Check for updates

Ontology-based Data Federation

Zhenzhen Gu
Free University of Bozen-Bolzano
Bolzano, Italy

Davide Lanti
Free University of Bozen-Bolzano
Bolzano, Italy

Alessandro Mosca
Free University of Bozen-Bolzano
Bolzano, Italy

Guohui Xiao
University of Bergen
Bergen, Norway
University of Oslo
Oslo, Norway
Ontopic S.r.l.
Bolzano, Italy

Jing Xiong
Free University of Bozen-Bolzano
Bolzano, Italy

Diego Calvanese
Free University of Bozen-Bolzano
Bolzano, Italy
Umeå University
Umeå, Sweden
Ontopic S.r.l.
Bolzano, Italy

ABSTRACT

Ontology-based data access (OBDA) is a well-established approach to information management which facilitates the access to a (single) relational data source through the mediation of a high-level ontology, and the use of a declarative mapping linking the data layer to the ontology. We formally introduce here the notion of *ontology-based data federation* (OBDF) to denote a framework that combines OBDA with a data federation layer where multiple, possibly heterogeneous sources are virtually exposed as a single relational database. We discuss opportunities and challenges of OBDF, and provide techniques to deliver efficient query answering in an OBDF setting. Such techniques are validated through an extensive experimental evaluation based on the Berlin SPARQL Benchmark.

CCS CONCEPTS

• **Information systems** → **Data federation tools**.

KEYWORDS

OBDA, Data federation, Query optimization

ACM Reference Format:
Zhenzhen Gu, Davide Lanti, Alessandro Mosca, Guohui Xiao, Jing Xiong, and Diego Calvanese. 2022. Ontology-based Data Federation. In *11th International Joint Conference On Knowledge Graphs (IJCKG 2022), October 27–28, 2022, Hangzhou, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3579051.3579070

1 INTRODUCTION

Ontology-based data access (OBDA) [10, 33, 42] is a well-established paradigm for querying data sources via a mediating ontology that has been successfully applied in many different domains [43]. In OBDA, the ontology is expressed in a lightweight conceptual language, such as OWL 2 QL [30], which has its formal foundations in the description logics of the DL-Lite family [12]. Typically, it is assumed that the underlying data are stored in a single relational data source, to which the ontology elements are mapped in a declarative way. Specifically, in each *mapping*, a SQL query over the source is mapped to a class / property of the ontology, specifying how the data retrieved from the database (DB) should be used to create instances and values that populate the class / property.

Notably, for query answering, OBDA follows a *virtual* approach, i.e., the data are not actually extracted from the source to populate the classes and properties, but instead a SPARQL query posed over the ontology is transformed on-the-fly into a SQL query over the data source. Such transformation takes into account both the ontology axioms (in what is generally called a *rewriting* step [12]) and the mappings (in an *unfolding* step [33, 34]), and typically may lead to a substantial blow-up in the size of the resulting SQL query w.r.t. the size of the original query. Due to this, sophisticated optimization techniques have been proposed and implemented in commercial and open source OBDA systems [10, 11, 38, 45]. Such techniques exploit the available information about constraints in the data source (e.g., primary and foreign keys), the form of the mappings, and the structure of the query in order to optimize the SPARQL-to-SQL query translation process and generate a final query that is not only as compact as possible but also efficient to execute [34, 44].

So far, such techniques have been tailored towards optimizing queries that are executed over a *single* data source to which the OBDA system is mapped. In many settings, however, there is the need to virtually access multiple, possibly heterogeneous, data sources in an integrated way. In this case, one can resort to *data federation* [20, 39], where multiple autonomous data sources are exposed transparently as a unified federated relational schema, usually called *virtual database*. Data federation is an active research area which has been extensively studied over the years, and many mature and highly-optimized data federation tools are currently available, both in the database community and in the Semantic Web community [19].

Data federation tools can be naturally used in combination with OBDA systems, by accessing them as if they were a single relational data source[1]. However, to the best of our knowledge, in current OBDA systems no provision is taken for the optimization of the generated SQL query to account for the fact that the evaluation of

[1]See, e.g., https://ontop-vkg.org/tutorial/federation/.

IJCKG 2022, October 27–28, 2022, Hangzhou, China

ACM ISBN 978-1-4503-9987-6/22/10.
https://doi.org/10.1145/3579051.3579070

a SQL query in a data federation system is fundamentally different from query evaluation by a standard relational DBMS engine. In this work, we consider specifically this issue and provide the following contributions to this novel setting that we call *Ontology-based Data Federation* (OBDF for short):

- We provide a formalization of OBDF systems, where a collection of multiple, possibly heterogeneous, federated data sources are accessed via mappings from an ontology (which captures the domain vocabulary and domain knowledge).
- We study the problem of query optimization in OBDF, by devising a set of optimization techniques specifically tailored towards federated data sources.
- We carry out an experimental evaluation over an adaptation of the Berlin SPARQL Benchmark (BSBM) [5] to the federation setting, in which we assess the effectiveness of the proposed optimization techniques.

2 PRELIMINARIES

We introduce technical preliminaries and notation that we will adopt throughout the remainder of this paper.

Relational Algebra. We assume the reader to be familiar with fundamental notions of relational algebra. As conventions, we use Σ to denote a (relational) *DB schema*, D to denote an *instance of a DB schema*, and $\mathsf{sig}(\mathsf{A})$ to denote the *signature* of a relational algebra expression A, which consists of the tuple $(a_1, \ldots, a_n)$ of *attributes* of the relation generated by A. When we want to make the signature explicit, we use the notation $\mathsf{A}(a_1, \ldots, a_n)$. We introduce the abbreviation $\pi_{r_1/a_1,\ldots,r_k/a_k}$ for the combination $\rho_{r_1/a_1,\ldots,r_k/a_k}\pi_{a_1,\ldots,a_k}$ of *projection* and *renaming*, $\emptyset_R$ to denote the *empty relation* of signature $\mathsf{sig}(R)$, and Ω_R to denote a *tuple* of null values with signature $\mathsf{sig}(R)$. Given a database instance D of Σ and an algebra expression A, $\mathsf{ans}(\mathsf{A}, D)$ denotes *the set of answers of* A over D. Given two expressions A and B over Σ, A *is contained in* B, denoted as $\mathsf{A} \subseteq \mathsf{B}$, if $\mathsf{ans}(\mathsf{A}, D) \subseteq \mathsf{ans}(\mathsf{B}, D)$ for every DB instance D of Σ. A and B are *equivalent*, denoted as $\mathsf{A} \equiv \mathsf{B}$, if $\mathsf{A} \subseteq \mathsf{B}$ and $\mathsf{B} \subseteq \mathsf{A}$.

Ontology-based Data Access (OBDA). We rely here on the classic framework from [42]. Due to space limitations, we assume the reader to be familiar with *ontologies* and *Description Logics* notation, and refer to the extensive literature on the subject [4].

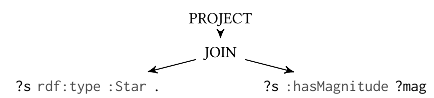
An *OBDA specification* $\mathcal{O}$ is a triple $(\mathcal{T}, \mathcal{M}, \Sigma)$, where $\mathcal{T}$ is an *ontology* consisting of *class inclusion axioms* $B \sqsubseteq C$ and *role inclusion axioms* $S \sqsubseteq T$; Σ is a relational DB schema; and $\mathcal{M}$ is a set of *OBDA-mappings* of the form $\mathsf{A} \rightsquigarrow C(\mathsf{f}(\mathbf{a}))$ or $\mathsf{A} \rightsquigarrow P(\mathsf{f}(\mathbf{a}), \mathsf{g}(\mathbf{b}))$, where A is a relational algebra expression over Σ, C and P are respectively a class and a property name of $\mathcal{T}$, and $\mathsf{f}(\mathbf{a})$ and $\mathsf{g}(\mathbf{b})$ are *(R2RML) IRI templates* [14], specifying how DB values are transformed into IRIs and RDF literals, respectively making use of the sets $\mathbf{a}$ and $\mathbf{b}$ of attributes in $\mathsf{sig}(\mathsf{A})$. We call A the *source part* and $C(\mathsf{f}(\mathbf{a})$ (resp., $P(\mathsf{f}(\mathbf{a}), \mathsf{g}(\mathbf{b}))$) the *target part* of the mapping. An *OBDA instance* is a pair $(\mathcal{O}, D)$, where D is a DB instance of Σ.

