On Rewriting and Answering Queries in OBDA Systems for Big Data (Short Paper)

Diego Calvanese[2], Ian Horrocks[3], Ernesto Jimenez-Ruiz[3], Evgeny Kharlamov[3], Michael Meier[1] Mariano Rodriguez-Muro[2], Dmitriy Zheleznyakov[3]

[1] fluid Operations AG, [2] Free University of Bozen-Bolzano, [3] Oxford University

Abstract. The project Optiqueaims at providing an end-to-end solution for scalable access to Big Data integration, were end users will formulate queries based on a familiar conceptualization of the underlying domain. From the users queries the Optique platform will automatically generate appropriate queries over the underlying integrated data, optimize and execute them. In this paper we discuss Optique's automatic generation of queries and two systems to support this process: QUEST and PEGASUS The automatic query generation is important and challenging, especially when the queries are over heterogeneous distributed databases and streams, and Optique will provide a scalable solution for it.

1 Introduction

A typical problem that end-users face when dealing with Big Data is the data access problem, which arises due to the three dimensions (so-called "3V") of Big Data: *volume*, since massive amounts of data have been accumulated over the decades, *velocity*, since the amounts keep increasing, and *variety*, since the data are spread over vast variety of formats and sources. In the context of Big Data, accessing the *relevant* information is an increasingly difficult problem.

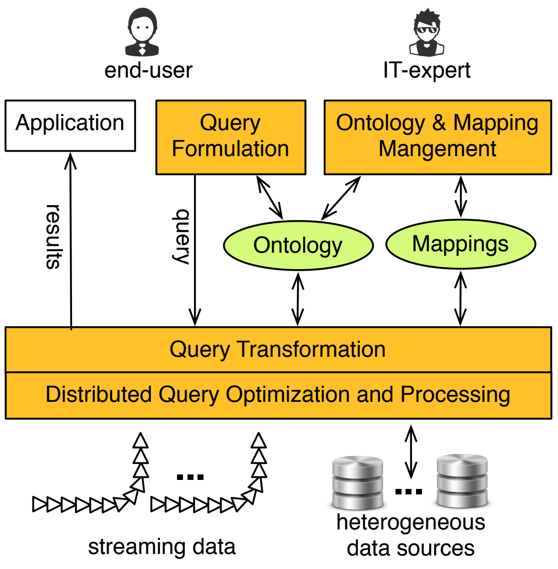

Fig. 1. General architecture of the Optique OBDA system

Existing approaches to data access limit it to a restricted set of predefined quires. In the situation when an end-user needs data that current applications cannot provide, the help of an IT-expert is required. The IT-expert translates the information need of the end-user to specialized queries and optimize them for an efficient execution. One of the major flaws of this approach is that accessing data can take several days. In data-intensive industries, engineers spend up to 80% of their time on data access problems [4]. Apart from the enormous direct cost, freeing up expert time would lead to even greater value creation through deeper analysis and improved decision making.

The approach, known as “Ontology-Based Data Access” (OBDA) [13,2], has the potential to address the data access problem by automating the translation process of the users information needs. The key idea is to use an ontology that confronts the user with a conceptual model of the problem domain. The user formulates their information needs, that is, requests in terms of the ontology and then receives the answers in the same understandable form. These requests should be executed over the data automatically, without the intervention of IT-experts. To this end, a set of mappings is maintained which describes the relationship between the terms in the ontology and data sources.

Automating the translation from users’ requests expressed, e.g., in SPARQL, to highly optimized executable code over data sources, e.g., SQL queries, is a key challenge in development of OBDA systems. In this paper we will discuss this automation in the context of the recently started EU project Optique [6]. We will present the Query Transformation component of the Optique OBDA platform. More generally, Optique aims at developing an OBDA system of the new generation integrated via the Information Workbench platform [7](cf. Figure 1). It will address a number of issues besides the automated query translation. In particular, it will address: *(i)* user-friendly query formulation interface, *(ii)* maintenance of an ontology and mappings *(iii)* processing and analytics over streaming data, and *(iv)* distributed query optimization and execution.

In Section 2, we present the architecture of Optique’s query transformation module. In Section 2.1, we present typical scenarios for the use of its component. In Sections 3.1, we introduce the QUEST system [13] that is intended to be the core of the Optique’s query transformation solution. In Section 3.2, we present a promising new system PEGASUS, which we plan to use to support the Optique’s query transformation.

