Object-Relational Queries over $\mathcal{CFDI}_{\text{nc}}$-Knowledge Bases: OBDA for the SQL-Literate

Jason St. Jacques, David Toman, and Grant Weddell
Cheriton School of Computer Science, University of Waterloo, Canada
jw.stjacques@gmail.com, {david,gweddell}@cs.uwaterloo.ca

Abstract
We consider how SQL-like query languages over object-relational schemata can be preserved in the setting of ontology based data access (OBDA), thus leveraging wide familiarity with relational technology. This is enabled by the adoption of the logic $\mathcal{CFDI}_{\text{nc}}$, a member of the $\mathcal{CFD}$ family of description logics (DLs). Of particular note is that this logic can fully simulate DL-Lite$_{\text{F\text{-}core}}$, a member of the DL-Lite family commonly used in the OBDA setting. Our main results present efficient algorithms that allow computation of certain answers with respect to $\mathcal{CFDI}_{\text{nc}}$ knowledge bases, facilitating direct access to a pre-existing row-based relational encoding of the data without any need for mappings to triple-based representations.

1 Introduction
Ontology based data access (OBDA) is concerned with computing query answers over (possibly incomplete) data sources for which background knowledge about the data, commonly captured in an ontology, is available. The background knowledge provides additional query answers that may not be explicit in the data itself. To address scalability issues relating to the volume of data, many current approaches to OBDA focus on conjunctive queries (CQ) and ontologies based on DL dialects for which CQ answering is in $\text{AC}^0$/PTIME with respect to data complexity. Moreover, to leverage advances in query processing in relational systems, approaches in which query answering can be reduced to SQL query evaluation over a relational encoding of the data are commonly sought. The two front-runners in this area are (i) the perfect rewriting-based approaches in which the given CQ is rewritten with the help of the ontological knowledge (typically formulated in one of the DL-Lite family of logics) in such a way that the resulting query can be executed over the original data sources to obtain desired answers [Calvanese et al., 2007], and (ii) the combined approaches in which the data is completed using ontological knowledge (formulated in DL-Lite or $\mathcal{EL}$ logics) in such a way that the original query (modulo filtering that only depends on role hierarchies in the ontology) can then be executed over the data completion [Kontchakov et al., 2010; 2011; Lutz et al., 2013; 2009]. The closest to our approach is the approach for Horn-$\mathcal{SHIQ}$ with rules [Eiter et al., 2012]; that approach deals with a logic incomparable with $\mathcal{CFDI}_{\text{nc}}$. Moreover, our approach handles keys and functional dependencies that are essential in database applications.

Recently, Toman and Weddell proposed $\mathcal{CFDI}_{\text{nc}}$ [Toman and Weddell, 2014], a dialect of the $\mathcal{CFD}$ family [Khizder et al., 2000; Toman and Weddell, 2009; 2013] that has PTIME complexity for many of the fundamental reasoning tasks, and can fully simulate DL-Lite$_{\text{core}}$ [Toman and Weddell, 2015].

In this paper, we show that CQ answering over $\mathcal{CFDI}_{\text{nc}}$ knowledge bases can be reduced to evaluating SQL queries over (a completion of) the data stored in a relational system. Our technique is based on a combination of query rewriting and data completion. Indeed, it is worth noting that, for CQs over $\mathcal{CFDI}_{\text{nc}}$ KBs, OBDA cannot be accomplished by either using (perfect) query rewriting alone, due to PTIME-completeness of CQ answering, or by exclusive use of the combined approach, due to the need to realize exponentially many prototypical anonymous witnesses to represent types induced by value restrictions. Our main technical contributions, in the order presented, are as follows:

- We exhibit an ABox completion procedure for a given logic knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with PTIME data complexity; the completion also serves as a basis for KB consistency checking;
- We define a query rewriting that produces a union of conjunctive queries $Q'$ from a given conjunctive query $Q$ and $\mathcal{T}$, and show that evaluating $Q'$ as a SQL query over the above ABox completion, viewed as a relational database, computes the certain answers of $Q$ over $\mathcal{K}$.
- Finally we show how a standard relational database schema can be naturally captured as a (fragment of a) $\mathcal{CFDI}_{\text{nc}}$ TBox, eliminating the need for additional mappings between data sources and virtual ABoxes that are typically utilized at this point, e.g., by [Calvanese et al., 2015]. We then show how such a rewritten query can be executed over an underlying relational representation without the need for object (id) invention.