For the semantics of an OBDA instance, we refer to [33]. Intuitively, an OBDA instance exposes a (virtual) *RDF graph* that can be queried through SPARQL [22]. The graph is virtual in the sense that RDF triples are not materialized. Instead, to answer a SPARQL query, the query is translated on-the-fly into an equivalent SQL query over the database, called its *translation*.

Unfolding. In OBDA, the process of producing a SQL translation q for a SPARQL query Q over an OBDA specification $(\mathcal{T}, \mathcal{M}, \Sigma)$ is called *unfolding* [33]. Different unfolding procedures have been proposed in the literature. For this work, we focus on two variants: the *classical one* aiming at producing a *union of conjunctive queries* (UCQ) [33], and the one aiming at producing a *join of union of conjunctive queries* (JUCQ) [7, 27]. One can switch from one variant to another by applying algebra transformations, specifically by pushing the joins at the bottom of the algebra-tree (UCQ) or the unions (JUCQ) through the *distributive rule*. We describe here the JUCQ variant, which we call $\mathsf{unfold}_{\mathsf{wrap}}$, by means of an example. As we will see in Section 5, $\mathsf{unfold}_{\mathsf{wrap}}$ provides the basis for the unfolding algorithm in our federated setting. Consider the following SPARQL query, asking for information about stars:

```
SELECT ?s ?mag WHERE { ?s rdf:type :Star; :hasMagnitude ?mag . }
```

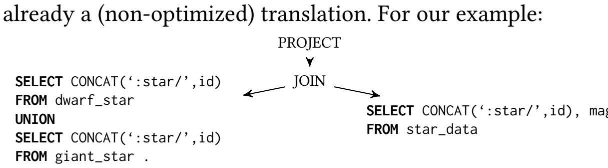
The query above corresponds to the following SPARQL algebra tree:



Assume the following mappings over a DB, providing SQL definitions for the class `:Star` and the data property `:hasMagnitude`.

```
SELECT id FROM dwarf_star    ⇝ :star/{id} rdf:type :Star .
SELECT id FROM giant_star    ⇝ :star/{id} rdf:type :Star .
SELECT id,mag FROM star_data ⇝ :star/{id} :hasMagnitude {mag} .
```

The $\mathsf{unfold}_{\mathsf{wrap}}$ algorithm traverses the algebra tree in a bottom-up fashion. It starts by replacing each leaf of the tree, that is, a triple pattern of the form (s, p, o), with the union of the *corresponding SQL definitions in the mappings.* In this step, the algorithm includes in the SQL query a function (usually implemented as string concatenation) which constructs the IRIs for s and o as specified by the IRI templates in the mappings. Once it finishes processing the leaves, the algorithm continues to the upper levels in the tree, where the SPARQL operators (PROJECT, JOIN, OPTIONAL, UNION, and FILTER) are translated into the corresponding SQL operators (Project, InnerJoin, LeftJoin, Union, and Filter, respectively). Once the root is translated, we have obtained an intermediate SQL query that is already a (non-optimized) translation. For our example:



At this point, a *crucial* optimization takes place, which leads to the creation of a proper JUCQ unfolding: the applications of the IRI templates (i.e., the `CONCAT` operations) are pushed as high as possible in the algebra tree. This allows the database to perform joins over (possibly, indexed) database values, rather than over the results of string concatenation. Without delving into technical details, during this phase the algebra tree might undergo some structural transformations: if a JOIN node is found in which the join condition equates two IRIs that are necessarily different (being the result of CONCAT operations that lead to different results, e.g., because of different prefixes), then that join is trivially empty and gets eliminated from the algebra tree.

In our example, at the end of the unfolding procedure $\mathsf{unfold}_{\mathsf{wrap}}$ we obtain the following query (expressed in relational algebra):

$\pi_{\mathsf{f}_{:\mathsf{star}}(id),mag}(\pi_{id}(\mathsf{dwarf\_star} \cup \mathsf{giant\_star}) \bowtie_{id=id1} \pi_{id1/id,mag}(\mathsf{star\_data}))$

where $\mathsf{f}_{:\mathsf{star}}(id)$ denotes the application of the IRI template `CONCAT(':star/',id)`.

Data Federation. Federating multiple, possibly heterogeneous data sources consists in exposing a unified view of such sources, usually called *virtual database* (VDB). In this paper, a data source, denoted by S, can be an RDB, a NoSQL DB, or of some other type. Consider a set $\mathbb{S} = \{S_1, \ldots, S_n\}$ of sources to be federated, and a function (given implicitly with $\mathbb{S}$) transforming the (possibly, non-relational) schema of each source S_i into a corresponding relational schema Σ_i. Then, the *federated VDB schema* (for $\mathbb{S}$) is the disjoint union $\Sigma_{\mathbb{S}} = \bigcup_{i=1}^{n} \Sigma_i$. In the following, we use letters T, U to denote database tables, and the subscript i (e.g., T_i) to indicate that S_i is the source of table T. Additionally, given an arbitrary relational algebra expression A, src(A) denotes the set of sources of the atoms in A, and $\mathsf{occ}(S_i, \mathsf{A})$ denotes the total number of occurrences in A of relations from S_i. A data federation instance $\mathbb{D}$ for $\Sigma_{\mathbb{S}}$ is the relational instance $\bigcup_i D_i$ made of the union of all instances of the (relational) source schemas in $\Sigma_{\mathbb{S}}$. Hence, given a query q, $\mathsf{ans}(q, \mathbb{D})$ denotes the set of answers of q evaluated over the federation instance $\mathbb{D}$.

Data Federation vs Data Integration. Real-world data federation systems often provide *data integration* capabilities, allowing users to specify an arbitrary VDB schema *integrating* the schemas of the various sources. The link between such a VDB schema and the schemas of the sources is realized through GAV, LAV, or GLAV mappings [15]. In this work, we always assume a federated VDB schema. The reason is that, in our setting, the integration is performed at the level of the ontology, by exploiting the definitions provided in the OBDA mappings (that can be interpreted, in fact, as GAV mappings coming from the context of data integration).

Local Operations vs Federated Operations. To compute the answers for a federated query, a data federation system can delegate operations (e.g., joins and unions) to the data sources, or perform the operations itself. In this paper, we distinguish between *local operations* (e.g., joins performed within a data source) and *federated operations* (e.g., joins across multiple sources, that have to be handled at the level of the federation system).

3 ONTOLOGY-BASED DATA FEDERATION

Our first contribution is the definition of a general framework for enriching OBDA with data federation capabilities.

Definition 3.1 (OBDF). Given an ontology $\mathcal{T}$, a federated VDB schema $\Sigma_{\mathbb{S}}$, and a set $\mathcal{M}$ of mappings from $\Sigma_{\mathbb{S}}$ to $\mathcal{T}$, an *ontology-based data federation (OBDF) specification* is the OBDA specification $\mathcal{F} = (\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}})$.

Hence, the notions of *OBDF instance* and *answers to a query over an OBDF instance* coincide with their OBDA counterpart. Figure 1 depicts the full process of query answering in an OBDF scenario. A federation engine (e.g., Teiid, Denodo, or Dremio) is responsible

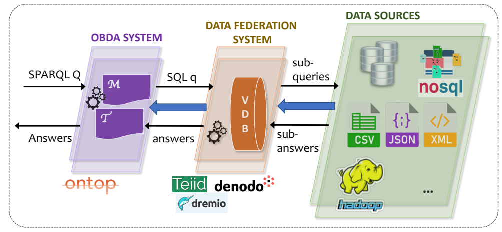


Figure 1: OBDF framework and query answering procedure.

for the federation of the data sources, and an OBDA system, in this case Ontop, interacts with the federation engine *as it would normally do with a single relational database.*

Opportunities and Challenges. In line with the FAIR principles[2], OBDA allows users to publish data according while complying to shared, agreed-upon vocabularies which enable interoperability between different applications. Furthermore, the ontology constitutes both a documentation about the data and a basis for enabling reasoning-based services, such as query answering w.r.t. the ontology. The added value of data federation is to extend the OBDA paradigm to multiple, possibly non-relational sources.

