# OBDA: Query Rewriting or Materialization? In Practice, Both!

Juan F. Sequeda[1], Marcelo Arenas[2], and Daniel P. Miranker[1]

[1]Department of Computer Science, The University of Texas at Austin
[2]Department of Computer Science, PUC Chile

**Abstract.** Given a source relational database, a target OWL ontology and a mapping from the source database to the target ontology, Ontology-Based Data Access (OBDA) concerns answering queries over the target ontology using these three components. This paper presents the development of Ultrawrap$^{\text{OBDA}}$, an OBDA system comprising bidirectional evaluation; that is, a hybridization of query rewriting and materialization. We observe that by compiling the ontological entailments as mappings, implementing the mappings as SQL views and materializing a subset of the views, the underlying SQL optimizer is able to reduce the execution time of a SPARQL query by rewriting the query in terms of the views specified by the mappings. To the best of our knowledge, this is the first OBDA system supporting ontologies with transitivity by using SQL recursion. Our contributions include: (1) an efficient algorithm to compile ontological entailments as mappings; (2) a proof that every SPARQL query can be rewritten into a SQL query in the context of mappings; (3) a cost model to determine which views to materialize to attain the fastest execution time; and (4) an empirical evaluation comparing with a state-of-the-art OBDA system, which validates the cost model and demonstrates favorable execution times.

## 1 Introduction

Given a source relational database, a target OWL ontology and a mapping from the relational database to the ontology, Ontology-Based Data Access (OBDA) concerns answering queries over the target ontology using these three components. Commonly, researchers have taken two approaches to developing OBDA systems: materialization or rewriting. In the materialization approach, the input relational database $D$, target ontology $\mathcal{O}$ and mapping $\mathcal{M}$ (from $D$ to $\mathcal{O}$) are used to derive new facts that are stored in a database $D_o$, which is the materialization of the data in $D$ given $\mathcal{M}$ and $\mathcal{O}$. Then the answer to a query $Q$ over the target ontology is computed by directly posing $Q$ over $D_o$ [3]. In the rewriting approach, three steps are executed. First, a new query $Q_o$ is generated from the query $Q$ and the ontology $\mathcal{O}$: the rewriting of $Q$ w.r.t to $\mathcal{O}$. The majority of the OBDA literature focuses on this step [19]. Second, the mapping $\mathcal{M}$ is used to compile $Q_o$ to a SQL query $Q_{\text{sql}}$ over $D$ [21, 22]. Finally, $Q_{\text{sql}}$ is evaluated on the database $D$, which gives us the answer to the initial query $Q$.

We develop an OBDA system, Ultrawrap$^{\text{OBDA}}$, which combines materialization and query rewriting. Our objective is to effect optimizations by pushing processing into the Relational Databases Management Systems (RDBMS) and closer to the stored data, hence making maximal use of existing SQL infrastructure. We distinguish two phases:

a compile and runtime phase. In the compile phase, the inputs are a relational database $D$, an ontology $\mathcal{O}$ and a mapping $\mathcal{M}$ from $D$ to $\mathcal{O}$. The first step is to embed in $\mathcal{M}$ the ontological entailments of $\mathcal{O}$, which gives rise to a new mapping $\mathcal{M}^\star$, the *saturation* of $\mathcal{M}$ w.r.t. $\mathcal{O}$. The mapping $\mathcal{M}^\star$ is implemented using SQL views. In order to improve query performance, an important issue is to decide which views should be materialized. This is the last step of the compilation phase. In the runtime phase, the input is a query $Q$ over the target ontology $\mathcal{O}$, which is written in the RDF query language SPARQL, and the problem is to answer this query by rewriting it into some SQL queries over the views. A key observation at this point is that some existing SQL optimizers are able to perform rewritings in order to execute queries against materialized views.

To the best of our knowledge, we present the first OBDA system which supports ontologies with transitivity by using SQL recursion. More specifically, our contributions are the following. (1) We present an efficient algorithm to generate saturated mappings. (2) We provide a proof that every SPARQL query over a target ontology can be rewritten into a SQL query in our context, where mappings play a fundamental role. It is important to mention that such a result is a minimal requirement for a query-rewriting OBDA system relying on relational database technology. (3) We present a cost model that help us to determine which views to materialize to attain the fastest execution time. And (4) we present an empirical evaluation using (i) Oracle, (ii) two benchmarks including an extension of the Berlin SPARQL Benchmark, and (iii) six different scenarios. This evaluation includes a comparison against a state-of-the-art OBDA system, and its results validate the cost model and demonstrate favorable execution times for Ultrawrap$^{\text{OBDA}}$.

**Related work.** This research builds upon the work of Rodriguez-Muro et. al. implemented in Ontop [24, 25] and our previous work on Ultrawrap [27]. Rodriguez-Muro et. al. uses the tree-witness rewriting algorithm and introduced the idea of compiling ontological entailments as mappings, which they named $\mathcal{T}$-Mappings. There are three key differences between Rodriguez-Muro et. al. and our work in this paper: (1) we have extended the work of Rodriguez-Muro et. al. to support more than hierarchy of classes and properties, including transitivity; (2) we introduce an efficient algorithm that generates saturated mappings while Rodriguez-Muro et. al. has not presented an algorithm before; and (3) we represent the mappings as SQL views and study when the views should be materialized. Ultrawrap is a system that encodes a fix mapping, the direct mapping [4, 26], of the database as RDF. These mappings are implemented using unmaterialized SQL views. The approach presented in this paper extends Ultrawrap in three important aspects: (1) supports a customized mapping language; (2) supports reasoning through saturated mappings; and (3) considers materializing views for query optimization. Another related work is the combined approach [16], which materializes entailments as data, without considering mappings, and uses a limited form of query rewriting. The main objective of this approach is to deal with the case of infinite materialization, which cannot occur for the type of ontologies considered in this paper.

## 2 Preliminaries

**Relational Databases.** Assume, a countably infinite domain $\mathbf{D}$. A *schema* $\mathbf{R}$ is a finite set of relation names, where for each $R \in \mathbf{R}$, $att(R)$ denotes the nonempty finite set

of attributes names associated to $R$ and $arity(R)$ is the arity of $R$ (that is, $arity(R)$ is the number of elements of the set $att(R)$). An instance $I$ of $\mathbf{R}$ assigns to each relation symbol $R \in \mathbf{R}$ a finite set $R^I = \{t_1, \ldots, t_\ell\}$ of tuples, where each tuple $t_j$ ($1 \leq j \leq \ell$) is a function that assigns to each attribute in $att(R)$ a value from $\mathbf{D}$. We use notation $t.A$ to refer to the value of a tuple $t$ in an attribute $A$. Moreover, we say that $R(t)$ is a fact in $I$ if $t \in R^I$, and we use notation $R(t) \in I$ in this case (that is, we also view instances as sets of facts).

*Example 1. We use a relational database for an organization as a running example. The schema of this database consists of the table* $\texttt{EMP}$ *with* $att(\texttt{EMP}) = \{\texttt{SID}, \texttt{NAME}, \texttt{JOB}\}$*, and the following is an instance of this schema:* $I = \{\texttt{EMP}(1, \texttt{Alice}, \texttt{CEO}), \texttt{EMP}(2, \texttt{Bob}, \texttt{JavaProgrammer}), \texttt{EMP}(3, \texttt{John}, \texttt{SysAdmin})\}$.

