OBDA Over Non-Relational Databases*

Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Martin Rezk, and Guohui Xiao

Free University of Bozen-Bolzano, Italy, *lastname*@inf.unibz.it

Abstract. The database landscape has been significantly diversified during the last decade, resulting in the emergence of a variety of non-relational (also called NoSQL) databases, e.g., XML and JSON-document databases, key-value stores, and graph databases. To facilitate access to such databases and to enable data integration of non-relational data sources, we generalize the well-known ontology-based data access (OBDA) framework so as to allow for querying arbitrary databases through a mediating ontology. We instantiate this framework to MongoDB, a popular JSON-document database, and implement a prototype extension of the virtual OBDA system *Ontop* for answering SPARQL queries over MongoDB.

1 Introduction

Accessing data using native query languages is getting a more and more involved task for users, as databases (DBs) increase in complexity and heterogeneity. The Ontology-Based Data Access (OBDA) paradigm [9] has emerged as a proposal to simplify this kind of access, by allowing users to write high-level ontological queries, which in the classical virtual approach are translated automatically into low-level queries that DB engines can handle. This separation of concerns between the conceptual level and the DB level has been proven successful in practice, notably when data sources have a complex structure and end-users have domain but not necessarily data management expertise [6,5,1]. The OBDA approach is implemented by connecting a DB to an ontology by means of mappings, where traditionally the ontology is expressed in the OWL 2 QL profile of the Web Ontology Language OWL 2 [7], the queries are formulated in SPARQL, the Semantic Web query language, and the DB is assumed to be relational [4].

As envisioned by Stonebraker and Cetintemel [11], a multitude of DB architectures is required to satisfy the needs of a wide variety of modern applications. This has been confirmed by the significant diversification of the DB landscape during the last decade. Some of these architectural changes have been proposed within the scope of relational DBs (e.g., column-oriented storage), while many have gone beyond them, causing the emergence of the so-called *NoSQL* (not only SQL) DBs. These *non-relational* DBs usually adopt one of four main data models, namely the column-family, key-value, document, and graph data models, and provide an (intimidating) number of query languages with varying querying capabilities. Notably, many of these new languages are limited [8] and thus clients might need to set up compensating post-processing techniques so as to satisfy advanced information needs. In this paper, to let users query non-relational DBs at the conceptual level, we propose to extend the OBDA framework to non-relational DBs, and then instantiate it to MongoDB, a popular document DB.

* This is an abridged version of [3].

2 Generalized OBDA framework

We consider fixed countably infinite sets $\mathbb{C}$ of DB values, and $\mathbb{E}_{\mathbf{D}}$ of elements built over $\mathbb{C}$, e.g., named tuples, trees, or XML documents. We assume to deal with a class $\mathbf{D}$ of DBs, where each DB in $\mathbf{D}$ is a finite subset of $\mathbb{E}_{\mathbf{D}}$, e.g., relational DBs, MongoDB, or XML DBs. Moreover, we assume that $\mathbf{D}$ comes equipped with:

- Suitable forms of constraints, which might express both information about the structure of the data in DBs of $\mathbf{D}$, e.g., the schema information in relational DBs, and "constraints" in the usual sense of DBs, e.g., primary and foreign key constraints for relational DBs. We call a collection of such constraints a $\mathbf{D}$-*schema*.
- A query language $\mathcal{Q}_{\mathbf{D}}$, such that, for each query $q \in \mathcal{Q}_{\mathbf{D}}$ and for each instance $D \in \mathbf{D}$, the answer $ans(q, D)$ of q over D is defined, and is itself a DB in $\mathbf{D}$.
- A relational *wrapper* $[\![\cdot]\!]_{\mathbf{D}}$, which is a function transforming a $\mathcal{Q}_{\mathbf{D}}$-query q into a new query $[\![q]\!]_{\mathbf{D}}$ that takes a DB in $\mathbf{D}$ and returns a relation over $\mathbb{C}$ (i.e., a relation in first normal form)[1]. The role of the wrapper is to present $ans(q, D)$ as a relation, by actually computing a query that retrieves from D what can be considered as the relational representation of $ans(q, D)$. In particular, when q is the identity query, $[\![q]\!]_{\mathbf{D}}$ is the query computing the relational view of a DB $D \in \mathbf{D}$.