2 Architecture

In Figure 1, you can see a general, conceptual overview of the Optique OBDA system. Due to this, it mentions only main components of the system, while the system includes more component, and shows the data flow, depicted with arrows.

Let us now see in details how the *Query Transformation* (QT) module works. You can find its architecture in Figure 2. Note that the figure mentions only those components that are relevant to the Query Transformation. The meaning of the arrows in Figure 2 is the following: an arrows goes from a (sub-)module X to a (sub-)module Y if X can call Y during the run.

The Query Transformation component includes the following subcomponents:

The Setup module can be thought of as the initialization module of the system. Its task is to receive configurations from the Configuration module (which is an Optique component external to the QT module) beforehand the actual query transformation happens, and distribute this configuration to the other modules of the QT module (cf. Section 2.1).

The Semantic indexing and **Materialization** modules are in charge of creation and maintenance of so-called semantic index [13].

The Query Transformation Manager is the main QT submodule. It coordinates the work of the other submodules and manages the query transformation process.

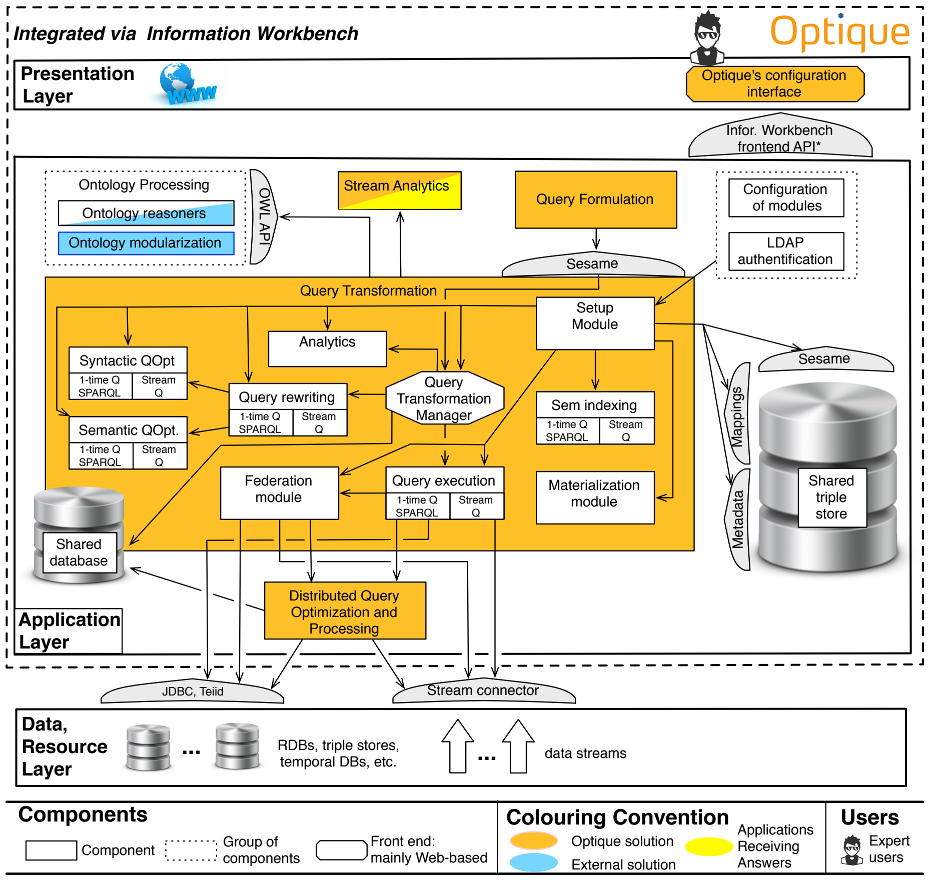

Fig. 2. Query transformation component of the Optique OBDA system

The Query rewriting module is in charge of query rewriting process. It transforms queries from the ontological vocabulary, e.g., SPARQL queries, into a format which is required to query the data sources, e.g., SQL (see Section 2.1 for details).

The Syntactic Query Optimization and **Semantic Query Optimization** modules are subroutines of the Query rewriting. They perform query optimization during query rewriting process (see Section 2.1 for details).

The Query execution module is in charge of the actual query evaluation.

The Federation module is a subroutine of the Query execution module that is needed to perform query answering over federated databases.

The Analytics module analyzes answers to a query and decides how to proceed with them further (see Section 2.1 for details).

