Ontop at Work

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 describe the architecture of the OBDA system *Ontop* and analyse its performance in a series of experiments. We 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

In this paper, we report on a series of experiments designed to test the performance of the ontology-based data access (OBDA) system *Ontop*[1] implemented at the Free University of Bozen-Bolzano. Our main concern was the quality of the query rewritings produced automatically by *Ontop* when given some standard queries, ontologies, databases and mappings from the database schemas to the ontologies.

Recall [4] that, in the OBDA paradigm, an ontology defines a high-level global schema of (already existing) data sources and provides a vocabulary for user queries. An OBDA system rewrites such queries into the vocabulary of the data sources and then delegates query evaluation to a relational database management system (RDBMS). The existing query rewriting systems include QuOnto [19], Nyaya [9], Rapid [7], Requiem [17]/Blackout [18], Clipper [8], Prexto [22] and the system of [14] (some of which use datalog engines rather than RDBMSs).

To illustrate how an OBDA system works, we take a simplified IMDb database (`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). The users are not supposed to know the structure of the database. Instead, they are given an ontology, say MO (`www.movieontology.org`), describing the application domain in terms of concepts (classes), such as *mo:Movie* and *mo:Person*, and roles and attributes (object and datatype properties), such as *mo:cast* and *mo:year*:

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

(we use the description logic parlance of *OWL 2 QL*). The user can query the data in terms of concepts and roles of the ontology; for example,

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

[1] `http://ontop.inf.unibz.it`

is a query asking for the titles of recent movies with their production year. To rewrite it to an SQL query over the data source, the OBDA system requires a mapping that relates the ontology terms to the database schema; for example:

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

By evaluating this mapping over a data instance with, say,

title

m	t	y
728	'Django Unchained'	2012

castinfo

p	m	r
n37	728	1
n38	728	1

we obtain the ground atoms

$$\mathit{mo{:}Movie}(728),\ \mathit{mo{:}title}(728, \text{'Django Unchained'}), \mathit{mo{:}year}(728, 2012),$$
$$\mathit{mo{:}Person}(\text{n37}),\ \mathit{mo{:}cast}(728, \text{n37}),\ \mathit{mo{:}Person}(\text{n38}),\ \mathit{mo{:}cast}(728, \text{n38})$$

that can be thought of as the ABox over which we can execute the query $\boldsymbol{q}(t, y)$ taking account of the consequences implied by the MO ontology. Such an ABox is not materialised and called *virtual* [21].

Thus, the OBDA system is facing three tasks: it has to (*i*) rewrite the original query to a query over the virtual ABox, (*ii*) unfold the rewriting, using the mapping, into an SQL query, and then (*iii*) evaluate it over the data instance using an RDBMS. The idea of OBDA stems from the empirical fact that answering conjunctive queries (CQs) in RDBMSs is very efficient in practice. So one can expect task (*iii*) to be smooth provided that the rewriting (*i*) and unfolding (*ii*) are reasonably small and standard.

However, the available experimental data (see, e.g., [20, 3]) as well as the recent complexity-theoretic analysis of rewritings show that they can be prohibitively large or complex. First, there exist CQs and ontologies for which any (first-order or datalog) rewriting results in an exponential blowup [12]; the polynomial datalog rewriting of [10] hides this blowup behind the existential quantification over special constants. Second, even for simple and natural ontologies and CQs, rewritings (*i*) become exponential when presented as (most suitable for RDBMSs) unions of CQs (UCQs) because they must include all sub-concepts/roles of each atom in the query induced by the ontology.

In *Ontop*, this bottleneck is tackled by making use of

- the tree-witness rewriting [13] that separates the *topology* of the CQ from the *taxonomy* defined by the ontology;
- an extended mapping (called a $\mathcal{T}$-mapping [21]) that takes account of the taxonomy and can be optimised using database integrity constraints and SQL features;
- an unfolding algorithm that employs the semantic query optimisation technique with database integrity constraints to produce small and efficient SQL queries.