While benefiting from both OBDA and data federation, OBDF combines the challenges of both. Next example shows possible issues that might arise from a naive implementation of OBDF.

Example 3.2. Consider an enterprise, whose data is spread across different sources $\mathbb{S} = \{S_1, \ldots, S_4\}$ that need to be integrated. Consider an OBDF specification $\mathcal{F} = (\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}})$, with $\mathcal{T}$ and $\mathcal{M}$ as in Figure 2, where each relation in $\mathcal{M}$ has a subscript i denoting the source S_i relative to that relation. For the SPARQL query Q in Figure 2, asking for products' and inspectors' information, the unfolding procedure would produce the SQL query q from the same figure. Observe that this query is already verbose, with 3 federated joins and 4 federated unions across the different sources. At this point, state-of-the-art OBDA systems typically apply structural optimizations transforming the JUCQ q into a UCQ, by pushing the join operators at the bottom level of the algebra tree. After this transformation, the reader can easily verify that the query obtained would consists of $2^4 = 16$ unions of CQs, where each CQ has 3 join operators, thus amounting to 48 joins in total. Hence, transforming JUCQs into UCQs blindly can largely increase the number of federated, thus inefficient, operations.

To complicate the picture, it is often the case that certain relations hold across the different sources: for instance, relation PerInfo might contain the names of all the employees in the enterprise, rendering the last union in q redundant. Similarly, the intersection of ConvenienceGoods and ShoppingGoods might be empty, rendering all joins between these two relations empty as well.

In the remainder of this work we discuss a novel unfolding procedure specific to the OBDF setting, able to choose the best strategy between UCQ and JUCQ unfoldings and to exploit relations holding across different data sources.

[2]https://www.go-fair.org/fair-principles/

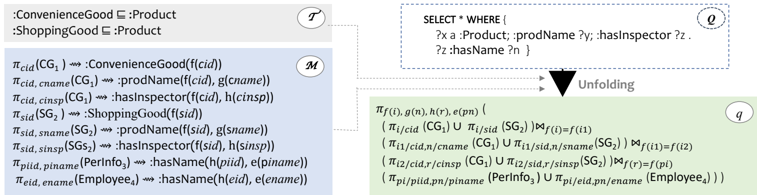


Figure 2: An example of unfolding SPARQL query into SQL query w.r.t. the ontology $\mathcal{T}$ and mapping set $\mathcal{M}$.

4 DATA HINTS

Under the standard formalization of OBDA [33], in which every IRI template behaves like an *injective* R2RML template, the assumption that the ontology language used is OWL 2 QL, and that query answering is performed under the SPARQL 1.1 entailment regime, it is possible to determine *a-priori* all the joins between relations that can occur in the SQL translation of a user query [27]. This can be done by an offline analysis of the OBDA specification, that is, by collecting pairs of atoms with compatible templates. In [27], such intuition is used to collect specific statistics about the OBDA instance, with the goal of improving the performance of query answering. We adopt here a similar approach, by introducing different kinds of meta-information, called *data hints* (or, simply, *hints*), that we use to optimize query answering in OBDF.

We identify three kinds of hints: *empty federated joins*, *containment redundancy*, and *materialized views*. The first kind of hint, *empty federated join*, annotates which joins are expected to be empty when evaluated over the current data federation instance. For the definitions in this section, we assume a fixed federated VDB schema $\Sigma_{\mathbb{S}}$.

Definition 4.1 (Hint 1: Empty Federated Join). Given an instance $\mathbb{D}$ of $\Sigma_{\mathbb{S}}$ and a federated join expression FJ over $\Sigma_{\mathbb{S}}$, we say that FJ is an *empty federated join w.r.t.* $\mathbb{D}$, denoted as $\mathsf{FJ} =_{\mathbb{D}} \emptyset$, if $\mathsf{ans}(\mathsf{FJ}, \mathbb{D}) = \emptyset$.

The second kind of hint, *containment redundancy*, annotates the presence of redundancy (typically across different data sources).

Definition 4.2 (Hint 2: Containment Redundancy). Given an instance $\mathbb{D}$ of $\Sigma_{\mathbb{S}}$ and two expressions A and B over $\Sigma_{\mathbb{S}}$, we say that A is *data-contained* in B, denoted as $\mathsf{A} \subseteq_{\mathbb{D}} \mathsf{B}$, if $\mathsf{ans}(\mathsf{A}, \mathbb{D}) \subseteq \mathsf{ans}(\mathsf{B}, \mathbb{D})$. We use $\mathsf{A} \equiv_{\mathbb{D}} \mathsf{B}$ to indicate that $\mathsf{A} \subseteq_{\mathbb{D}} \mathsf{B}$ and $\mathsf{B} \subseteq_{\mathbb{D}} \mathsf{A}$.

Materialized views [1, 21] can improve the overall performance of query answering. The ability to specify materialized views is provided by a few data federation systems, such as Teiid or Dremio. In our formalization, we assume the presence of an extra source to store the materialization of the views, where such source could be the federation system itself. This is motivated by the fact that it is often impossible or impractical to store the views directly in the sources, due to access policies, source ownership, etc.

Definition 4.3 (Hint 3: VDB Schema with Views). Let M be a set of view definitions. We denote by $\Sigma_{\mathbb{S}}^{\mathsf{M}}$ the VDB schema $\Sigma_{\mathbb{S}} \cup \Sigma_{\mathsf{M}}$, where Σ_{M} is the relational schema of a special data source S^{M} materializing the views defined in M.

Consequently, an instance $\mathbb{D}^{\mathsf{M}}$ of $\Sigma_{\mathbb{S}}^{\mathsf{M}}$ is a VDB instance $\mathbb{D} \cup D_{\mathsf{M}}$ such that $\mathbb{D}$ is an instance of $\Sigma_{\mathbb{S}}$ and D_{M} is an instance of Σ_{M} conforming to the view definitions in M.

Finally, we assume two labeling functions: the first one characterizes whether a source is *efficient* or *inefficient* when answering queries, whereas the second one characterizes whether the data source is *dynamic* (i.e., its content is expected to change frequently), or *static* (i.e., its content is not expected to change).

5 QUERY OPTIMIZATION IN OBDF

We now discuss our solution to optimize SQL translations of SPARQL queries posed over an OBDF system. The main intuition is that, in OBDF, the ontology and mappings contain information to guide the discovery of the data hints discussed in the previous section. The overall method consists of two parts: *1)* an offline *hints pre-computation part*, and *2)* an on-line *translation optimization part*. For the remainder of this section, we assume a fixed OBDF specification $\mathcal{F} = (\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}})$.

5.1 Pre-Computation of Hints

We exploit the mappings and the ontology in an OBDF specification to guide the gathering of hints. Basically, we will enumerate in advance all possible SQL joins and unions between pairs of relations that the system can possibly produce during the unfolding.

