Ontology-Based Data Access: *Ontop* of Databases

Mariano Rodríguez-Muro[1], Roman Kontchakov[2] and Michael Zakharyaschev[2]

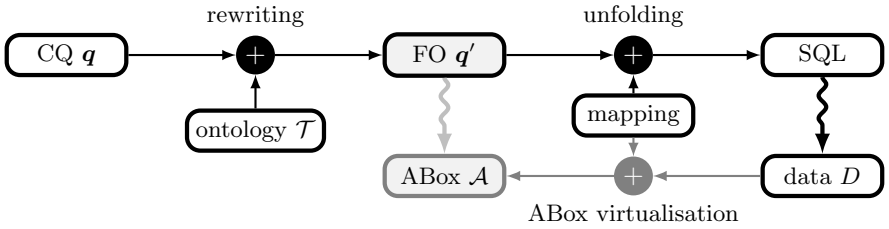
[1] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
[2] Department of Computer Science and Information Systems, Birkbeck, University of London, U.K.

Abstract. We present the architecture and technologies underpinning the OBDA system *Ontop* and taking full advantage of storing data in relational databases. We discuss the theoretical foundations of *Ontop*: the tree-witness query rewriting, $\mathcal{T}$-mappings and optimisations based on database integrity constraints and SQL features. We analyse the performance of *Ontop* in a series of experiments and demonstrate that, for standard ontologies, queries and data stored in relational databases, *Ontop* is fast, efficient and produces SQL rewritings of high quality.

1 Introduction

Ontology-based data access (OBDA) [6, 11, 22] is regarded as a key ingredient for the new generation of information systems, especially for Semantic Web applications that involve large amounts of data. In the OBDA paradigm, an ontology defines a high-level global schema and provides a vocabulary for user queries, thus isolating the user from the details of the structure of data sources (which can be relational databases, triple stores, datalog engines, etc.). The OBDA system transforms user queries into the vocabulary of the data and then delegates the actual query evaluation to the data sources.

In this paper, we concentrate on OBDA with ontologies given in OWL 2 QL, a profile of OWL 2 designed to support rewriting of conjunctive queries (CQs) over ontologies into first-order (FO) queries. A standard architecture of such an OBDA system over relational data sources can be represented as follows:



The user is given an OWL 2 QL ontology $\mathcal{T}$ and can formulate CQs $\boldsymbol{q}(\boldsymbol{x})$ in the signature of $\mathcal{T}$. The system rewrites $\boldsymbol{q}$ and $\mathcal{T}$ into an FO-query $\boldsymbol{q}'(\boldsymbol{x})$, called a *rewriting* of $\boldsymbol{q}$ and $\mathcal{T}$, such that $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$ iff $\mathcal{A} \models \boldsymbol{q}'(\boldsymbol{a})$, for any set $\mathcal{A}$ of ground atoms (called an ABox) in the signature of $\mathcal{T}$ and any tuple $\boldsymbol{a}$

of individuals in $\mathcal{A}$. A number of different rewriting techniques have been proposed and implemented for OWL 2 QL (PerfectRef [22], Presto/Prexto [27, 26], Rapid [5], the tree-witness rewriting [15]) and its extensions ([16], Nyaya [9], Requiem/Blackout [20, 21], Clipper [7]).

The rewriting $\boldsymbol{q}'$ is formulated in the signature of $\mathcal{T}$ and, before evaluation, has to be further transformed into the vocabulary of the data source D. For instance, $\boldsymbol{q}'$ can be *unfolded* into an SQL query by means of a GAV mapping $\mathcal{M}$ relating the signature of $\mathcal{T}$ to the vocabulary of D. Strangely enough, mappings and unfoldings have largely been ignored by query rewriting algorithms (with Mastro-I [22] being an exception), partly because the data was assumed to be given as an ABox (say, as a universal table in a database or as a triple store). We consider the query transformation process as consisting of two steps—query rewriting and unfolding—and argue that this brings practical benefits (even in the case of seemingly trivial mappings for universal tables or triple stores).

The performance of first OBDA systems based on the architecture above was marred by large rewritings that could not be processed by RDBMSs, which led the OBDA community to intensive investigations of rewriting techniques and optimisations. There are 3 main reasons for large CQ rewritings and unfoldings:

(E) Sub-queries of $\boldsymbol{q}$ with existentially quantified variables can be folded in many different ways to match the canonical models of possible knowledge bases $(\mathcal{T}, \mathcal{A})$, all of which must be reflected in the rewriting $\boldsymbol{q}'$.

(H) Classes/properties occurring in $\boldsymbol{q}$ can have many subclasses/subproperties according to $\mathcal{T}$, which all have to be included in the rewriting $\boldsymbol{q}'$.

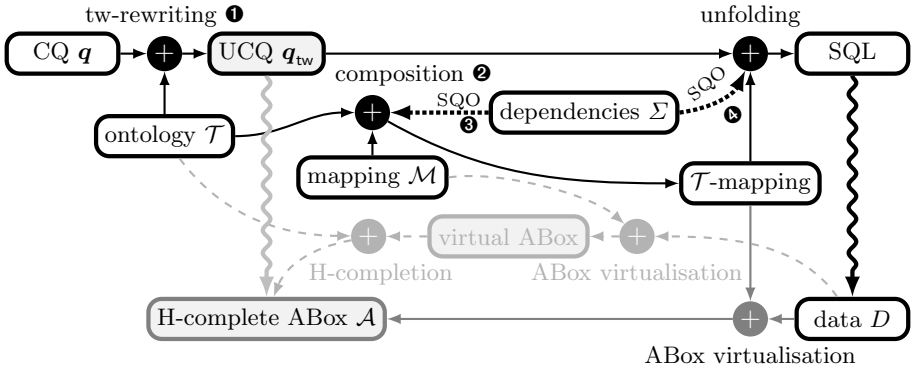
(M) The mapping $\mathcal{M}$ can have multiple definitions of the ontology terms, which may result in an exponential blowup when $\boldsymbol{q}'$ is unfolded into a (most suitable for RDBMSs) union of Select-Project-Join queries.

In fact, most of the proposed techniques produce rewritings in the form of unions of CQs (UCQs) and try to tame (E) using various optimisations in unification strategies to reduce the size of rewritings, with expensive CQ containment as the last resort. Presto [27] and the tree-witness rewriting [15] use nonrecursive datalog to deal with (H); this, however, is of little help if a further transformation to a UCQ is required. The combined approach [17] constructs finite representations of (in general) infinite canonical models of $(\mathcal{T}, \mathcal{A})$ thereby totally removing (H). It also solves (E) for ontologies without role inclusions; otherwise, rewritings can still be of exponential size, or the filtering procedure [19] may have to run exponentially many times.

In theory, (E) turns out to be incurable under the architecture above: there are CQs and OWL 2 QL ontologies for which any FO- (or nonrecursive datalog) rewriting is superpolynomial (or exponential) [13, 14], which happens independently of the contribution of (H) and (M); the polynomial rewriting of [10] hides this blowup behind extra existential quantifiers. Fortunately, it seems that only (artificially) complex CQs and ontologies trigger issues with (E). Our experiments show that, for standard benchmark CQs and ontologies, the number of foldings in (E) is small and can be efficiently dealt with by the tree-witness rewriting.