Having these building blocks at hand, we now define $\mathbf{D}$-mapping assertions and their semantics. We start by introducing the notion of variable-to-RDF-term map, which is a generalization of the RDF-term template in relational OBDA. We say that $f(x_1, \ldots, x_n)$ is an *RDF term constructor* if it is a (partial) function $f : \mathbb{C}^n \to \mathbf{I} \cup \mathbf{L}$, where $\mathbf{I}$ is the set of IRIs and $\mathbf{L}$ is the set of RDF literals. Then, a *variable-to-(RDF)-term map* κ (for variable $?X$) has the form $?X \mapsto f(x_1, \ldots, x_n)$. In the following, $\pi_{x_1,\ldots,x_n}$ denotes the standard projection operator of relational algebra.

Definition 1. *A* $\mathbf{D}$-mapping assertion m *is an expression* $q \rightsquigarrow_K h$ *where:*

- q *is a* $\mathcal{Q}_{\mathbf{D}}$*-query, called* source query*;*
- h *is an RDF triple pattern, called* target*, of the form* $(?X_1$ `rdf:type` $A)$ *or* $(?X_1\ P\ ?X_2)$*, where* A *is a class name, and* P *is a property name;*
- K *is a set of variable-to-term maps, one for each variable* $?X_i$ *appearing in* h*.*

The mapping assertion m *is* safe *if for each* $?X_i \mapsto f(x_1, \ldots, x_n)$ *in* K*, we have that the function* $\pi_{x_1,\ldots,x_n} \circ [\![q]\!]_{\mathbf{D}}$ *is well-defined.*

A $\mathbf{D}$-mapping $\mathcal{M}$ *is a finite set of* $\mathbf{D}$*-mapping assertions.*

Notice that, the mapping assertion m is safe if and only if, for each variable-to-term map $?X_i \mapsto f(x_1, \ldots, x_n)$ used by m, the wrapper $[\![\cdot]\!]_{\mathbf{D}}$, when applied to the source query q of m, returns a query producing a relation whose attributes contain $x_1, \ldots, x_n$.

Definition 2. *Let* $\mathbf{D}$ *be a class of DBs,* $m = q \rightsquigarrow_K h$ *a* $\mathbf{D}$*-mapping assertion, and* $D \in \mathbf{D}$*. The* RDF graph $m(D)$ generated by m from D *is defined as follows:*

- *When* $h = (?X_1$ `rdf:type` $A)$*, then*

$$m(D) = \big\{(f(\boldsymbol{v})\ \texttt{rdf:type}\ A) \mid K = \{?X_1 \mapsto f(\boldsymbol{x})\},\ \boldsymbol{v} \in \pi_{\boldsymbol{x}}([\![q]\!]_{\mathbf{D}}(D))\big\}.$$

[1] Discussing the specific form and properties of wrappers is outside the scope of this paper.

– *When* $h = (?X_1\ P\ ?X_2)$*, then*

$$m(D) = \big\{(f_1(\boldsymbol{v}_1)\ P\ f_2(\boldsymbol{v}_2)) \mid K = \{?X_1 \mapsto f_1(\boldsymbol{x}_1),\ ?X_2 \mapsto f_2(\boldsymbol{x}_2)\},\ \boldsymbol{v}_1 \in \pi_{\boldsymbol{x}_1}([\![q]\!]_{\mathbf{D}}(D)),\ \boldsymbol{v}_2 \in \pi_{\boldsymbol{x}_2}([\![q]\!]_{\mathbf{D}}(D))\big\}.$$

For a $\mathbf{D}$*-mapping* $\mathcal{M}$*, the RDF graph* $\mathcal{M}(D)$ *is defined as* $\bigcup_{m\in\mathcal{M}} m(D)$.

Now, an *OBDA specification for* $\mathbf{D}$ is a triple $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}\rangle$, where $\mathcal{T}$ is an ontology, $\mathcal{S}$ is a $\mathbf{D}$-schema, and $\mathcal{M}$ is a $\mathbf{D}$-mapping. An *OBDA instance for* $\mathbf{D}$ consists of an OBDA specification $\langle \mathcal{T}, \mathcal{M}, \mathcal{S}\rangle$ for $\mathbf{D}$ and an instance $D \in \mathbf{D}$ satisfying $\mathcal{S}$. The semantics of such an instance is derived naturally from the semantics of $\mathbf{D}$-mapping assertions.

