be able to query $?x$ `owl:sameAs`$?x$. In principle, we could also move transitivity and symmetry to the query, but it will not reduce the SQL query rewriting.

**Theorem 1.** *Given OBDA setting* $(\mathcal{T}, \mathcal{A}_S, \mathcal{M}, D)$ *and a query* $(Q, V)$*, we have that* $[\![Q]\!]^{DL}_{\mathcal{T},G_{\mathcal{M},D}\cup\mathcal{A}_S}|_V = [\![\varphi(Q)]\!]^{QL}_{\mathcal{T},G_{\mathcal{M}_S,D}}|_V$.

## 6 Optimization

The technique presented in Section 5 can cause excessive overhead on the query size and therefore on the query execution time, since it has to extend *every* triple pattern with `owl:sameAs` statements. In this section we show how to remove the `owl:sameAs` statements that do not contribute to the answer. For instance, in our running example the property `hasLicense` is defined over the companies in $D_4$, which are not linked with the other 3 databases. Thus, the `owl:sameAs` statements should not contribute to "populate" this property.

To translate SPARQL to SQL, in the literature [15] and in the implementation, we encode the SPARQL algebra tree as a logic program. Intuitively, each SPARQL operator is represented by a rule in the program as illustrated in Example 4. The translation algorithm employs a well-known process in Logic Programming called *partial evaluation* [10]. Intuitively, the partial evaluation of a SPARQL query $Q$ (represented as a logic program) is another query $Q'$, that represents the *partial execution* of $Q$. This process iterates over the structure of the query and *specializes* the query going from the highly abstract query to the concrete SQL query over the database. It starts by replacing the atoms that correspond to leaves in the algebra tree (triple patterns) with the union of *all* its definitions in the mappings, and then it iterates over remaining atoms trying to replace the atoms by their definitions. This procedure is done without executing any SQL query over the databases.

```
Select * WHERE {
?v :hasLicense ?w .
}
```

(A)

```
Select * WHERE {
{?v :hasLicense ?w .} UNION {
?v owl:sameAs [ :hasLicense :w ] . } }
```

(B)

**Fig. 5.** Optimizable Queries

We detect and remove `owl:sameAs` statements that do not contribute to the answer using this procedure. It is critical to notice that this optimization can be performed because we intentionally added two constraints: *(i)* we disallow mappings modeling reflexivity; and *(ii)* we force unique URIs for each pair of category/database. We illustrate this optimization in the following example.

*Example 4 (Companies).* Consider the query asking for the list of companies and licenses shown in Figure 5 (A). This query is translated into the query (fragment) shown in Figure 5 (B). Since we know that only wellbore are linked through the different datasets, it is clear that there is no need for `owl:sameAs` statements (nor unions) in this query. In the following, we show how the system *partially evaluates* the query to remove such pointless union. This translated query is represented as the following program encoding the SPARQL algebra tree:

```
(1)answer(v,w)← union(v,w)
(2)  union(v,w)← bgp₁(v,w)
(3)       bgp₁(v,w) ← hasLicense(v,w)
(4)  union(v,w)← bgp₂(v,w)
(5)       bgp₂(v,w) ← owl:sameAs(v,x), hasLicense(x,w)
```

The next step is to replace the leaves of the SPARQL tree (the triple patterns `owl:sameAs` and `hasLicense` ) with their definitions (fragment without including transitivity and symmetry):

```
(6) hasLicense(uri4(v),uri4(w))← sql(v,w)
(7) owl:sameAs(uri1(v),uri2(x)) ← T12(v,w)
(8) owl:sameAs(uri2(v),uri3(x)) ← T23(v,w)
(9) owl:sameAs(uri1(v),uri3(x)) ← T13(v,w)
```

Thus, the system try to replace `hasLicense`(x,w) in (5) by its definition in (6), and analogously with `owl:sameAs` (5 by the union of 7-9) Using partial evaluation, the system will try to unify the head of (6) with `hasLicense` in (5). The result is:

```
(5')bgp₂(v,uri4(w)) → owl:sameAs(v,uri4(x)), sql(uri4(x),uri4(w))
```

In the next step, the algorithm will try to unify the `owl:sameAs` in $(5')$ with the head of at least one of the rules (7), (8), (9) (if all matched, it would add the union of the tree). Given that the URI template (represented as a function) *uri*4 does not occur in any of the rules, the whole branch will be removed. The resulting program is:

```
(1)answer(v,w)→ union(v,w)
(2)      union(v,w)→ bgp₁(v,w)
(4)          bgp₁(v,w) → hasLicense(v,w)
(5)                hasLicense(uri4(v),uri4(w))→ sql(v,w)
```

This query without `owl:sameAs` overhead is now ready to be translated into SQL. □

This process will also take care of eliminating unnecessary SQL queries used to define `owl:sameAs`. For instance, if the user queries for wellbores, it will remove all the SQL queries used for linking company names. This is why we require a unique URI for each pair category/dataset.