Moreover, we show that the potential exponential blowup of the query rewriting cannot be avoided in general, since the combined complexity for CQ answering in $\mathcal{CFDI}_{\text{nc}}$ is PSPACE-complete (unless NP=PSPACE). However, the exponential blowup originating from concept hierarchies is
eliminated/minimized thanks to the data completion.

**Example 1** In the rest of the paper we use an object-relational UNIV schema depicted in Figure 1 in which single arrows denote named features, double arrows inheritance between classes, and where primary keys are underlined. We can then query instances of this schema using familiar SQL/OQL-style syntax for queries, for example:

```
select distinct s.name as n
from STUDENT s, TAKES t, CLASS c
where s = t.student and
  c.time = t.class.time and
  c.inst.dept.name = :p1 and c.num = :p2
```

The remainder of the paper is organized as follows: preliminaries are given next in which we introduce $\mathcal{CFDI}_{nc}$ and conjunctive queries. In Section 3, we present the theoretical basis of our approach, and in Section 4 discuss practical matters connected with the use of a relational system. We then outline some preliminary experimental results and follow with a summary, including some brief comments on directions for future work.

## 2 Preliminaries

All members of the $\mathcal{CFD}$ family of DLs are fragments of FOL with underlying signatures based on disjoint sets of unary predicate symbols called *primitive concepts*, constant symbols called *individuals* and unary function symbols called *features*. Formally, the logics are defined with respect to the sets $F$, $PC$ and $IN$ of (names of) features, primitive concepts, and individuals, respectively. A *path function* $\mathsf{Pf}$ is a word in $F^*$ with the usual convention that the empty word is denoted by $id$ and concatenation by $\cdot$. *Concept descriptions* $C$ and $D$ are defined by the grammars on the left-hand side of Figure 2. A concept $\forall \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k : Pf$ is called a path func-

![Figure 1: UNIV Schema.](image)

**Figure 2:** $\mathcal{CFDI}_{nc}$ Concepts.

- **Syntax**
  - $C ::= A$
  - $\forall \mathsf{Pf}.C$
  - $\exists f^{-1}.C$
  - $C \subseteq \Delta$
  - $\neg C$
  - $\forall \mathsf{Pf}.D$
  - $C : \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf}$

- **Semantics:** “$(\cdot)^\Delta$”
  - $A^\Delta \subseteq \Delta$
  - $\forall \mathsf{Pf}.C : \{ x \mid \mathsf{Pf}^\Delta(x) \in C^\Delta \}$
  - $\exists f^{-1}.C : \{ x \mid \exists y \in \Delta : f^\Delta(y) = x \}$
  - $C^\Delta \subseteq \Delta$
  - $\neg C : \Delta \setminus C^\Delta$
  - $\forall \mathsf{Pf}.D : \{ x \mid \mathsf{Pf}^\Delta(x) \in D^\Delta \}$
  - $C : \mathsf{Pf}_1, \ldots, \mathsf{Pf}_k \rightarrow \mathsf{Pf} : \{ x \mid \forall y \in C^\Delta : (\bigwedge_{i=1}^k \mathsf{Pf}_i^\Delta(x) = \mathsf{Pf}_i^\Delta(y)) \Rightarrow \mathsf{Pf}^\Delta(x) = \mathsf{Pf}^\Delta(y) \}$

---

**Example 3** Sample constraints for the UNIV schema de-
picted in Figure 1 can be captured in $C^{FD\Delta}_{NC}$ as follows:

1. PERSON $\sqsubseteq \neg$DEPT,
2. PERSON $\sqsubseteq \text{\textbackslash{}nname\textbackslash{}.STRING}$,
3. PERSON $\sqsubseteq$ PERSON : $name \rightarrow id$,
4. PERSON $\sqsubseteq$ DEPT : $name \rightarrow id$,
5. PROF $\sqsubseteq$ PERSON,
6. $\text{\textbackslash{}vreports\textbackslash{}.CHAIR} \sqsubseteq$ PROF,
7. $\text{\textbackslash{}head}^{-1} \sqsubseteq$ CHAIR,
8. CLASS $\sqsubseteq$ CLASS : $dept$, $num \rightarrow id$, etc.

2.1 Conjunctive Queries and Certain Answers

Conjunctive queries are, as usual, formed from atomic queries (or atoms) of the form “$C(x)$” and “$x \cdot Pf_1 = y \cdot Pf_2$”, where $x$ and $y$ are variables, using conjunction and existential quantification. To simplify notation, we conflate conjunctive queries with the set of its constituent atoms and a set of answer variables.

Definition 4 (Conjunctive Query) Let $\varphi$ be a set of atoms $C(x_1)$ and $x_1 \cdot Pf_1 = x_2 \cdot Pf_2$, where $C$ is a concept description (defined in Figure 2), $Pf_1$ are path functions, and $x$ a tuple of variables. We call the expression $\{ x \mid \varphi \}$ a conjunctive query (CQ).

A conjunctive query “$\{ x \mid \varphi \}$” is therefore a notational variant of the formula “$\exists y. \forall w \in \varphi \ y^{\varphi \circ w}$” in which $y$ contains all variables appearing in $\varphi$ but not in $x$. The usual definition of certain answers is given by the following:

Definition 5 (Certain Answer) Let $\mathcal{K}$ be a $C^{FD\Delta}_{NC}$ knowledge base and $Q = \{ x \mid \varphi \}$ a CQ. A certain answer to $Q$ over $\mathcal{K}$ is a substitution of constant symbols $\bar{a}$, $[x \mapsto \bar{a}]$, such that $\mathcal{K} \models Q[x \mapsto \bar{a}]$.

As is the case with TBoxes and ABoxes, a CQ can be represented in a normal form, a form in which all atoms in the CQ are of the form “$A(x)$” or “$x \cdot f = y$”, where $A$ is a primitive concept and $f$ a feature. This can be easily achieved by introducing additional non-answer (existentially quantified) variables.

Example 6 The OQL query from Example 1 above is now formally captured as the following CQ in normal form:

\[
\{(n, p1, p2) \mid \text{STUDENT}(s), \text{TAKES}(t), \text{CLASS}(c), n = s\cdot name, p2 = c\cdot num, p1 = d\cdot name, c = t\cdot class, p1 = c\cdot inst, d = y\cdot dept, s = t\cdot student, t' = c\cdot time, t' = d\cdot time\}.
\]

For the remainder of the paper, we assume CQs are always connected. (Evaluating disconnected CQs is easily achieved by considering each component separately.)

3 Query Answering (OBDA)

We begin by introducing OBDA over abstract ABoxes. From the introduction: we need a first step to obtain an ABox completion $A_T$ depending only on $A$ and $T$ that is polynomial in both $|A|$ and $|T|$, and a second step to obtain a query rewriting $Q_T$ depending only on $Q$ and $T$ that is polynomial in $|T|$, such that $\bar{a}$ is a certain answer to $Q$ in $K = (T, A)$ if and only if $\bar{a}$ is an answer to $Q_T$ when evaluated over $A_T$.

3.1 Inverse-Induced PFDs

Allowing inverse features affects how PFDs interact with an ABox. In particular, PFDs in which all path interact with a common prefix may apply to (pairs of) anonymous individuals mandated by the existence of anonymous inverse features for existing ABox individuals. Consider the following example:

Example 7 For an ABox $A = \{A(a), A(b), a.g = b.g\}$ and a TBox $T = \{A \sqsubseteq \exists f^{-1}, \forall f, A \sqsubseteq B, B \sqsubseteq B : f.g \rightarrow id\}$, it is easy to see that both $a$ and $b$ must have an $f$-predecessor to which the PFD in $T$ applies and, in combination with functionality of features, yields $a = b$.