In what follows, we assume some familiarity with the syntax and semantics of first-order logic. In particular, we assume that a formula over a relational schema $\mathbf{R}$ is constructed by using the relation names in $\mathbf{R}$, the equality predicate $=$ and the elements (also referred as constants) in $\mathbf{D}$. Moreover, a tuple of variables is denoted by $\bar{x}$ and a tuple of elements from $\mathbf{D}$ is denoted by $\bar{c}$, notation $\varphi(\bar{x})$ is used to indicate that the free variables of $\varphi$ are exactly the variables in $\bar{x}$, and $\varphi(\bar{c})$ is the formula obtained from $\varphi(\bar{x})$ by replacing every variable in $\bar{x}$ by the corresponding element in $\bar{c}$. Finally, given an instance $I$ over a relational schema $\mathbf{R}$ and a set $\Sigma$ of first-order formulae over $\mathbf{R}$, notation $I \models \psi$ is used to indicate that a first-order formula $\psi$ over $\mathbf{R}$ holds in $I$, while notation $\Sigma \models \psi$ is used to indicate that $\psi$ is implied by $\Sigma$.

**RDF and Ontologies.** Assume there are disjoint countably infinite sets $\mathbf{U}$ (URIs) and $\mathbf{L}$ (literals). A tuple $(s, p, o) \in \mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L})$ is called an RDF triple,[1] where $s$ is the subject, $p$ is the predicate and $o$ is the object. A finite set of RDF triples is called an RDF graph.

In order to define the notion of ontology, define $\mathbf{O}$ as the following set of reserved keywords: $\{\texttt{subClass}, \texttt{subProp}, \texttt{dom}, \texttt{range}, \texttt{type}, \texttt{equivClass}, \texttt{equivProp}, \texttt{inverse}, \texttt{symProp}, \texttt{transProp}\}$, and assume that $\mathbf{O} \subseteq \mathbf{U}$. Moreover, following [28] say that an RDF triple $(a, b, c)$ is ontological if : (1) $a \in (\mathbf{U} \setminus \mathbf{O})$, and (2) either $b \in (\mathbf{O} \setminus \{\texttt{type}\})$ and $c \in (\mathbf{U} \setminus \mathbf{O})$, or $b = \texttt{type}$ and $c$ is either $\texttt{symProp}$ or $\texttt{transProp}$. Additionally, say that an RDF triple $(a, b, c)$ is assertional if $(a, b, c)$ is not ontological. Then an ontology $\mathcal{O}$ is simply defined as a finite set of ontological triples. The semantics of an ontology $\mathcal{O}$ is usually defined by representing it as a set of description logic axioms, and then relying on the semantics of this logic [5] (which, in turn, is inherited from the semantics of first-order logic). For our purpose, it is more convenient to directly define a set $\Sigma_{\mathcal{O}}$ of first-order formulae encoding the ontology $\mathcal{O}$. More precisely, assume that $\texttt{triple}$ is a ternary predicate that is used to store RDF graphs in the obvious way: every triple $(a, b, c) \in G$ is stored as $\texttt{triple}(a, b, c)$. Then for every triple $t \in \mathcal{O}$, define a first-order formula $\varphi_t$ over $\texttt{triple}$ as follows:

[1] For simplicity, we do not consider blank nodes as a skolemization process can be used to replace them by URIs.

$$
\begin{aligned}
\varphi_{(a,\texttt{subClass},b)} &= \forall x\,(\texttt{triple}(x,\texttt{type},a) \rightarrow \texttt{triple}(x,\texttt{type},b))\\
\varphi_{(a,\texttt{subProp},b)} &= \forall x \forall y\,(\texttt{triple}(x,a,y) \rightarrow \texttt{triple}(x,b,y))\\
\varphi_{(a,\texttt{dom},b)} &= \forall x \forall y\,(\texttt{triple}(x,a,y) \rightarrow \texttt{triple}(x,\texttt{type},b))\\
\varphi_{(a,\texttt{range},b)} &= \forall x \forall y\,(\texttt{triple}(x,a,y) \rightarrow \texttt{triple}(y,\texttt{type},b))\\
\varphi_{(a,\texttt{equivClass},b)} &= \forall x\,(\texttt{triple}(x,\texttt{type},a) \leftrightarrow \texttt{triple}(x,\texttt{type},b))\\
\varphi_{(a,\texttt{equivProp},b)} &= \forall x \forall y\,(\texttt{triple}(x,a,y) \leftrightarrow \texttt{triple}(x,b,y))\\
\varphi_{(a,\texttt{inverse},b)} &= \forall x \forall y\,(\texttt{triple}(x,a,y) \leftrightarrow \texttt{triple}(y,b,x))\\
\varphi_{(a,\texttt{type},\texttt{symProp})} &= \forall x \forall y\,(\texttt{triple}(x,a,y) \rightarrow \texttt{triple}(y,a,x))\\
\varphi_{(a,\texttt{type},\texttt{transProp})} &= \forall x \forall y \forall z\,(\texttt{triple}(x,a,y) \wedge \texttt{triple}(y,a,z) \rightarrow \texttt{triple}(x,a,z)),
\end{aligned}
$$

and define $\Sigma_{\mathcal{O}}$ as $\{\varphi_t \mid t \in \mathcal{O}\}$.

Note that `subClass`, `subProp`, `dom`, `range`, `type`, `equivClass`, `equivProp` are in RDFS [7], `inverse`, `symProp` are in OWL 2 QL but not in OWL 2 EL and `transProp` is in OWL 2 EL but not in OWL 2 QL [17]. Actually, the ontologies we consider are in the non-standard profiles known as RDFS-Plus [1], OWL-LD[2] and RDFS 3.0[3].

In an OBDA system, we need to map relational databases into RDF graphs, which forces us to transform every constant from $\mathbf{D}$ into either a URI or a literal. This process is usually carried over by using some built-in transformation functions [4, 26]. For the sake of simplicity, we assume that $\mathbf{D} = (\mathbf{U} \setminus \mathbf{O}) \cup \mathbf{L}$, which allows us to use the constants in a relational database directly as URIs or literals in an RDF graph, and which also allows us to view a set of facts of the form $\texttt{triple}(a,b,c)$ as an instance over the relational schema $\{\texttt{triple}\}$. Notice that the keywords in $\mathbf{O}$ are not allowed in $\mathbf{D}$, as these as reserved for the specification of ontologies.

## 3 Mapping Relational Databases to RDF for OBDA

We describe how mappings are handled in our approach. We start by defining the mappings from relational databases to RDF, which are called RDB2RDF mappings, and then introduce the notion of saturation of an RDB2RDF mapping w.r.t. an ontology, which plays a fundamental role in our approach. We provide an efficient algorithm to compute it for ontologies not containing transitive predicates, and then show how our results can be extended to deal with transitive predicates. Finally, we show how RDB2RDF mappings are implemented in our system.

### 3.1 Relational databases to RDF mappings

We introduce the notion of mapping from a relation database to RDF, which is denoted as an RDB2RDF mapping. Two proposals to standardize this notion can be found in [4, 11]. However, we follow here an alternative approach that has been widely used in the data exchange [3] and data integration areas [15], and which is based on the use of first-order logic and its semantics to define mappings. More precisely, given a relational

[2] http://semanticweb.org/OWLLD/

[3] http://www.w3.org/2009/12/rdf-ws/papers/ws31

schema $\mathbf{R}$ such that $\texttt{triple} \notin \mathbf{R}$, a class RDB2RDF-rule $\rho$ over $\mathbf{R}$ is a first-order formula of the form:

$$\forall s \forall p \forall o \forall \bar{x}\ \alpha(s, \bar{x}) \wedge p = \texttt{type} \wedge o = c \rightarrow \texttt{triple}(s, p, o), \tag{1}$$

where $\alpha(s, \bar{x})$ is a domain-independent first-order formula over $\mathbf{R}$ and $c \in \mathbf{D}$. Moreover, a predicate RDB2RDF-rule $\rho$ over $\mathbf{R}$ is a first-order formula of the form:

$$\forall s \forall p \forall o \forall \bar{x}\ \beta(s, o, \bar{x}) \wedge p = c \rightarrow \texttt{triple}(s, p, o), \tag{2}$$

where $\beta(s, o, \bar{x})$ is a domain-independent first-order formula over $\mathbf{R}$ and $c \in \mathbf{D}$. Finally, an RDB2RDF-rule over $\mathbf{R}$ is either a class or a predicate RDB2RDF-rule over $\mathbf{R}$. In what follows, we omit the universal quantifiers $\forall s \forall p \forall o \forall \bar{x}$ from RDB2RDF rules, and we implicitly assume that these variables are universally quantify.