Analyzing the mappings. We start by showing how the analysis is carried out for joins. Trivially, join conditions appearing in the source part of mapping assertions give an indication on what columns of the DB can be joined. Other joins are the result of the translation of a SPARQL join into a SQL join. As proposed in [27, 28], in order to determine these we can analyze the IRI templates appearing in the mapping, and pre-compute all pairs of *compatible templates* (i.e., templates that generate the same IRI for some DB instance). For instance, consider the following mapping assertions:

$$\mathsf{m}_1 : \mathsf{T}_1(a) \rightsquigarrow A(\mathsf{f}(a)), \quad \mathsf{m}_2 : \mathsf{T}_2(b,c) \rightsquigarrow P(\mathsf{f}(b), \mathsf{g}(c))$$
$$\mathsf{m}_3 : \mathsf{T}_3(d,e) \rightsquigarrow P(\mathsf{h}(d), \mathsf{i}(e))$$

and the SPARQL query `SELECT ?x ?y WHERE {?x a A; P ?y}`, retrieving A-individuals and their P-successors. During $\mathsf{unfold}_{\mathsf{wrap}}$ execution, we reach the following intermediate translation:

$$(\pi_{\mathsf{f}(a),\mathsf{g}(c)}(\mathsf{T}_1(a) \bowtie_{\mathsf{f}(a)=\mathsf{f}(b)} \mathsf{T}_2(b,c))) \cup (\pi_{\mathsf{f}(a),\mathsf{i}(e)}(\mathsf{T}_1(a) \bowtie_{\mathsf{f}(a)=\mathsf{h}(d)} \mathsf{T}_3(d,e)))$$

Note that the second join expression can only be empty, since the join condition $\mathsf{f}(a) = \mathsf{h}(d)$ can never be satisfied for any instantiation of a and d. Hence, $\mathsf{unfold}_{\mathsf{wrap}}$ will remove this empty join, simplify the join between IRIs with the same template into a join between the underlying database attributes, and finally return:

$$\pi_{\mathsf{f}(a),\mathsf{g}(c)}(\mathsf{T}_1(a) \bowtie_{a=b} \mathsf{T}_2(b,c))$$

A similar analysis can be carried out for unions. For instance, consider the following additional mapping:

$$\mathsf{m}_4 : \pi_b(\mathsf{T}_2(b,c)) \rightsquigarrow A(\mathsf{f}(b))$$

Algorithm 1: $\mathsf{hintify}(((\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}}), \mathbb{D}), \mathsf{Joins}, \mathsf{Unions})$

Input: An OBDF instance $((\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}}), \mathbb{D})$, set Joins of joins, set Unions of unions.
Output: An OBDF instance $((\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}}^{\mathsf{M}}), \mathbb{D}^{\mathsf{M}})$ with materialized views M, a set $\mathcal{E}$ of empty join hints, a set C of containment redundancy hints.

foreach j *in* Joins **do**
  if $j =_{\mathbb{D}} \emptyset$ **then**
    $\mathcal{E} \hookleftarrow \mathcal{E} \cup \{j =_{\mathbb{D}} \emptyset\}$
  else if j *is a federated join* **or** *it is over an inefficient source* **then**
    if s *is static for each* $s \in \mathsf{src}(j)$ **then**
      $\mathsf{M} \hookleftarrow \mathsf{M} \cup \{\mathsf{V}(\ldots) \leftarrow j\}$
foreach $\mathsf{A} \cup \mathsf{B}$ *in* Unions **do**
  if $\mathsf{A} \subseteq_{\mathbb{D}} \mathsf{B}$ **then**
    $C \hookleftarrow C \cup \{\mathsf{A} \subseteq_{\mathbb{D}} \mathsf{B}\}$
  if $\mathsf{B} \subseteq_{\mathbb{D}} \mathsf{A}$ **then**
    $C \hookleftarrow C \cup \{\mathsf{B} \subseteq_{\mathbb{D}} \mathsf{A}\}$
return $((\mathcal{T}, \mathcal{M}, \Sigma_{\mathbb{S}}^{\mathsf{M}}), \mathbb{D}^{\mathsf{M}}), \mathcal{E}, C$

Now, the unfolding for the same query becomes:

$$\pi_{\mathsf{f}(a),\mathsf{g}(c)}((\mathsf{T}_1(a) \cup \pi_{b/a}(\mathsf{T}_2(b,c))) \bowtie_{a=b} \mathsf{T}_2(b,c))$$

Again, the union is between SQL columns (and not over the result of applying the templates to DB values) only because the templates are compatible. The same considerations apply to the case where the SPARQL query itself contains a union operation (e.g., a query retrieving all individuals in A union all those having a P-successor).

The role of the ontology. Ontology axioms increase the number of unions. Consider again the mappings $\mathsf{m}_1, \mathsf{m}_2$ above, and an ontology axiom $\exists P \sqsubseteq A$, stating that the domain of property P is class A. Then, our SPARQL query will unfold into the last SQL query from the previous paragraph. In fact, it is well-known [35] that answering SPARQL[3] queries over an OWL 2 QL ontology and a set $\mathcal{M}$ of mappings can be reduced to the problem of answering SPARQL queries over an empty ontology and an enriched set of mappings, called T-mapping, obtained by *compiling* the ontology into $\mathcal{M}$. In our example, it turns out that the set $\{m_1, m_2, m_4\}$ of mappings is a T-mapping compiling the axiom $\exists P \sqsubseteq A$ into $\{m_1, m_2\}$.

Constructing the hints. Summing up, under the assumption of using $\mathsf{unfold}_{\mathsf{wrap}}$ for the translation, we derive the following observations: *(1)* the enumeration of possible joins between DB columns can be carried out by an offline mappings analysis identifying compatible templates; *(2)* if the ontology is empty, the only possible unions between database columns are those induced by mappings with compatible templates; *(3)* the case of a non-empty ontology can always be reduced to the case of an empty ontology by exploiting the T-mapping technique. From *(1)–(3)*, we conclude that it is possible to enumerate all possible SQL joins and unions by analyzing the T-mapping of an OBDF specification. These are used by Algorithm 1 to compute data hints for an OBDF instance. The number of hints is bound to n^2, where n is the number of attributes in the DB schema. The worst case captures the scenario where every DB attribute is mapped to the same URI template. Algorithm 1 needs to be re-ran every time the sources get updated, which makes our approach less suited to scenarios where sources are, e.g., streams of data.

[3]Assuming the SPARQL 1.1 Entailment Regime and OWL 2 QL.

5.2 Query Optimization Rules

We introduce a set of query optimization rules based on pre-computed hints. The top part of Figure 3 shows well-known rules commonly applied in OBDA systems [10, 35], and which are relevant also in our approach. The bottom part introduces the novel rules based on hints, which we now explain in detail.

The *empty join elimination rule* **ejr** removes empty joins from SQL queries. To check the applicability of this rule, we rely on the observation that, if a join between two relations appears in the unfolding, then we have pre-computed it during the computation of hints. Therefore, given a set $\mathcal{E}$ of empty join hints, computed as in Algorithm 1, to verify $\mathsf{A} \bowtie \mathsf{B} =_{\mathbb{D}} \emptyset$ it suffices to check whether this expression belongs to $\mathcal{E}$.

The *containment elimination rule* **cr** removes redundant unions from SQL queries. As discussed for **ejr**, the applicability of this rule can be verified through a membership check over the set C of containment redundancies computed by Algorithm 1.

The *equivalence elimination rule* **er** replaces the operands of joins or left outer joins with expressions. The applicability of the rule involves checking a data containment (as for rule **cr**), and a condition † that is verified only if the *cost* of the resulting expression is less than the cost of the original one. In Section 5.3 we discuss a way to compute such cost.

Finally, the *materialization rule* **mtr** replaces the federated joins in the SQL query according to the views computed by Algorithm 1.

5.3 Cost Model

The application of the rules introduced in Section 5.2 is guided by a *cost model*, which associates an *evaluation cost* to each SQL query. Our cost model is based on the following heuristic arguments:

(1) Local joins are preferred to federated joins.
(2) Efficient sources should be favored over inefficient ones.
(3) Redundant and empty sub-expressions should be eliminated.
(4) Whenever available, materialized views are preferred.

Besides such heuristics specific to the federated setting, we also assume the standard heuristics for the OBDA setting [35]:

(5) URI templates should be applied at the highest level possible of the algebra tree.
(6) Joins should be pushed inside unions (see rule **dlr**).
(7) Self-joins and redundant unions should be eliminated (see, e.g., rule **sjr**).

We now introduce our cost measure, inspired by these heuristic assumptions. For the remainder of this section, we fix a VDB instance $\mathbb{D}$ for $\Sigma_{\mathbb{S}}$.