Observe with this example, however, that the TBox logically implies another PFD, in particular “$A \sqsubseteq A : g \rightarrow id$”, and that this PFD applies to objects explicitly present in $A$. In general, to enforce PFDs on an ABox while avoiding any need to explicitly create anonymous predecessor objects, we add additional logically implied PFDs to a given TBox as follows:

Definition 8 (PFD Enrichment for Inverses) Let $T$ be a $C^{FD\Delta}_{NC}$ TBox such that $T \sqsubseteq B : f, Pf_1, \ldots, f \cdot Pf_k \rightarrow f, Pf \in T$ ($A \sqsubseteq B : f, Pf_1, \ldots, f \cdot Pf_k \rightarrow id \in T$) where $Pf_1 \neq id$ for all $1 \leq i \leq k$. Then we require that $A \sqsubseteq \forall f. A', B \sqsubseteq \forall f. A', and A' \sqsubseteq B' : Pf_1, \ldots, f \cdot Pf_k \rightarrow Pf$ ($A' \sqsubseteq B' : Pf_1, \ldots, f \cdot Pf_k \rightarrow id$), where $A'$ and $B'$ are fresh primitive concepts, are also in $T$.

It is easy to see that the value restrictions and PFDs added to $T$ logically infer the original PFD and the necessary existence of $f$-inverses for $A'$ and $B'$ that originate in $A$ and $B$, respectively. Thus, we assume that TBoxes satisfy this condition for the remainder of the paper.

3.2 Abstract ABox Completion

The first step of query answering, ABox completion, is defined by the rules in Figure 3. In particular, the rules extend a given ABox with all implied concept memberships and feature agreements. (Note that this step cannot be accomplished by a FO query since it requires path exploration and is therefore hard for NLOGSPACE.)

Definition 9 Let $(T, A)$ be a $C^{FD\Delta}_{NC}$ knowledge base. We define an ABox completion $\Gamma(A)$ to be the least ABox $A'$ such that $A \subseteq A'$ and $A'$ is closed under the rules in Figure 3.

Observe that individuals can only be declared to be members of primitive concepts since $A$ is in normal form. Also, if UNA were to be assumed (or assumed for elements of specific primitive concepts, such as INT and STRING), the equalities in rules (a) and (c) lead to KB inconsistency (without any need for additional “firing of rules”). It is also easy to see that completion terminates since it can add at most $|T|^2 |A|^2$ new objects, one for every pair of existing objects and a feature name.

3.3 Query Rewriting

The second step in query answering relies on query reformulation with respect to $T$. This step is also necessary as $C^{FD\Delta}_{NC}$ can force exponentially many anonymous objects with distinct class membership to exist:
if \( a = b \in A \) then add \( b = a \) to \( A \)
if \( a = b, \varphi \in A \) then add \( \varphi[b/a] \) to \( A \)

(a) ABox Equality Interactions

if \( A(a) \in A \) and \( T = \{ A \subseteq B \} \) then add \( B(a) \) to \( A \)
if \( \{ A(a), a, f = b \} \subseteq A \) and \( T = \{ A \subseteq \forall f.B \} \) then add \( b \) to \( A \)
if \( \{ A(a), b, f = a \} \subseteq A \) and \( T = \{ \forall f.A \subseteq B \} \) then add \( b \) to \( A \)

(b) ABox–TBox Interactions

if \( A(a), B(b) \in A, a, \text{Pf}_i = c_i, b, \text{Pf}'_i = c_i \in A \) for \( 0 < i \leq k \), and \( A \subseteq B : \text{Pf}_1, \ldots, \text{Pf}_k \rightarrow \text{Pf} \in T \) then
1. if \( a, \text{Pf} = c, b, \text{Pf} = d \in A \) and \( c = d \notin \text{A} \) then add \( c = d \) to \( \text{A} \); or
2. if \( \text{Pf} \) is of the form \( \alpha \beta \) and \( a, \text{Pf} = c, b, \text{Pf} = d \) and \( c = d \notin \text{A} \) then add \( c, f = e, d, f = e \) to \( \text{A} \);
where \( \text{Pf}'_i \) is a prefix of \( \text{Pf}_i, c \) and \( d \) are \( \text{A} \) individuals, and \( c \) is a new individual.