*Example 2. Consider the relational database from Example 1. Then the following RDB2RDF rule maps all the instances of the* `EMP` *table as instances of the* `Employee` *class:* $\texttt{EMP}(s, x_1, x_2) \wedge p = \texttt{type} \wedge o = \texttt{Employee} \rightarrow \texttt{triple}(s, p, o)$.

Let $\mathbf{R}$ be a relational schema. An RDB2RDF mapping $\mathcal{M}$ over $\mathbf{R}$ is a finite set of RDB2RDF rules over $\mathbf{R}$. Given an RDB2RDF mapping $\mathcal{M}$ and an instance $I$ over $\mathbf{R}$, the result of applying $\mathcal{M}$ over $I$, denoted by $[\![\mathcal{M}]\!]_I$, is an instance over the schema $\{\texttt{triple}\}$ that is defined as the result of the following process. For every RDB2RDF rule of the form (1) and value $c_1 \in \mathbf{D}$, if there exists a tuple of values $\bar{d}$ from $\mathbf{D}$ such that $I \models \alpha(c_1, \bar{d})$,[4] then $\texttt{triple}(c_1, \texttt{type}, c)$ is included as a fact of $[\![\mathcal{M}]\!]_I$, and likewise for every RDB2RDF rule of the form (2). Notice that this definition coincides with the notion of canonical universal solution in the context of data exchange [3]. Besides, notice that $[\![\mathcal{M}]\!]_I$ represents an RDF graph and, thus, mapping $\mathcal{M}$ can be considered as a mapping from relational databases into RDF graphs.

*Example 3. Consider the relational database from our running example, and let $\mathcal{M}$ be an RDB2RDF mapping consisting of the rule in Example 2 and the following rule:*

$$\texttt{EMP}(s, x_1, \texttt{CEO}) \wedge p = \texttt{type} \wedge o = \texttt{Executive} \rightarrow \texttt{triple}(s, p, o). \tag{3}$$

*If $I$ is the instance from Example 1, then $[\![\mathcal{M}]\!]_I$ consists of the following facts:*

$$\texttt{triple}(1, \texttt{type}, \texttt{Employee}),\ \texttt{triple}(2, \texttt{type}, \texttt{Employee}),$$
$$\texttt{triple}(3, \texttt{type}, \texttt{Employee}),\ \texttt{triple}(1, \texttt{type}, \texttt{Executive}).$$

### 3.2 Saturation of RDB2RDF mappings

As mentioned in Section 1, being able to modify an RDB2RDF mapping to embed a given ontology is a fundamental step in our approach. This process is formalized by means of the notion of saturated mapping.

[4] Given that $\alpha(s, \bar{x})$ is domain-independent, there exists a finite number of tuples $(c_1, \bar{d})$ such that $I \models \alpha(c_1, \bar{d})$.

**Definition 1 (Saturated mapping).** *Let $\mathcal{M}$ and $\mathcal{M}^\star$ be RDB2RDF mappings over a relational schema $\mathbf{R}$ and $\mathcal{O}$ an ontology. Then $\mathcal{M}^\star$ is a saturation of $\mathcal{M}$ w.r.t. $\mathcal{O}$ if for every instance $I$ over $\mathbf{R}$ and assertional RDF-triple $(a, b, c)$:*

$$[\![\mathcal{M}]\!]_I \cup \Sigma_\mathcal{O} \models \texttt{triple}(a, b, c) \quad \textit{iff} \quad \texttt{triple}(a, b, c) \in [\![\mathcal{M}^\star]\!]_I.$$

In this section, we study the problem of computing a saturated mapping from a given mapping and ontology. In particular, we focus on the case of ontologies not mentioning any triple of the form $(a, \texttt{type}, \texttt{transProp})$, which we denote by non-transitive ontologies. In Section 3.3, we extend these results to the case of arbitrary ontologies.

In our system, the saturation step is performed by exhaustively applying the inference rules in Table 1, which allow us to infer new RDB2RDF rules from the existing ones and the input ontology. More precisely, given an inference rule $t{:}\frac{\rho_1}{\rho_2}$ from Table 1, where $t$ is a triple and $\rho_1$, $\rho_2$ are RDB2RDF rules, and given an RDB2RDF mapping $\mathcal{M}$ and an ontology $\mathcal{O}$, we need to do the following to apply $t{:}\frac{\rho_1}{\rho_2}$ over $\mathcal{M}$ and $\mathcal{O}$. First, we have to replace the letters `A` and `B` in $t$ with actual URIs, say $a \in \mathbf{U}$ and $b \in \mathbf{U}$, respectively.[5] Second, we need to check whether the triple obtained from $t$ by replacing `A` by $a$ and `B` by $b$ belongs to $\mathcal{O}$, and whether the RDB2RDF rule obtained from $\rho_1$ by replacing `A` by $a$ belongs to $\mathcal{M}$. If both conditions hold, then the inference rule can be applied, and the result is an RDB2RDF mapping $\mathcal{M}'$ consisting of the rules in $\mathcal{M}$ and the rule obtained from $\rho_2$ by replacing `A` by $a$ and `B` by $b$.

*Example 4. Consider the RDB2RDF rule* (3) *from Example 3, and assume that we are given an ontology $\mathcal{O}$ containing the triple* $(\texttt{Executive}, \texttt{subClass}, \texttt{Employee})$. *Then by applying the first inference rule in Table 1, we infer the following RDB2RDF rule:* $\texttt{EMP}(s, x_1, \texttt{CEO}) \wedge p = \texttt{type} \wedge o = \texttt{Employee} \rightarrow \texttt{triple}(s, p, o)$.

Given an RDB2RDF mapping $\mathcal{M}$ and an ontology $\mathcal{O}$, we denote by $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ the RDB2RDF mapping obtained from $\mathcal{M}$ and $\mathcal{O}$ by successively applying the inference rules in Table 1 until the mapping does not change. The following theorem shows that $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ is a saturation of $\mathcal{M}$ w.r.t. $\mathcal{O}$, which justifies its use in our system.

**Theorem 1.** *For every RDB2RDF mapping $\mathcal{M}$ and ontology $\mathcal{O}$ in RDFS, it holds that $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ is a saturation of $\mathcal{M}$ w.r.t. $\mathcal{O}$.*

Theorem 1 is a corollary of the fact that the first six rules in Table 1 encode the rules to infer assertional triples from an inference system for RDFS given in [18]. A natural question at this point is whether $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ can be computed efficiently. In our setting, the approach based on exhaustively applying the inference rules in Table 1 can be easily transformed into a polynomial time algorithm for this problem. However, if this transformation is done in a naïve way, then the resulting algorithm is not really efficient. For this reason, we present here an efficient algorithm to compute $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ that is linear in the size of the input RDB2RDF mapping $\mathcal{M}$ and ontology $\mathcal{O}$, which are denoted by $\|\mathcal{M}\|$ and $\|\mathcal{O}\|$, respectively.