3 OBDA Framework over MongoDB

In this section we instantiate the generalized OBDA framework to MongoDB, a popular DB that stores collections of semi-structured JSON-style documents. A sample document consisting of (possibly nested) key-value pairs and arrays is given below.

```
{ "_id": 4,
  "awards": [
    { "award": "Rosing Prize", "year": 1999, "by": "Norwegian Data Association" },
    { "award": "Turing Award", "year": 2001, "by": "ACM" },
    { "award": "IEEE John von Neumann Medal", "year": 2001, "by": "IEEE" } ],
  "birth": "1926-08-27",
  "contribs": ["OOP", "Simula"],
  "death": "2002-08-10",
  "name": {
      "first": "Kristen", "last": "Nygaard" }
}
```

This instantiation relies on work in [2], where we obtain the following results:

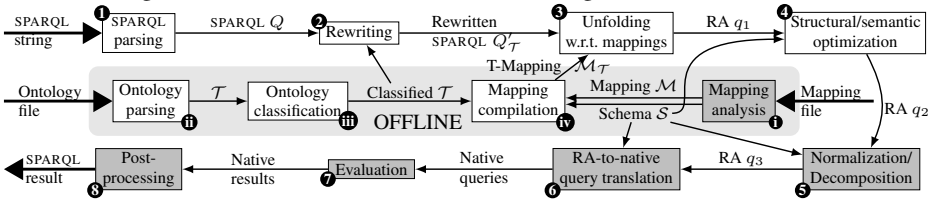
– We formalize a fragment of MongoDB aggregate queries that consists of match, unwind, project, group, and lookup stages, and that we call MUPGL.
– We propose a notion of MongoDB type constraints. These constraints allow one to specify that certain paths (concatenations of keys) must point to an array (e.g., `awards`), or an atomic value (e.g., `name.last`), or an object (e.g., `name`).
– We define a relational view over MongoDB with respect to a set of type constraints.
– We develop a translation from RA expressions (over the relational view) to MUPGL queries. This translation shows that full RA can be captured by MUPGL, while RA over a single collection can be captured by MUPG.

We start by introducing a compact notation for MongoDB mapping assertions, which we also use to specify type constraints. We consider two extensions of paths called *array paths*. An *array index path* is a path where a key is followed by any combination of keys and '#', separated by dots. An *array element path* is the concatenation of an array index path with '.[#]'. Intuitively, for (simple) paths p_1 and p_2, any of the array paths $p_1.\#.p_2$, $p_1.\#$, or $p_1.[\#]$ imply that p_1 must point to an array. Moreover, the path $p_1.\#.p_2$ (e.g., `awards.#.year`) is used to access the value of the path p_2 inside the array pointed to by p_1, while $p_1.\#$ (e.g., `awards.#`) is used to denote the indices and $p_1.[\#]$ (e.g., `contribs.[#]`) to denote the single elements of such an array. Hence, the presence of '#' or '[#]' requires that the path preceding it points to an array, whereas a path that does not end with '#' or '[#]' must point to atomic values. We use *extended paths* to refer both to normal paths and to array paths.

The source queries we consider are a restricted form of MUP queries and can be represented as a pair (C, φ), where C is a collection name and φ is a criterion constructed using extended paths. Let K be a set of variable-to-term maps $?X \mapsto f(p_1, \ldots, p_n)$, where each p_i is an extended path. Then, a *MongoDB mapping assertion* is an expression of the form $(C, \varphi) \rightsquigarrow_K h$, where the variables in h constitute the domain of K. Given a MongoDB mapping $\mathcal{M}$, we can extract from it the MongoDB schema $\mathcal{S}_\mathcal{M}$ (a set of type constraints), and then define a relational wrapper $[\![\cdot]\!]_{\text{MongoDB}}$ for the source queries in $\mathcal{M}$ (see [3] for more details).

We built a prototype implementation for answering SPARQL queries over MongoDB as an extension of the state-of-the-art OBDA system *Ontop* [4].

