# Realizing Bidirectional Virtual Knowledge Graphs Using Ontop

Romuald Esdras Wandji[1][0009−0008−5036−2452] and
Diego Calvanese[2][0000−0001−5174−9693]

[1] Department of Computing Science, Umeå Universitet, Umeå, Sweden
romuald.esdras.wandji@umu.se
[2] Faculty of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy
diego.calvanese@unibz.it

**Abstract.** The Virtual Knowledge Graph (VKG) paradigm enables querying heterogeneous relational databases through a unified semantic layer comprising an ontology and declarative mappings (typically in R2RML). While querying is well-supported, propagating updates from the virtual RDF graph (ABox) back to the source databases remains a challenge. This paper addresses the problem of translating updates (expressed in SPARQL Update) applied over the ABox into equivalent SQL updates over the underlying databases, specifically within the Ontop VKG system. Key difficulties arise from the non-injectivity inherent in R2RML mappings, where a single ABox update can correspond to multiple source update possibilities, and the potential for these source updates to cause unintended side effects—additional insertions or deletions in the ABox beyond the user's original intent. While relying on *Ontop*'s query rewriting engine, our method employs lineage computation to identify source tuples for deletion and a strategy for handling existential variables during insertion. It generates candidate SQL translations, analyzes their potential side effects on the virtual ABox, and selects the ones that minimize these unintended consequences. This work represents a step toward closing the Linked Data Life Cycle loop, allowing changes in the knowledge graph to be reflected in the corresponding source.

**Keywords:** Knowledge Representation · Virtual Knowledge Graph (VKG) · Ontology-based Data Access · View Updates

## 1 Introduction

In the evolving landscape of data integration, Virtual Knowledge Graphs (VKGs), previously known as Ontology-Based Data Access (OBDA), have emerged as a powerful paradigm for querying relational data sources through an ontological lens. VKGs enable users to pose high-level semantic queries using standards like SPARQL over an ontology, which are automatically rewritten and unfolded into executable low-level queries (e.g., SQL) via declarative schema mappings, typically expressed in languages like R2RML [10]. These mappings link a lightweight ontology, often specified in a description logic such as DL-Lite [5], to the underlying data source, facilitating efficient query answering through query rewriting and unfolding [38]. Current approaches, such as *Ontop* [4], Morph [29], SparqlMap [32], and Mastro [6] primarily focus on exposing structured data as virtual RDF for integration into Linked Data workflows without the need for costly data extraction, transformation, and loading (ETL) processes. However, these systems lack mechanisms for bidirectional integration, where updates applied to the virtual layer are propagated back to the source.

The task of updating VKGs, particularly translating *instance-level* (ABox) updates into equivalent updates over relational data sources, is a relatively underexplored area that intersects OBDA [36], view updates in databases [1], and SPARQL-to-SQL translation [31]. Updating the extensional data (ABox) of a knowledge base (KB) while preserving the ontology is a well-studied problem in description logics (DLs) and knowledge representation. The complexity arises from logical inference in the ontology, which can lead to inconsistencies and therefore to multiple update outcomes, aiming at (minimally) removing the inconsistencies [22,16]. Existing semantics for ABox updates are broadly classified into formula-based and model-based approaches. Formula-based semantics represent updates as logical formulas, achieving consistency by minimally removing conflicting ABox facts [7,14]. For instance, [7] proposed a formula-based semantics for DL-Lite, leveraging its first-order rewritability to ensure a unique repaired ABox after an update. Model-based semantics, conversely, update all models of the KB [35,13]. The ComputeUpdate algorithm proposed in [13] extends [35] to compute ABox updates for DL-Lite$_{\mathcal{F}}$ KBs. More expressive DLs, however, face expressibility challenges for updates [26].

Translating KB updates into source data updates is closely related to the view update problem in relational databases, where updates to a view must be reflected in the base tables [12,11,1]. The ambiguity introduced by mappings, where multiple source updates may correspond to a single view update, poses significant challenges [24,25]. SPARQL-to-SQL translation is a cornerstone of OBDA systems, enabling relational data to be queried as virtual RDF. State-of-the-art systems like *Ontop* [4], Morph-RDB [29], and SparqlMap [32] use R2RML mappings [10] to rewrite SPARQL queries into SQL. However, these systems primarily focus on read-only access, with no support for updates. SPARQL Update [19] provides a standardized mechanism for modifying RDF data, but translating such updates into SQL DML statements remains challenging due to schema constraints and mapping ambiguity.

D2RQ/update [15] introduced a schema-aware SPARQL Update processor that optimizes SQL statement generation and prevents constraint violations by grouping related triples. D2RQ++ [30] proposed integrating a native triple store to handle unmapped triples or those violating constraints, later propagating them to the database when feasible. OntoAccess [21] developed a custom mapping language (R3M) for bidirectional mappings, addressing insert and delete operations considering schema constraints. Additionally, [18] systematically explored SPARQL-to-SQL translations for both read and write scenarios, building on [8]. However, this work is conceptual and lacks detailed handling of complex R2RML templates or empirical evaluation.

The literature reveals several gaps: existing KB update semantics do not account for source data translation via mappings, which limit their application to VKG, *(1)* view update solutions are not tailored to the complex, ontology-driven mappings in VKGs, and *(2)* SPARQL-to-SQL update mechanisms lack robust strategies for minimizing side effects. In this paper, we address these gaps by proposing a comprehensive framework for *bidirectional VKGs* within *Ontop* that translates ABox updates into source updates using the mappings, allowing combined insertion and deletion operations to achieve minimal or no side effects. Thus we improve on our prior work [33], which restricted the translation of ABox insertions and deletions to corresponding source updates of the same type. Building upon the *Ontop* system, we propose an implementation of the translation algorithms proposed in [33] that prioritizes translations with minimal side effects, with R2RML mappings to ensure precise propagation of updates.

The rest of the paper is organized as follows. In Section 2 we provide some technical preliminaries on VKGs, the *Ontop* system, and updates in VKGs. In Section 3 we discuss how maximum recoveries are computed in *Ontop*, and in Section 4 how *Ontop* performs the translation of updates. In Section 5 we present some experimental results, and in Section 6 we conclude the paper.

## 2 Preliminaries

Formerly referred to as Ontology-Based Data Access (OBDA), the VKG paradigm is designed to facilitate access to heterogeneous data sources, in particular relational databases, through a high-level conceptual schema called *ontology* [4]. Formally, a VKG *specification* is a triple $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ consisting of *(i)* an *ontology*, which provides to the user a convenient and familiar vocabulary (of classes and properties), and extends the data with background knowledge, *(ii)* a relational *source schema* $\mathcal{S}$ for the underlying repositories, and *(iii)* a *schema mapping* $\mathcal{M}$, which describes through queries over the signature of $\mathcal{S}$ how to populate the classes and properties in $\mathcal{T}$. A VKG instance is given as a pair $\langle \mathcal{P}, D \rangle$ where $\mathcal{P}$ is a VKG specification and $D$ is a source instance compliant with $\mathcal{S}$. While component *(ii)* is specified in standard SQL, W3C published recommended languages for specifying components *(i)* and *(iii)*, namely OWL 2 QL [27] and R2RML [10], respectively.