**Theorem 2.** *There exists an algorithm that, given an RDB2RDF mapping $\mathcal{M}$ and a non-transitive ontology $\mathcal{O}$, computes $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ in time $O(\|\mathcal{M}\| \cdot \|\mathcal{O}\|)$.*

[5] If $t = (\texttt{A}, \texttt{type}, \texttt{symProp})$, then we only need to replace `A` by $a$.

$$(\mathtt{A}, \mathtt{subClass}, \mathtt{B}) : \frac{\alpha(s,\bar{x}) \wedge p = \mathtt{type} \wedge o = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\alpha(s,\bar{x}) \wedge p = \mathtt{type} \wedge o = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$(\mathtt{A}, \mathtt{subProp}, \mathtt{B}) : \frac{\beta(s,o,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\beta(s,o,\bar{x}) \wedge p = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$(\mathtt{A}, \mathtt{dom}, \mathtt{B}) : \frac{\beta(s,o,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\beta(s,y,\bar{x}) \wedge p = \mathtt{type} \wedge o = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$(\mathtt{A}, \mathtt{range}, \mathtt{B}) : \frac{\beta(s,o,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\beta(y,s,\bar{x}) \wedge p = \mathtt{type} \wedge o = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$\begin{array}{r}(\mathtt{A}, \mathtt{equivClass}, \mathtt{B}) \\ \text{or } (\mathtt{B}, \mathtt{equivClass}, \mathtt{A})\end{array} : \frac{\alpha(s,\bar{x}) \wedge p = \mathtt{type} \wedge o = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\alpha(s,\bar{x}) \wedge p = \mathtt{type} \wedge o = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$\begin{array}{r}(\mathtt{A}, \mathtt{equivProp}, \mathtt{B}) \\ \text{or } (\mathtt{B}, \mathtt{equivProp}, \mathtt{A})\end{array} : \frac{\beta(s,o,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\beta(s,o,\bar{x}) \wedge p = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$\begin{array}{r}(\mathtt{A}, \mathtt{inverse}, \mathtt{B}) \\ \text{or } (\mathtt{B}, \mathtt{inverse}, \mathtt{A})\end{array} : \frac{\beta(s,o,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\beta(o,s,\bar{x}) \wedge p = \mathtt{B} \rightarrow \mathtt{triple}(s,p,o)}$$

$$(\mathtt{A}, \mathtt{type}, \mathtt{symProp}) : \frac{\beta(s,o,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}{\beta(o,s,\bar{x}) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s,p,o)}$$

**Table 1.** Inference rules to compute saturated mappings.

We now give the main ingredients of the algorithm mentioned in Theorem 2. Fix a mapping $\mathcal{M}$ and an ontology $\mathcal{O}$. In the first place, the algorithm transforms $\mathcal{O}$ into an instance $I_{\mathcal{O}}$ over the relational schema $\{\mathtt{triple}\}$, which satisfies that for every $(a,b,c) \in \mathcal{O}$: (1) if $b \in \{\mathtt{subClass}, \mathtt{subProp}, \mathtt{dom}, \mathtt{range}, \mathtt{type}\}$, then $\mathtt{triple}(a,b,c) \in I_{\mathcal{O}}$; (2) if $b = \mathtt{equivClass}$, then $\mathtt{triple}(a, \mathtt{subClass}, c) \in I_{\mathcal{O}}$ and $\mathtt{triple}(c, \mathtt{subClass}, a) \in I_{\mathcal{O}}$; (3) if $b = \mathtt{equivProp}$, then $\mathtt{triple}(a, \mathtt{subProp}, c) \in I_{\mathcal{O}}$ and $\mathtt{triple}(c, \mathtt{subProp}, a) \in I_{\mathcal{O}}$; and (4) if $b = \mathtt{inverse}$, then $\mathtt{triple}(a, \mathtt{inverse}, c) \in I_{\mathcal{O}}$ and $\mathtt{triple}(c, \mathtt{inverse}, a) \in I_{\mathcal{O}}$. Obviously, $I_{\mathcal{O}}$ can be computed in time $O(\|\mathcal{O}\|)$.

In the second place, the algorithm transforms as follows $\mathcal{M}$ into an instance $I_{\mathcal{M}}$ over a relational schema consisting of binary predicates $\mathtt{F}_{\mathtt{class}}$ $\mathtt{F}_{\mathtt{pred}}$, $\mathtt{Ch}$, $\mathtt{R}_{\mathtt{s}}$ and $\mathtt{R}_{\mathtt{o}}$. First, for every class RDB2RDF-rule in $\mathcal{M}$ of the form (1), a fresh natural number $m$ is assigned as an identifier of formula $\alpha(s,\bar{x})$, and then fact $\mathtt{F}_{\mathtt{class}}(m,c)$ is included in $I_{\mathcal{M}}$ (thus, $\mathtt{F}_{\mathtt{class}}$ is used to store the class RDB2RDF-rules in $\mathcal{M}$). Second, for every predicate RDB2RDF-rule in $\mathcal{M}$ of the form (2), a fresh natural number $n$ is assigned as an identifier of formula $\beta(s,o,\bar{x})$, and then fact $\mathtt{F}_{\mathtt{pred}}(n,c)$ is included in $I_{\mathcal{M}}$ (thus, $\mathtt{F}_{\mathtt{pred}}$ is used to store the predicate RDB2RDF-rules in $\mathcal{M}$). Moreover, in this case fresh natural numbers $k_1$, $k_2$ and $k_3$ are assigned as identifiers of formulae $\beta(o,s,\bar{x})$, $\beta(s,y,\bar{x})$ and $\beta(y,s,\bar{x})$ (where $y$ is a fresh variable), respectively, and then the facts $\mathtt{Ch}(n,k_1)$, $\mathtt{Ch}(k_1,n)$, $\mathtt{R}_{\mathtt{s}}(n,k_2)$ and $\mathtt{R}_{\mathtt{o}}(n,k_3)$ are included in $I_{\mathcal{M}}$ (thus, these predicates are used to store some syntactic modifications of formulae that are needed in the inference rules in Table 1). Finally, in this case fresh natural numbers $\ell_1$ are $\ell_2$ are assigned as iden-

tifiers of formulae $\beta(o, z, \bar{x})$ and $\beta(z, o, \bar{x})$ (where $z$ is a fresh variable), respectively, and then the facts $\mathtt{R_s}(k_1, \ell_1)$ and $\mathtt{R_o}(k_1, \ell_2)$ are included in $I_{\mathcal{M}}$. It is easy to see that $I_{\mathcal{M}}$ can be computed in time $O(\|\mathcal{M}\|)$.

With all the previous terminology, the problem of computing $\textsc{Sat}(\mathcal{M}, \mathcal{O})$ can be reduced to the problem of computing the minimal model of a Datalog program $\mathit{\Pi}_{\mathcal{M},\mathcal{O}}$, which consists of the facts in $(I_{\mathcal{O}} \cup I_{\mathcal{M}})$ together with the following set $\Delta$ of rules representing the inference rules in Table 1:

$$\begin{aligned}
\mathtt{triple}(X, \mathtt{subClass}, Y), \mathtt{F_{class}}(U, X) &\rightarrow \mathtt{F_{class}}(U, Y)\\
\mathtt{triple}(X, \mathtt{subProp}, Y), \mathtt{F_{pred}}(U, X) &\rightarrow \mathtt{F_{pred}}(U, Y)\\
\mathtt{triple}(X, \mathtt{dom}, Y), \mathtt{F_{pred}}(U, X), \mathtt{R_s}(U, V) &\rightarrow \mathtt{F_{class}}(V, Y)\\
\mathtt{triple}(X, \mathtt{range}, Y), \mathtt{F_{pred}}(U, X), \mathtt{R_o}(U, V) &\rightarrow \mathtt{F_{class}}(V, Y)\\
\mathtt{triple}(X, \mathtt{inverse}, Y), \mathtt{F_{pred}}(U, X), \mathtt{Ch}(U, V) &\rightarrow \mathtt{F_{pred}}(V, Y)\\
\mathtt{triple}(X, \mathtt{type}, \mathtt{symProp}), \mathtt{F_{pred}}(U, X), \mathtt{Ch}(U, V) &\rightarrow \mathtt{F_{pred}}(V, X),
\end{aligned}$$

where $X$, $Y$, $U$ and $V$ are variables. Notice that $\Delta$ is a fixed set of rules (it depends neither on $\mathcal{M}$ nor on $\mathcal{O}$), and also that $\Delta$ does not include rules for the keywords `equivClass` and `equivProp`, as these are represented in $I_{\mathcal{O}}$ by using the keywords `subClass` and `subProp`, respectively.