A *terminal federated join* is a query of the form $\mathsf{FJ} = (\bigcup_{i=1}^{n} \mathsf{A}_i) \circ (\bigcup_{j=1}^{m} \mathsf{B}_j)$, where $\circ \in \{\bowtie, ⟕\}$, and each A_i and B_j is an expression over $\Sigma_{\mathbb{S}}$ not containing joins. The *cost of evaluating* FJ over $\mathbb{D}$ is estimated as: $\mathsf{Cost}_{\mathsf{base}}(\mathsf{FJ}) = n + m$. In compliance with heuristic (1) above, this measure favors the structure where unions are folded into a single federated join, as opposed to the UCQ structure (which would contain $n * m$, possibly federated, joins). In compliance with (3) and (7), removing redundant operations from either operand will decrease the cost of the expression.

Consider a SQL query q over $\Sigma_{\mathbb{S}}$ containing k terminal federated joins $\mathsf{FJ}_1, \ldots, \mathsf{FJ}_k$, where $\mathsf{Cost}_{\mathsf{base}}(\mathsf{FJ}_i) = c_i$, for $1 \leq i \leq k$, and

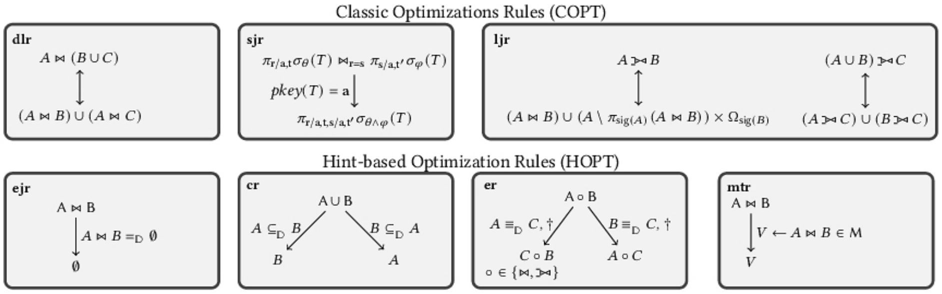


Figure 3: List of "standard" optimization rules applied by OBDA systems (first row), and the new rules specific to OBDF. The former is not complete (e.g., we omit trivial transformations (e.g., commutativity) and the sjr variant for left-joins).

let $\#\mathsf{ineff} = \Sigma_{S \in \{S \in \mathrm{src}(q) \mid S \text{ is inefficient}\}} \mathrm{occ}(S, q)$ denote the total number of occurrences in q of relations over inefficient sources. Then the *cost of evaluating* q *over* $\mathbb{D}$ is estimated as:

$$\mathsf{Cost}(q) = (\Sigma_{i=1}^{k} c_i, \#\mathsf{ineff})$$

Finally, we define a partial order for comparing costs.

Definition 5.1 (Cost comparison). Consider two queries q_1, q_2 such that $\mathsf{Cost}(q_1) = (n_1, n_2)$ and $\mathsf{Cost}(q_2) = (m_1, m_2)$. The *cost comparison relations* $\preceq$ and $\prec$ are defined as:

- $\mathsf{Cost}(q_1) \preceq \mathsf{Cost}(q_2)$, if $n_i \leq m_i$ for each $i \in \{1, 2\}$;
- $\mathsf{Cost}(q_1) \prec \mathsf{Cost}(q_2)$ if $\mathsf{Cost}(q_1) \preceq \mathsf{Cost}(q_2)$, and $n_i < m_i$ for some $i \in \{1, 2\}$.

Note that our cost model is compliant with heuristics (1)-(7). For instance, if a structural transformation increases the number of joins in an expression, but these joins are local, then the cost of such expression will still decrease (see heuristics (5) and (6)). Also, whenever a materialized view is used, an inefficient or federated join is removed from the expression, reducing either argument of the cost value (heuristic (4)).

5.4 Hints-Based Unfolding Algorithm

We present a novel algorithm, $\mathsf{unfold}_{\mathsf{OBDF}}$, for translating federated SPARQL queries over an OBDF system into SQL queries over a VDF system. The algorithm relies on pre-computed hints, and applies the optimization rules discussed in Section 5.2, guided by the cost-model described in Section 5.3.

Algorithm 2 starts by computing a translation of a SPARQL query Q w.r.t. $(\mathcal{T}, \mathcal{M}, \mathbb{\Sigma}_{\mathbb{S}})$ (Line 1). For the translation it adopts $\mathsf{unfold}_{\mathsf{wrap}}$, discussed in Section 2, because such an algorithm *maximizes* the number of unions between source relations.

By hint construction, each redundant union must occur in C. Therefore, identifying redundant unions reduces to a set membership check (Lines 3 and 8).

Line 4 uses the cost-model to determine the specific application branch of rule **cr**, in case the containment holds in both directions.

After having removed as many redundant unions as possible, the algorithm attempts to remove joins. To this aim, a UCQ-like form is more suited, since it maximizes the number of join operations between source relations. To perform this structural transformation, the algorithm iteratively applies rule **dlr** (Line 11).

Algorithm 2: $\mathsf{unfold}_{\mathsf{OBDF}}(Q, ((\mathcal{T}, \mathcal{M}, \mathbb{\Sigma}_{\mathbb{S}}^{\mathsf{M}}), \mathbb{D}^{\mathsf{M}}), \mathcal{E}, C)$

Input: A SPARQL query Q over the OBDF instance $((\mathcal{T}, \mathcal{M}, \mathbb{\Sigma}_{\mathbb{S}}^{\mathsf{M}}), \mathbb{D}^{\mathsf{M}})$ with materialized views M, a set $\mathcal{E}$ of empty join hints, a set C of containment redundancy hints;
Output: A translation q' of Q w.r.t. $(\mathcal{T}, \mathcal{M})$ and the view definitions M.

1 $q \hookleftarrow \mathsf{unfold}_{\mathsf{wrap}}(Q)$;
2 **foreach** $\bigcup_{i=1}^{n} \mathsf{A}_i$ *in* q *s.t. there exists data redundancy btw* A_k *and* A_ℓ **do**
  /* Apply exahustively rule **cr**. */
3   **if** $\mathsf{A}_k \equiv_{\mathbb{D}} \mathsf{A}_l \in C$ **then**
4   **if** $\mathsf{Cost}(q[\bigcup_{i=1}^{n} \mathsf{A}_i / \bigcup_{i=1, i \neq k}^{n} \mathsf{A}_i]) \prec \mathsf{Cost}(q[\bigcup_{i=1}^{n} \mathsf{A}_i / \bigcup_{i=1, i \neq l}^{n} \mathsf{A}_i])$ **then**
5       $q \hookleftarrow q[\bigcup_{i=1}^{n} \mathsf{A}_i / \bigcup_{i=1, i \neq k}^{n} \mathsf{A}_i]$;
6   **else**
7       $q \hookleftarrow q[\bigcup_{i=1}^{n} \mathsf{A}_i / \bigcup_{i=1, i \neq l}^{n} \mathsf{A}_i]$;
8   **else if** $\mathsf{A}_k \subseteq_{\mathbb{D}} \mathsf{A}_\ell \in C$ **then**
9     $q \hookleftarrow q[\bigcup_{i=1}^{n} \mathsf{A}_i / \bigcup_{i=1, i \neq k}^{n} \mathsf{A}_i]$;
10 **foreach** *terminal federated join* $\mathsf{FJ} : (\bigcup_{i=1}^{n} \mathsf{A}_i) \bowtie_\theta (\bigcup_{j=1}^{m} \mathsf{B}_j)$ *in* q *where* $n, m \geq 1$ **do**
  /* Push joins into unions by applying rules **dlr** */
11   $\mathsf{E} \hookleftarrow \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} (\mathsf{A}_i \bowtie_\theta \mathsf{B}_j)$;
  /* Apply rules for join optimizations */
12   $\mathsf{E}' \hookleftarrow \mathsf{applyExaustively}(\mathsf{E}, \{\mathbf{ejr}, \mathbf{er}, \mathbf{mtr}, \mathbf{sjr}\})$;
  /* Fold unions of joins into joins of unions */
13   **while** *there exists appplication:* $\mathsf{E}' \rightarrow_{\mathbf{dlr}^-} \mathsf{E}''$ ***and*** $\mathsf{Cost}(\mathsf{E}'') \prec \mathsf{Cost}(\mathsf{E}')$ **do**
14     $\mathsf{E}' \hookleftarrow \mathsf{E}''$
  /* Accept the change only if there is improvement. */
15   **if** $\mathsf{Cost}(q[\mathsf{FJ}/\mathsf{E}']) \prec \mathsf{Cost}(q)$ **then**
16     $q \hookleftarrow q[\mathsf{FJ}/\mathsf{E}']$;
17 **foreach** *terminal federated join* $\mathsf{FJ} : (\bigcup_{i=1}^{n} \mathsf{A}_i) ⟕_\theta (\bigcup_{j=1}^{n} \mathsf{B}_j)$ *in* q *where* $n, m \geq 1$ **do**
18   Proceed similarly as for inner joins;
19 **return** q;