For example, a rewriting of the query $\boldsymbol{q}(t, y)$ above can be split into the CQ

$$\boldsymbol{q}'(t, y) \leftarrow \mathit{ext{:}Movie}(m),\ \mathit{mo{:}title}(m, t),\ \mathit{mo{:}year}(m, y),\ (y > 2010)$$

and the datalog rules for the *ext:Movie* predicate:

$$\mathit{ext{:}Movie}(m) \leftarrow \mathit{mo{:}Movie}(m), \tag{1}$$
$$\mathit{ext{:}Movie}(m) \leftarrow \mathit{mo{:}cast}(m, p), \tag{2}$$
$$\mathit{ext{:}Movie}(m) \leftarrow \mathit{mo{:}title}(m, t). \tag{3}$$

The former inherits the topology of the original CQ, while the latter represents the taxonomy defined by the ontology. In theory, the topological part can contain exponentially many rules (reflecting possible matches in the canonical models) [12], but this never happens in practice, and usually there are very few of them (see the experiments below). The taxonomical component is independent from the CQ and combines with the mapping into a $\mathcal{T}$-mapping [21], which can then be drastically simplified using the database integrity constraints. For example, since *castinfo* has a foreign key (its movie ID attribute references ID in *title*), every virtual ABox of IMDb will satisfy the axiom $\exists \mathit{mo{:}cast} \sqsubseteq \mathit{mo{:}Movie}$, making (2) redundant; moreover, (3) and (1) give rise to the same rule, resulting in a $\mathcal{T}$-mapping with a single rule for *mo:Movie*. Thus, a rewriting *over IMDb ABoxes* will be a single CQ. In contrast, any UCQ rewriting over arbitrary ABoxes contains three CQs which simply duplicate the answers because the data respects the integrity constraints (a query with a few more atoms may give rise to a UCQ rewriting with thousands CQs).

By straightforwardly applying the unfolding algorithm to $\boldsymbol{q}'$ and the $\mathcal{T}$-mapping $\mathcal{M}$ above, we obtain the query

$$\boldsymbol{q}''_0(t, y) \leftarrow \mathit{title}(m, t_0, y_0),\ \mathit{title}(m, t, y_1),\ \mathit{title}(m, t_2, y),\ (y > 2010),$$

which requires two (potentially) expensive JOIN operations. However, if we use the fact that the ID attribute is a primary key of *title* (uniquely defining the title and production year), then $\boldsymbol{q}'$ can be unfolded into a much simpler

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

In fact, such multiple JOINs are very typical in OBDA because n-ary relations of data sources are reified by ontologies into binary roles and attributes.

The aim of this paper is to (*i*) present the rewriting and optimisation techniques that allow *Ontop* to produce optimised queries as discussed above, and (*ii*) evaluate the performance of *Ontop* using three use cases. We demonstrate that—at least in these cases—*Ontop* produces query rewritings of reasonably high quality and its performance is comparable to that of traditional RDBMSs.

2 *OWL 2 QL* and Databases

The language of *OWL 2 QL* contains *individual names* a_i, *concept names* A_i, and *role names* P_i ($i \geq 1$). *Roles* R and *basic concepts* B are defined by

$$R \quad ::= \quad P_i \quad | \quad P_i^-, \qquad\qquad B \quad ::= \quad \bot \quad | \quad A_i \quad | \quad \exists R.$$

A *TBox* (or an *ontology*), $\mathcal{T}$, is a finite set of *inclusions* of the form

$$B_1 \sqsubseteq B_2, \qquad B_1 \sqsubseteq \exists R.B_2, \qquad B_1 \sqcap B_2 \sqsubseteq \bot, \qquad R_1 \sqsubseteq R_2, \qquad R_1 \sqcap R_2 \sqsubseteq \bot.$$

An *ABox*, $\mathcal{A}$, is a set of atoms of the form $A_k(a_i)$ or $P_k(a_i, a_j)$. The semantics for *OWL 2 QL* is defined in the usual way based on interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ [2]. The set of individual names in $\mathcal{A}$ is denoted by $\mathsf{ind}(\mathcal{A})$.

A *conjunctive query* $\boldsymbol{q}(\boldsymbol{x})$ is a first-order formula $\exists \boldsymbol{y}\, \varphi(\boldsymbol{x}, \boldsymbol{y})$, where φ is a conjunction of atoms of the form $A_k(t_1)$ or $P_k(t_1, t_2)$, and each t_i is a *term* (an individual or a variable in $\boldsymbol{x}$ or $\boldsymbol{y}$). We use the datalog notation for CQs, writing $\boldsymbol{q}(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{y})$ (without existential quantifiers), and call $\boldsymbol{q}$ the *head* and φ the *body* of the rule. A tuple $\boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A})$ is a *certain answer* to $\boldsymbol{q}(\boldsymbol{x})$ over $(\mathcal{T}, \mathcal{A})$ if $\mathcal{I} \models \boldsymbol{q}(\boldsymbol{a})$ for all models $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$; in this case we write $(\mathcal{T}, \mathcal{A}) \models \boldsymbol{q}(\boldsymbol{a})$.