In practice, each assertion in the provided mapping $\mathcal{M}$ of a traditional VKG has one of the forms

$$\begin{array}{lcl} \texttt{sql}(\boldsymbol{y}) & \rightsquigarrow & f(\boldsymbol{x})\ \texttt{:rdfType C}, \\ \texttt{sql}(\boldsymbol{y}) & \rightsquigarrow & f_1(\boldsymbol{x}_1)\ \texttt{:P}\ f_2(\boldsymbol{x}_2), \end{array}$$

where $\boldsymbol{x} \subseteq \boldsymbol{y}$, $\boldsymbol{x}_1 \subseteq \boldsymbol{y}$, $\boldsymbol{x}_2 \subseteq \boldsymbol{y}$, `sql` is an SQL query over the signature of $\mathcal{S}$ projecting the columns $\boldsymbol{y}$, and $f(\boldsymbol{x})$, $f_1(\boldsymbol{x}_1)$, and $f_2(\boldsymbol{x}_2)$ are *IRI-templates*[3], obtained by embedding the answer variables of the source query (enclosed in curly brackets) into a fixed string. For instance, if we have the IRI-template `:univ/student/{id}`, where `id` is a placeholder for a student ID, then we can obtain an IRI `:univ/student/7` by instantiating the variable `id` with the value "7". In this paper, we make the common assumption that IRIs in VKGs are generated in a unique way by IRI-templates, i.e., IRI-templates are *injective*. Moreover we assume that for each IRI-template $f(x_1, \ldots, x_n)$, we have also $n$ *inverse templates* $f^1, \ldots, f^n$ such that $f^i(f(v_1, \ldots, v_n)) = v_i$, for each $i \in [1..n]$ and for all possible values $v_1, \ldots, v_n$ instantiating the variables $x_1, \ldots, x_n$. When a mapping $\mathcal{M}$ is applied over a source instance $D$, we obtain a set $\mathcal{M}(D)$ of (*virtual*) RDF triples that represent a knowledge graph (also called ABox).

[3] IRI-templates correspond to the R2RML *string templates*.

In VKGs, queries are expressed over the terms in the ontology $\mathcal{T}$ and are formulated in SPARQL [27]. They are evaluated over the knowledge base $\langle \mathcal{T}, \mathcal{M}(D) \rangle$. Notice that the knowledge graph is not materialized, but is kept virtual, which ensures that the VKG system always exposes the actual up-to-date information.

### 2.1 The Ontop System

The *Ontop* framework is an open-source[4] VKG system designed to expose relational databases as virtual RDF graphs, allowing SPARQL queries to be answered over a semantic layer defined by an ontology [4]. Its virtual approach avoids materializing RDF triples by rewriting SPARQL queries into optimized SQL queries, which are then executed by the underlying database engine.

*Ontop* supports all W3C standards relevant to VKG, including SPARQL 1.0 (with partial SPARQL 1.1 support), OWL 2 QL [28], RDFS [3] and is compatible with major relational databases (e.g., PostgreSQL, MySQL, Oracle) and data federation systems, such as Teiid[5], Dremio[6], and Denodo[7], enabling data integration across heterogeneous sources. A core feature of *Ontop* is its query rewriting mechanism, which translates SPARQL queries over the ontology into SQL queries executable on the underlying database. Since version 3, *Ontop* has adopted an algebra-based data structure called Intermediate Query (IQ) to facilitate this translation. While Ontop focuses on query answering, understanding its query rewriting process is crucial for exploring update translation in VKGs, where similar techniques may be adapted to propagate ontology-level updates to the database.

IQ is a variant of relational algebra designed to represent both SPARQL queries and SQL queries from the mapping, bridging the semantic gap between the two [37]. Based on that notion, when the user poses a SPARQL query over a VKG instance, *Ontop* proceeds as follows: *(i)* the query is translated into IQ; *(ii)* the translated IQ is expanded taking into account the ontology (e.g., subclass and subproperty axioms, and domain and range assertions); *(iii)* the expanded IQ is *unfolded* using the mapping; and *(iv)* after a series of optimizations carried out by *Ontop*, the optimized IQ is converted into an SQL query for the underlying data source.

*Example 1.* As an example, assume that our mapping $\mathcal{M}$ has the following assertions:

```
T_1(x,y) ⇝ :b/{x} :p y ,
T_2(x,y) ⇝ :b/{x} :p y .
```

where `T_1` and `T_2` are source relations with `x` as primary key of type `TEXT`, and `y` of type `INTEGER`. Each assertion in $\mathcal{M}$ translates the tuples in the source relation into RDF triples with IRI subjects `:b/{x}`, predicate `:p` and object `y`. Assume that the user poses the following SPARQL query:

```
SELECT ?x ?y WHERE { ?x :p ?y. }
```

Then, in Step *(i)*, the query is translated into the following IQ:

[4] https://github.com/ontop/ontop
[5] https://teiid.io/
[6] https://www.dremio.com/
[7] https://www.denodo.com/

```
ans(x,y) INTENSIONAL triple(x,:p,y)
```

For simplicity, we assume that the ontology does not contain any axiom, hence Step *(ii)* leaves the IQ unchanged. In Step *)iii*, the triple atom obtained in the previous step is replaced with the mapping assertions for `:p`, which in this case involve `T_1` and `T_2`. The unfolded IQ is:

```
ans(x,y)
CONSTRUCT [x,y] [x/RDF(:b{}(z),IRI), y/RDF(i2t(z1), xsd:integer)]
    UNION [z,z1]
        EXTENSIONAL T_1(z,z1)
        EXTENSIONAL T_2(z,z1)
```

In Step *(iv)*, the final one, the IQ is translated into SQL, tailored to the DBMS dialect. For example, in PostgreSQL, it might become:

```
SELECT z AS x, CAST(z1 AS TEXT) AS y FROM T_1
UNION ALL
SELECT z AS x, CAST(z2 AS TEXT) AS y FROM T_2
```
$\triangleleft$

In the above example, we observe that using IQ facilitates handling complex SPARQL queries and aligns with SQL semantics, which is a potential foundation for update translation in VKGs. Extending this framework to support ontology-to-database updates involves defining update operations in IQ and ensuring consistency, which is what we are investigating in this paper.