Architecture of *Ontop*. Query answering (QA) in *Ontop* under the OWL 2 QL entailment regime involves an offline and an online stage:



The offline stage (highlighted in gray) takes as input the mapping and the ontology files and produces three entities used by the online QA stage: the classified ontology, the DB schema (extracted from the mapping file), and the *T-mapping*, constructed by 'compiling' the classified ontology into the input mapping [10]. The online stage handles individual SPARQL queries: ① the input SPARQL query is parsed, ② rewritten w.r.t. the ontology, and ③ unfolded w.r.t. the T-mapping; ④ (the internal representation of) the resulting RA query is simplified by applying structural and semantic optimization techniques; ⑤ the RA query is normalized and (possibly) decomposed so that it can be directly handled by the RA-to-native-query translator; ⑥ each normalized RA (sub)query is translated into a native query, which is then ⑦ evaluated by the DB engine; ⑧ the native results are post-processed into SPARQL results.

Implementation for MongoDB. We observe that steps ⓘⓘ–ⓘⓥ and ①–④ are independent of the actual class $\mathbf{D}$ of DBs, while steps ⓘ and ⑤–⑧ require specific implementations according to $\mathbf{D}$. Therefore, our prototype implements the latter five components. The current implementation supports MongoDB 3.2 and is able to return sound and complete answers to the subset of SPARQL queries that *(i)* correspond to BGPs with filters consisting of comparisons, and *(ii)* can be translated into MUPG queries.

4 Conclusions

We are currently working on providing support for (most of) SPARQL 1.0. In the future we want to explore the tradeoff between delegating all of query processing to MongoDB vs. a fine-grained decomposition of the input query together with post-processing to combine the results of the sub-queries. We will test our techniques on real-world large-scale use-cases, so as to assess the practical feasibility of OBDA over MongoDB.

References

1. N. Antonioli, F. Castanò, C. Civili, S. Coletta, S. Grossi, D. Lembo, M. Lenzerini, A. Poggi, D. F. Savo, and E. Virardi. Ontology-based data access: the experience at the Italian Department of Treasury. In *Proc. of the Industrial Track of the 25th Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, volume 1017 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, pages 9–16, 2013.
2. E. Botoeva, D. Calvanese, B. Cogrel, M. Rezk, and G. Xiao. A formal presentation of MongoDB (Extended version). CoRR Technical Report abs/1603.09291, arXiv.org e-Print archive, 2016. Available at `http://arxiv.org/abs/1603.09291`.
3. E. Botoeva, D. Calvanese, B. Cogrel, M. Rezk, and G. Xiao. OBDA beyond relational DBs: A study for MongoDB. In *Proc. of the 29th Int. Workshop on Description Logics (DL)*, volume 1577 of *CEUR Electronic Workshop Proceedings,* `http://ceur-ws.org/`, 2016.
4. D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web J.*, 2016. To appear.
5. D. Calvanese, P. Liuzzo, A. Mosca, J. Remesal, M. Rezk, and G. Rull. Ontology-based data integration in EPNet: Production and distribution of food during the Roman Empire. *Engineering Applications of Artificial Intelligence*, 2016. To appear.
6. M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jiménez-Ruiz, D. Lanti, M. Rezk, G. Xiao, Ö. L. Özçep, and R. Rosati. Optique: Zooming in on Big Data. *IEEE Computer*, 48(3):60–67, 2015.
7. B. Motik, A. Fokoue, I. Horrocks, Z. Wu, C. Lutz, and B. Cuenca Grau. OWL Web Ontology Language profiles. W3C Recommendation, World Wide Web Consortium, Oct. 2009. Available at `http://www.w3.org/TR/owl-profiles/`.
8. K. W. Ong, Y. Papakonstantinou, and R. Vernoux. The SQL++ query language: Configurable, unifying and semi-structured. CoRR Technical Report abs/1405.3631, arXiv.org e-Print archive, 2014. Available at `http://arxiv.org/abs/1405.3631`.
9. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
10. M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In *Proc. of the 12th Int. Semantic Web Conf. (ISWC)*, volume 8218 of *Lecture Notes in Computer Science*, pages 558–573. Springer, 2013.
11. M. Stonebraker and U. Cetintemel. "one size fits all": An idea whose time has come and gone. In *Proc. of the 21st IEEE Int. Conf. on Data Engineering (ICDE)*, pages 2–11, 2005.