In order to compute the minimal model of $\mathit{\Pi}_{\mathcal{M},\mathcal{O}}$, we instantiate the variables in the above rules to generate a ground Datalog program $\mathit{\Pi}'_{\mathcal{M},\mathcal{O}}$ having the same minimal model as $\mathit{\Pi}_{\mathcal{M},\mathcal{O}}$. The key observation here is that $\mathit{\Pi}'_{\mathcal{M},\mathcal{O}}$ can be computed in time $O(\|\mathcal{M}\| \cdot \|\mathcal{O}\|)$, which proves Theorem 2 as the minimal model of a ground Datalog program can be computed in linear time [10] and the time needed to compute $(I_{\mathcal{O}} \cup I_{\mathcal{M}})$ is $O(\|\mathcal{M}\| + \|\mathcal{O}\|)$. More precisely, $\mathit{\Pi}'_{\mathcal{M},\mathcal{O}}$ is defined as $(I_{\mathcal{O}} \cup I_{\mathcal{M}}) \cup \Delta'$, where $\Delta'$ is generated from $\Delta$ as follows. For every fact $\mathtt{triple}(a, b, c) \in I_{\mathcal{O}}$, we look for the only rule in $\Delta$ where this fact can be applied, and we replace this rule by a new one where $X$ is replaced by $a$ and $Y$ is replaced by $c$ (or just $X$ is replaced by $a$ if $b = \mathtt{type}$ and $c = \mathtt{symProp}$). For example, if we consider a triple $\mathtt{triple}(a, \mathtt{subClass}, c)$, then we generate the rule $\mathtt{triple}(a, \mathtt{subClass}, b), \mathtt{F_{class}}(U, a) \rightarrow \mathtt{F_{class}}(U, b)$. Let $\Delta_1$ be the result of this process. Given that the set of rules $\Delta$ is fixed, we have that $\Delta_1$ can be computed in time $O(\|\mathcal{O}\|)$. Now for every rule $\rho$ in $\Delta_1$, we do the following to transform $\rho$ into a ground rule. We first replace the variable $U$ in $\rho$ by a value $n$ in $I_{\mathcal{M}}$. If $\rho$ also contains a variable $V$, then we notice that there exists at most one value $m$ in $I_{\mathcal{M}}$ for which the antecedent of the rule could hold, as there exists at most one value $m$ such that $\mathtt{R_s}(n, m)$ holds, and likewise for predicates $\mathtt{R_o}$ and $\mathtt{Ch}$. Thus, in this case we replace variable $V$ in $\rho$ for such a value $m$ to generate a ground Datalog rule, and we conclude that the resulting set $\Delta'$ of ground Datalog rules is computed in time $O(\|\mathcal{M}\| \cdot \|\mathcal{O}\|)$ (given that the size of $\Delta_1$ is $O(\|\mathcal{O}\|)$). This concludes the sketch of the proof of Theorem 2.

### 3.3 Dealing with transitive predicates

We show here how the approach presented in the previous section can be extended with recursive predicates. This functionality is of particular interest as the current work on

OBDA does not consider transitivity, mainly because the query language considered in that work is SQL without recursion [8]. From now on, given a first-order formula $\varphi(x, y)$, we use $\mathrm{TC}_{\varphi}(x, y)$ to denote the transitive closure of $\varphi(x, y)$. This formula can be written in many different formalisms. For example, if $\varphi(x, y)$ is a conjunction of relational atoms, then $\mathrm{TC}_{\varphi}(x, y)$ can be written as follows in Datalog:

$$\varphi(x, y) \rightarrow \mathrm{TC}_{\varphi}(x, y), \qquad \varphi(x, z), \mathrm{TC}_{\varphi}(z, y) \rightarrow \mathrm{TC}_{\varphi}(x, y).$$

In our system, $\mathrm{TC}_{\varphi}(x, y)$ is written as an SQL query with recursion. Then to deal with an ontology $\mathcal{O}$ containing transitive predicates, the set of inference rules in Table 1 is extended with the following inference rule:

$$(\mathtt{A}, \mathtt{type}, \mathtt{transProp}) : \frac{\{\beta_i(s, o, \bar{x}_i) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s, p, o)\}_{i=1}^{k}}{\mathrm{TC}_{[\bigvee_{i=1}^{k} \exists \bar{x}_i \beta_i]}(s, o) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s, p, o)}.$$

This rule tell us that given a transitive predicate A, we can take any number $k$ of RDB2RDF rules $\beta_i(s, o, \bar{x}_i) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s, p, o)$ for this predicate, and we can generate a new RDB2RDF rule for A by putting together the conditions $\beta_i(s, o, \bar{x}_i)$ in a formula $\gamma(s, o) = \bigvee_i \exists \bar{x}_i \beta_i(s, o, \bar{x}_i)$, and then using the transitive closure $\mathrm{TC}_{\gamma}(s, o)$ of $\gamma$ in an RDB2RDF rule $\mathrm{TC}_{\gamma}(s, o) \wedge p = \mathtt{A} \rightarrow \mathtt{triple}(s, p, o)$. In order for this approach to work, notice that we need to extend the syntax of RDB2RDF rules (1) and (2), so that formulae $\alpha$ and $\beta$ in them can be arbitrary formulae in a more expressive formalism such as (recursive) Datalog.

### 3.4 Implementing RDB2RDF mappings as views

We conclude this section by showing how RDB2RDF mappings are implemented in our system. Inspired by our previous work on Ultrawrap [27], every RDB2RDF rule is implemented as a triple-query, that is, as a SQL query which outputs triples. For example, the RDB2RDF rules:

$$\mathtt{EMP}(s, x_1, \mathtt{CEO}) \wedge p = \mathtt{type} \wedge o = \mathtt{Employee} \rightarrow \mathtt{triple}(s, p, o)$$
$$\mathtt{EMP}(s, x_1, \mathtt{SysAdmin}) \wedge p = \mathtt{type} \wedge o = \mathtt{Employee} \rightarrow \mathtt{triple}(s, p, o)$$

give rise to the following triple-queries:

```
SELECT SID as S, "type" as P, "Employee" as O FROM EMP WHERE JOB = "CEO"
SELECT SID as S, "type" as P, "Employee" as O FROM EMP WHERE JOB = "SysAdmin"
```

In practice, the triple-queries may include additional projections in order to support indexes, URI templates, datatypes and languages. However, for readability, we will consider here this simple version of these queries (we refer the reader to [27] for specific details). Then to implement an RDB2RDF mapping, all the class (resp. predicate) RDB2RDF-rules for the same class (resp. predicate) are grouped together to generate a triple-view, that is, a SQL view comprised of the union of the triple-queries for this class (resp. predicate). For instance, in our previous example the following is the triple-view for the class Employee:

```
CREATE VIEW EmployeeView AS
SELECT SID as S, "type" as P, "Employee" as O FROM EMP WHERE JOB = "CEO"  UNION ALL
SELECT SID as S, "type" as P, "Employee" as O FROM EMP WHERE JOB = "SysAdmin"
```

## 4 Executing SPARQL Queries

In this section, we describe how SPARQL queries are executed and optimized over the RDBMS through a cost model that determines which views should be materialized.

### 4.1 SPARQL rewriting

The runtime phase executes SPARQL queries on the RDBMS. We reuse Ultrawrap's approach of translating SPARQL queries to SQL queries in terms of the views defined for every class and property, which are denoted as triple-views in our system (see Section 3.4). Thus, we make maximal use of existing query optimization tools in commercial RDBMS, such as Oracle, to do the SPARQL query execution and rewriting [27].