### 2.2 Updates in VKGs and Instance Recoveries

In this paper, we focus on two types of instance-level updates in VKGs: either a set of *deletions* or a set of *insertions* over the *retrieved* ABox $\mathcal{M}(D)$. We are interested in translating such ABox update requests into suitable source update requests. For our techniques to be applicable, We need to make the simplifying assumption that the source of each mapping assertion in $\mathcal{M}$ is a union of conjunctive queries (UCQs). Considering more general forms of mappings is left for future work.

We now present the notion of update translation in VKGs as introduced in [33]. We start with the definition of translation for ABox deletions $\mathcal{U}_A^-$ in a VKG instance $\mathcal{J}$. The goal is to find a set of source deletions $\mathcal{U}_D^-$ that can be applied over the database $D$ to generate the desired (*virtual*) ABox through the mapping $\mathcal{M}$. We consider here *direct translations*, which are translations that are of the same nature as the original ABox update, i.e., ABox deletions translate to source deletions, and similarly, ABox insertions translate to source insertions.[8]

**Definition 1 (Direct deletion translation).** Let $\mathcal{U}_A^-$ be a set of ABox deletions. A source deletion $\mathcal{U}_D^-$ is a *direct translation of* $\mathcal{U}_A^-$ *in* $\mathcal{J}$ if $\mathcal{U}_A^- \cap \mathcal{M}(D \setminus \mathcal{U}_D^-) = \emptyset$. We say that $\mathcal{U}_D^-$ is an *exact direct translation of* $\mathcal{U}_A^-$ *in* $\mathcal{J}$ if $\mathcal{M}(D \setminus \mathcal{U}_D^-) = \mathcal{M}(D) \setminus \mathcal{U}_A^-$.$\triangleleft$

Note that in our definition, we use the term *exact translation* for translations whose execution over the data source would create a (*virtual*) ABox that perfectly reflects the

[8] More general forms of translations, where ABox deletions (or insertions) are translated using both deletions and insertions are considered in [34].

requested deletion. However, translations that are not exact will lead to extra deletions in the ABox, and these unintendedly deleted ABox assertions are referred to as *side effects*. We provide a similar definition of translation for ABox insertions.[9]

**Definition 2 (Direct insertion translation).** Let $\mathcal{U}_A^+$ be a set of ABox insertions. A source insertion $\mathcal{U}_D^+$ is a *direct translation of* $\mathcal{U}_A^+$ *in* $\mathcal{J}$ if $\mathcal{U}_A^+ \subseteq \mathcal{M}(\mathcal{U}_D^+)$ and $\langle \mathcal{T}^-, \mathcal{M}(D \cup \mathcal{U}_D^+)\rangle$ is consistent. $\mathcal{U}_D^+$ is an *exact direct translation of* $\mathcal{U}_A^+$ *in* $\mathcal{J}$ if $\mathcal{M}(D \cup \mathcal{U}_D^+) = \mathcal{M}(D) \cup \mathcal{U}_A^+$. ◁

We also require that translations comply with the minimal change principle, i.e., the state of the new VKG system should remain as close as possible to the previous one [16,17].

**Definition 3 (Minimal direct translation).** Let $\mathcal{U}_A^-$ be a set of ABox deletions. A direct translation $\mathcal{U}_D^-$ of $\mathcal{U}_A^-$ in $\mathcal{J}$ is *minimal* if every proper subset of $\mathcal{U}_D^-$ is not a translation of $\mathcal{U}_A^-$ in $\mathcal{J}$. Similarly, let $\mathcal{U}_A^+$ be a set of ABox insertions. A direct translation $\mathcal{U}_D^+$ of $\mathcal{U}_A^+$ in $\mathcal{J}$ is *minimal* if every proper subset of $\mathcal{U}_D^+$ is not a translation of $\mathcal{U}_A^+$ in $\mathcal{J}$. ◁

The main challenge related to translating instance-level updates in VKGs (which is also strongly related to the *view update problem*), is to compute the set of source tuples from which a given set of ABox assertions (to be deleted or inserted) can be generated through the provided schema mapping. In other words, given an ABox deletion $\mathcal{U}_A^-$, how can we compute its *lineage* in the source? Similarly, given an ABox insertion $\mathcal{U}_A^+$, how can we compute the minimal sets of source tuples that generate $\mathcal{U}_A^+$? Given the *non-injective* nature of VKG mappings, there is no unique way to map ABox assertions to their corresponding tuples in the source. To address this, [33] has proposed to compute a reverse mapping $\widehat{\mathcal{M}}$ that approximates the inverse of the original VKG mapping $\mathcal{M}$. This reverse mapping, called the *MR-os-mapping*[10] of $\mathcal{M}$, is constructed by associating each target atom in $\mathcal{T}$ with its rewriting over $\mathcal{S}$ via $\mathcal{M}$. Conceptually, applying $\widehat{\mathcal{M}}$ to an ABox assertion $f$ computes its logical rewriting over the source, thereby capturing all possible source tuples that could produce $f$.

*Example 2.* Assume that our mapping $\mathcal{M}$ contains the assertion `T1(x,y,z)` $\rightsquigarrow$ `:b/{x} :p y`. Then, when the MR-os-mapping $\widehat{\mathcal{M}}$ of $\mathcal{M}$ is applied to the ABox assertion $f =$`(:b/{a} :p c)`, it generates $\widehat{\mathcal{M}}(\{f\}) = \exists z.$`T1(a,c,`$z$`)`. ◁

Our goal is to compute the source tuples from which a set of ABox assertions can be derived. To achieve this, we leverage the MR-os-mapping recovery of each assertion, producing a logical condition over $\mathcal{S}$ that describes all generating source sets. We formalize this with the notion of *instance recovery*, defined as follows, where we assume that databases are expressed over a predefined domain $\Delta$ of values, and we use $\mathcal{S}(\Delta)$ to express the set of all facts obtained by applying the predicates in $\mathcal{S}$ to the values in $\Delta$.

[9] Notice that we adopt here the definition of direct translation of ABox insertions proposed in [34], where the translation contains *all* source tuples necessary to generate the inserted ABox facts, including tuples already present in the data source, as opposed to the definition of [33], where the translation contains only the new tuples to be added to the data source to accomplish the ABox insertion.

[10] MR-os-mapping stands for *maximum recovery* from the ontology to the source schema.