Sub-procedure applyExaustively (Line 12) removes redundant self-joins, and transforms federated joins or joins between inefficient sources either into local joins through the application of rule **er**, or into queries over the materialized views through the application of rule **mtr**. Observe that procedure applyExaustively is terminating: each application of **ejr**, **sjr**, and **mtr** reduces the size of E, and each application of **er** reduces $\mathsf{Cost}(\mathsf{E})$. Again, to decide the applicability of each of these rules it suffices to solve a membership problem in $\mathcal{E}$, C, or M.

Note that the application of rule **dlr** in Line 11 potentially introduces a *large* number of joins, many of which might even be federated or over inefficient sources. Hence, even after the optimizations applied in Line 12, the algorithm tries to fold-back the structure of the query (Lines 13 and 14) through an inverse application of **dlr**.

After these steps, if the optimization of the federated join led to an improvement of the overall cost, then the changes are accepted and incorporated in q (Lines 15 and 16).

After a similar strategy for left outer joins is applied, the algorithm returns the final translation, whose cost is lower than or equal to the cost of the initial unfolding.

Example 5.2. Consider again the OBDF specification and SPARQL query from Example 3.2, and an OBDF instance $(\mathcal{F}, \mathbb{D})$ of $\mathcal{F}$. Suppose all "id" columns to be primary keys for the respective tables. Suppose we have the empty federated join hint $\mathsf{CG}_1 \bowtie_{cid=sid} \mathsf{SG}_2 =_{\mathbb{D}} \emptyset$ and the containment redundancy hint $\pi_{pid/piid,pname/piname}(\mathsf{PerInfo}_3) \equiv_{\mathbb{D}} \pi_{pid/eid,pname/ename}(\mathsf{Employee}_4)$, and that S_1, S_2, and S_3 have been labelled as *efficient*, while S_4 as *inefficient*. Then, Figure 4 illustrates how the above hints and labels are exploited in order to further unfold the SQL translation q of Q. The translation goes as follows, where we assume that the operators in the query expression are processed in order from left to right:

(1) The federated join $\bowtie_{i=i1}$ is first unfolded into a union of 4 joins and then traslated into $\pi_{i/cid,i1/cid,n/cname}(\mathsf{CG}_1) \cup \pi_{i/sid,i1/sid,n/sname}(\mathsf{SG}_2)$ on the basis of the empty join hint and the application of the rule **sjr**.
(2) Similarly, the intermediate q_1 is translated into query q_2.
(3) On the basis of containment redundancy hint and the given source labelling, the union between $\mathsf{PerInfo}_3$ and $\mathsf{Employee}_4$ is then removed by the application of rule **cr**, and only the projection over the fastest source $\mathsf{PerInfo}_3$ is kept in the resulting query q_3.

Each unfold step reduces the query's cost, and $\mathrm{Cost}(q_3) \prec \mathrm{Cost}(q)$.

6 EVALUATION

We have carried out an extensive experiment to verify the effectiveness of the proposed optimizations. We ran the experiments on an HP Proliant server with 2 Intel Xeon X5690 Processors (each with 12 logical cores at 3.47 GHz), 106GB of RAM and five 1TB 15K RPM HDs. For database systems, we have employed PostgreSQL 14, DB2 11.5.7.0, MySQL 8.0.28 and MS SQL Server 2019 as relational systems, MongoDB 4.4.13 as the NoSQL system, and Teiid 16.0.0 as the federation engine.

The material for reproducing the experiments and the appendix of this work are available at: https://github.com/efghk321456/sc.

6.1 Experiment Setup

Our experiment is based on the well-known Berlin SPARQL Benchmark (BSBM) [5]. BSBM is built around an e-commerce use case in which a set of products is offered by different vendors and consumers have posted reviews about products. The data is organized as ten relational tables.

Data sets. We generate 5 data sets out of the original BSBM tables, while introducing data partitioning and redundancy to simulate the scenarios where data from different sources are mapped to the same classes in the ontology. We split the tables Product, ProductFeatureProduct, and ProductTypeProduct "horizontally" according to their product IDs, into two data sets $\mathcal{D}_1$ and $\mathcal{D}_2$. We make a copy of the table Review and put it into $\mathcal{D}_1$. Dataset $\mathcal{D}_3$ contains ProductType and ProductFeature; $\mathcal{D}_4$ contains Offer, Producer, and Vendor; $\mathcal{D}_5$ contains Review and Person.

Data sources. We store the 5 data sets in different database systems and derive 8 data sources in total. $\mathcal{D}_1$–$\mathcal{D}_5$ are stored in RDBs and data sources S_1–S_5 are obtained. We convert the tables in $\mathcal{D}_2$ and $\mathcal{D}_4$ to CSV files to obtain two more data sources S_2' and S_4'. We convert $\mathcal{D}_5$ into JSON files, stored them in MongoDB, and obtain a data source S_5'.

OBDA/OBDF specifications. We use the ontology and mappings from [44] for OBDA, with minor modifications on the mappings for handling the different DBs. As base lines, we generate two OBDA specifications using two centralized RDBs: sc1, containing the original BSBM tables, and sc2, containing the tables in $S_1 - S_5$. We create two OBDF specifications over Teiid: a homogeneous (relational) one, Hom, defined over sources $S_1 - S_5$, and a heterogeneous one (in which some data are also in CSV files and MongoDB) het, defined over the sources $\{S_1, S_2', S_3, S_4', S_5'\}$. For each OBDF specification, the hints include 3 empty federated joins, 1 data redundancy, and 6 materialized views (see the appendix). The materialized views are stored in a local PostgreSQL DB.

6.2 Query Evaluation and Result Analysis

Table 1: Performance change compared with the sc1 setting

	sc1	sc2	hom	$\mathrm{hom}_{\mathrm{opt}}$	$\mathrm{hom}_{\mathrm{opt}}^{\mathrm{matv}}$	het	$\mathrm{het}_{\mathrm{opt}}$	$\mathrm{het}_{\mathrm{opt}}^{\mathrm{matv}}$
200K	1.0	8.4	8.5	0.6	0.4	254.9	54.6	12.2
2M	1.0	59.7	67.5	1.8	0.6	922.9	205.7	52.1

For scalability testing, we generate three groups of instances using the BSBM data generating tool, setting the numbers of products to be 20K, 200K and 2M. In this way each OBDA/OBDF specification has 3 instances. For each OBDF instance, the hints, including data redundancy, empty federated joins, and materialized views, are pre-computed following the approach in Section 5.1. For space reasons, we only report the results on 200K and 2M. Further details are listed in the Appendix.

We consider the 12 SPARQL queries Q_1 to Q_{12} from the BSBM benchmark. The SQL queries without optimization with hints are generated by Ontop, and the optimized ones are computed manually following the algorithms in Section 5. All queries were ran 12 times: 2 warm-up runs, and 10 test runs. In each run, the queries were instantiated with different parameters. The tests were ran by using the testing platform of the NPD benchmark [26].

The SQL queries evaluation times are reported in Figure 5. In such figure, $\mathrm{hom}_{\mathrm{opt}}$ and $\mathrm{het}_{\mathrm{opt}}$ denote the evaluation with the hints of empty federated joins and redundancies; and $\mathrm{hom}_{\mathrm{opt}}^{\mathrm{matv}}$ and $\mathrm{het}_{\mathrm{opt}}^{\mathrm{matv}}$ employed all the hints including materialized views.