We assume that the data comes from a relational database rather than an ABox. We view databases [1] as triples $(\boldsymbol{R}, \Sigma, \boldsymbol{I})$, where $\boldsymbol{R}$ is a database schema, containing predicate symbols for both stored database relations and views (together with their definitions in terms of stored relations), Σ is a set of integrity constraints over $\boldsymbol{R}$ (in the form of inclusion and functional dependencies), and $\boldsymbol{I}$ is a data instance over $\boldsymbol{R}$ (satisfying Σ). The vocabularies of $\boldsymbol{R}$ and $\mathcal{T}$ are linked together by means of mappings. A *mapping*, $\mathcal{M}$, from $\boldsymbol{R}$ to $\mathcal{T}$ is a set of (GAV) rules of the form

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

where S is a concept or role name in $\mathcal{T}$ and $\varphi(\boldsymbol{x}, \boldsymbol{z})$ a conjunction of atoms with stored relations and views from $\boldsymbol{R}$ 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 $\mathcal{M}$, the atoms $S(\boldsymbol{a})$, for $S(\boldsymbol{x}) \leftarrow \varphi(\boldsymbol{x}, \boldsymbol{z})$ in $\mathcal{M}$ 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 a TBox $\mathcal{T}$ linked by a mapping $\mathcal{M}$ to a database $(\boldsymbol{R}, \Sigma, \boldsymbol{I})$ as certain answers to $\boldsymbol{q}$ over $(\mathcal{T}, \mathcal{A}_{\boldsymbol{I},\mathcal{M}})$.

3 The Architecture of *Ontop*

We now briefly describe the main ingredients of *Ontop*: the tree-witness rewriting over complete ABoxes, $\mathcal{T}$-mappings and the unfolding algorithm. Suppose we are given a CQ $\boldsymbol{q}$ over an ontology $\mathcal{T}$ and a mapping $\mathcal{M}$ from a database schema $\boldsymbol{R}$ to $\mathcal{T}$. The tree-witness rewriting of $\boldsymbol{q}$ and $\mathcal{T}$, denoted $\boldsymbol{q}_{\mathsf{tw}}$, presupposes that the underlying ABox $\mathcal{A}$ is *H-complete with respect to* $\mathcal{T}$ in the sense that

$$S(\boldsymbol{a}) \in \mathcal{A} \quad \text{whenever} \quad S'(\boldsymbol{a}) \in \mathcal{A} \text{ and } \mathcal{T} \models S' \sqsubseteq S,$$

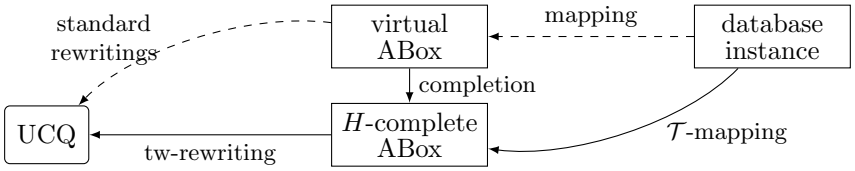
for all concept names S and basic concepts S' and for all role names S and roles S' (we identify $P^-(b, a)$ and $P(a, b)$ in ABoxes and assume $\exists R(a) \in \mathcal{A}$ if $R(a, b) \in A$, for some b). An obvious way to define H-complete ABoxes is to

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 identify $P^-(y, x)$ with $P(x, y)$ in the heads of the mapping rules.) 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 $\mathcal{A}_{\boldsymbol{I},\mathcal{M}^{\mathcal{T}}}$:

$$(\mathcal{T}, \mathcal{A}_{\boldsymbol{I},\mathcal{M}}) \models \boldsymbol{q}(\boldsymbol{a}) \quad \text{iff} \quad \mathcal{A}_{\boldsymbol{I},\mathcal{M}^{\mathcal{T}}} \models \boldsymbol{q}_{\mathsf{tw}}(\boldsymbol{a}), \quad \text{for any } \boldsymbol{I} \text{ and } \boldsymbol{a} \subseteq \mathsf{ind}(\mathcal{A}_{\boldsymbol{I},\mathcal{M}}).$$



OBDA systems such as QuOnto [19] and Prexto [22] first construct rewritings over *arbitrary* ABoxes and only then unfold them, using mappings, into UCQs which are evaluated by an RDBMS (dashed lines above). The same result can be obtained by unfolding rewritings over H-complete ABoxes with the help of the composition $\mathcal{M}^{\mathcal{T}}$ (solid lines above). However, in practice the resulting UCQs very often turn out to be too large [20].