**Definition 4 ($\widehat{\mathcal{M}}$-instance recovery [33]).** Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be a VKG instance and $\widehat{\mathcal{M}}$ the MR-os-mapping of $\mathcal{M}$. The $\widehat{\mathcal{M}}$-*instance recovery* is the function $\widehat{\mathcal{M}}_{\mathcal{S}}$ from ABox facts over $\mathcal{T}$ to $2^{2^{\mathcal{S}(\Delta)}}$ such that

$$\widehat{\mathcal{M}}_{\mathcal{S}}(f) = \{B \subseteq \mathcal{S}(\Delta) \mid B \text{ is true in } \widehat{\mathcal{M}}(\{f\}), \text{ and } \forall B' \subsetneq B, B' \text{ is not true in } \widehat{\mathcal{M}}(\{f\})\}. \quad \triangleleft$$

Notice that the above definition of instance recovery differs from the one provided in [33], which relied on a specific source instance $D$. Here instead, we abstract from such a source instance and consider instead the minimal source instances $B$ contained in the (infinite) set $\mathcal{S}(\Delta)$ of facts from which an ABox fact $f$ can be generated through $\mathcal{M}$. Notice that, while each $B$ in $\widehat{\mathcal{M}}_{\mathcal{S}}(f)$ is finite (since minimal), in general $\widehat{\mathcal{M}}_{\mathcal{S}}(f)$ will contain an infinite number of such source instances. For ABox deletions, we will restrict the set of such source instances to those contained in the given source instance $D$ (for $\mathcal{S}$). Instead, for ABox insertions we will have to find suitable representatives in $\widehat{\mathcal{M}}_{\mathcal{S}}(f)$ that lead to minimal insertion side-effects through $\mathcal{M}$.

## 3 Maximum Recovery in *Ontop*

As we mentioned earlier, *Ontop* excels at exposing relational databases as virtual RDF graphs through the ontologies and the mappings, which allows SPARQL queries to be answered without materializing the triples. *Ontop* 's query answering process relies on rewriting SPARQL queries into SQL queries using mappings and ontology inference [4]. However, translating ABox assertions (instance-level facts) into database tuples for updates is not supported in *Ontop*. In this section, we show how we can exploit the rewriting capabilities of *Ontop* to efficiently compute the rewriting of ABox assertions, which then allows us to compute source updates for ABox updates.

In *Ontop*, given a VKG specification $\langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, a SPARQL query $\mathcal{Q}_{\mathcal{T}}$ over $\mathcal{T}$ is translated into an SQL query $\mathcal{Q}_{\mathcal{S}}$ over $\mathcal{S}$ by using Intermediate Query (IQ), which is a language combining basic algebraic operations like `UNION`, `JOIN`, `PROJECT`, and `SLICE`. In our proposed method, we leverage that translation by deriving from the target triple (e.g., $f$ = `(:b/{a} :p b)`) a ground `ASK` query (e.g., `ASK {` $f$ `.}`), and *Ontop*'s query engine then translates this SPARQL query into an IQ. For `ASK` queries, the unfolded IQ typically includes a `SLICE` (or `LIMIT` 1) construct which checks for the existence of any set of source tuples satisfying the IQ. By programmatically removing this `SLICE` construct from the unfolded IQ, the resulting IQ transforms the existential check (with the `SLICE` constructor) into a comprehensive retrieval mechanism.

*Example 3 (Example 1 cont'd).* If we have the RDF triple $f$ = `(:b/{a} :p b)`, from which we form the `ASK` query `ASK {` $f$ `. }`, the unfolded IQ will be:

```
ans1() SLICE limit=1
UNION []
    SLICE limit=1 EXTENSIONAL T_1(0:a^^TEXT,1:b^^INTEGER)
    SLICE limit=1 EXTENSIONAL T_2(0:a^^TEXT,1:b^^INTEGER)
```
◁

Another challenge in computing the MR-os-mapping is matching the assertions' IRIs to mapping assertions, which requires handling IRI templates and their inverses.

For an ABox assertion $f$ = (:b/{a} :p b), *Ontop* identifies mappings whose target triples unify with $f$. This involves: *(i)* matching p to the mapping's predicate, *(ii)* applying the inverse templates to :b/{a} and b to extract database values, and *(iii)* using the extracted values to construct the unfolded IQ. In the next sections, we will show how the notion of MR-os-mapping can be used in the translation of updates within the *Ontop* framework.

## 4 Update Translation in *Ontop*

SPARQL [20] is known as the standard query language for RDF graphs, however, it is limited to read-only access to a knowledge graph. In order to overcome the limitation, W3C introduced SPARQL Update [19] as a standardized language for modifying RDF graphs, supporting operations such as insertions and deletions of RDF triples. Insertions are specified using the `INSERT DATA` construct, which adds new triples to the graph. For example, `INSERT DATA { :student/1 :hasName "Alice" }` adds a triple stating that a student has a specific name. Similarly, deletions are specified using `DELETE DATA` to remove specific triples from the graph. In VKGs, these update operations must be translated into equivalent SQL update statements that modify the underlying database through mappings. In this section, we first provide a running example that we use throughout the rest of the paper to illustrate the update problems in VKGs and our solution in *Ontop*.

### 4.1 Running Example

We present a running example involving a university database with two relational tables: `Student(id,name,faculty)` and `Faculty(id,course)`. The `Student` table stores student records with a unique identifier (`id`), name (`name`), and their enrolled faculty (`faculty`). The `Faculty` table specifies the mandatory courses (`course`, e.g., "Math") for each faculty. Based on that source schema, we define the following mappings:

– Mapping $m_1$ maps `Student` records to the `:hasName` data property:
```
SELECT id, name FROM Student
  ⇝ :uni/student/{id} :hasName {name}.
```
– Mapping $m_2$ maps `Student` and `Faculty` records to the `:isTaking` data property:
```
SELECT s.id AS id, f.course AS course
FROM Student s JOIN Faculty f ON s.faculty = f.id
  ⇝ :uni/student/{id} :isTaking {course}^^xsd:string .
```

The above mapping $\mathcal{M} = \{m_1, m_2\}$ is specified in *Ontop*'s native mapping language[11], where each mapping assertion consist of *(i)* a source, which is an SQL query, and *(ii)* a target, which is an RDF triple pattern with placeholders (enclosed in '{' and '}') for the values of answer variables from the source query. When applied over a

[11] See https://github.com/ontop/ontop/wiki/ontopOBDAModel for a full description of *Ontop*'s mapping language.

source instance, these mappings create a virtual RDF graph where students are linked to their names and the courses they are taking. Consider the database $D$ shown below on the left-hand side:

| *Student* | | |
|---|---|---|
| *id* | *name* | *faculty* |
| s1 | john | f1 |
| s1 | john | f2 |
| s2 | paul | f2 |

| *Faculty* | |
|---|---|
| *id* | *course* |
| f1 | ethics |
| f2 | ethics |
| f1 | law |

$\mathcal{M}(D)$:

```
:uni/student/s1 :hasName "john"
:uni/student/s2 :hasName "paul"
:uni/student/s1 :isTaking "ethics"
:uni/student/s1 :isTaking "law"
:uni/student/s2 :isTaking "ethics"
```

By applying the mapping $\mathcal{M}$ over $D$, we obtain the (virtual) ABox $\mathcal{M}(D)$ shown above on the right-hand side. Assume now that we have also an OWL 2 QL ontology containing the following inclusion assertion:

```
:isTaking rdfs:subPropertyOf :isStudying
```

By using T-mappings [23] w.r.t. the above ontology (which essentially encode the TBox axioms into the mapping), the ABox is enriched with the assertions: (`:uni/student/s1 :isStudying "ethics"`), (`:uni/student/s1 :isStudying "law"`), and (`:uni/student/s2 :isStudying "ethics"`).