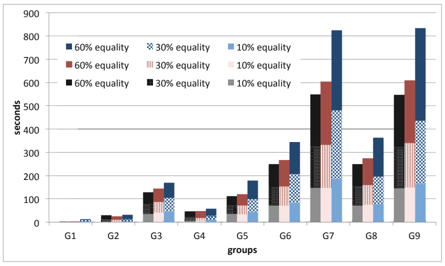
## 7 Experiments

In this section we present a sets of experiments evaluating the performance of queries over crossed-linked datasets. We integrated EPDS and the NPD fact pages at Statoil extending the existing ontology and the set of mappings, and creating the linking tables. We ran 22 queries covering real information needs of end-users over this integrated OBDA setting. Since EPDS is a production server with confidential data, and its loads changes constantly, and in addition the OBDA setting is too complex to isolate different features of this approach, we also created a controlled OBDA environment in our own server to perform a careful study our technique. In addition, we exported the triples of this controlled environment and load them into the commercial triple store Stardog[8] (v3.0.1).

To perform the controlled experiments, we setup an OBDA cross-linked environment based on the Wisconsin Benchmark [5].[9] The Wisconsin benchmark was designed for the systematic evaluation of database performance with respect to different query characteristics. It comes with a schema that is designed so one can quickly understand the structure of each table and the distribution of each attribute value. This allows easy construction of queries that isolate the features that need to be tested. The schema can be used to instantiate multiple tables. These tables, which we now call "Wisconsin tables", contain 16 attributes, and a primary key.

Observe that *Ontop* does not perform SQL federation, therefore it usually relies on systems such as Teiid [10] or EXAREME [17] (a.k.a. ADP) to integrate multiple databases. These systems expose to *Ontop* a set of tables coming from the different databases. Thus, to mimic this scenario we created a single database with 10 tables: 4 Wisconsin tables, representing different datasets, and 6 linking tables. Each Wisconsin table contains 100M rows, the 6 tables occupied ca. 100GB of disk space, exposing +1.8B triples.

[8] http://stardog.com
[9] All the material to reproduce the experiments can be found online: https://github.com/ontop/ontop-examples/tree/master/iswc-crosslinked
[10] http://teiid.jboss.org



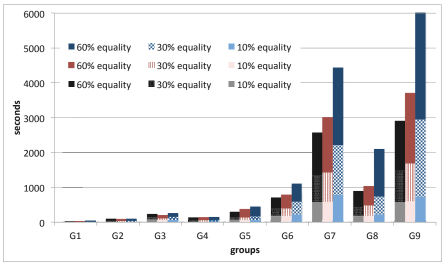


**Fig. 6.** Worst Execution Time including fetching time - 2 linked-DS (left) and 3 linked-DS (right)

The following experiments evaluate the overhead of equality reasoning when answering SPARQL queries. The variables we considered are: *(i)* Number of SPARQL joins (1-4); *(ii)* Number and type of properties (0-4 /data-object); *(iii)* Number of linked datasets (2-3); *(iv)* Selectivity of the query (0.001%, 0.01%, 0.1%); *(v)* Number of equal objects between datasets (10%,30%,60%). In total we ran 1332 queries. The SPARQL queries have the following template:

```
 SELECT *  WHERE {
?x rdf:type :Class_i . // i =1..4
?x :DataProperty_{j-1} ?y1 . ?x :DataProperty_j ?y2 . // j =0..4
?x :ObjectProperty_{k-1} ?z1 . ?x :ObjectProperty_k ?z2 . // k =0..4
Filter( ?y < k% ) }
```

where a 0 or negative subindex means that the property is not present in the query. When we evaluated 2 datasets we included equalities between elements of the classes $A_1$ and $A_2$. When we evaluated 3 datasets the equality was between $A_1$, $A_2$ and $A_4$. The class $A_3$ and the properties $S_3$ and $R_3$ are isolated. We group the queries in 9 groups: (G1) No properties (c), (G2) 1 d. prop. 0 obj. prop. (1d), (G3) 0 d. prop. 1 obj. prop. (1o),..., (G9)2 d. prop. 2 obj. prop. (2d2o).