(c) ABox–PFD Interactions

Example 10 Consider a \( \text{CFD} \text{L}_{\text{nc}} \) TBox
\[ T = \{ B_i \subseteq \bigwedge_{\text{t} \in \text{prime}} f_i.B_i \mid i \leq k \}. \]

Asserting \( (B_0 \cap \ldots \cap B_k)(a) \) would require exponentially many anonymous objects belonging to distinct concept combinations to be created as prototypical witnesses when completing the ABox along the lines of the combined approach [Lutz et al., 2009].

To avoid the need for "expensive" ABox completion, our approach treats matches to anonymous objects belonging to distinct concept combinations to be created as prototypical witnesses when completing the ABox along the lines of [Calvanese et al., 2007]:

Definition 11 Let \( Q = \{ \bar{x} \mid \varphi \} \) be a CQ. We write \( \text{Fold}_T(Q) \) to denote the set of CQs (implicitly denoting the union of their results) that is obtained by an exhaustive application of the following on the initial set \( \{ \{ \bar{x} \mid \varphi \} \} \).

For a CQ \( \{ \bar{y} \mid \psi \} \in \text{Fold}(Q) \), apply rewrite rules:
1. If \( \{ A(x), B(x) \} \subseteq \psi \) and \( T \models A \subseteq \neg B \) then \( \text{Fold}(Q) := \text{Fold}(Q) - \{ \bar{y} \mid \psi \} \).
2. If \( \{ x, f = y, x, f = z \} \subseteq \psi \) then \( \text{Fold}(Q) := \text{Fold}(Q) - \{ \bar{y} \mid \psi \} \cup \{ \bar{y} \mid \psi \}[z/y] \).
3. If \( \{ x, f = z, y, f = z \} \subseteq \psi \) then \( \text{Fold}(Q) := \text{Fold}(Q) \cup \{ \bar{y} \mid \psi \}[x/y] \).
4. If \( \{ A(x), B(x) \} \subseteq \psi \) and \( T \models A \subseteq B \) then \( \text{Fold}(Q) := \text{Fold}(Q) - \{ \bar{y} \mid \psi \} \cup \{ \bar{y} \mid \psi \} \setminus \{ A(x) \} \).

The key idea underlying this definition is that, to find query answers, it is now sufficient to match queries in \( \text{Fold}_T(Q) \) explicitly against the (extended) ABox; matches outside the ABox are captured by query reformation: removed parts of the query are implied by \( T \).

Lemma 12 Let \( Q \) be a CQ with at least one answer variable. Then \( \bar{a} \) is a certain answer to \( Q \) over \( K = (T, A) \) if and only if \( \bar{a} \) is an answer for some \( \{ \bar{x} \mid \psi \} \in \text{Fold}_T(Q) \) over \( A_T \).

Proof (sketch): Observing that the extended ABox \( A_T \) is essentially a part of the minimal model of \( K \) (since \( K \) is Horn) and that every element of \( \text{Fold}_T(Q) \) implies \( Q \), it is easy to see that whenever (1-6) are satisfied, there is a match of \( Q \) in the minimal model and thus \( \bar{a} \) is an answer. Conversely, if a match of \( Q \) in a minimal model exists yielding \( \bar{a} \) as an answer, then part of the match will be realized in the ABox (since at least the answer variables must be bound to ABox individuals) and the reminder of the match must be forest-like. Hence, one of the queries in \( \text{Fold}_T(Q) \) matches in the ABox since the remaining conjunctions must be implied by \( T \).