## 4.2 Translation of SPARQL `DELETE`

To translate a SPARQL `DELETE` operation into an update over the underlying data source in *Ontop*, we propose an approach grounded in the notion of data *lineage* [12,9] and MR-os-mapping for VKG mappings (see Section 3). When a user requests the deletion of an ABox assertion $f$, our goal is to compute the minimal set of deletions over the source instance that ensures $f$ is no longer entailed in the virtual ABox generated by the VKG. Central to our approach is the observation that if $f$ currently holds in the virtual ABox, it must have been generated by at least one mapping assertion applied to a subset of the database. We define the *lineage* of $f$ as the set of all minimal subsets of the source instance (called *lineage branches*) that individually suffice to produce $f$. This is captured by the following definition:

**Definition 5 (Lineage [33]).** Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ be a VKG specification, $\langle \mathcal{P}, D \rangle$ a VKG instance, and $f \in \mathcal{M}(D)$ an ABox assertion. A subset $B \subseteq D$ is a *lineage branch of* $f$ if $f \in \mathcal{M}(B)$, and for every $B' \subsetneq B$, $f \notin \mathcal{M}(B')$. The *lineage of* $f$, denoted $\mathsf{lineage}(f, \mathcal{P}, D)$, is the set of all lineage branches of $f$. $\triangleleft$

Each lineage branch represents a distinct witness of $f$ in the data source through $\mathcal{M}$. It was proven in [33] that for the MR-os-mapping $\widehat{\mathcal{M}}$, every set $B$ in $\widehat{\mathcal{M}}_{\mathcal{S}}(f)$ such that $B \subseteq D$ is a lineage branch of $f$. Hence, to formally capture these branches in a source instance $D$, we select from its instance recovery $\widehat{\mathcal{M}}_{\mathcal{S}}(f)$ those sets that are included in $D$. In practice, these branches are obtained by searching sets of tuples in $D$ that are true in $\widehat{\mathcal{M}}(\{f\})$ (which is the source IQ).

*Example 4 (Continued from Section 4.1).* The lineage in $D$ of the ABox assertion $f =$ (`:uni/student/s1 :isTaking "ethics"`) is $\mathsf{lineage}(f, \mathcal{P}, D) = \{B_1, B_2\}$, where

$$\begin{aligned} B_1 &= \{\texttt{Student(s1,john,f1)},\ \texttt{Faculty(f1,ethics)}\}, \\ B_2 &= \{\texttt{Student(s1,john,f2)},\ \texttt{Faculty(f2,ethics)}\}. \end{aligned}$$ $\triangleleft$

Since the lineage branches are minimal (according to their definition), removing at least one tuple from each of them is enough to delete $f$ from $\mathcal{M}(D)$. However, to minimize side effects on other ABox assertions, we select combinations of deletions that include exactly one tuple per branch, and that intersect minimally with the lineage of other assertions. This corresponds to computing a *hitting set* over the lineage branches, constrained to avoid tuples that also contribute to other assertions in the ABox.

*Example 5 (Continued from Example 4).* The *hitting set* over the lineage branches of $f$ is $\mathbf{T} = \{T_1, T_2, T_3, T_4\}$ where

$$\begin{aligned} T_1 &= \{\texttt{Student(s1,john,f1)},\ \texttt{Student(s1,john,f2)}\},\\ T_2 &= \{\texttt{Student(s1,john,f1)},\ \texttt{Faculty(f2,ethics)}\},\\ T_3 &= \{\texttt{Faculty(f1,ethics)},\ \texttt{Student(s1,john,f2)}\},\\ T_4 &= \{\texttt{Faculty(f1,ethics)},\ \texttt{Faculty(f2,ethics)}\}. \end{aligned}$$ ◁

We observe that each *hitting set* over the lineage branches of an ABox assertion represents a minimal set of source tuples to delete in $D$ to realize the deletion of that assertion. Algorithm 1 formalizes the translation of a SPARQL `DELETE` request into a minimal SQL `DELETE` request over the source instance of $\mathcal{J}$. Our algorithm takes as input $\mathcal{J}$, a SPARQL `DELETE` request $\mathcal{Q}^{-}_{\mathcal{T}}$, and a distance metric $\Delta^{-}_{\mathcal{J}}$ to guide side-effect minimization[12]. The algorithm begins by extracting from $\mathcal{Q}^{-}_{\mathcal{T}}$ the set $F$ of ABox assertions targeted for deletion (at Line 1), and the lineage $\mathbf{B}$ of $F$ in $D$ (which is the union of the lineage branches of all assertions in $F$). Since deleting the set $F$ amounts to deleting at least one tuple in each lineage branch of its elements, the algorithm computes the *hitting set* of $\mathbf{B}$ (at Line 3) using the COMPUTEHITTINGSETS algorithm. Then the algorithm selects the subset $\mathbf{U}^{-}_{D} \subseteq \mathbf{T}$ of minimal translations according to $\Delta^{-}_{\mathcal{J}}$. Finally, each selected translation is converted into a concrete SQL `DELETE` over $\mathcal{S}$ (using CONVERTTODELSQL) and returned to the user as a possible translation.

*Example 6 (Cont.'d from Example 5).* Consider the request $\mathcal{Q}^{-}_{\mathcal{J}}$: `DELETE DATA {` $f$ `}`. For any comparison metric $\Delta^{-}_{\mathcal{J}}$, the only set of tuples to delete is $T_3$, since its deletion leads to an empty side-effect. Therefore, Algorithm 1 returns the SQL `DELETE`:

```
DELETE FROM Student WHERE id="s1" AND name="john" AND faculty="f2";
DELETE FROM Faculty WHERE id="f1" AND course="ethics".
```
◁

### 4.3 Translation of SPARQL `INSERT`

Unlike deletions, the translation of a SPARQL `INSERT` request for a set $F$ of ABox assertions in VKGs presents a fundamentally different challenge: the ABox assertions targeted for insertion do not yet exist in the database. Therefore, we cannot simply manipulate the database to realize the intended insertion. Instead, we must construct new source tuples such that, when the VKG mapping is applied, the desired assertion appears in the virtual ABox. To address this, we apply the MR-os-mapping over the set of ABox assertions to be inserted. This leads to an unfolded IQ, which serves as a logical

[12] $\Delta^{-}_{\mathcal{J}}$ can also encode heuristics such as tuple preservation or data provenance cost.