Continuing with the example in Section 3.4, consider now a SPARQL query which asks for all the Employees: `SELECT ?x WHERE {?x type Employee}`. It is clear that this query needs to be rewritten to ask for the CEO and SysAdmin. The `EmployeeView` triple-view in Section 3.4 implements the mappings to the Employee class which consists of two triple-queries, one each for CEO and SysAdmin. Therefore, it is sufficient to generate a SQL query in terms of the `EmployeeView`. Given that a triple-view models a table with three columns, a SPARQL query is syntactically translated to a SQL query in terms of the triple-view. The resulting SQL query is `SELECT t1.s AS x FROM EmployeeView t1`.

A natural question at this point is whether every SPARQL query has an equivalent SQL query in our context, where RDB2RDF mappings play a fundamental role. In what follows we give a positive answer to this question, but before we introduce some terminology. Due to the lack of space, we do not formally present the syntax and semantics of SPARQL. Instead, we refer the reader to [23, 20] for this definition, and we just point out here that $[\![P]\!]_G$ denotes the answer to a SPARQL query $P$ over an RDF graph $G$, which consists of a set of solution mappings, that is, a set of functions that assign a value to each selected variable in $P$. For example, if $P$ is the SPARQL query `SELECT ?x WHERE {?x type ?y}`, and $G$ is an RDF graph consisting of the triples $(1, \texttt{type}, \texttt{Employee})$ and $(2, \texttt{type}, \texttt{Employee})$, then $[\![P]\!]_G = \{\mu_1, \mu_2\}$, where $\mu_1$ and $\mu_2$ are functions with domain $\{?\texttt{x}\}$ such that $\mu_1(?\texttt{x}) = 1$ and $\mu_2(?\texttt{x}) = 2$. Moreover, given a SQL query $Q$ (that may use recursion) over a relational schema $\mathbf{R}$ and an instance $I$ of $\mathbf{R}$, we use notation $[\![Q]\!]_I$ to represent the answer of $Q$ over $I$, which consists of a set of tuples in this case. Finally, to compare the answer of a SQL query with the answer of a SPARQL query, we make use of a function *tr* to transform a tuple into a solution mapping (this function is defined in the obvious way, see [26]). Then given an RDB2RDF mapping $\mathcal{M}$ over a relational schema $\mathbf{R}$ and a SPARQL query $P$, an SQL query $Q$ over $\mathbf{R}$ is said to be a SQL-rewriting of $P$ under $\mathcal{M}$ if for every instance $I$ of $\mathbf{R}$, it holds that $[\![P]\!]_{[\![\mathcal{M}]\!]_I} = tr([\![Q]\!]_I)$. Moreover, $P$ is said to be SQL-rewritable under $\mathcal{M}$ if there exists a rewriting of $P$ under $\mathcal{M}$.

**Theorem 3.** *Given an RDB2RDF mapping $\mathcal{M}$, every SPARQL query is SQL-rewritable under $\mathcal{M}$.*

The proof that the previous condition holds is by induction on the structure of a SPARQL query $P$ and, thus, it gives us a (naïve) bottom-up algorithm for translating $P$ into an equivalent SQL query $Q$ (given the mapping $\mathcal{M}$). More precisely, in the base case we are given a triple pattern $t = \{\texttt{s p o}\}$, where each one of its component is either a URI or a literal or a variable. This triple pattern is first translated into a SPARQL query $P_t$, where each position in $t$ storing a URI or a literal is replaced by a fresh variable, a filter condition is added to ensure that these fresh variables are assigned the corresponding URIs or literals, and a SELECT clause is added to ensure that the output variables of $t$ and $P_t$ are the same. For example, if $t = \{?\texttt{x type Employee}\}$, then $P_t$ is the following SPARQL query: $\texttt{SELECT ?x WHERE } \{?\texttt{x ?y ?z}\} \texttt{ FILTER } (?\texttt{y} = \texttt{type \&\& } ?\texttt{z} = \texttt{Employee})$. Then a SQL-rewriting of $P_t$ under $\mathcal{M}$ is computed just by replacing a triple pattern of the form $\{?\texttt{s } ?\texttt{p } ?\texttt{o}\}$ by a union of all the triple-queries representing the RDB2RDF rules in $\mathcal{M}$, and also replacing the SPARQL filter condition in $P_t$ by a filter condition in SQL.

In the inductive step, we assume that the theorem holds for two SPARQL queries $P_1$ and $P_2$. The proof then continues by presenting rewritings for the SPARQL queries constructed by combining $P_1$ and $P_2$ through the operators SELECT, AND (or '.' operator), OPTIONAL, FILTER and UNION, which is done by using existing approaches to translate SPARQL to SQL [2, 9].

### 4.2 Cost Model for View Materialization

A common approach for query optimization is to use materialized views [13]. Given that we are implementing RDB2RDF mappings as views, it is a natural to pursue this option. There are three implementation alternatives: (1) Materialize all the views: This approach gives the best query response time. However, it consumes the most space. (2) Materialize nothing: In this approach, every query needs to go to the raw data. However, no extra space is needed. (3) Materialize a subset of the views: Try to find a trade-off between the best query response time and the amount of space required.

In this section, we present a cost model for these three alternatives. First we must introduce some terminology. We consider ontologies consisting of hierarchy of classes which form a tree with a unique root, where a root class of an ontology is a class that has no superclasses. Then a leaf class of an ontology is a class that has no subclasses, and the depth of a class is the number of subclass relationships from the class to the root class (notice that there is a unique path from a class to the root class). Moreover, the depth of an ontology is the maximum depth of all classes present in the ontology.

First, we consider the cost of answering a query $Q$ is equal to the number of rows present in the relation used to construct Q. For example, if a relation $R$ has 100 rows, then the cost of the query $\texttt{SELECT } * \texttt{ FROM R}$ is 100. Second, assume we have a single relation $R$ and that mappings are from a query on the relation $R$ with a selection on an attribute $A$, to a class in the ontology. In Example 3, the relation $R$ is EMP, the attribute $A$ is JOB and the mapping is to the class Executive. Finally, we consider a query workload of queries asking for the instances of a class in the ontology, i.e. $\texttt{SELECT ?x WHERE } \{?\texttt{x type C}\}$, which can be translated into the triple-view implementing the mapping to the class C.

Our cost model is the following: If all the views implementing mappings are materialized, the query cost is $n \times N_R \times S(A, R)$ where $n$ is the number of leaf classes

underneath the class that is being queried for, $N_R$ is the number of tuples of the relation $R$ in the mapping, and $S(A, R)$ is the selectivity of the attribute $A$ of the relation $R$ in the mapping. The space cost is $N_R + (N_R \times d)$ where $d$ is the depth of the ontology. The reason for this cost is because the number of rows in a materialized view depends on the selectivity of the attribute and the number of leaf classes. Additionally, the sum of all the rows of each triple-view representing the mapping to classes in a particular depth $d$ of an ontology, is equivalent at most to the number of rows of the relation. If no views are materialized, then the query cost is $n \times N_R$, assuming there are no indices. The space cost is simply $N_R$. The reason for this cost is because to answer a query, the entire relation needs to be accessed $n$ times because there are no indices[6].

The question now is: How can we achieve the query cost of materializing all the views while keeping space to a minimum? Our hypothesis is the following: If a RDBMS rewrites queries in terms of materialized views, then by only materializing the views representing mappings to the leaf classes, the query cost would be $n \times N_R \times S(A, R)$, the same as if we materialized all the views, and the space cost would only be $2 \times N_R$. The rationale is the following: A triple-view representing a mapping to a class, can be rewritten into the union of triple-views representing the mapping to the child classes. Subsequently, a triple-view representing the mapping to any class in the ontology can be rewritten into a union of triple-views representing the mappings to leaf classes of an ontology. Finally, given a set of triple-views representing mappings from a relation to each leaf class of an ontology, the sum of all the rows in the set of triple-views is equivalent to the number of rows in the relation.

Given the extensive research of answering queries using views [14] and the fact that Oracle implements query rewriting on materialized views[7], we strongly suspect that our hypothesis will hold. The following evaluation section provides empirical results supporting our hypothesis.