In *Ontop*, we also start with $\mathcal{M}^{\mathcal{T}}$. But before applying it to unfold $\boldsymbol{q}_{\mathsf{tw}}$, we first simplify and reduce the size of the mapping by exploiting the database integrity constraints. Following [21], a mapping $\mathcal{M}$ from $\boldsymbol{R}$ to $\mathcal{T}$ is called a *$\mathcal{T}$-mapping over integrity constraints* Σ if the virtual ABox $\mathcal{A}_{\boldsymbol{I},\mathcal{M}}$ is H-complete w.r.t. $\mathcal{T}$, for any data instance $\boldsymbol{I}$ satisfying Σ. (The composition $\mathcal{M}^{\mathcal{T}}$ is a $\mathcal{T}$-mapping over any Σ.) *Ontop* transforms $\mathcal{M}^{\mathcal{T}}$ to a much simpler $\mathcal{T}$-mapping by taking account of database integrity constraints (dependencies), and SQL features such as disjunctions in filter conditions.

3.1 Tree-Witness Rewriting

We explain the essence of the tree-witness rewriting using an example. Consider an ontology $\mathcal{T}$ with the axioms

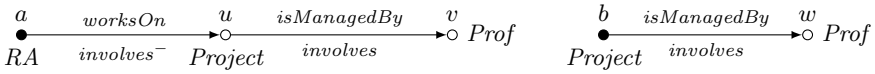
$$RA \sqsubseteq \exists worksOn.Project, \qquad Project \sqsubseteq \exists isManagedBy.Prof, \tag{4}$$

$$worksOn^- \sqsubseteq involves, \qquad isManagedBy \sqsubseteq involves \tag{5}$$

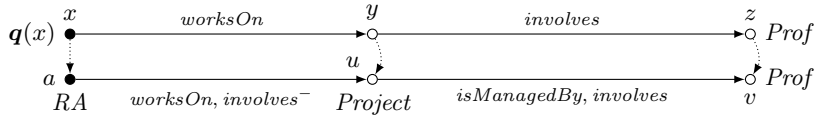
and the CQ asking to find those who work with professors:

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

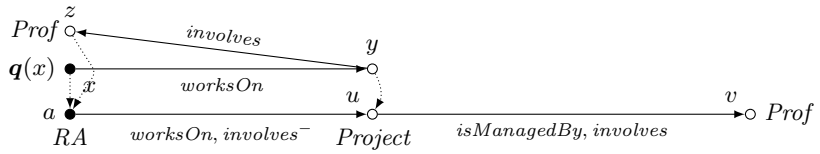
Observe that if a model $\mathcal{I}$ of $(\mathcal{T}, \mathcal{A})$, for some $\mathcal{A}$, contains individuals $a \in RA^{\mathcal{I}}$ and $b \in Project^{\mathcal{I}}$, then $\mathcal{I}$ must also contain the following fragments:



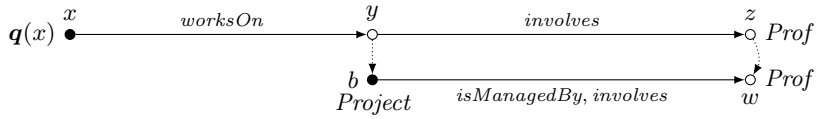
Here the points ∘ are not necessarily named individuals from the ABox, but can be generated by the axioms (4) as (anonymous) witnesses for the existential quantifiers. It follows then that a is an answer to $\boldsymbol{q}(x)$ if $a \in RA^{\mathcal{I}}$, in which case the atoms of $\boldsymbol{q}$ are mapped to the fragment generated by a as follows:



Alternatively, if a is in both $RA^{\mathcal{I}}$ and $Prof^{\mathcal{I}}$, then we obtain the following match:



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:



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 model are called *tree witnesses*. The tree witnesses for $\boldsymbol{q}$ found above give the following tree-witness rewriting $\boldsymbol{q}_{\mathsf{tw}}$ of $\boldsymbol{q}$ and $\mathcal{T}$ over H-complete ABoxes:

$$\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{worksOn}(x, y), \mathit{involves}(y, z), \mathit{Prof}(z), \tag{6}$$

$$\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{RA}(x), \tag{7}$$

$$\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{RA}(x), \mathit{Prof}(x), \tag{8}$$

$$\boldsymbol{q}_{\mathsf{tw}}(x) \leftarrow \mathit{worksOn}(x, y), \mathit{Project}(y). \tag{9}$$

(Note that $\boldsymbol{q}_{\mathsf{tw}}$ is not a rewriting over arbitrary ABoxes.)