To highlight the differences between the setups, we compute the *aggregated ratios* of running times compared with sc1, reported in Table 1. Concretely, for each setup c, the values are computed by $\left(\prod_{i=1}^{N} \frac{t_i^{\mathrm{c}}}{t_i^{\mathrm{sc1}}}\right)^{\frac{1}{N}}$, where N=12 is the number of queries, and t_i^{c} and t_i^{sc1} are the times of evaluating Q_i in setup c and sc1, respectively.

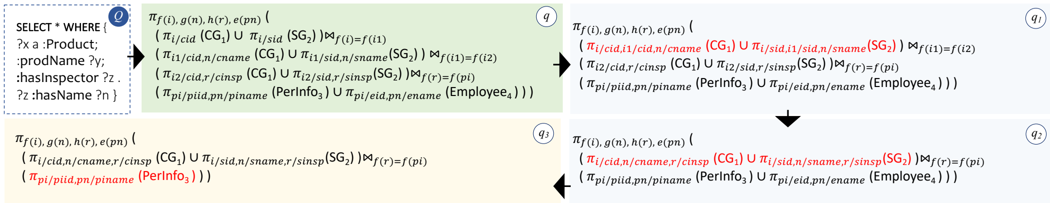


Figure 4: An example of translating a SPARQL query to SQL under hints.

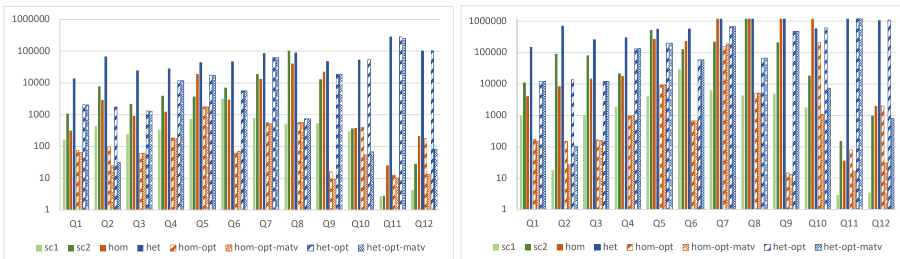


Figure 5: Results (time in ms) of SQL evaluation with factors 200K (left) and 2M (right).

Table 2: Usage of hints in federated queries

Query	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}	Q_{11}	Q_{12}
Hints	EFJ	EFJ, MatV	EFJ	EFJ	EFJ	EFJ, DR	DR, EFJ	DR	DR	MatV	−−	MatV
Before	S_1, S_2	S_1, S_2, S_4	S_1, S_2	S_1, S_2	S_1, S_2	$S_1 - S_5$	S_1, S_2, S_4, S_5	S_1, S_5	S_1, S_5	S_1, S_2, S_4	S_4	S_1, S_2, S_4
After	S_1, S_2	S^M	S_1, S_2	S_1, S_2	S_1, S_2	S_1, S_3, S_5	S_1, S_2, S_4, S_5	S_5	S_1, S_5	S^M, S_4	S_4	S^M, S_4

Table 2 reports the hints used for each query, and the data sources accessed before and after the optimization with respect to the hints.

Hereafter, we summarize the main outcomes of our experiments:

(a) Comparing sc1 and sc2, we conclude that data partitioning alone can render query answering less efficient.
(b) Comparing hom and sc2, we notice that simply adding a federation layer does not have a significant impact on query answering. In some cases, the federated queries are evaluated even faster. Most probably, for those queries involving unions of joins, the evaluation is performed by Teiid in a parallel way.
(c) Comparing hom and het, we see that federation of heterogeneous data sources shows a significant performance decrease, which can be explained by the fact that accessing non-relational sources is rather expensive.
(d) Looking at the three cases about hom, we see that optimization with hints is very effective. In particular, when all the hints are employed, the performance is much better than sc2, and even closer to sc1. Actually, from Figure 5, almost half of the queries in the 200K and 2M product cases perform better than sc1.
(e) In the heterogeneous cases, optimization also helps, but the impact is not as significant as for the homogeneous cases. Still, we do observe that when materialized views can be used, the performance can be improved dramatically.

7 RELATED WORK

OBDA [12, 33] is a semantic technology-based paradigm that has been developed since the mid 2000s [42, 43] with the aim to ease the access to legacy data stored in relational data source. Most research in this field has focused on *query rewriting* [8, 9, 13, 18, 24, 25, 32, 41], that is, on the problem of reformulating queries over the virtual RDF graph into an equivalent query over the data source.

Data federation studies the problem of accessing multiple, distributed and possibly heterogeneous data sources [2, 3, 16, 17, 23, 29, 31, 36, 37, 39, 40] via a unified schema. Many mature and high-optimized data federation systems have been developed in both academia and industry (e.g., *ANAPSID*, Teiid, and Denodo). For a comprehensive and up-to-date survey of these systems, and a brief overlook on the main optimization techniques used therein, the interested reader might refer to [19].

8 CONCLUSION

This work introduces the Ontology-based Data Federation setting and studies the problem of optimizing query translations in this setting. We provide techniques to address this problem, which are based on source data information that can be automatically computed in an offline stage by exploiting the information encoded in an OBDF specification. We performed an extensive empirical evaluation, showing that our techniques have a significant impact on the overall performance of query answering.

In this work we laid the foundations of OBDF. In future work we plan on further investigating hint-based optimizations, as well as implementing our algorithms in an actual system and applying OBDF to tackle complex, real-world scenarios. A more sophisticated handling of static and dynamic sources (e.g., [6]) might be necessary in these scenarios.

ACKNOWLEDGMENTS

This research has been partially supported by the EU H2020 project INODE (grant agreement No. 863410), by the Italian PRIN project HOPE (2019-2022), by the Free University of Bozen-Bolzano through the project MP4OBDA, and by the "Fusion Grant" project HIVE.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison Wesley Publ. Co.

[2] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. 2011. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 7031)*, Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist (Eds.). Springer, 18–34. https://doi.org/10.1007/978-3-642-25073-6_2

[3] Rana Alotaibi, Bogdan Cautis, Alin Deutsch, Moustafa Latrache, Ioana Manolescu, and Yifei Yang. 2020. ESTOCADA: Towards Scalable Polystore Systems. *Proc. VLDB Endow.* 13, 12 (2020), 2949–2952. https://doi.org/10.14778/3415478.3415516

[4] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2007. *The Description Logic Handbook: Theory, Implementation and Applications* (2nd ed.). Cambridge University Press.

[5] Christian Bizer and Andreas Schultz. 2009. The Berlin SPARQL Benchmark. *Int. J. on Semantic Web and Information Systems* 5, 2 (2009), 1–24.

[6] Carlos Bobed, Fernando Bobillo, Sergio Ilarri, and Eduardo Mena. 2014. Answering Continuous Description Logic Queries: Managing Static and Volatile Knowledge in Ontologies. *Int. J. Semant. Web Inf. Syst.* 10, 3 (jul 2014), 1–44. https://doi.org/10.4018/IJSWIS.2014070101

[7] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. 2015. Reformulation-based Query Answering in RDF: Alternatives and Performance. *Proc. of the VLDB Endowment* 8, 12 (2015), 1888–1891. http://www.vldb.org/pvldb/vol8/p1888-bursztyn.pdf

[8] Damian Bursztyn, François Goasdoué, and Ioana Manolescu. 2016. Teaching an RDBMS about ontological constraints. *Proc. VLDB Endow.* 9, 12 (2016), 1161–1172. https://doi.org/10.14778/2994509.2994532

[9] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2012. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Semant.* 14 (2012), 57–83. https://doi.org/10.1016/j.websem.2012.03.001

[10] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. 2017. Ontop: Answering SPARQL Queries over Relational Databases. *Semantic Web J.* 8, 3 (2017), 471–487. https://doi.org/10.3233/SW-160217

[11] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. 2011. The Mastro System for Ontology-Based Data Access. *Semantic Web J.* 2, 1 (2011), 43–53.