In this paper, we attack both (H) and (M) at *the same time* using two key observations. First, the schema and integrity constraints (dependencies), Σ, of the data source D together with the mapping $\mathcal{M}$ often provide valuable information about the class of possible ABoxes over which the user CQ is rewritten. (These ABoxes are *virtual* representations of D and are not materialised.) For example, if we know that all our virtual ABoxes $\mathcal{A}$ are $\exists$-*complete* with respect to $\mathcal{T}$ (that is, contain witnesses for all $\exists R$ in $\mathcal{T}$) then we can ignore (E); if all $\mathcal{A}$ are *H-complete* (that is, $\mathcal{A}$ contains $A(a)$ whenever it contains $B(a)$ and $\mathcal{T} \models B \sqsubseteq A$, and similarly for properties) then the problem (H) does not exist. Second, we can make the virtual ABoxes H-complete by taking the composition of $\mathcal{T}$ and $\mathcal{M}$ as a new mapping. This composition, called a $\mathcal{T}$-*mapping* [24], can be simplified with the help of Σ and the features of the target query language before being used in the unfolding. As the simplifications use Σ, they preserve correct answers only over database instances satisfying Σ. (Even if the mappings are trivial and the data comes from a universal table or a triple store, it often has a certain structure and satisfies certain constraints, which could be taken into account to make query answering more efficient [12]).

These observations underpin the system *Ontop* (`ontop.inf.unibz.it`) implemented at the Free University of Bozen-Bolzano and available as a plugin for Protégé 4, SPARQL end-point and OWLAPI and Sesame libraries. The process of query rewriting and unfolding in *Ontop* with all optimisations is shown below (the dashed lines show processes that aid explanations but do *not* take place):



This architecture, which is our main theoretical contribution, will be discussed in detail in Section 2. Here we only emphasise the key ingredients:

- ❶ the tree-witness rewriting $\boldsymbol{q}_{\text{tw}}$ assumes the virtual ABoxes to be H-complete; it separates the topology of $\boldsymbol{q}$ from the taxonomy defined by $\mathcal{T}$, is fast in practice and produces short UCQs;
- ❷ the $\mathcal{T}$-mapping combines the system mapping $\mathcal{M}$ with the taxonomy of $\mathcal{T}$ to ensure H-completeness of virtual ABoxes;
- ❸ the $\mathcal{T}$-mapping is simplified using the Semantic Query Optimisation (SQO) technique and SQL features; the $\mathcal{T}$-mapping is constructed and optimised for the given $\mathcal{T}$ and Σ only once, and is used to unfold all rewritings $\boldsymbol{q}_{\text{tw}}$;
- ❹ the unfolding algorithm uses SQO to produce small and efficient SQL queries.

In Section 3, we evaluate the performance of *Ontop* and compare it with other systems using a number of standard ontologies, including LUBM with generated data and the Movie Ontology with real data. Our experimental results show that UCQ rewritings over arbitrary ABoxes are not scalable in the presence of class and property hierarchies; in contrast to that, rewritings of real-world queries and ontologies over H-complete ABoxes (or equivalent datalog rewritings) turn out to be unions of few (at most two, in our experiments) CQs whose size does not exceed (in fact, is often smaller than) the size of the original query. Class and property hierarchies can be tackled by optimisations of $\mathcal{T}$-mappings and the SQO, which use the structure of databases and integrity constraints, so that *Ontop* automatically produces SQL queries of reasonably high quality. As a result, *Ontop* successfully competes with and often outperforms systems based on materialisation of inferences.

2 The Architecture of *Ontop*

We begin by describing the three main ingredients of *Ontop*: the tree-witness rewriting over H-complete ABoxes, $\mathcal{T}$-mappings and the unfolding algorithm. To avoid long formulas, we use the DL parlance [2] for OWL 2 QL ontologies and the datalog notation for conjunctive queries. Thus, subclass axioms are of the form $A_1 \sqsubseteq A_2$, for concept (class) names A_i; property inclusions are $R_1 \sqsubseteq R_2$, where the R_i are role (object and datatype property) names or their inverses; and property P domain and range axioms are $\exists P \sqsubseteq A_1$ and $\exists P^- \sqsubseteq A_2$, respectively. Conjunctive queries (CQs) are of the form $\boldsymbol{q}(\boldsymbol{x}) \leftarrow \alpha_1, \dots, \alpha_n$, where $\boldsymbol{x}$ is a vector of answer variables and each α_i is a unary or binary atom (the variables in the α_i that are not in $\boldsymbol{x}$ are *existentially quantified*). Throughout the paper, we identify atoms $P^-(y, x)$ and $P(x, y)$ (in query heads, bodies and ABoxes).

Suppose we are given a CQ $\boldsymbol{q}(\boldsymbol{x})$ and an OWL 2 QL ontology $\mathcal{T}$. *Ontop* starts its work by constructing the *semantic-based* tree-witness rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes. We say that an ABox $\mathcal{A}$ is *H-complete with respect to* $\mathcal{T}$ in case it satisfies the following conditions:

$$A(a) \in \mathcal{A} \quad \text{if} \quad A'(a) \in \mathcal{A}, \mathcal{T} \models A' \sqsubseteq A \quad \text{or} \quad R(a, b) \in \mathcal{A}, \mathcal{T} \models \exists R \sqsubseteq A,$$
$$P(a, b) \in \mathcal{A} \quad \text{if} \quad R(a, b) \in \mathcal{A} \text{ and } \mathcal{T} \models R \sqsubseteq P.$$

2.1 Tree-Witness Rewriting over H-Complete ABoxes

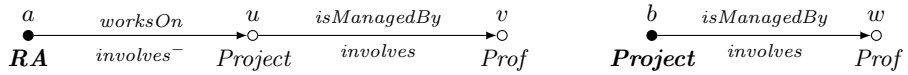
We explain the essence of the tree-witness rewriting using an example; a formal definition can be found in [25]. Consider an ontology $\mathcal{T}$ with the axioms

$$RA \sqsubseteq \exists worksOn.Project, \qquad Project \sqsubseteq \exists isManagedBy.Prof, \tag{1}$$
$$worksOn^- \sqsubseteq involves, \qquad isManagedBy \sqsubseteq involves, \tag{2}$$

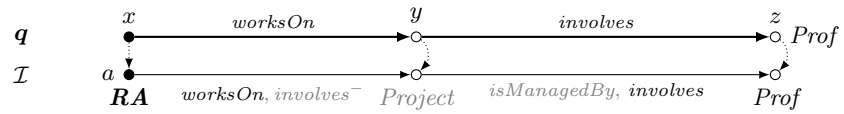
and the CQ $\boldsymbol{q}(\boldsymbol{x})$ asking to find those who work with professors:

$$\boldsymbol{q}(x) \leftarrow worksOn(x, y),\ involves(y, z),\ Prof(z).$$

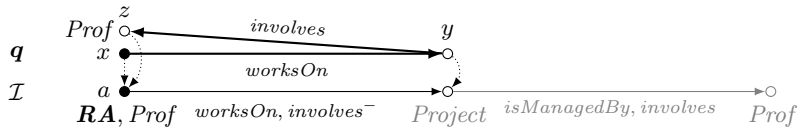
Observe that if a model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$, for some ABox $\mathcal{A}$, contains individuals $a \in RA^{\mathcal{I}}$ and $b \in Project^{\mathcal{I}}$ then $\mathcal{I}$ must also contain the following fragments:



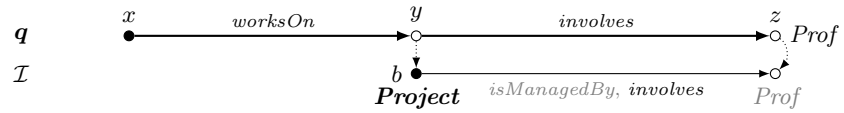
where the points u, v, w are not necessarily named individuals from the ABox, but can be (anonymous) witness for the existential quantifiers of (1) (or labelled nulls in the chase); we say that these fragments are *generated* by RA and $Project$, respectively, and use the bold-faced font to indicate that. It follows that a is an answer to $\boldsymbol{q}(x)$ whenever a is an instance of $RA^{\mathcal{I}}$, in which case the atoms of $\boldsymbol{q}$ (thick lines) are mapped to the fragment generated by RA as follows:



Alternatively, we obtain the following match (provided that a is also in $Prof^{\mathcal{I}}$):



Another option is to map x and y to ABox individuals, a and b, and if b is in $Project^{\mathcal{I}}$, then the last two atoms of $\boldsymbol{q}$ can be mapped to the anonymous part generated by $Project$:



Finally, all the atoms of $\boldsymbol{q}$ can be mapped to ABox individuals. The possible ways of mapping parts of the CQ to the anonymous part of the models are called *tree witnesses*. The tree-witnesses for $\boldsymbol{q}$ found above give the following UCQ *tree-witness rewriting* $\boldsymbol{q}_{\mathsf{tw}}(x)$ of $\boldsymbol{q}(x)$ and $\mathcal{T}$ over H-complete ABoxes:

$$
\begin{aligned}
&\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \boldsymbol{RA}(x),\\
&\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{Prof}(x), \boldsymbol{RA}(x),\\
&\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{worksOn}(x, y), \boldsymbol{Project}(y),\\
&\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{worksOn}(x, y), \mathit{involves}(y, z), \mathit{Prof}(z).
\end{aligned}
$$

(It is to be noted that $\boldsymbol{q}_{\mathsf{tw}}(x)$ is *not* a rewriting of $\boldsymbol{q}(x)$ and $\mathcal{T}$ over *all* ABoxes.)

Having computed the UCQ $\boldsymbol{q}_{\mathsf{tw}}$, *Ontop* simplifies it using two optimisations. First, it applies a subsumption algorithm to remove redundant CQs from the union: for example, the first query in the example above subsumes the second, which can be safely removed. It also reduces the size of the individual CQs in the union using the following observation: any CQ $\boldsymbol{q}$ (viewed as a set of atoms)

has the same certain answers over *H-complete* ABoxes as

$$\boldsymbol{q} \setminus \{A(x)\}, \qquad \text{if } A'(x) \in \boldsymbol{q} \text{ and } \mathcal{T} \models A' \sqsubseteq A \text{ with } A' \neq A, \tag{3}$$

$$\boldsymbol{q} \setminus \{A(x)\}, \qquad \text{if } R(x,y) \in \boldsymbol{q} \text{ and } \mathcal{T} \models \exists R \sqsubseteq A, \tag{4}$$

$$\boldsymbol{q} \setminus \{P(x,y)\}, \qquad \text{if } R(x,y) \in \boldsymbol{q} \text{ and } \mathcal{T} \models R \sqsubseteq P \text{ with } R \neq P, \tag{5}$$

Surprisingly, such a simple optimisation, especially (4) for domains and ranges, makes rewritings substantially shorter [27, 9].

We have to bear in mind, however, that in theory, the size of the resulting UCQ rewritings can be very large: there exists [13, 14] a sequence of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ generating exponentially many (in $|\boldsymbol{q}_n|$) tree witnesses, and *any* first-order (or nonrecursive datalog) rewriting of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ is of superpolynomial (or exponential) size (unless it employs $|\boldsymbol{q}_n|$-many additional existentially quantified variables [10]). On the other hand, to generate many tree witnesses, the CQ $\boldsymbol{q}$ must have many subqueries that can be matched in the canonical models, which requires both $\boldsymbol{q}$ and $\mathcal{T}$ to be quite sophisticated, with $\boldsymbol{q}$ 'mimicking' parts of the canonical models for $\mathcal{T}$. To the best of our knowledge, this never happens in real-world CQs and ontologies used for OBDA. More often than not, they do not generate tree witnesses at all; see Section 3.1. It is also known [15, Theorem 21] that, if the query and ontology do not contain fragments as in the example considered above, then the number of tree witnesses is polynomial.

2.2 Optimising $\mathcal{T}$-Mappings

In a typical scenario for *Ontop*, the data comes from a relational database rather than an ABox. A database schema [1] contains predicate symbols (with their arity) for both stored database relations and views (with their definitions in terms of stored relations) as well as a set Σ of integrity constraints (in the form of inclusion and functional dependencies). Any instance $\boldsymbol{I}$ of the database schema must satisfy its integrity constraints Σ. The vocabularies of a database schema and an ontology are linked together by means of mappings. We define a *mapping*, $\mathcal{M}$, as a set of GAV rules of the form

$$S(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{z}),$$

where S is a class or property name in the ontology and $\varphi(\boldsymbol{x}, \boldsymbol{z})$ a conjunction of atoms with database relations (both stored relations and views) and a *filter*, that is, a Boolean combination of built-in predicates such as $=$ and $<$. (Note that, by including views in the schema, we can express any SQL query in mappings.) Given a mapping $\mathcal{M}$ and a data instance $\boldsymbol{I}$, the ground atoms

$$S(\boldsymbol{a}), \qquad \text{for } S(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{z}) \text{ in } \mathcal{M} \text{ and } \boldsymbol{I} \models \exists \boldsymbol{z}\, \varphi(\boldsymbol{a}, \boldsymbol{z}),$$

comprise the ABox, $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$, which is called the *virtual ABox* for $\mathcal{M}$ over $\boldsymbol{I}$. We can now define *certain answers* to a CQ $\boldsymbol{q}$ over an ontology $\mathcal{T}$ linked by a mapping $\mathcal{M}$ to a database instance $\boldsymbol{I}$ as certain answers to $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A}_{\boldsymbol{I},\mathcal{M}})$.

The tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ works only for H-complete ABoxes. An obvious way to define such ABoxes is to take the composition $\mathcal{M}^{\mathcal{T}}$ of $\mathcal{M}$ and the inclusions in $\mathcal{T}$ given by

$$\begin{aligned} A(x) &\leftarrow \varphi(x, \boldsymbol{z}) && \text{if } A'(x) \leftarrow \varphi(x, \boldsymbol{z}) \in \mathcal{M} \text{ and } \mathcal{T} \models A' \sqsubseteq A, \\ A(x) &\leftarrow \varphi(x, y, \boldsymbol{z}) && \text{if } R(x, y) \leftarrow \varphi(x, y, \boldsymbol{z}) \in \mathcal{M} \text{ and } \mathcal{T} \models \exists R \sqsubseteq A, \\ P(x, y) &\leftarrow \varphi(x, y, \boldsymbol{z}), && \text{if } R(x, y) \leftarrow \varphi(x, y, \boldsymbol{z}) \in \mathcal{M} \text{ and } \mathcal{T} \models R \sqsubseteq P \end{aligned}$$

(we do not distinguish between $P^-(y, x)$ and $P(x, y)$). Thus, to compute answers to $\boldsymbol{q}$ over $\mathcal{T}$ with $\mathcal{M}$ and a database instance $\boldsymbol{I}$, it suffices to evaluate the rewriting $\boldsymbol{q}_{\mathsf{tw}}$ over $\mathcal{A}_{\boldsymbol{I},\mathcal{M}^{\mathcal{T}}}$: for any $\boldsymbol{I}$ and any tuple $\boldsymbol{a}$ of individuals in $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$,

$$(\mathcal{T}, \mathcal{A}_{\boldsymbol{I},\mathcal{M}}) \models \boldsymbol{q}(\boldsymbol{a}) \quad \text{iff} \quad \mathcal{A}_{\boldsymbol{I},\mathcal{M}^{\mathcal{T}}} \models \boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{a}). \tag{6}$$

Given a CQ $\boldsymbol{q}$ and an ontology $\mathcal{T}$, most OBDA systems first construct a rewriting of $\boldsymbol{q}$ and $\mathcal{T}$ over *arbitrary* ABoxes and then unfold it, using a mapping $\mathcal{M}$, into a union of Select-Project-Join (SPJ) queries, which is forwarded for execution to an RDBMS. By (6), the same result can be achieved by unfolding a rewriting over H-complete ABoxes with the help of the composition $\mathcal{M}^{\mathcal{T}}$. In principle, this may bring some benefits if the SQL query is represented as a union of SPJ queries over views for class and property names, but only if the RDBMS can evaluate such queries efficiently (each view is a union of simple queries, for rules in $\mathcal{M}^{\mathcal{T}}$ listing subclasses and subproperties). On the other hand, there will be no benefit if the query is unfolded into a union of SPJ queries either by the RDBMS or by the OBDA system itself. However, the resulting query will produce duplicating answers if the ontology axioms express the same properties of the application domain as the integrity constraints of the database [23].

For this reason, before applying $\mathcal{M}^{\mathcal{T}}$ to unfold the tree-witness rewriting in *Ontop*, we optimise the mapping using the database integrity constraints Σ. This allows us to (*a*) reduce redundancy in answers, and (*b*) substantially shorten the SQL queries. We say that a mapping $\mathcal{M}$ is a *$\mathcal{T}$-mapping over Σ* if the ABox $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$ is H-complete with respect to $\mathcal{T}$, for any data instance $\boldsymbol{I}$ satisfying Σ. (The composition $\mathcal{M}^{\mathcal{T}}$ is trivially a $\mathcal{T}$-mapping over any Σ.)

To illustrate the optimisations, we take a simplified IMDb (`www.imdb.com/interfaces`) whose schema contains relations $\mathit{title}[m, t, y]$ with information about movies (ID, title, production year), and $\mathit{castinfo}[p, m, r]$ with information about movie casts (person ID, movie ID, person role), and an ontology MO (`www.movieontology.org`) describing the application domain in terms of, for example, classes *mo:Movie* and *mo:Person*, and properties *mo:cast* and *mo:year*:

$$\begin{aligned} \mathit{mo{:}Movie} &\equiv \exists \mathit{mo{:}title}, & \mathit{mo{:}Movie} &\sqsubseteq \exists \mathit{mo{:}year}, \\ \mathit{mo{:}Movie} &\equiv \exists \mathit{mo{:}cast}, & \exists \mathit{mo{:}cast}^- &\sqsubseteq \mathit{mo{:}Person}. \end{aligned}$$

A mapping $\mathcal{M}$ that relates the ontology terms to the database schema contains, for example, the following rules:

$$\begin{aligned} \mathit{mo{:}Movie}(m), \mathit{mo{:}title}(m, t), \mathit{mo{:}year}(m, y) &\leftarrow \mathit{title}(m, t, y), \\ \mathit{mo{:}cast}(m, p), \mathit{mo{:}Person}(p) &\leftarrow \mathit{castinfo}(p, m, r). \end{aligned}$$

Inclusion Dependencies. Suppose $\mathcal{M} \cup \{S(\boldsymbol{x}) \leftarrow \psi_1(\boldsymbol{x}, \boldsymbol{z})\}$ is a $\mathcal{T}$-mapping over Σ. If there is a more specific rule than $S(\boldsymbol{x}) \leftarrow \psi_1(\boldsymbol{x}, \boldsymbol{z})$ in $\mathcal{M}$, then $\mathcal{M}$ itself is also a $\mathcal{T}$-mapping. To discover such 'more specific' rules, we run the standard query containment check (see, e.g., [1]), but taking account of the inclusion dependencies. For example, since $\mathcal{T} \models \exists \mathit{mo{:}cast} \sqsubseteq \mathit{mo{:}Movie}$, the composition $\mathcal{M}^{\mathrm{MO}}$ of mapping $\mathcal{M}$ and MO contains the following rules for *mo:Movie*:

$$\begin{array}{rcl} \mathit{mo{:}Movie}(m) & \leftarrow & \mathit{title}(m, t, y), \\ \mathit{mo{:}Movie}(m) & \leftarrow & \mathit{castinfo}(p, m, r). \end{array}$$

The latter is redundant as IMDb contains the foreign key (inclusion dependency)

$$\forall m \left(\exists p, r\ \mathit{castinfo}(p, m, r) \rightarrow \exists t, y\ \mathit{title}(m, t, y)\right).$$

Disjunctions in SQL. Another way to reduce the size of a $\mathcal{T}$-mapping is to identify pairs of rules whose bodies are equivalent up to *filters w.r.t. constant values*. This optimisation deals with the rules introduced due to the so-called type (discriminating) attributes [8] in database schemas. For example, the mapping $\mathcal{M}$ for IMDb and MO contains six rules for subclasses of *mo:Person*:

$$\begin{array}{c} \mathit{mo{:}Actor}(p) \leftarrow \mathit{castinfo}(c, p, m, r), (r = 1), \\ \cdots \\ \mathit{mo{:}Editor}(p) \leftarrow \mathit{castinfo}(c, p, m, r), (r = 6). \end{array}$$

Then the composition $\mathcal{M}^{\mathrm{MO}}$ contains six rules for *mo:Person* that differ only in the last condition $(r = k)$, $1 \leq k \leq 6$. These can be reduced to a single rule:

$$\mathit{mo{:}Person}(p) \leftarrow \mathit{castinfo}(c, p, m, r), (r = 1) \vee \cdots \vee (r = 6).$$

Note that such disjunctions lend themselves to efficient evaluation by RDBMSs.