In theory, the size of the rewriting $\boldsymbol{q}_{\mathsf{tw}}$ can be large [12]: there exists a sequence of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ generating exponentially many (in $|\boldsymbol{q}_n|$) tree witnesses, and any rewriting of $\boldsymbol{q}_n$ and $\mathcal{T}_n$ is of exponential size (unless it employs $|\boldsymbol{q}_n|$-many additional existentially quantified variables [10]). Our experiments (see Section 4) demonstrate, however, that in practice, real-world ontologies and CQs generate small and simple tree-witness rewritings.

There are two ways to simplify tree-witness rewritings further. First, we can use a subsumption algorithm to remove redundant CQs from the union: for example, (7) subsumes (8), which can be safely removed. Second, we can reduce the size of the individual CQs in the union using the following observation: for

any CQ $\boldsymbol{q}$ (viewed as a set of atoms),

$$\begin{array}{rcll} \boldsymbol{q} \cup \{A(x), A'(x)\} & \equiv_c & \boldsymbol{q} \cup \{A(x)\}, & \text{if } \mathcal{T} \models A \sqsubseteq A', \\ \boldsymbol{q} \cup \{A(x), R(x,y)\} & \equiv_c & \boldsymbol{q} \cup \{R(x,y)\}, & \text{if } \mathcal{T} \models \exists R \sqsubseteq A, \\ \boldsymbol{q} \cup \{P(x,y), R(x,y)\} & \equiv_c & \boldsymbol{q} \cup \{R(x,y)\}, & \text{if } \mathcal{T} \models R \sqsubseteq P, \end{array}$$

where $\equiv_c$ reads 'has the same certain answers over *H-complete* ABoxes' (we again identify $P^-(y,x)$ with $P(x,y)$). Surprisingly, such a simple optimisation, especially for the domain/range constraints, makes rewritings substantially shorter [23, 9].

3.2 Optimising $\mathcal{T}$-mappings

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 the mapping in the introduction 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 rule is redundant since IMDb contains the foreign key

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

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*. For example, the mapping $\mathcal{M}$ for IMDb and MO contains 6 rules for sub-concepts of *mo:Person*:

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

So, the composition $\mathcal{M}^{\mathrm{MO}}$ contains six rules for *mo:Person* that differ only in the last condition $(r=k)$, for $1 \leq k \leq 6$. 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.

3.3 Unfolding with Semantic Query Optimisation (SQO)

The unfolding procedure [19] 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 in [15]). *Ontop* applies SQO [6] to rules obtained at the intermediate steps of unfolding. In particular, this eliminates redundant JOIN operations caused by reification of database relations by means of concepts and roles. We saw in the introduction that the primary key m of *title*, i.e., following two functional dependencies with determinant m:

$$\forall m \big(\exists y\, \mathit{title}(m, t_1, y) \land \exists y\, \mathit{title}(m, t_2, y) \to (t_1 = t_2)\big),$$
$$\forall m \big(\exists t\, \mathit{title}(m, t, y_1) \land \exists t\, \mathit{title}(m, t, y_2) \to (y_1 = y_2)\big),$$

remove the two JOIN operations in $\mathit{title}(m, t_0, y_0)$, $\mathit{title}(m, t, y_1)$, $\mathit{title}(m, t_2, y)$, resulting in a single atom $\mathit{title}(m, t, y)$. Note that these two JOIN operations were introduced to reconstruct the ternary relation from its reification by means of 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 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 redundancy.

4 Experiments

We illustrate the performance of *Ontop* by three use cases. All experiments were run on Ubuntu 12.04 64-bit with an Intel Core i5 650, 4 cores@3.20GHz, 16 GB RAM and 1 TB@7200 rpm HD. We used a Java 7 virtual machine for *Ontop* with MySQL 5.5 for Cases 1 and 3, and with PostgreSQL 9.1 for Case 2. Full details of the experiments are available at `obda.inf.unibz.it/data/owled13`.