[12] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. *J. of Automated Reasoning* 39, 3 (2007), 385–429. https://doi.org/10.1007/s10817-007-9078-x

[13] Alexandros Chortaras, Despoina Trivela, and Giorgos B. Stamou. 2011. Optimized Query Rewriting for OWL 2 QL. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6803)*, Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer, 192–206. https://doi.org/10.1007/978-3-642-22438-6_16

[14] Souripriya Das, Seema Sundara, and Richard Cyganiak. 2012. *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. World Wide Web Consortium. Available at http://www.w3.org/TR/r2rml/.

[15] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration.* Morgan Kaufmann.

[16] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Rec.* 44, 2 (2015), 11–16. https://doi.org/10.1145/2814710.2814713

[17] Kemele M. Endris, Philipp D. Rohde, Maria-Esther Vidal, and Sören Auer. 2019. Ontario: Federated Query Processing Against a Semantic Data Lake. In *Database and Expert Systems Applications - 30th International Conference, DEXA 2019, Linz, Austria, August 26-29, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11706)*, Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil (Eds.). Springer, 379–395. https://doi.org/10.1007/978-3-030-27615-7_29

[18] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. 2014. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.* 39, 3 (2014), 25:1–25:46. https://doi.org/10.1145/2638546

[19] Zhenzhen Gu, Francesco Corcoglioniti, Davide Lanti, Alessandro Mosca, Guohui Xiao, Jing Xiong, and Diego Calvanese. 2022. A systematic overview of data federation systems. *Semantic Web J.* (2022). To appear in print. Available at tinyurl.com/48tpyy88.

[20] Laura M. Haas, Eileen Tien Lin, and Mary A. Roth. 2002. Data Integration through Database Federation. *IBM Systems J.* 41, 4 (2002), 578–596.

[21] Alon Y. Halevy. 2001. Answering Queries Using Views: A Survey. *Very Large Database J.* 10, 4 (2001), 270–294.

[22] Steve Harris and Andy Seaborne. 2013. *SPARQL 1.1 Query Language*. W3C Recommendation. World Wide Web Consortium. Available at http://www.w3.org/TR/sparql11-query.

[23] Yasar Khan, Antoine Zimmermann, Alokkumar Jha, Vijay Gadepally, Mathieu d'Aquin, and Ratnesh Sahay. 2019. One Size Does Not Fit All: Querying Web Polystores. *IEEE Access* 7 (2019), 9598–9617. https://doi.org/10.1109/ACCESS.2018.2888601

[24] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. 2012. Conjunctive Query Answering with OWL 2 QL. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*, Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith (Eds.). AAAI Press.

[25] Mélanie König, Michel Leclère, Marie-Laure Mugnier, and Michaël Thomazo. 2013. On the Exploration of the Query Rewriting Space with Existential Rules. In *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7994)*, Wolfgang Faber and Domenico Lembo (Eds.). Springer, 123–137. https://doi.org/10.1007/978-3-642-39666-3_10

[26] Davide Lanti, Martin Rezk, Guohui Xiao, and Diego Calvanese. 2015. The NPD Benchmark: Reality Check for OBDA Systems. In *Proc. of the 18th Int. Conf. on Extending Database Technology (EDBT)*. OpenProceedings.org, 617–628. https://doi.org/10.5441/002/edbt.2015.62

[27] Davide Lanti, Guohui Xiao, and Diego Calvanese. 2017. Cost-Driven Ontology-Based Data Access. In *Proc. of the 16th Int. Semantic Web Conf. (ISWC) (Lecture Notes in Computer Science, Vol. 10587)*. Springer, 452–470. https://doi.org/10.1007/978-3-319-68288-4_27

[28] Davide Lanti, Guohui Xiao, and Diego Calvanese. 2019. VIG: Data Scaling for OBDA Benchmarks. *Semantic Web J.* 10, 2 (2019), 413–433. https://doi.org/10.3233/SW-180336

[29] Mohamed Nadjib Mami, Damien Graux, Simon Scerri, Hajira Jabeen, Sören Auer, and Jens Lehmann. 2019. Squerall: Virtual Ontology-Based Access to Heterogeneous and Large Data Sources. In *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11779)*, Chiara Ghidini, Olaf Hartig, Maria Maleshkova, Vojtech Svátek, Isabel F. Cruz, Aidan Hogan, Jie Song, Maxime Lefrançois, and Fabien Gandon (Eds.). Springer, 229–245. https://doi.org/10.1007/978-3-030-30796-7_15

[30] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2012. *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation. World Wide Web Consortium. Available at http://www.w3.org/TR/owl2-profiles/.

[31] Andriy Nikolov, Peter Haase, Johannes Trame, and Artem Kozlov. 2017. Ephedra: Efficiently Combining RDF Data and Services Using SPARQL Federation. In *Knowledge Engineering and Semantic Web - 8th International Conference, KESW 2017, Szczecin, Poland, November 8-10, 2017, Proceedings (Communications in Computer and Information Science, Vol. 786)*, Przemyslaw Rózewski and Christoph Lange (Eds.). Springer, 246–262. https://doi.org/10.1007/978-3-319-69548-8_17

[32] Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. 2010. Tractable query answering and rewriting under description logic constraints. *J. Appl. Log.* 8, 2 (2010), 186–209. https://doi.org/10.1016/j.jal.2009.09.004

[33] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2008. Linking Data to Ontologies. *J. on Data Semantics* 10 (2008), 133–173. https://doi.org/10.1007/978-3-540-77688-8_5

[34] Freddy Priyatna, Oscar Corcho, and Juan F. Sequeda. 2014. Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using morph. In *Proc. of the 23rd Int. World Wide Web Conf. (WWW)*. 479–490. https://doi.org/10.1145/2566486.2567981

[35] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. 2013. Ontology-Based Data Access: Ontop of Databases. In *Proc. of the 12th Int. Semantic Web Conf. (ISWC) (Lecture Notes in Computer Science, Vol. 8218)*. Springer, 558–573. https://doi.org/10.1007/978-3-642-41335-3_35

[36] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. 2014. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings (Lecture Notes in*

Computer Science, Vol. 8465), Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai (Eds.). Springer, 176–191. https://doi.org/10.1007/978-3-319-07443-6_13
[37] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. 2011. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *Proc. of the 10th Int. Semantic Web Conf. (ISWC) (Lecture Notes in Computer Science, Vol. 7031)*. Springer, 601–616.
[38] Juan F. Sequeda and Daniel P. Miranker. 2013. Ultrawrap: SPARQL Execution on Relational Data. *J. of Web Semantics* 22 (2013), 19–39.
[39] Amit P. Sheth and James A. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *Comput. Surveys* 22, 3 (1990), 183–236.
[40] Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. 2018. Comunica: A Modular SPARQL Query Engine for the Web. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11137)*, Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl (Eds.). Springer, 239–255. https://doi.org/10.1007/978-3-030-00668-6_15
[41] Michaël Thomazo. 2013. Compact Rewritings for Existential Rules. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, Francesca Rossi (Ed.). IJCAI/AAAI, 1125–1131. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6826
[42] Guohui Xiao, Diego Calvanese, Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. 2018. Ontology-Based Data Access: A Survey. In *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. IJCAI Org., 5511–5519. https://doi.org/10.24963/ijcai.2018/777
[43] Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego Calvanese. 2019. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. *Data Intelligence* 1, 3 (2019), 201–223. https://doi.org/10.1162/dint_a_00011
[44] Guohui Xiao, Roman Kontchakov, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. 2018. Efficient Handling of SPARQL Optional for OBDA. In *Proc. of the 17th Int. Semantic Web Conf. (ISWC) (Lecture Notes in Computer Science)*. Springer, 354–373.
[45] Guohui Xiao, Davide Lanti, Roman Kontchakov, Sarah Komla-Ebri, Elem Güzel-Kalayci, Linfang Ding, Julien Corman, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. 2020. The Virtual Knowledge Graph System Ontop. In *Proc. of the 19th Int. Semantic Web Conf. (ISWC) (Lecture Notes in Computer Science, Vol. 12507)*. Springer, 259–277. https://doi.org/10.1007/978-3-030-62466-8_17