Materialised ABoxes and Semantic Index. In addition to working with proper relational data sources, *Ontop* supports ABox storage in the form of structureless *universal tables*: a binary relation $\mathit{CA}[\mathit{id}, \mathit{class\text{-}id}]$ and a ternary relation $\mathit{RA}[\mathit{id}_1, \mathit{id}_2, \mathit{property\text{-}id}]$ represent class and property membership assertions. The universal tables give rise to trivial mappings, and *Ontop* implements a technique, the *semantic index* [24], that takes advantage of SQL features in $\mathcal{T}$-mappings for this scenario. The key observation is that since the IDs in the universal tables *CA* and *RA* can be chosen by the system, each class and property name in the TBox $\mathcal{T}$ can be assigned a numeric *index* and a set of numeric *intervals* in such a way that the resulting $\mathcal{T}$-mapping contains simple SQL queries with interval filter conditions. For example, in IMDb, we have

$$\mathit{mo{:}Actor} \sqsubseteq \mathit{mo{:}Artist}, \quad \mathit{mo{:}Artist} \sqsubseteq \mathit{mo{:}Person}, \quad \mathit{mo{:}Director} \sqsubseteq \mathit{mo{:}Person};$$

so we can choose index 1 and interval [1,1] for *mo:Actor*, 2 and [1,2] for *mo:Artist*, 3 and [3,3] for *mo:Director* and 6 and [1,6] for *mo:Person*. This will generate a

$\mathcal{T}$-mapping with, for instance,

$$\begin{aligned} \mathit{mo{:}Person}(p) &\leftarrow \mathit{CA}(p, \mathit{class\text{-}id}), (1 \leq \mathit{class\text{-}id} \leq 6), \\ \mathit{mo{:}Artist}(p) &\leftarrow \mathit{CA}(p, \mathit{class\text{-}id}), (1 \leq \mathit{class\text{-}id} \leq 2). \end{aligned}$$

So, by choosing appropriate class and property IDs, we effectively construct H-complete ABoxes *without* the expensive forward chaining procedure (and the need to store large amounts of derived assertions). On the other hand, the semantic index $\mathcal{T}$-mappings are based on range expressions that can be evaluated efficiently by RDBMSs using standard B-tree indexes [8].

2.3 Unfolding with Semantic Query Optimisation (SQO)

The unfolding procedure [22] applies SLD-resolution to $\boldsymbol{q}_{\mathsf{tw}}$ and the $\mathcal{T}$-mapping, and returns those rules whose bodies contain only database atoms (cf. partial evaluation [18]). *Ontop* applies SQO [4] to rules obtained at the intermediate steps of unfolding. In particular, it eliminates redundant self-JOIN operations caused by reification of database relations by means of classes and properties. Consider, for example, the CQ

$$\boldsymbol{q}(t, y) \leftarrow \mathit{mo{:}Movie}(m), \mathit{mo{:}title}(m, t), \mathit{mo{:}year}(m, y), (y > 2010).$$

It has no tree witnesses, and so $\boldsymbol{q}_{\mathsf{tw}} = \boldsymbol{q}$. By straightforwardly applying the unfolding to $\boldsymbol{q}_{\mathsf{tw}}$ and the $\mathcal{T}$-mapping $\mathcal{M}$ above, we obtain the query

$$\boldsymbol{q}'_{\mathsf{tw}}(t, y) \leftarrow \mathit{title}(m, t_0, y_0), \mathit{title}(m, t, y_1), \mathit{title}(m, t_2, y), (y > 2010),$$

which requires two (potentially) expensive JOIN operations. However, by using the primary key m of *title*:

$$\begin{aligned} &\forall m \, \forall t_1 \forall t_2 \left(\exists y \, \mathit{title}(m, t_1, y) \land \exists y \, \mathit{title}(m, t_2, y) \rightarrow (t_1 = t_2) \right), \\ &\forall m \, \forall y_1 \forall y_2 \left(\exists t \, \mathit{title}(m, t, y_1) \land \exists t \, \mathit{title}(m, t, y_2) \rightarrow (y_1 = y_2) \right) \end{aligned}$$

(a functional dependency with determinant m), we reduce two JOIN operations in the first three atoms of $\boldsymbol{q}'_{\mathsf{tw}}$ to a single atom $\mathit{title}(m, t, y)$:

$$\boldsymbol{q}''_{\mathsf{tw}}(t, y) \leftarrow \mathit{title}(m, t, y), (y > 2010).$$

Note that these two JOIN operations were introduced to reconstruct the ternary relation from its reification by means of the roles *mo:title* and *mo:year*.

The role of SQO in OBDA systems appears to be much more prominent than in conventional RDBMSs, where it was initially proposed to optimise SQL queries. While some of the SQO techniques reached industrial RDBMSs, it never had a strong impact on the database community because it is costly compared to statistics- and heuristics-based methods, and because most SQL queries are written by highly-skilled experts (and so are nearly optimal anyway). In OBDA scenarios, in contrast, SQL queries are generated automatically, and so SQO becomes the only tool to avoid redundant and expensive JOIN operations [28].

A & S	a_1	a_2	a_3	a_4	a_5	s_1	s_2	s_3	s_4	s_5
tree witnesses	1	1	0	1	0	0	0	0	0	0
CQs in $\boldsymbol{q}_{\mathsf{tw}}$	**2**	**2**	**1**	**2**	**1**	**1**	**1**	**1**	**1**	**1**
atoms in $\boldsymbol{q}$	2	3	5	3	5	1	3	5	5	7
atoms in $\boldsymbol{q}_{\mathsf{tw}}$	2+2	1+3	5	2+3	5	1	1	3	2	4

$\text{LUBM}^{\exists}_{20}$	r_1	r_2	r_3	r_4	r_5	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
tree witnesses	0	0	0	0	0	1	1	0	1	0	0	0	3	1
CQs in $\boldsymbol{q}_{\mathsf{tw}}$	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**1**	**2**	**1**	**1**	**1**	**1**	**1**
atoms in $\boldsymbol{q}$	2	3	6	3	4	8	4	6	8	5	8	13	13	34
atoms in $\boldsymbol{q}_{\mathsf{tw}}$	2	1	4	1	2	4+6	3+4	5	5+8	4	6	12	6	33

Table 1. Tree-witness UCQ rewritings over H-complete ABoxes.

3 Experiments

In this section, we present the results of experiments conducted to evaluate the performance of *Ontop* in comparison with other systems (for details see `sites.google.com/site/ontopiswc13`). We begin by testing the tree-witness rewriter.

3.1 Tree Witnesses: The Topology of *Ontop* Rewritings

We ran the *Ontop* tree-witness rewriter on the usual set of ontologies and CQs: Adolena (A) and StockExchange (S) [20] with the original queries a_1–a_5 and s_1–s_5, respectively, and $\text{LUBM}^{\exists}_{20}$ [19] with queries r_1–r_5 from the Requiem evaluation [20], q_1–q_6 from the combined approach evaluation [19], and q_7–q_9 from the Clipper evaluation [30]. Our aim was to understand the size of the *topological* part of the rewritings that reflects matches into the anonymous part of the canonical models (as opposed to the taxonomical one). Table 1 shows the number of tree witnesses, the number of CQs in the rewriting, and the number of atoms in the input query and in each of the CQs of the rewriting.