## 5 Experimental evaluation

**Benchmarks:** The evaluation requires benchmarks consisting of a relational database schema and data, ontologies, mappings from the database to ontologies and a query workload. Thus, we created a synthetic benchmark, the *Texas Benchmark*, inspired by the Wisconsin Benchmark [12] and extended the Berlin SPARQL Benchmark (BSBM) Explore Use Case [6]. More precisely, the Texas Benchmark is composed of a single relation with 1 million rows. The relation has a first attribute which serves as a primary key, a set of additional filler attributes in order to take up space and then a set of six different integer-valued attributes which are non-unique. The main purpose of these attributes is to provide a systematic way to model a wide range of selectivity factors. Each attribute is named after the range of values the attribute assumes: `TWO`, `FIVE`, `TEN`, `TWENTY`, `FIFTY` and `HUNDRED`. For example, the attribute `FIVE` assumes a range of values from 1 to 5. Thus, the selection $\texttt{FIVE} = 1$ will have a 20% selectivity. In addition to the data, we created five different ontologies, consisting of a depth between 2-5. The branching factor is uniform and the number of leaves is 100 for each ontology.

[6] In the evaluation, we also consider the case when indices are present.

[7] http://docs.oracle.com/cd/B28359_01/server.111/b28313/qrbasic.htm

| | ontop | or index | or leaves | union index | union leaves | all mat |
|---|---|---|---|---|---|---|
| D6_100 | 7.31 | 7.37 | 7.44 | 5.28 | 5.42 | 4.80 |
| D6_50 | 7.44 | 7.55 | 7.54 | 5.14 | 5.76 | 4.92 |
| D6_20 | 9.76 | 9.88 | 9.88 | 6.28 | 5.71 | 5.26 |
| D6_10 | 10.51 | 10.74 | 10.43 | 7.20 | 6.90 | 6.37 |
| D6_5 | 17.24 | 17.26 | 10.97 | 19.75 | 13.42 | 9.09 |
| D6_2 | 22.54 | 22.88 | 20.36 | 23.21 | 21.17 | 20.18 |
| D5_100 | 5.38 | 5.39 | 5.42 | 4.97 | 4.38 | 3.85 |
| D5_50 | 5.56 | 5.84 | 5.61 | 4.92 | 4.52 | 4.02 |
| D5_20 | 9.01 | 8.52 | 8.41 | 6.16 | 5.47 | 5.05 |
| D5_10 | 9.86 | 9.30 | 9.04 | 7.20 | 6.52 | 6.13 |
| D5_5 | 17.32 | 17.80 | 9.71 | 20.24 | 10.98 | 9.32 |
| D5_2 | 21.80 | 25.28 | 19.33 | 23.27 | 19.67 | 20.13 |
| D4_100 | 5.63 | 5.88 | 5.81 | 4.92 | 4.59 | 3.96 |
| D4_50 | 5.83 | 6.08 | 6.25 | 5.10 | 4.82 | 4.18 |
| D4_20 | 6.28 | 6.56 | 6.46 | 5.51 | 5.34 | 5.00 |
| D4_10 | 7.43 | 7.76 | 7.52 | 6.47 | 6.10 | 5.80 |
| D4_5 | 17.54 | 17.60 | 10.84 | 20.96 | 11.89 | 9.74 |
| D4_2 | 22.33 | 23.30 | 20.25 | 23.40 | 19.96 | 19.88 |
| D3_100 | 6.21 | 7.36 | 6.48 | 3.80 | 3.92 | 3.53 |
| D3_50 | 6.40 | 6.64 | 10.15 | 4.09 | 3.76 | 3.54 |
| D3_20 | 6.93 | 7.15 | 6.94 | 4.41 | 4.25 | 3.90 |
| D3_10 | 8.30 | 8.47 | 8.08 | 5.91 | 5.38 | 5.14 |
| D3_5 | 16.55 | 16.93 | 7.85 | 16.80 | 7.61 | 7.58 |
| D3_2 | 23.61 | 23.43 | 19.91 | 22.78 | 20.09 | 19.98 |
| D2_100 | 1.13 | 1.28 | 1.24 | 1.23 | 1.10 | 1.05 |
| D2_50 | 1.41 | 1.50 | 1.48 | 1.45 | 1.28 | 1.30 |
| D2_20 | 2.23 | 2.40 | 2.20 | 2.23 | 1.99 | 1.99 |
| D2_10 | 4.22 | 4.42 | 3.99 | 4.26 | 3.91 | 3.81 |
| D2_5 | 16.39 | 16.98 | 10.47 | 16.77 | 7.67 | 8.14 |
| D2_2 | 23.43 | 23.19 | 20.24 | 22.66 | 19.59 | 20.18 |
| Average | 10.85 | 11.16 | 9.34 | 10.21 | 8.11 | 7.59 |

**Fig. 1.** Results of Texas Benchmark (sec)

The query workload consists of asking for an instance of a class at each depth of the ontology. On the other hand, the extension of BSBM replicates the query workload of an e-commerce website. Products have a type that is part of a ProductType ontology. Every product is mapped to one leaf class of the ProductType ontology. In our experiments, we created a dataset consisting of 1 million products with the benchmark driver, hence the product table has 1 million rows. The resulting ProductType ontology has a depth of 4 and consists of 3949 classes from which 3072 are leaf-level classes. The selectivity of the attribute in the mappings to ProductTypes is approximately 0.1%. In order to replicate the results of the Texas Benchmark, the query workload also consists of asking for an instance of a class at each depth of the ontology. In order to evaluate queries with transitivity, we use the child-parent relationship in the ProductType table which models the subclass relationship. The query workload for the transitivity part consists of asking for Products of a particular ProductType including the label and a numeric property of the Products, therefore including joins. More details about the benchmarks can be found at `http://obda-benchmark.org`

**Measurements and scenarios:** The objective of our experiments is to observe the behavior of a commercial relational database, namely Oracle, and its capabilities of supporting subclass and transitivity reasoning under our proposed approach. Therefore, the evaluation compares execution time and queries plans of SPARQL queries. With the Texas Benchmark, we compare w.r.t. two dimensions: depth of an ontology and selectivity of the attribute that is being mapped. In BSBM, because we are using a fixed 1 million product dataset, the depth of the hierarchy and selectivity is also fixed. We ran each query ten times and averaged the execution time, hence the experiments ran on a warm cache. In the evaluation we considered six scenarios: (**all-mat**) all the views are materialized; (**union-leaves**) only views representing mappings to the leaf classes are materialized, implemented with UNION; (**or-leaves**) same as in the previous scenario but with the views implemented with OR instead of UNION, (**union-index**) none of the views, implemented with UNION, are materialized, instead an index on the respective attributes have been added, (**or-index**) same as in the previous scenario but with the views implemented with OR; and (**ontop**) we compare against Ontop, a state of the art OBDA system [25]. We only compare against Ontop because to the best of our knowl-

edge, this is the only OBDA system that supports RDB2RDF mappings and SPARQL. The experiments were conducted on Oracle 11g R2 EE installed on a Sun Fire X4150 with a four core Intel Xeon X7460 2.66 GHz processor and 16 GB of RAM, running Microsoft Windows Server 2008 R2 Standard on top of VMWare ESX 4.0.

**Results:** An initial assessment suggests the following four expected observations: (1) The fastest execution time is *all-mat*; (2) our hypothesis should hold, meaning that the execution time of *union-leaves* should be comparable, if not equal, to the execution time of *all-mat*; (3) given that the Ontop system generates SQL queries with OR instead of UNION [25], the execution time of *ontop* and *or-index* should be comparable if not equal; (4) with transitivity, the fastest execution time is when the views are materialized.