**Algorithm 1:** ImpTransDel

```
input : A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$ with $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$ where $\mathcal{T}$ is specified in
        OWL 2 QL and $\mathcal{M}$ in R2RML.
        A SPARQL DELETE request $\mathcal{Q}^{-}_{\mathcal{T}}$ in $\mathcal{T}$ and distance metric $\Delta^{-}_{\mathcal{J}}$.
output: A set of SQL DELETE request expressed in $\mathcal{S}$.
1 $F \leftarrow$ Set of ABox assertions contained in $\mathcal{Q}^{-}_{\mathcal{T}}$;
2 $\mathbf{B} \leftarrow \bigcup_{f \in F}$ lineage$(f, \mathcal{P}, D)$;
3 $\mathbf{T} \leftarrow$ ComputeHittingSets$(\mathbf{B})$;
4 $\mathbf{U}^{-}_{D} \leftarrow$ Translations in $\mathbf{T}$ that are minimal according to $\Delta^{-}_{\mathcal{J}}$;
5 return $\{$ConvertToDelSQL$(T) \mid T \in \mathbf{U}^{-}_{D}\}$
```

specification of the source data needed to generate $F$. The problem then becomes one of finding a suitable set of source tuples that satisfy the unfolded IQ.

To accomplish this, we propose to rewrite the set $F$ as a whole in the source schema. A key advantage of using *Ontop* to achieve this, lies in the fact that the resulting rewritten IQ reflects a unified logical condition that captures all possible ways to jointly generate $F$ through the original VKG mapping $\mathcal{M}$. *Ontop* leverages a unification mechanism at the level of the source schema $\mathcal{S}$, which merges repeated variables and shared join paths across the different assertions in $F$, and also correctly deals with IRI templates. This avoids redundancy and reveals shared existential dependencies.

*Example 7 (Cont.'d from Section 4.1).* Consider the set $F$ of ABox assertions:

```
{ (:uni/student/s3 :hasName "smith");
  (:uni/student/s3 :isTaking "art")   }
```

Rewriting each assertion separately yields two IQs that are respectively equivalent to the CQs $\exists id.\texttt{Student}(\texttt{"s3"}, \texttt{"smith"}, id)$ and $\exists name, id.\texttt{Student}(\texttt{"s3"}, name, id) \wedge \texttt{Faculty}(id, \texttt{"art"})$. Instead, by rewriting the entire set $F$ as a single query, *Ontop* produces the unified IQ $\exists id.\texttt{Student}(\texttt{"s3"}, \texttt{"smith"}, id) \wedge \texttt{Faculty}(id, \texttt{"art"})$, where the only existential variable $id$ is shared across the two atoms. ◁

The first challenge when translating SPARQL INSERT operations arises when the set of RDF triples includes multiple distinct subjects. Since each subject typically corresponds to a distinct entity in the domain, a set of triples with different subjects will generate several distinct IQs in the source. Therefore, to ensure an efficient translation, we group the ABox assertions by subject before applying the rewriting. Within each group, the triples collectively represent attributes or relationships of the same entity and are therefore more likely to share a common source predicate or join path.

Once an MR-os-mapping $\widehat{\mathcal{M}}$ is applied to each group of ABox assertions in the provided SPARQL INSERT request, we obtain an unfolded IQ $\widehat{\mathcal{M}}(F)$ over the source, from which we have to search for sets $B$ of source tuples such that $B$ is true in $\widehat{\mathcal{M}}(F)$ with minimum side-effect. Since the sets $B$ are not restricted to $B \subseteq D$ (as is the case for deletion), the search space of such $B$ can be infinite, due to the presence of existential variables in $\widehat{\mathcal{M}}(F)$. To narrow this space, we decompose each CQ $q$ in $\widehat{\mathcal{M}}(F)$ and compute its *maximum satisfiable subsets* [2] — the subqueries that evaluate to true over $D$, with the additional condition that all variables that occur in such a maximum satisfiable

**Algorithm 2:** ImpTransIns

```
input : A VKG instance $\mathcal{J} = \langle \mathcal{P}, D \rangle$ with $\mathcal{P} = \langle \mathcal{T}, \mathcal{M}, \mathcal{S} \rangle$, where $\mathcal{T}$ is specified in
        OWL 2 QL and $\mathcal{M}$ in R2RML.
        A SPARQL INSERT request $\mathcal{Q}^+_{\mathcal{T}}$ in $\mathcal{T}$ and distance metric $\Delta^+_{\mathcal{J}}$.
output: A set of SQL INSERT request expressed in $\mathcal{S}$.
1  $\mathbf{T} \leftarrow \{\}$;    $F \leftarrow$ set of ABox assertions contained in $\mathcal{Q}^+_{\mathcal{T}}$;
2  for each group $G \in$ GroupBySubject$(F)$ do
3      for each $q \in \widehat{\mathcal{M}}(G)$ do
4          $\Sigma \leftarrow$ SatSubset$(q, D) \cup \{(\emptyset, \emptyset)\}$;
5          for each $(B_\sigma, \sigma) \in \Sigma$ do
6              $v_q \leftarrow \mathsf{vars}(q \setminus B_\sigma)$;
7              $\Delta_q = \{b_1, \ldots, b_{|v_q|}\} \cup \Delta_A$, where $\Delta_A$ are the constants in $G$ and each $b_i$
                 is a fresh value not in $\Delta_A$ and not appearing in $D$;
8              $\Gamma \leftarrow$ Assign$(q \setminus B_\sigma, \Delta_q, D)$;
9              for each $(B_\gamma, \gamma) \in \Gamma$ do
10                 $\mathbf{T} \leftarrow \mathbf{T} \cup \{B_\sigma[\sigma] \cup B_\gamma[\gamma]\}$;
11     $\mathbf{U}^+_D \leftarrow$ translations in $\mathbf{T}$ that are minimal according to $\Delta^+_{\mathcal{J}}$
12 return $\{$ConvertToInsSQL$(T) \mid T \in \mathbf{U}^+_D\}$
```

subset $s \subseteq q$ do not occur in any atom of $q$ not in $s$. These satisfiable fragments allow for partial grounding of variables, from which we infer the values of the remaining existential variables. Notice that the additional condition that no variable $x$ occurring in $s$ occurs also outside of $s$ is necessary in order to ensure that we do not lose solutions with minimal side-effect. Indeed, when considering a maximum satisfiable subset $s$, we enforce the instantiation of all variables in $s$ via the facts already in $D$, and we want to avoid that such instantiation constains also atoms (not already in $D$) that need to be inserted to accomplish the ABox insertion according to $q$.