Case 1 is a simulation of a railway network for cargo delivery developed by the University of Genoa with the industrial partner Intermodal Logistics [5]. The ILog ontology, mapping and queries are used to monitor the status of the network. The case includes an ontology with 70 concepts and roles, a mapping with 43 rules and 11 queries (`www.mind-lab.it/~gcicala/isf2012`). For our experiments, we generated data for 30 days.

Case 2 uses the Movie Ontology (MO) over the real data from the Internet Movie Database (IMDb) with a mapping created by the *Ontop* development team. We use nine complex, yet natural queries, e.g.,

```
SELECT DISTINCT ?x ?title ?actor_name ?prod_year ?rating
WHERE {
    ?m a mo:Movie;
         mo:title ?title;
         mo:imdbrating ?rating;
         dbpedia:productionStartYear ?prod_year;
         mo:hasActor ?x;
         mo:hasDirector ?x .
     ?x dbpedia:birthName ?actor_name .
     FILTER ( ?rating > '7.0' && ?prod_year >= 2000 && ?prod_year <= 2010 )
}
ORDER BY desc(?rating) ?prod_year
LIMIT 25
```

(full details are available at the URL above). Most queries are of high selectivity and go beyond CQs, using inequalities, ORDER BY/LIMIT and DISTINCT operators. Both the SQL database and the ontology were developed independently by third parties (IMDb and the University of Zurich) for purposes different from benchmarking.

Case 3 is based on the Lehigh University Benchmark (LUBM, `swat.cse.lehigh.edu/projects/lubm`), which comes with an OWL ontology, 14 simple CQs of varying degree of selectivity and a data generator. We approximated the ontology in *OWL 2 QL* and created a database schema to store the data for 200 universities (1 university $\approx$ 130K assertions). Note that although the data has some degree of randomness, it is not arbitrary and follows what can be regarded as a natural pattern: each university has 15–25 departments, each department has 7–10 full professors, every person has a name, etc. These considerations were taken into account to produce a *normalised* database schema with relations of appropriate arity together with primary and foreign keys (instead of the standard universal tables storing RDF triples).

case	ontology		rules in	assertions in
	predicates	inclusions	mapping	virtual ABox
ILog	45	70	43	1m
IMDb-MO	137	157	271	42m
LUBM	75	93	79	26m

The tables below give the results of the experiments with ILog and IMDb-MO.

ILog

query	no optimisations				with optimisations					
	rewriting		unfolding		rewriting		unfolding		SQL execution	
	CQs	time	CQs	time	CQs	time	CQs	time	time	answer size
Q1	1	0.005	1	0.004	1	0.000	1	0.004	0.008	582
Q2	1	0.001	4	0.004	1	0.000	1	0.001	0.900	41,594
Q3	1	0.003	1	0.002	1	0.000	1	0.002	0.015	6,710
Q4	1	0.004	1	0.004	1	0.000	1	0.003	0.660	41,000
Q5	1	0.001	2	0.004	1	0.000	2	0.004	0.008	7
Q6	1	0.007	1	0.002	1	0.000	1	0.001	0.009	170
Q7	2	0.001	2	0.002	2	0.000	2	0.001	0.080	269
Q8	1	0.000	1	0.001	1	0.000	1	0.001	0.008	621
Q9	2	0.001	2	0.007	2	0.000	2	0.007	0.009	94
Q10	1	0.004	1	0.001	1	0.000	1	0.002	1.000	22,666

IMDb

Q1	1	0.008	2	0.003	1	0.010	1	0.003	1.652	25
Q2	1	0.001	2	0.004	1	0.001	1	0.003	3.371	53,400
Q3	1	0.004	7	0.012	1	0.004	1	0.000	1.316	10,681
Q4	1	0.000	2	0.002	1	0.001	1	0.001	0.219	25
Q5	2	0.010	126	0.037	2	0.016	3	0.006	1.335	327
Q6	1	0.001	2	0.000	1	0.000	1	0.000	0.085	69
Q7	1	0.001	4	0.001	1	0.000	1	0.001	0.070	4
Q8	1	0.000	156	0.026	1	0.000	1	0.003	4.144	3
Q9	1	0.001	1	0.001	1	0.001	1	0.001	0.365	37