Figure 1 shows the results of the Texas Benchmark in a form of a heat map, which evaluates subclass reasoning. The darker colors corresponds to the fastest query execution time. The x-axis consists of the six scenarios. In the y-axis, D6_100 means Depth 6 on Selectivity of 100. The values are the average execution time of the query workload. Notice that the expected observations (1), (2) and (3) hold. The fastest execution time corresponds to *all-mat*. The execution time of *union-leaves* is comparable, if not equal, to the execution time of *all-mat*, because Oracle was able to rewrite queries in terms of the materialized views. The number of rows examined is equivalent to the number of rows in the views where everything was materialized. This result provides evidence supporting our hypothesis and validates our cost model. Finally the execution time of *ontop* and *or-index* are comparable.

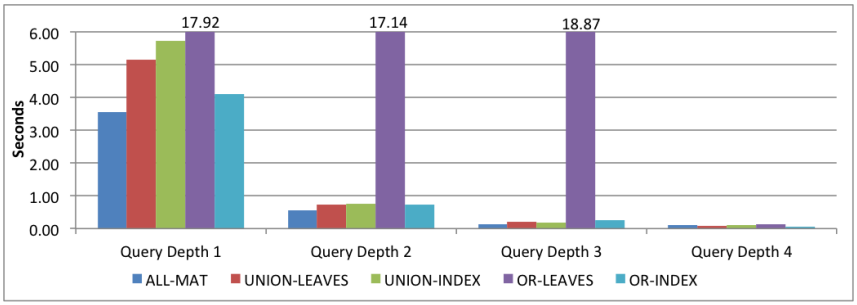


**Fig. 2.** Results of Subclass reasoning on BSBM

Figure 2 shows the results of the BSBM Benchmark for subclass reasoning. Our expected observations also hold in this case. Note that we do not report results for Ontop because the setup of the SPARQL endpoint timed-out after 2 hours.[8] Given that the selectivity is much lower compared to the selectivities in the Texas Benchmark, we observe that for queries asking for instances of classes that are in depth 1 (child of the root Class), the or-index outperforms union-leaves. We speculate that there is a slight overhead when rewriting queries over a large amount of views. However, for the rest of the queries, the overhead diminishes. We observe that the execution time of or-leaves is the worst because the database is not able to rewrite the query into the materialized views when the views are implemented with OR. Finally, throughout both benchmarks, we observe that *or-index* is competitive w.r.t *union-leaves*.

[8] We have reported the issue to the Ontop developers.

Figure 3 shows the results of the transitivity experiments on the BSBM Benchmark. Notice that the expected observations (4) holds. Given that Ontop does not support transitivity, we cannot compare with them. Therefore we only compare between materialized and non-materialized views. The Simple query requests all the ancestors of the given ProductType. The Join query requests all ancestors of the given ProductType and its corresponding Products. Therefore there is a join between ProductType and Product. The More Join query is similar to Join query, however it requests the name and a numeric property of the products, hence there are more joins. It is clear that materializing the view outperforms the non-materialized view for the following reasons: when the view is materialized, the size of the view is known beforehand and the optimizer is able to do a range scan with the index. However, when the view is not materialized, the size is not known therefore the optimizer does a full scan of the table. Detailed results can be found at `http://ribs.csres.utexas.edu/ultrawrap`

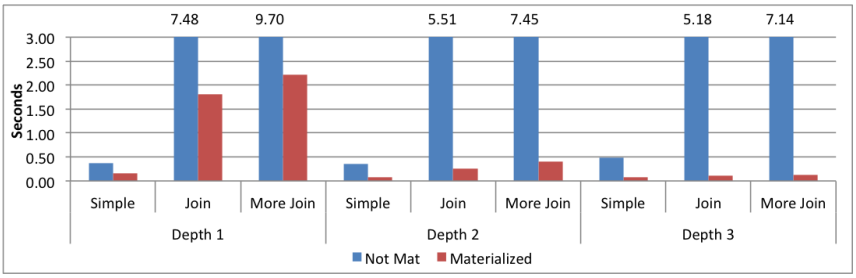


**Fig. 3.** Results of Transitivity reasoning on BSBM

## 6 Concluding Remarks

We presented Ultrawrap$^{\text{OBDA}}$, which to the best of our knowledge, is the first OBDA system supporting ontologies with transitivity by using SQL recursion. Ultrawrap$^{\text{OBDA}}$ is able to push processing into the RDBMS by implementing mappings using materialized views and taking advantage of existing query rewriting techniques.

Per related work, existing OBDA approaches only exploit the relational algebra capabilities of RDBMS. Our experimental results provide evidence that existing advanced capabilities implemented in RDBMS, such as recursion and query rewriting using materialized views, can be utilized for OBDA. We are not saying that we should rely exclusively on RDBMS technology to do the heavy lifting. RDBMS such as MySQL lack these advanced optimizations. However, we strongly suggest that the OBDA community should exploit the advanced optimizations in existing RDBMS.

Several open questions remain unanswered: What is the cost of maintaining views when the underlying data is updated? What is the state of the art of other RDBMS's optimizers in order to support OBDA? How does this approach respond to complex query workload? What is the trade-off between reasoning over relational databases with mappings and using native RDF databases supporting reasoning? We believe that these questions can be answered by developing systematic and real-world benchmark consisting of relational database schemas, data, ontologies and mappings and evaluating beyond just query rewriting. The Texas Benchmark and OBDA-Benchmark.org is a

first step in this process and we invite the community to contribute. As future work, we plan to evaluate Ultrawrap$^{\text{OBDA}}$ on other RDBMSs and compare against additional OBDA systems and native RDF databases that support reasoning.

**Acknowledgments.** Sequeda and Miranker were supported by NSF Grant IIS 1018554. Arenas was supported by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004 and Fondecyt grant 1131049. We thank the anonymous referees for many helpful comments.

## References

1. D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann Publishers Inc., 2008.
2. R. Angles and C. Gutierrez. The expressive power of sparql. In *ISWC*, pages 114–129, 2008.
3. M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.
4. M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda. Direct mapping of relational data to RDF. W3C Recomendation 27 September 2012.
5. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
6. C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.
7. D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema, W3C recommendation, February 2004.
8. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. *Artif. Intell.*, 195:335–360, 2013.
9. A. Chebotko, S. Lu, and F. Fotouhi. Semantics preserving sparql-to-sql translation. *Data Knowl. Eng.*, 68(10):973–1000, 2009.
10. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
11. S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF mapping language. W3C Recomendation 27 September 2012.
12. D. J. DeWitt. The wisconsin benchmark: Past, present, and future. In *The Benchmark Handbook*, pages 119–165. 1991.
13. A. Gupta and I. S. Mumick. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, 1999.
14. A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.
15. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
16. C. Lutz, I. Seylan, D. Toman, and F. Wolter. The combined approach to obda: Taming role hierarchies using filters. In *ISWC*, pages 314–330, 2013.
17. B. Motik, B. C. Grau, I. Horrocks, Z. Wu, and A. F. amd Carsten Lutz. Owl 2 web ontology language profiles (second edition), W3C recommendation, December 2012.
18. S. Muñoz, J. Pérez, and C. Gutierrez. Simple and efficient minimal rdfs. *J. Web Sem.*, 7(3):220–234, 2009.
19. M. Ortiz and M. Simkus. Reasoning and query answering in description logics. In *Reasoning Web*, pages 1–53, 2012.
20. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.
21. F. D. Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In *EDBT*, 2013.
22. A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
23. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation 15 January 2008, `http://www.w3.org/TR/rdf-sparql-query/`.
24. M. Rodríguez-Muro and D. Calvanese. Dependencies: Making ontology based data access work in practice. In *AMW*, 2011.
25. M. Rodríguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *ISWC*, pages 558–573, 2013.
26. J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to rdf and owl. In *WWW*, pages 649–658, 2012.
27. J. F. Sequeda and D. P. Miranker. Ultrawrap: Sparql execution on relational data. *J. Web Sem.*, 22:19–39, 2013.
28. J. Weaver and J. A. Hendler. Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *International Semantic Web Conference*, pages 682–697, 2009.