Note that these CQs and ontologies have very few tree witnesses. More precisely, in 67% of the cases there are *no tree witnesses* at all, and in 29% we have only one. Even for the specially designed q_8, the structure of tree witnesses is simpler than in our example from Section 2.1 (e.g., they do not overlap). And although q_8 and q_9 do have tree witnesses, the resulting UCQs contain only one CQ since these tree witnesses are generated by other atoms of the queries. In fact, all tree-witness rewritings in our experiments contain at most two CQs: one of them is an optimised original CQ (in particular, by the domain/range optimisation (4) in s_2–s_5, r_2–r_5, q_1, q_3, q_5–q_8) and the other is obtained by replacing the atoms of the tree-witness with its generator. Thus, each of the CQs in the rewritings is not larger than the input query and has a very similar structure.

To illustrate, consider the following subquery of q_8:

$$\boldsymbol{q}_0(x_0) \leftarrow \mathit{Publication}(x_0),\ \mathit{publicationAuthor}(x_0, x_{11}),\ \mathit{Subj1Professor}(x_{11}),\\ \mathit{worksFor}(x_{11}, x_{12}),\ \mathit{Department}(x_{12}),$$

	r_1	r_2	r_3	r_4	r_5	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
UCQ (number of CQs)														
Requiem	2	1	23	2	10	DNF	2	DNF	14,880	690	DNF	DNF	DNF	DNF
Nyaya	2	1	23	2	10	DNF	2	DNF	DNF	690	DNF	DNF	DNF	DNF
IQAROS	2	1	23	2	10	DNF	**1**	15,120	14,400	690	23,552	DNF	DNF	DNF
Rapid	2	1	23	2	10	3,887	2	15,120	14,880	690	23,552	DNF	1	16
datalog (number of non-taxonomical rules)														
Rapid	1	1	1	1	1	2	3	1	2	1	1	1	27	1
Clipper	1	1	1	1	1	8	7	1	5	1	1	1	512	16
tw-rewriter	1	1	1	1	1	2	2	1	2	1	1	1	1	1

Table 2. The size of rewritings over $\text{LUBM}^{\exists}_{20}$ (DNF = Did Not Finish in 600s).

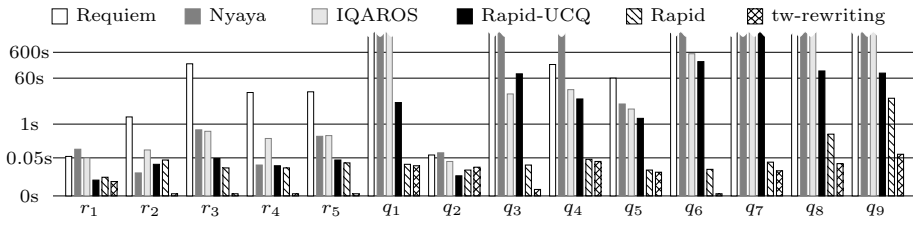


Fig. 1. Rewriting time for queries over $\text{LUBM}^{\exists}_{20}$.

where x_{11}, x_{12} do not occur in the rest of q_8. This CQ has a tree witness comprising the last two atoms because of the $\text{LUBM}^{\exists}_{20}$ axiom $\mathit{Faculty} \sqsubseteq \exists \mathit{worksFor}$. However, *Subj1Professor* is a subclass of *Faculty*, and so any of its instances is always connected to *Department* by *worksFor* (either in the ABox or in the anonymous part). Thus, the last two atoms of $\boldsymbol{q}_0$ do not affect its answers and can be removed. The first atom is redundant by (4) with the domain axiom $\exists \mathit{publicationAuthor} \sqsubseteq \mathit{Publication}$, which results in the following rewriting: $\boldsymbol{q}'_0(x_0) \leftarrow \mathit{publicationAuthor}(x_0, x_{11}),\ \mathit{Subj1Professor}(x_{11})$. As $\boldsymbol{q}_0$ represents a natural and common pattern for expressing queries—select a *Publication* whose *publicationAuthor* is a *Subj1Professor*, etc.—any OBDA system should be able to detect such redundancies automatically.

For comparison, we computed the rewritings of the CQs over $\text{LUBM}^{\exists}_{20}$ using Requiem [20], Nyaya [9], IQAROS (v 0.2) [29], Rapid (v 0.3) [5] and Clipper (v 0.1) [7]. The first four return UCQ rewritings, the numbers of CQs in which are shown in Table 2. The last two return nonrecursive datalog rewritings over *arbitrary* ABoxes. These rewritings consist of a number of 'main' rules and a number of taxonomical rules for completing the ABoxes by subclasses/subproperties; to compare with *Ontop*, Table 2 shows only the number of the 'main' rules. Interestingly, Clipper and Rapid return single-rule rewritings in the cases without tree witnesses, but generate more rules than *Ontop* (e.g., q_8 and q_9) otherwise.

Figure 1 shows the time required for rewriting (it was impossible to separate rewriting from DLV execution in Clipper, but it terminated within 1.5s on every query). The UCQ-based systems do not finish in many cases and re-

quire a substantial amount of memory (up to 1GB in some cases). In contrast, the datalog-based systems and *Ontop* produce rewritings very quickly. Observe that the rewritings returned by the four UCQ-based systems can be obtained from the tree-witness rewritings by replacing each class/property with its subclasses/subproperties (IQAROS's rewritings of q_2 and possibly q_4 are incorrect): for instance, q_7 gives 216,000 ($= 30^3 \times 2^3$) CQs, q_3 gives 15,120 ($=4 \times 5 \times 21 \times 36$) CQs and q_1 gives 3,887 ($= 23 + 2 \times 4 \times 21 \times 23$) CQs as *Student*, *Faculty* and *Professor* have 23, 36 and 30 subclasses, respectively, *worksFor* has 2 subproperties, etc. Such an operation (if needed) could be performed in fractions of seconds.

The experiments reported in this section imply that dealing efficiently with class/property hierarchies is the most critical component of any OBDA system. We discuss how *Ontop* copes with this task in the next section.

3.2 $\mathcal{T}$-mappings: Class and Property Hierarchies

We compare the query execution time in *Ontop*, Stardog 1.2 [21] and OWLIM [3]. Both Stardog and OWLIM use internal data structures to store RDF triples. Stardog is based on rewriting into UCQs (as we saw above, such systems can run out of memory during the rewriting stage, even before accessing the data). OWLIM is based on inference materialisation (forward chaining); but the implemented algorithm is known to be incomplete for OWL 2 QL [3].

It was impossible to compare *Ontop* with other systems: Rapid and IQAROS are just query rewriting algorithms; Clipper (v 0.1) supports only the DLV datalog engine that reads queries and triples at the same time (which would be a serious disadvantage for large datasets). The experiments were run on an HP Proliant with 24 Intel Xeon 6-core 3.47GHz CPUs, 106GB RAM and a 1TB@15000rpm HD under 64-bit Ubuntu 12.04 with Java 7, MySQL 5.6 and DB2 10.1.

Data as Triples: The Semantic Index We first compare the performance of the three systems for the case where the data is stored in the form of triples. In this case, *Ontop* uses universal tables, and the SQO optimisations do not play any role. We took $\text{LUBM}^{\exists}_{20}$ with the data created by the modified LUBM data generator [19] for 50, 200 and 1000 universities (5% incompleteness) with 7m, 29m and 143m triples, respectively.