We start with a few observations on the performance of query rewriting and unfolding. The size of the produced SQL queries is given in the column CQs under 'unfolding.' We can see that the *Ontop* optimisations produce very few Select-Project-Join queries in the union, quite often only one. In contrast, when the optimisations are disabled, the size of the union is in line with QuOnto, Rapid and Requiem and can grow considerably, in particular, when large hierarchies are involved (cf. Q5 and Q8 in IMDb-MO where the union contains > 100 queries). To explain this behaviour, we observe first that almost all of the queries

in ILog and IMDb-MO have *no tree witnesses*, and so the rewriting returns the original query (note that Q7, Q9 of ILog have a union in the original query). The only exception is Q5 in IMDb-MO with one tree witness, which generates two CQs in the rewriting. Second, the ratio of the number of rules in a mapping per concept/role in both scenarios is very low when our optimisations are applied: most have at most one rule (even in the case of large hierarchies with many domain/range axioms). So, the unfolding with such a mapping produces a small number of SELECT-PROJECT-JOIN queries in the union. These observations support our claim that, in practice, there are few tree witnesses and that our $\mathcal{T}$-mapping optimisations can handle efficiently concept and role hierarchies, domain and range constraints.

The time required for query rewriting and optimisation is negligible and stays within 4ms. In contrast, the time required to generate queries without optimisations is higher, especially for queries involving large hierarchies ($\geq$ 25ms): in particular, Q5 and Q8 in IMDb-MO, where our optimisations reduce the time of unfolding from 37/26ms to 3/6ms. Similarly to other systems, *Ontop* applies CQ containment (CQC) checks to reduce the number of SELECT-PROJECT-JOIN queries, and these checks prove to be costly on large unfoldings without optimisations. Although few milliseconds might seem negligible, the performance of such systems as RDF triple stores and DBs is measured in *queries per second* and is usually expected to be in thousands. With such requirements, an overhead of 20–30ms per query is not acceptable.

The execution time for SQL queries produced by *Ontop*, in MySQL or Postgres, is within 100ms for simple, high selectivity queries (with few results). Although Q2, Q4 and Q10 in ILog and Q1, Q2, Q3, Q5 and Q8 in IMDb-MO take up to 4s to execute, their SQL rewritings *are* optimal (in the sense that they coincide with hand-crafted queries), and their relatively long execution time is due to DISTINCT/ORDER BY over large relations.

LUBM

query	no optimisations				with optimisations					
	rewriting		unfolding		rewriting		unfolding		SQL execution	
	CQs	time	CQs	time	CQs	time	CQs	time	time	answer size
Q1	1	0.014	1	0.003	1	0.021	1	0.003	0.009	4
Q2	1	0.001	13	0.018	1	0.001	1	0.002	0.012	499
Q3	1	0.000	1	0.001	1	0.000	1	0.001	0.009	6
Q4	1	0.001	14	0.017	1	0.001	10	0.015	0.010	306
Q5	1	0.000	7	0.001	1	0.000	7	0.001	0.011	720
Q6	1	0.000	2	0.000	1	0.000	2	0.000	10.805	2,088,195
Q7	1	0.000	8	0.006	1	0.000	2	0.001	0.008	67
Q8	1	0.000	2	0.003	1	0.000	2	0.003	0.086	7790
Q9	1	0.001	128	0.115	1	0.000	6	0.006	17.419	54,285
Q10	1	0.000	1	0.000	1	0.000	1	0.000	0.008	4
Q11	1	0.002	1	0.001	1	0.001	1	0.000	0.010	224
Q12	1	0.000	2	0.001	1	0.001	2	0.001	0.007	12
Q13	1	0.001	31	0.015	1	0.000	13	0.001	0.009	917
Q14	1	0.000	1	0.000	1	0.000	1	0.000	4.494	1,584,743

In the LUBM case, the queries have *no tree witnesses*, which results in tree-witness rewritings that coincide with the original queries. LUBM is, however, the only case where *Ontop* generated unions with hundreds SELECT-PROJECT-JOIN queries, which is due to a higher ratio of mappings per concept/role. This is a consequence of the database structure and the way in which mappings construct

object URIs from integers and strings in the database (known as impedance mismatch [19]). The generated SQL queries are still optimal in the sense that they correspond to human-generated queries for the given database schema.