Algorithm 2 outlines the process of translating a SPARQL `INSERT` request into SQL `INSERT` statements over a VKG instance $\mathcal{J}$. The algorithm begins by grouping the target ABox assertions $F$ by subject to ensure that related triples are rewritten together (using GroupBySubject), capturing shared existential dependencies. For each group $G$, the algorithm applies the MR-os-mapping to get $\widehat{\mathcal{M}}(G)$ (at Line 3), returning a set of CQs in $\mathcal{S}$. Each CQ is then analyzed to identify its maximal satisfiable subsets with respect to the current database $D$ (with the additional condition on variables described above). This is achieved by using the SatSubset algorithm, which takes as input a CQ $q$ and a source instance $D$, and returns a set of pairs $(B_\sigma, \sigma)$, where $\sigma$ is a partial binding of variables in $q$ with values in $D$, and $B_\sigma \subseteq q$ is the set of atoms corresponding to $\sigma$. For a variable binding $\sigma$ and a set $B_\sigma$ of atoms, we use $B_\sigma[\sigma]$ to denote the set of atoms obtained from $q$ by binding the variables in $q$ according to $\sigma$.

The remaining variables in $q \setminus B_\sigma$ (from the unsatisfiable subset) are instantiated either from known values in $D$ or by introducing fresh values (at Line 8). This is achieved with the Assign function, which takes as input $q \setminus B_\sigma$ and by using its existential variables as answer variables, queries from $D$ the set of possible assignments, on top of which the fresh values from $\Delta_q$ are added. For each assignment returned by the Assign function, we create an insertion candidate, which we add to $\mathbf{T}$ (at Lines 10) from which a minimal subset $\mathbf{U}_D^+$ is selected using a comparison metric $\Delta_{\mathcal{J}}^+$. The selected instantiations are then translated into SQL `INSERT` statements (using CONVERTTOINSSQL), ensuring that the intended ABox assertions are derivable from the updated source.

*Example 8 (Cont.'d from Example 7).* Consider the request $\mathcal{Q}_{\mathcal{T}}^+$ : `INSERT DATA {` $F$ `}`. From its unified source IQ (which in this case has no satisfiable subset), we observe that a fresh value assigned to the variable $id$ will lead to an empty side-effect (assuming $\Delta_{\mathcal{J}}^+$ is based on cardinality). Therefore, a possible source translation is the following:

```
INSERT INTO student VALUES (s3, "smith", f8)
INSERT INTO faculty VALUES (f8, "art")
```

◁

## 5 Empirical Results

To evaluate the practical applicability and scalability of our proposed translation algorithms for SPARQL Update, we conducted two complementary experiments based on our running example. In the first experiment, we varied the database size from 1 to 2,000 students and from 1 to 500 faculties (with each faculty having two students). For insertion, we tested adding a fresh grouped triple that requires creating new source tuples, since no satisfiable rewriting exists. For deletion, we focused on removing an existing `foaf:isTaking` triple, which consists of two relations in the source schema. The results show that insertion time grows significantly with database size due to the need to search for translations that minimize side effects, while deletion time remains relatively stable since it depends only on the fixed lineage branch of the target assertion.

In the second experiment, we fixed the database size but progressively increased the number of lineage branches per assertion from 1 to 15. We observed that deletion time can grow exponentially with the number of branches, reaching up to 23 minutes for 15 branches. However, by employing early stopping strategies (terminating as soon as an exact translation is found or one below a side-effect threshold), runtime was drastically reduced. Together, these experiments show that the insertion cost scales primarily with the database size, while the deletion cost scales with lineage complexity, and both can be significantly optimized using practical heuristics such as early stopping, prioritizing fresh value insertions, or even allowing null values in the data source.

## 6 Conclusions

In this paper, we addressed the challenge of translating instance-level update requests in Virtual Knowledge Graphs into executable source-level operations within the *Ontop* framework. We introduced a novel approach based on maximum recovery, which enables us to systematically rewrite ABox assertions into source queries and thus compute the minimal source updates. We have demonstrated how such a mapping can be obtained by leveraging the *Ontop* rewriting engine, and we proposed translation algorithms for both insertion and deletion of ABox assertions. Through extensive experiments, we demonstrated how the insertion cost scales with the database size, while the deletion cost scales with lineage complexity, and we showed that practical heuristics

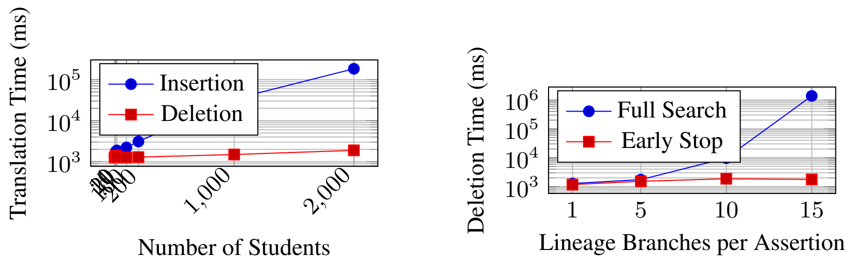

Fig. 1: Translation time scaling: (a) with database size; (b) with lineage branch complexity.

such as early stopping can significantly improve performance. Our results highlight the feasibility of supporting precise and side-effect-aware updates in VKG systems like *Ontop*, paving the way for more expressive and update-capable virtualized semantic data management.

**Acknowledgments.** This research has been partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, by the HEU project CyclOps (GA n. 101135513), by the Province of Bolzano and FWF through project OnTeGra (DOI 10.55776/PIN8884924). by the Province of Bolzano and EU through projects ERDF-FESR 1078 CRIMA, and ERDF-FESR 1047 AI-Lab, by MUR through the PRIN project 2022XERWK9 S-PIC4CHU, and by the EU and MUR through the PNRR project PE0000013-FAIR.

## References

1. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM Trans. on Database Systems **6**(4), 557–575 (1981). https://doi.org/10.1145/319628.319634
2. Bendík, J.: On decomposition of maximal satisfiable subsets. In: Proc. of Formal Methods in Computer Aided Design (FMCAD). pp. 212–221. IEEE (2021). https://doi.org/10.34727/2021/ISBN.978-3-85448-046-4_30
3. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF Schema. W3C Recommendation, World Wide Web Consortium (Feb 2004), https://www.w3.org/TR/rdf-schema/
4. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web J. **8**(3), 471–487 (2017). https://doi.org/10.3233/SW-160217
5. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R.: Ontologies and databases: The *DL-Lite* approach. In: Reasoning Web: Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW), Lecture Notes in Computer Science, vol. 5689. Springer (2009). https://doi.org/10.1007/978-3-642-03754-2_7
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The Mastro system for ontology-based data access. Semantic Web J. **2**(1), 43–53 (2011). https://doi.org/10.3233/SW-2011-0029

7. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Updating ABoxes in *DL-Lite*. In: Proc. of the 4th Alberto Mendelzon Int. Workshop on Foundations of Data Management (AMW). CEUR Workshop Proceedings, vol. 619, pp. 3.1–3.12. CEUR-WS.org (2010)
8. Chebotko, A., Lu, S., Fotouhi, F.: Semantics preserving SPARQL-to-SQL translation. Data and Knowledge Engineering **68**(10), 973–1000 (2009). https://doi.org/10.1016/J.DATAK.2009.04.001
9. Cui, Y., Widom, J.: Lineage tracing for general data warehouse transformations. The VLDB J. **12**(1), 41–58 (2003). https://doi.org/10.1007/s00778-002-0083-8
10. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C Recommendation, World Wide Web Consortium (Sep 2012), https://www.w3.org/TR/r2rml/
11. Dayal, U., Bernstein, P.A.: On the updatability of relational views. In: Proc. of the 4th Int. Conf. on Very Large Data Bases (VLDB). vol. 78, pp. 368–377 (1978). https://doi.org/10.14778/3415478.3415503
12. Dayal, U., Bernstein, P.A.: On the correct translation of update operations on relational views. ACM Trans. on Database Systems **7**(3), 381–416 (1982). https://doi.org/10.1145/319732.319740
13. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On instance-level update and erasure in description logic ontologies. J. of Logic and Computation **19**(5) (2009). https://doi.org/10.1093/logcom/exn051
14. De Giacomo, G., Oriol, X., Rosati, R., Savo, D.F.: Instance-level update in DL-Lite ontologies through first-order rewriting. J. of Artificial Intelligence Research **70**, 1335–1371 (2021). https://doi.org/10.1613/jair.1.12414
15. Eisenberg, V., Kanza, Y.: D2RQ/update: Updating relational data via virtual RDF. In: Proc. of the 21st Int. World Wide Web Conf. (WWW). pp. 497–498 (2012). https://doi.org/10.1145/2187980.2188095
16. Eiter, T., Gottlob, G.: On the complexity of propositional knowledge base revision, updates and counterfactuals. Artificial Intelligence **57**, 227–270 (1992). https://doi.org/10.1016/0004-3702(92)90018-S
17. Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change: Classification and survey. Knowledge Engineering Review **23**(2), 117–152 (2008). https://doi.org/10.1017/S0269888908001367
18. Garrote Hernandez, A., García, M.N.M.: RESTful writable APIs for the web of Linked Data using relational storage solutions. In: Proc. of WWW Workshop on Linked Data on the Web (LDOW). CEUR Workshop Proceedings, vol. 813. CEUR-WS.org (2011), https://ceur-ws.org/Vol-813/ldow2011-paper04.pdf
19. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 update. W3C Recommendation, World Wide Web Consortium (Mar 2013), https://www.w3.org/TR/sparql11-update/
20. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C Recommendation, World Wide Web Consortium (Mar 2013), https://www.w3.org/TR/sparql11-query
21. Hert, M., Reif, G., Gall, H.C.: Updating relational data via SPARQL/update. In: Proc. of the EDBT/ICDT Workshops. pp. 1–8 (2010). https://doi.org/10.1145/1754239.1754266
22. Katsuno, H., Mendelzon, A.: On the difference between updating a knowledge base and revising it. In: Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR). pp. 387–394 (1991)
23. Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., Zakharyaschev, M.: Answering SPARQL queries over databases under OWL 2 QL entailment regime. In: Proc. of the 13th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, Springer (2014). https://doi.org/10.1007/978-3-319-11964-9_35
24. Kotidis, Y., Srivastava, D., Velegrakis, Y.: Updates through views: A new hope. In: Proc. of the 22nd IEEE Int. Conf. on Data Engineering (ICDE). p. 2 (2006). https://doi.org/10.1109/ICDE.2006.167

25. Langerak, R.: View updates in relational databases with an independent scheme. ACM Trans. on Database Systems **15**(1), 40–66 (1990). https://doi.org/10.1145/77643.77645
26. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Foundations of instance level updates in expressive description logics. Artificial Intelligence **175**(18), 2170–2197 (2011). https://doi.org/10.1016/J.ARTINT.2011.08.003
27. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, World Wide Web Consortium (Dec 2012), https://www.w3.org/TR/owl2-profiles/
28. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language profiles (second edition). W3C Recommendation, World Wide Web Consortium (Dec 2012), http://www.w3.org/TR/owl2-profiles/
29. Priyatna, F., Corcho, O., Sequeda, J.F.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In: Proc. of the 23rd Int. World Wide Web Conf. (WWW). pp. 479–490 (2014). https://doi.org/10.1145/2566486.2567981
30. Ramanujam, S., Khadilkar, V., Khan, L., Seida, S., Kantarcioglu, M., Thuraisingham, B.: Bi-directional translation of relational data into virtual RDF stores. In: Proc. of the 4th IEEE Int. Conf. on Semantic Computing (ICSC). pp. 268–276. IEEE Computer Society (2010). https://doi.org/10.1109/ICSC.2010.61
31. Rodriguez-Muro, M., Rezk, M.: Efficient SPARQL-to-SQL with R2RML mappings. J. of Web Semantics **33**, 141–169 (2015). https://doi.org/10.1016/j.websem.2015.03.001
32. Unbehauen, J., Martin, M.: Executing SPARQL queries over mapped document store with SparqlMap-M. In: Proc. of the 12th Int. Conf. on Semantic Systems (SEMANTICS). pp. 137–144. ACM (2016). https://doi.org/10.1145/2993318.2993326
33. Wandji, R.E., Calvanese, D.: Ontology-based update in Virtual Knowledge Graphs via schema mapping recovery. In: Proc. of the 8th Int. Joint Conf. on Rules and Reasoning (RuleML+RR). Lecture Notes in Computer Science, vol. 15183, pp. 59–74. Springer (2024). https://doi.org/10.1007/978-3-031-72407-7_6
34. Wandji, R.E., Calvanese, D.: Minimizing side-effects in virtual knowledge graph updates. In: Proc. of the 9th Int. Joint Conf. on Rules and Reasoning (RuleML+RR). Lecture Notes in Computer Science, Springer (2025)
35. Winslett, M.: A model-based approach to updating databases with incomplete information. ACM Trans. on Database Systems **13**(2), 167–196 (1988). https://doi.org/10.1145/42338.42386
36. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI). pp. 5511–5519. IJCAI Org. (2018). https://doi.org/10.24963/ijcai.2018/777
37. Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalayci, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., Botoeva, E.: The virtual knowledge graph system Ontop. In: Proc. of the 19th Int. Semantic Web Conf. (ISWC). Lecture Notes in Computer Science, vol. 12507, pp. 259–277. Springer (2020). https://doi.org/10.1007/978-3-030-62466-8_17
38. Zheleznyakov, D., Kharlamov, E., Nutt, W., Calvanese, D.: On expansion and contraction of DL-Lite knowledge bases. J. of Web Semantics **57**, 100484 (2019). https://doi.org/10.1016/j.websem.2018.12.002