Example 13 \( \text{Fold}_T(Q) \), where \( Q \) is the normal form query in Example 6, will contain the following CQ
\[ \{ (n, p1, p2) \mid \text{STUDENT}(s), \text{TAKES}(t), \text{CLASS}(c), \}
\[ n = s.\text{name}, p2 = c.\text{num}, p1 = d.\text{name}, \]
\[ c = t.\text{class}, p' = c.\text{inst}, d' = p'.\text{dept}, \]
\[ s = t.\text{student}, \]
which is obtained from \( Q \) by applying rules (3) and (5).

For CQ without answer variables, we need an additional step that checks if the query (when equivalent to a concept) matches in the tree part of every interpretation of \( K \). We therefore extend the query rewriting in Definition 11 as follows:

7. If \( \bar{x} = \emptyset \) in \( Q = \{ \bar{x} \mid \varphi \} \) and \( \varphi \) is equivalent to a conjunction of concepts \( A_1, \ldots, A_k \), then \( \text{Fold}(Q) := \text{Fold}(Q) \cup \{ \{ \bar{a} \mid A_i \wedge \ldots \wedge A_k \} \} \) for every combination of primitive concepts \( A_1, \ldots, A_k \) that force \( C_1, \ldots, C_n \) in \( T \).

This construction accounts for matches in the anonymous part of the minimal model of \( K \), and yields the following Lemma:
Lemma 14 Let \( Q \) be a CQ without answer variables. Then \( K \models Q \) if and only if at least one \( \{ y \mid \psi \} \in \text{Fold}_T(Q) \) evaluates to true over \( \mathcal{A}_T \).

3.4 Anonymous Object Invention

To handle equalities generated by PFDs, ABox completion, in particular completion rule (c.2) in Figure 3, requires inventing new ABox individuals (denoted \( e \) in the figure). These individuals then allow agreement in queries to be resolved within the extended ABox. However, in many situations, individual invention (which, in turn leads to primary key invention as we will see in Section 4) may not be possible in many practical applications (indeed, Section 4 will rely on this).

Here, we explore an alternative based on query reformulation. Unfortunately, it is easy to see for the PFDs of the second form (those that lead to individual invention), that this is not possible. In particular, consider the following case:

Example 15 Assume \( T = \{ A \subseteq \forall f.A \subseteq A : f.h, g \rightarrow h \} \) and consider the CQ \( \{ \langle x, y \rangle \mid A(x), A(y), x.h = z_0, y.h = z_0 \} \). Then a rewriting of the query must contain \( \{ \langle x, y \rangle \mid A(x), A(y), x.g = z_0, y.g = z_0, x.f = x_1, y.f = y_1, A(x_1), A(y_1), x.h = z_1, y.h = z_1 \} \), and \( \{ \langle x, y \rangle \mid A(x), A(y), x.g = z_0, y.g = z_0, x.f = x_1, y.f = y_1, A(x_1), A(y_1), x.g = z_1, y.g = z_1, x.f = x_2, y.f = y_2, A(x_2), A(y_2), x_2.h = z_2, y_2.h = z_2 \} \), and so on, which yields a rewriting that is necessarily infinite.\(^1\)

However, if the second form of PFDs is restricted to standard relational FDs of the form \( A \subseteq A : f_1, \ldots, f_n \rightarrow f' \), then rewriting becomes possible:

8. If \( \{ x.f = z, y.f = z, A_1(z), \ldots, A_k(z) \} \subseteq \psi \) and \( A \subseteq B : f_1, \ldots, f_n \rightarrow f \in T \) then \( \text{Fold}(Q) := \text{Fold}(Q) \cup \{ \{ y \mid \psi' \} \} \) for all \( \psi' \) of the form \( \psi' = \{ x.f = y.f = z, A_1(z), \ldots, A_k(z) \} \cup \{ A^x(x), A^y(y) \} \cup \{ B_{i,j}^x(x), \ldots, B_{i,j}^y(y) \} \cup \{ x.f_1 = z_1, y.f_1 = z_2, \ldots, x.f_n = z_n, y.f_n = z_n \} \).