Query execution appears to be optimal for all queries (but Q6, Q9 and Q14), with response times under 12ms even for queries with JOIN and filter operations over large tables. This corresponds to the expected performance of an optimised RDBMS, in which most operations can be performed using in-memory indexes (provided that SQL queries have the right structure for the query planner). Q6, Q9 and Q14 have low selectivity (large number of results) and the execution time is dominated by disk access.

It is to be noted that although we used an *OWL 2 QL* approximation of LUBM, most queries return the same results as for the original LUBM ontology. The only exceptions are Q11 and Q12: all answers to Q11 are recovered with an extra mappings simulating transitivity (up to a predefined depth) by means of self-JOINs on the transitive property; similarly, for all answers to Q12, we include an extra mapping rule expressing $\exists R.B \sqsubseteq A$ on the elements of the virtual ABox. The execution times in the table are given for the extensions described above, which ensure completeness (w.r.t. the original LUBM) of the returned answers.

Finally, by comparing the performance of *Ontop* (see `ontop.inf.unibz.it`) with that of other open-source or commercial systems [11, 16], we see that *Ontop* is much faster than Sesame or Jena (open-source), and similar to OWLIM (commercial), but does not pay the heavy price for inference materialisation, which can take days or hours.

5 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 standard ontologies, queries and data stored in relational databases. In such cases, query rewriting together with SQO and SQL optimisations are fast, efficient and produce SQL queries of high quality.

Acknowledgements. We thank Giuseppe Cicala and Armando Taccella for helping us with setting up the ILog experiments, and the *Ontop* development team (Josef Hardi, Timea Bagosi and Mindaugas Slusnys) for their help with running the experiments and collecting the results.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
5. M. Casu, G. Cicala, and A. Tacchella. Ontology-based data access: An application to intermodal logistics. *Information Systems Frontiers*, pages 1–23, 2012.
6. U. S. Chakravarthy, D. H. Fishman, and J. Minker. *Semantic query optimization in expert systems and database systems*. Benjamin-Cummings Publishing Co., Inc., 1986.
7. A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting for OWL 2 QL. In *Proc. of CADE-23*, volume 6803 of *LNCS*, pages 192–206. Springer, 2011.
8. T. Eiter, M. Ortiz, M. Šimkus, T.-K. Tran, and G. Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI 2012*. AAAI Press, 2012.
9. G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*, pages 2–13. IEEE Computer Society, 2011.
10. G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *Proc. of KR 2012*. AAAI Press, 2012.
11. V. Khadilkar, M. Kantarcioglu, B. M. Thuraisingham, and P. Castagna. Jena-HBase: A distributed, scalable and efficient RDF triple store. In *Proc. of ISWC*, volume 914 of *CEUR-WS*, 2012.
12. S. Kikot, R. Kontchakov, V. Podolskii, and M. Zakharyaschev. Exponential lower bounds and separation for query rewriting. In *Proc. of ICALP 2012, Part II*, volume 7392 of *LNCS*, pages 263–274. Springer, 2012.
13. S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In *Proc. of KR 2012*. AAAI Press, 2012.
14. M. König, M. Leclère, M.-L. Mugnier, and M. Thomazo. A sound and complete backward chaining algorithm for existential rules. In *Proc. of RR 2012*, volume 7497 of *LNCS*, pages 122–138. Springer, 2012.
15. J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. *The Journal of Logic Programming*, 11(3-4):217–242, October 1991.
16. Ontotext. OWLIM performance with Jena, 2011. `http://www.ontotext.com/owlim/benchmark-results/owlim-jena-performance`.
17. H. Pérez-Urbina, B. Motik, and I. Horrocks. A comparison of query rewriting techniques for DL-lite. In *Proc. of DL 2009*, volume 477 of *CEUR-WS*, 2009.
18. H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, and E. Sirin. Evaluation of query rewriting approaches for OWL 2. In *Proc. of SSWS+HPCSW 2012*, volume 943 of *CEUR-WS*, 2012.
19. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
20. M. Rodríguez-Muro. *Tools and Techniques for Ontology Based Data Access in Lightweight Description Logics*. PhD thesis, KRDB Research Centre for Knowledge and Data, Free Univ. of Bozen-Bolzano, 2010.
21. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work. In *Proc. of AMW 2011*, volume 749. CEUR-WS.org, 2011.
22. R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proc. of EWSC 2012*, volume 7295 of *LNCS*, pages 360–374. Springer, 2012.
23. R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. of KR 2010*. AAAI Press, 2010.