The average start-up time is ≈5 seconds. Observe that SPARQL engines based on materialization can take hours to start-up with OWL-DL ontologies [9]. The results are summarized in Figure 6. We show the *worst* execution time in each group including the time that it takes to fetch the results.

**Discussion:** The results confirm that reasoning over OBDA-based integrated data has a high cost, but this cost is not prohibitive. The execution times at Statoil range from 3.2 seconds to 12.8 minutes, with mean 53 secs, and median 8.6 secs. An overview of the execution times are shown in Fig. 7. The most complex query had 15 triple patterns, using object and data properties coming from both data sources.

In the controlled environment, in the 2 linked-datasets scenario, with 120M equal objects (60%), even in the worst case most of the queries run in ≈ 5min. The query that performs the worst in this setting, (4 joins, 2 data properties,

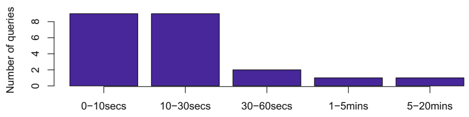


**Fig. 7.** Overview of query execution times for tests on EPDS at Statoil.

2 object properties, $10^5$ selectivity) returns 480.000 results, and takes $\approx$ 13min. When we move to the 3 linked-datasets scenario, most executions (again worst time in every group) take around than 15min. In this case, the worst query in $G9$ takes around 1.5hs and returns 1.620.000 results. One can see that the number of linked datasets is the variable that impacts the most on the query performance. The second variable is the number of object properties since its translation is more complex than the one for data properties. The third variable, is the selectivity. It is worth noticing that these results measure an almost pathological case taking the system to its very limit. In practice, it is unlikely that 60% of the all the objects of a 300M integrated dataset will be equal and belong to the same category. Recall that if they are not in the same category, the optimization presented in Section 6 removes the unnecessary SQL subqueries. For instance, in the integration of EPDS and NPD there are less than 10.000 equal wellbores and there are millions of objects of different categories. Moreover, even 1.5hs is a reasonable time. Recall that Statoil users required weeks to get an answer for this sort of queries.

Because of the partial evaluation-based optimizations proposed in Section 6, with 2 datasets 30 out of 48 queries (52 out of 100 with 3 datasets) get optimized and executed in a few milliseconds. These queries are the ones that join elements in $A_{1,2,4}$ (3 datasets) with $A3$, $S_3$ and $R_3$ elements. Since there is no equality between these elements, neither through `owl:sameAs`, nor with standard equality, the SPARQL translation produces an empty SQL, and no SQL query gets executed returning automatically 0 answers.

To load the data into Stardog we used *Ontop* to materialize the triples. The materialization took 11hs, and it took another 4hs to load the triples into Stardog. The default semantics that Stardog gives to `owl:sameAs` is not compliant with the official OWL semantics since "Stardog designates one canonical individual for each `owl:sameAs` equivalence set"; however, one can force Stardog to consider all the URIs in the equivalence set. Our experiments show that Stardog does not behave according to the claimed semantics. Details can be found in [3].

## 8 Related Work

The treatment of `owl:sameAs` in reasoning and query evaluation has received considerable interest in recent years. After all, many data sources in the Linked Opend Data (LOD) cloud give `owl:sameAs` links to equivalent URIs, so it would be desirable to use them. Surprisingly, to the best of our knowledge, there has been no attempt to extend OBDA to take into account `owl:sameAs`. Next we discuss several approaches that handle `owl:sameAs` trough rewriting.

Balloon Fusion [16] is a line of work that attempts to make use of `owl:sameAs` information in the LOD cloud for query answering. The approach is similar to ours in that it is based on *rewriting a query* to take into account equality inferences, before executing it. The treatment of `owl:sameAs` is semantically very incomplete however, since the rewriting only applies to URIs stated explicitly in the query. No equality reasoning is applied to the *variables* in the query, which is a main point of our work.