This additional rule makes it safe to omit completion rules from Figure 3(b) when concepts corresponding to data types are involved.

Example 17 Consider the query \( Q = \{ \langle x \rangle \mid \text{STRING}(x) \} \).

We get reformulations \( \{ \langle x \rangle \mid \text{PERSON}(y), y.name = x \} \) and \( \{ \langle x \rangle \mid \text{DEPT}(y), y.name = x \} \) in \( \text{Fold}(Q) \), which in turn retrieve all STRINGS recorded in the KB.

The constructions used in Lemmas 12 and 14 (and optionally Lemma 16 and Section 3.5(9)) also yield the upper bounds. In all cases, computing certain answers reduces to evaluating an union of conjunctive queries over a polynomially-sized completion of an ABox. Lower bounds follow from reductions from graph reachability, Horn-SAT, and the DFA intersection problem [Kozen, 1977], respectively. Note that PSPACE-hardness holds even for very simple CQs of the form \( \exists x \{ A_1(x) \land \ldots \land A_k(x) \} \).

Corollary 18 Data complexity for CQ query answering over \( \mathcal{CFD}_{\text{HC}} \) KB is in \( \text{PTIME} \). CQ query answering over \( \mathcal{CFD}_{\text{HC}} \) KB is \( \text{NLOGSPACE} \)-hard for data complexity (in \( |\mathcal{A}| \)), \( \text{PTIME} \)-hard in \( |T + \mathcal{A}| \), and \( \text{PSPACE} \)-hard for combined complexity.

4 Relational Back-end

Hereon, we assume any non-key PFD occurring in a knowledge base \( K \) is a key or conforms to a standard relational FD. (Recall that this enables our framework to avoid anonymous object invention.) Our objective in this section is to show that this leads to a very practical front-end to existing relational engines in which \( K \) consists of a TBox \( T \) that embeds a relational schema \( S \), and a virtual ABox \( \mathcal{A} \) that is given by the tuples occurring in relational tables. W.l.o.g., we assume the correspondence is as follows:

1. Primitive concepts (save concepts capturing concrete values such as INT and STRING in Figure 1), hereon called abstract, correspond to tables, and identities of objects belonging to such concepts are captured by primary key values in the tables;
2. Features between abstract concepts, hereon also called abstract, are captured by foreign keys between the corresponding tables. Note that a single such feature may correspond to a multi-arity foreign key;

\(^1\)This, without resorting to regular path functions as in [Toman and Weddell, 2005] which, however, would preclude the consequent use of an SQL engine.

\(^2\)This is the approach SQL and OQL have taken.
3. Features between abstract concepts and concepts capturing concrete values, hereon called concrete, correspond to attributes in the table corresponding to the abstract concept with an appropriate concrete data type, for which UNA is assumed (as is common in SQL).

Note also that non primary key attributes may contain SQL's null values to account for data incompleteness.

In our running example, S consists of “create table” commands that include declarations for primary and foreign keys, such as the following in the case of the CLASS primitive concept for our UNIV TBox:

create table CLASS-TAB (dname, num, iname, room, time, primary key (dname, num),
foreign key (dname) to DEPT-TAB,
foreign key (iname) to PROF-TAB).

We write Tab(CLASS), Attr(CLASS) and PK(CLASS) to respectively denote table CLASS-TAB, its set of attributes (dname, etc.) and the sequence of its primary key attributes (dname, num). For abstract features, such as dept, we use DOMattr(dept), DOMTab(dept), and RANTab(dept) to denote the list of source attributes (dname), the originating table (CLASS-TAB), and the target table (DEPT-TAB), respectively, in order to describe the associated foreign key.