OWLIM requires a considerable amount of time for loading and materialising the inferences—14min, 1h 23min and 8h 4min, respectively—expanding the data by 93% and obtaining 13m, 52m and 252m triples. Neither Stardog nor *Ontop* need this expensive loading stage. The results of executing the queries from Section 3.1 are given in Table 3 (in order to reduce the influence of the result size, which are quite large in some cases, we executed queries that counted the number of distinct tuples rather than returned the tuples themselves). We note first that Stardog runs out of memory on 50% of the queries, with a likely cause being the query rewriting algorithm, which is an improved version of Requiem (cf. Table 2). On the remaining queries, Stardog is fast, which is probably due

		r_1	r_2	r_3	r_4	r_5	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
							50 universities								
Ontop	DB2	**0**	**0.03**	**0.50**	**0.01**	**0**	25.2	0.47	0.39	**0.04**	**1.37**	**0.07**	0.51	**0.13**	**0**
Ontop	MySQL	**0**	0.19	3.76	0.08	**0**	31.0	2.48	10.54	0.13	4.22	2.19	0.48	**0.13**	**0**
OWLIM		0.01	0.78	2.43	0.28	0.17	**12.9**	2.68	**0.21**	0.29	3.95	0.78	**0.23**	0.23	0.04
Stardog		0.01	0.79	1.16	0.34	0.10	DNF	**0.10**	DNF	DNF	DNF	DNF	DNF	DNF	0.04
result size		–	102k	12k	34k	–	1.2m	–	89	–	205k	–	–	–	–
							200 universities								
Ontop	DB2	**0**	**0.08**	7.33	**0.07**	**0**	522.9	1.75	3.48	**0.12**	**5.52**	**0.26**	**0.86**	**0.25**	**0**
Ontop	MySQL	**0**	1.21	14.6	0.32	**0**	260.4	9.30	34.02	0.49	16.11	8.45	1.66	0.54	**0**
OWLIM		0.01	3.10	9.28	0.94	0.79	**46.4**	10.52	**0.89**	15.15	16.91	3.32	0.92	0.92	0.05
Stardog		0.01	3.22	**2.92**	1.12	0.27	DNF	**0.33**	DNF	DNF	DNF	DNF	DNF	DNF	0.06
result size		–	410k	48k	137k	–	4.6m	–	399	–	825k	–	–	–	–
							1000 universities								
Ontop	DB2	**0**	**0.24**	**11.6**	**0.19**	**0**	2761	3.29	11.3	**0.58**	**12.7**	**1.15**	**5.38**	**1.23**	**0**
Ontop	MySQL	**0**	1.54	70.9	0.85	**0**	1232	90.9	185.3	2.37	132.7	2.86	7.75	2.48	**0**
OWLIM		0.01	18.9	63.6	6.40	3.38	**308**	65.7	**5.11**	94.3	105.7	2.76	5.84	5.79	0.10
Stardog		0.21	20.7	13.7	9.36	1.11	DNF	**3.07**	DNF	DNF	DNF	DNF	DNF	DNF	0.17
result size		–	2m	239k	685k	–	23m	–	2k	–	4.1m	–	–	–	–

Table 3. Query execution time (in seconds) and the result size over $\text{LUBM}^{\exists}_{20}$.

to its optimised triple store. Unlike Stardog, both OWLIM and *Ontop* return answers to all queries, and their performance is comparable. In fact, in 83% of the cases *Ontop* with DB2 outperforms OWLIM.

It is to be emphasised that *Ontop* can work with a variety of database engines and that, as these experiments demonstrate, *Ontop* with MySQL in many case is worse in executing queries than with DB2 (but is still competitive with OWLIM). Two techniques turned out to be crucial to improve the performance of the engines. First, in the universal relations $CA[id, \textit{class-id}]$ and $RA[id_1, id_2, \textit{property-id}]$, we store integer URI identifiers rather than URIs themselves, with a special relation $URI[id, uri]$ serving as a dictionary to de-reference the URI identifiers. Second, a significant improvement of performance was achieved by creating indexes on sequences of attributes of the universal relations: for example, CA has indexes on (*id*, *class-id*), (*id*) and (*class-id*). The full impact of such indexes on storing data in the form of RDF triples is yet to be investigated.

Finally, we observe that some queries do not need evaluation because *Ontop* simplifies them to empty queries: in fact, r_1, r_5 and q_9 contain atoms that have no instances in the generated data, and only 6 out of the 14 CQs return any answers (which probably reflects the artificial nature of the benchmark).

These experiments confirm once again that rewritings into UCQs over arbitrary ABoxes can be prohibitively large even for high-performance triple stores such as Stardog. The materialisation approach should 'by definition' cope with large taxonomies. We have demonstrated that the semantic index used in *Ontop*

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
atoms in query	4	15	6+f	4+f	9	6	8	6	9+f	3+f
UCQ rewriting	2	48	2	1	24	2	4	16	4	1
tables in SQL	3	13	3	2	6	4	5	4	8	2
result size	6	14,688	15,010	2,224	1,921	84	4	59,211	48	26
Stardog result size	0	0	4,047	0	9	0	0	27,804	0	26
Ontop-DB2	**0.003**	0.626	**0.495**	**0.355**	7.525	**0.005**	**0.001**	**0.699**	0.167	**0.358**
Ontop-MySQL	0.005	6.138	0.679	0.571	9.190	0.009	0.023	3.563	0.460	0.457
OWLIM	0.005	**0.562**	5.413	2.833	**0.681**	0.009	0.007	4.307	**0.046**	0.836
Stardog	0.030	1.136	1.329	2.227	1.389	0.029	0.038	1.277	0.409	0.584

Table 4. Query and rewriting metrics, result sizes and execution times (in seconds).

is able to deal with this problem as efficiently as (and often better than) inference materialisation, without the considerable overhead expense of the latter.

***Ontop* of Databases** We now evaluate the performance of the $\mathcal{T}$-mapping approach to answering queries over OWL 2 QL ontologies with mappings to real-world databases. We use the Movie Ontology (MO, `www.movieontology.org`) and the data from the SQL version of the Internet Movie Database (IMDb, `www.imdb.com/interfaces`). Both the database and ontology were developed independently by third parties for purposes different from benchmarking; the mapping was created by the *Ontop* development team. MO has 137 class and property names and 157 inclusion axioms; the mapping contains 271 rules and the virtual ABox has 42m assertions. We tested 10 natural queries to IMDb: e.g., q_3 retrieves the companies from East Asia and the movies they produced between 2006 and 2010.