The question of equality handling becomes quite different in nature in the context of a single data store that is already in triple format. Equality can then be handled essentially by rewriting equal URIs to one common representative. E.g. [13] report on doing this for an in-memory triple store, while simultaneously saturating the data with respect to a set of forward chaining inference rules. Observe that in many scenarios (such as the Statoil scenario discussed here) this approach is not possible, both due to the fact that the data should be moved from the original source, and because of the amount of data that should be loaded into memory. In a query rewriting, OBDA setting, this corresponds to the idea of making sure that mappings will map equivalent entities from several sources to the same URI – which is often not practical or even impossible.

Our approach is only valid when the links between records really mean semantic identity. When the links are uncertain, query answering then requires the use of probabilistic database methods, as discussed e.g. in [7] for a limited type of queries. Extending these methods to handle arbitrary SPARQL-style queries is not trivial.

## 9 Conclusions

In this paper we showed how to represent links over database as `owl:sameAs` statements, we propose a mapping-based framework that carefully constructs `owl:sameAs` statements to minimize the performance impact of equality reasoning. To recover rewritability of SPARQL into SQL we imposed a suitable set of restrictions on the linking mechanisms that are fully compatible with real world requirements, and together with the `owl:sameAs`-mappings make it possible to do the SPARQL-to-SQL translation. We showed how to answer SPARQL queries over crossed linked datasets using query transformation. and how to optimize the translation to improve the performance of the produced SQL query. To empirically support this claim, we provided an extensive set of experiments over real enterprise data, and also in a controlled environment.

**Acknowledgments.** This paper is supported by the EU under the large-scale integrating project (IP) Optique (Scalable End-user Access to Big Data), grant agreement n. FP7-318338.

## References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. of Artificial Intelligence Research **36**, 1–69 (2009)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. Autom. Reasoning **39**(3), 385–429 (2007)
3. Calvanese, D., Giese, M., Hovland, D., Rezk, M.: Ontology-based integration of cross-linked datasets (2015). http://www.inf.unibz.it/~mrezk/pdf/techRep-ISWC15.pdf (accessed April 30, 2015)
4. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, W3C (September 2012). http://www.w3.org/TR/r2rml/
5. DeWitt, D.J.: The wisconsin benchmark: past, present, and future. In: Gray, J. (ed.) The Benchmark Handbook. Morgan Kaufmann (1992)
6. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann (2012)
7. Ioannou, E., Nejdl, W., Niederée, C., Velegrakis, Y.: On-the-fly entity-aware query processing in the presence of linkage. PVLDB **3**(1), 429–438 (2010)
8. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Proc. of IJCAI 2011, pp. 2656–2661 (2011)
9. Kontchakov, R., Rezk, M., Rodríguez-Muro, M., Xiao, G., Zakharyaschev, M.: Answering SPARQL queries over databases under OWL 2 QL entailment regime. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 552–567. Springer, Heidelberg (2014)
10. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer-Verlag New York Inc, Secaucus (1993)
11. Marnette, B.: Generalized schema-mappings: from termination to tractability. In: PODS 2009, pp. 13–22. ACM, New York (2009)
12. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language profiles, 2nd edn. W3C Recommendation, W3C (December 2012). http://www.w3.org/TR/owl2-profiles/
13. Motik, B., Nenov, Y., Piro, R.E.F., Horrocks, I.: Handling owl:sameAs via rewriting. In: Bonet, B., Koenig, S. (eds) Proc. 29th AAAI, pp. 231–237. AAAI Press (2015)
14. Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Ontology-based data access: *Ontop* of databases. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 558–573. Springer, Heidelberg (2013)
15. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. J. of Web Semantics **33**, 141–169 (2015)
16. Schlegel, K., Stegmaier, F., Bayerl, S., Granitzer, M., Kosch, H.: Balloon fusion: SPARQL rewriting based on unified co-reference information. In: Proc. of the 30th Int. Conf. on Data Engineering Workshops (ICDE 2014), pp. 254–259. IEEE (2014)
17. Tsangaris, M.M., Kakaletris, G., Kllapi, H., Papanikos, G., Pentaris, F., Polydoras, P., Sitaridi, E., Stoumpos, V., Ioannidis, Y.E.: Dataflow processing and optimization on grid and cloud infrastructures. IEEE Bull. on Data Engineering **32**(1), 67–74 (2009)