We assume that A ⊆ ¬B ∈ T whenever PK(A) ≠ PK(B) and that Attr(B) ⊆ Attr(A) whenever T |= A ⊆ B.3

With these assumptions, ABox completion for a consistent knowledge base will only be required for ABox-TBox interactions (see Figure 3), and is accomplished by repeatedly executing the following updates until no changes occur:4

- (when T |= A ⊆ B)
  \[
  \text{Tab}(A) := \text{Tab}(A) \cup (\text{Tab}(A) \bowtie_{\text{PK}(A)=\text{PK}(B)} \text{Tab}(B))
  \]
- (when T |= A ⊆ ∀f.B and f is abstract)
  \[
  \text{Tab}(B) := \text{Tab}(B) \cup \pi_{\text{Attr}(B)}(\text{Tab}(A))
  \]
- (when T |= ∀f.B ⊆ A and f is abstract)
  \[
  \text{Tab}(A) := \text{Tab}(A) \cup \pi_{\text{Attr}(A)}(\text{DOMTab}(f) \bowtie_{\text{DOMattr}(f)=\text{PK}(B)} \text{Tab}(B))
  \]

As a consequence, this approach effectively equates object identity with primary key values already occurring in a relational data source, and therefore entirely avoids any need to create Skolem constants that would otherwise be needed for an explicit ABox realisation. The final step consists of converting Fold(Q) to SQL. This is rather clerical task and the main steps consist of replacing:

1. concept descriptions by the associated tables;
2. object references by primary keys; and
3. abstract features by corresponding foreign keys.

Finally, to obtain well-formed SQL query, we need to separate the conjuncts to the SELECT, FROM, and WHERE clauses as dictated by the (rather rigid) SQL syntax.

5 Experiments

To confirm practicality, we applied our framework to a modified version of the LUBM benchmark in which we introduce a relational schema. We then defined a knowledge base TBox that embeds this schema and supplied various virtual ABoxes according to a number of configurations of generated data for the LUBM benchmark [Guo et al., 2005]. We performed the tests on MacBook Pro with 3ghz Core i7 CPU, 16gb RAM (OSX 10.11.1), and Postgres 9.4. Figure 4 reports the time needed to load the raw data, to index it, and to compute the ABox completion. The results show that the completion is comparable to raw data loading (a factor of about 2).

<table>
<thead>
<tr>
<th>LUBM scale</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw data load</td>
<td>14.9</td>
<td>117.4</td>
<td>234.4</td>
<td>518.1</td>
<td>1341</td>
<td>3034</td>
</tr>
<tr>
<td>indexing</td>
<td>10.2</td>
<td>69.7</td>
<td>167.9</td>
<td>377.6</td>
<td>1063</td>
<td>2308</td>
</tr>
<tr>
<td>completion</td>
<td>5.9</td>
<td>134.2</td>
<td>286.6</td>
<td>769.5</td>
<td>2452</td>
<td>6853</td>
</tr>
</tbody>
</table>

Due to space constraints we do not report the individual running times for the LUBM queries. However, all the queries complete within few seconds (typically under a second), even on the largest instances (except for queries 2 and 9 that took 73s and 65s, respectively, on the LUBM 1000 instance). These results are not surprising for reasonably indexed relational instances and can be considered representative of the approach.

6 Summary

The paper presents a novel approach to OBDA over knowledge bases formulated in $C^FDI_{NC}^\forall$, a DL that has been designed to capture relational and object-relational schemata in a natural way. Indeed, there are additional benefits to using $C^FDI_{NC}^\forall$ as the underlying DL:

**Example 19:** Consider the two queries generated in Example 17. Since the $C^FDI_{NC}^\forall$ TBox allows us to reason about keys and concept disjointness, we can deduce that neither of the two queries needs the distinct duplicate elimination in their SQL select clauses (since name is a key for both PERSON and DEPT) and, moreover, the union of these two queries can use the union all operator (since the two subqueries are disjoint), thus completely avoiding duplicate elimination in the final query.

For more complete development and benefits of these optimizations see [Khizder et al., 2000; Liu et al., 2002; Toman and Weddell, 2011]. There are two additional avenues for extending this proposal: first, the data completion could be performed incrementally and second, the restrictions on concept compatibility based on the primary keys of the underlying tables could be relaxed using techniques based on referring expressions recently proposed in [Borgida et al., 2016].
References