The metrics of the queries and their rewritings, the numbers of returned tuples, and the execution times by *Ontop* with DB2 and MySQL, OWLIM and Stardog over the materialised ABox are shown in Table 4. The line 'atoms in query' gives the number of atoms in the input query (+f denotes a FILTER expression). Each query coincides with its tree-witness rewriting (there are no tree witnesses, and none of the atoms is redundant). The line 'UCQ rewriting' shows the number of CQs in the rewritings over *arbitrary* ABoxes, which reflects the size of class and property hierarchies. The resulting SQL query contains a single SELECT-PROJECT-JOIN component with the number of tables given by 'tables in SQL'—this corresponds to the number of JOINs in the SQL query. Because of the SQO, the SQL queries have fewer tables and JOINs than the original one (or the rewriting). For example, q_3 with 6 atoms produces a single SPJ query with 3 tables (and one disjunction over 7 country codes rather than 7 subqueries):

```
SELECT DISTINCT Q3.name, Q1.title, Q1.production_year
FROM title Q1, movie_companies Q2, company_name Q3
WHERE (Q1.id = Q2.movie_id) AND (Q2.company_id = Q3.id) AND
  (('[tw]' = Q3.country_code) OR ... OR ('[kr]' = Q3.country_code)) AND
  (Q1.production_year <= 2010) AND (Q1.production_year >= 2006)
```

Note that Stardog, on the same set of triples as OWLIM, returns *fewer* tuples

in *all cases* but q_{10}, which may explain the better execution times (one of the Stardog optimisations [21] removes empty CQs from the rewriting and may be responsible for the missing tuples).

In 70% cases, *Ontop* with DB2 outperforms OWLIM (and is efficient even with MySQL). Moreover, OWLIM takes 45min to load the data into the triple store (and will have to do this again every time the data is changed). This demonstrates that on-the-fly inference over real-world databases by means of the tree-witness rewriting and $\mathcal{T}$-mappings is efficient enough to successfully compete with materialisation-based techniques. Moreover, the usual problems associated with query-rewriting-based approaches disappear in *Ontop*: $\mathcal{T}$-mappings efficiently deal with hierarchical reasoning avoiding the exponential blowup, and the SQO improves the performance of the produced SQL queries by taking account of the structure and integrity constraints of the database.

4 Conclusions

To conclude, we believe this paper shows that—despite the negative theoretical results on the worst-case OWL 2 QL query rewriting and sometimes disappointing experiences of the first OBDA systems—high-performance OBDA is achievable in practice when applied to real-world ontologies, queries and data stored in relational databases. In such cases, query rewriting together with SQO and SQL optimisations is fast, efficient and produces SQL queries of high quality.

Acknowledgements. We thank G. Orsi for providing Nyaya, G. Xiao for providing queries and the *Ontop* development team (J. Hardi, T. Bagosi, M. Slusnys) for the help with the experiments. This work was supported by the EU FP7 project Optique (grant 318338) and UK EPSRC project ExODA (EP/H05099X).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
3. Bishop, B., Bojanov, S.: Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM. In: Proc. of OWLED 2011. CEUR-WS, vol. 796 (2011)
4. Chakravarthy, U.S., Fishman, D.H., Minker, J.: Semantic query optimization in expert systems and database systems. Benjamin-Cummings Publ. Co., Inc. (1986)
5. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: Proc. of CADE-23. LNCS, vol. 6803, pp. 192–206. Springer (2011)
6. Dolby, J., Fokoue, A., Kalyanpur, A., Ma, L., Schonberg, E., Srinivas, K., Sun, X.: Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In: Proc. of ISWC 2008. LNCS, vol. 5318, pp. 403–418. Springer (2008)
7. Eiter, T., Ortiz, M., Šimkus, M., Tran, T.K., Xiao, G.: Query rewriting for Horn-SHIQ plus rules. In: Proc. of AAAI 2012. AAAI Press (2012)

8. Elmasri, R., Navathe, S.: Fundamentals of Database Systems. Addison-Wesley, 6th edn. (2010)
9. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization. In: Proc. of ICDE 2011. pp. 2–13. IEEE Computer Society (2011)
10. Gottlob, G., Schwentick, T.: Rewriting ontological queries into small nonrecursive datalog programs. In: Proc. of KR 2012. AAAI Press (2012)
11. Heymans, S., Ma, L., Anicic, D., Ma, Z., Steinmetz, N., Pan, Y., Mei, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Feier, C., Hench, G., Wetzstein, B., Keller, U.: Ontology reasoning with large data repositories. In: Ontology Management, Semantic Web, Semantic Web Services, and Business Applications, pp. 89–128. Springer (2008)
12. Kementsietsidis, A., Bornea, M., Dolby, J., Srinivas, K., Dantressangle, P., Udrea, O., Bhattacharjee, B.: Building an efficient RDF store over a relational database. In: Proc. of SIGMOD 2013. ACM (2013)
13. Kikot, S., Kontchakov, R., Podolskii, V., Zakharyaschev, M.: Exponential lower bounds and separation for query rewriting. In: Proc. of ICALP 2012, Part II. LNCS, vol. 7392, pp. 263–274. Springer (2012)
14. Kikot, S., Kontchakov, R., Podolskii, V., Zakharyaschev, M.: Query rewriting over shallow ontologies. In: Proc. of DL 2013. CEUR-WS, vol. 1014 (2013)
15. Kikot, S., Kontchakov, R., Zakharyaschev, M.: Conjunctive query answering with OWL 2 QL. In: Proc. of KR 2012. AAAI Press (2012)
16. König, M., Leclère, M., Mugnier, M.L., Thomazo, M.: A sound and complete backward chaining algorithm for existential rules. In: Proc. of RR. Springer (2012)
17. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in DL-Lite. In: Proc. of KR 2010. AAAI (2010)
18. Lloyd, J., Shepherdson, J.: Partial Evaluation in Logic Programming. The Journal of Logic Programming 11(3–4), 217–242 (1991)
19. Lutz, C., Seylan, İ., Toman, D., Wolter, F.: The combined approach to OBDA: Taming role hierarchies using filters. In: Proc. of SSWS+HPCSW 2012. (2012)
20. Pérez-Urbina, H., Motik, B., Horrocks, I.: A comparison of query rewriting techniques for DL-lite. In: Proc. of DL 2009. CEUR-WS, vol. 477 (2009)
21. Pérez-Urbina, H., Rodríguez-Díaz, E., Grove, M., Konstantinidis, G., Sirin, E.: Evaluation of query rewriting approaches for OWL 2. In: Proc. of SSWS+HPCSW 2012. CEUR-WS, vol. 943 (2012)
22. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. Journal on Data Semantics 10, 133–173 (2008)
23. Rodríguez-Muro, M.: Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics. Ph.D. thesis, Free Univ. of Bozen-Bolzano (2010)
24. Rodríguez-Muro, M., Calvanese, D.: Dependencies: Making ontology based data access work. In: Proc. of AMW 2011. CEUR-WS, vol. 749 (2011)
25. Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M.: Query rewriting and optimisation with database dependencies in Ontop. In: Proc. of DL 2013. (2013)
26. Rosati, R.: Prexto: Query rewriting under extensional constraints in DL-Lite. In: Proc. of EWSC 2012. LNCS, vol. 7295, pp. 360–374. Springer (2012)
27. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: Proc. of KR 2010. AAAI Press (2010)
28. Sequeda, J., Miranker, D.: Ultrawrap: SPARQL execution on relational data. Tech. Rep. TR-12-10, Dept. of Computer Science, University of Texas at Austin (2012)
29. Venetis, T., Stoilos, G., Stamou, G.: Query extensions and incremental query rewriting for OWL 2 QL ontologies. Journal on Data Semantics (2013)
30. Xiao, G.: Personal communication (2013)