Also, the QT module interacts with other components of the Optique OBDA system:

The Query Formulation module provides a query interface for end-users. This module receives queries from end-users and send them to the QT module, e.g., via Sesame API.

The Configuration module provides the configuration for the QT module that is required for query transformation performance.

The Ontology Processing is a group of components such as ontology reasoners, modularization systems, etc. It is called by the QT module to perform semantic optimization.

The Distributed Query Optimization and Processing component receives rewritten queries from the QT module and performs their evaluation over data sources.

The Shared database contains the technical data required for data answering process, such as semantic indices, answers to queries, etc.

The Shared triple store contains the data that can be used by (the most of) the components of the Optique OBDA system. E.g., it contains the ontology, the mappings, the query history, etc. The QT module calls this store, for example, for reasoning over the ontology during the semantic optimization process.

The Stream analytics module provides analysis of answers to the stream quires.

Data sources (RDBs, triple stores, data streams) can be also accessed by the QT module during the query execution (see Section 2.1 for details).

2.1 Query Transformation Process

In this section we will discuss how the QT module performs the query transformation task. This process can be divided into several stages. Assume that the QT module receives a query from the Query Formulation module. Then the first stage of the query transformation process is the initialization of the process.

Initialization. At this stage the Configuration module sends the configuration to the Setup module, which, in turn, configure the other modules of the QT module. The initialization includes several steps in which the input ontology and mappings get analyzed and optimized so as to allow the rewriting and optimization algorithms to be fast, and the query evaluation over the data sources to be more efficient (find more details in [14,12]). This includes the application of the Semantic Index technic.

Query rewriting. After the initialization, the query transformation itself starts. The Query Transformation Manager receives a (SPARQL) query Q from the Query Formulation module. Then it resends the query to the Query Rewriting module that rewrites the query in the required format, e.g., SQL for querying relational databases, or Streaming SPARQL for querying data streams. Further, for the sake of simplicity, we will assume that the target query format is SQL. Along with the rewriting, the Query Rewriting module optimizes the rewritten query both syntactically and semantically.

Syntactic optimization. During the transformation process, the initial query may be turned into a number of SQL queries $Q_1, \ldots, Q_n$ such that their union is equivalent to Q. In the Syntactic optimization stage, these queries get optimized to improve the performance, e.g., redundant joins, conditions, etc. within this SQL queries are deleted. The methods, used to detect what parts of the queries have to be optimized, are syntactical, that is they are based only on the shape of a query and do not involve any reasoning.

Semantic optimization. Then the semantic optimization of the queries is performed. The queries get optimized in the similar manner as in the case of Syntactic optimization. The difference is that the methods, used in Semantic optimization, are semantic, that is they take into account query containment, integrity constraints of the data sources, etc.

Query execution. After rewriting and optimization, the queries $Q'_{i_1}, \dots, Q'_{i_m}$ are returned to the Query Transformation Manager. It sends them to the Query Execution module. This module decides what queries of $Q'_{i_1}, \dots, Q'_{i_m}$, if any, need to be evaluated using distributing query execution, and what can be evaluated directly by the standard query answering facilities. In the former case, the corresponding queries are sent to the Distributed Query module. In the latter case, the corresponding queries are evaluated over the data sources by standard means. If some of the queries have to be evaluated over over a federated database system, the Query Execution module entrusts this task to the Federation module.

Query answer management. After the query evaluation process, the answers to the queries, that has been sent directly to the data sources, are returned to the Query Transformation Manager. The manager transform them into the required format and send them back to the Query Formulation module, which takes care of representing the answers to end-users.

The queries that have been sent to the Distributed Query Optimization and Processing module do not necessarily go directly to the Query Transformation Manager, but rather to the Shared database. The reason is that the answer can be very big (up to several GBs), so sending them directly to the QT component would hang the system. The Query Transformation Manager receives the signal that the answers are in the Shared database, and some metadata about the answer. Then, together with the Analytics module, it decides how to proceed further. The answers to one-time queries, e.g. SQL queries over relational databases, eventually go to the Query Formulation module, while the answers to stream queries go to the Stream analytics module.

3 Possible Implementations

In this section we will discuss two options for the implementation of the Query Transformation component: QUEST and PEGASUS.

3.1 Quest

One of the options for the QT module is the QUEST implementation [14]. QUEST can be used in several stages of the query transformation process. In particular, it performs the following tasks: *(i)* the initialization stage of the process in the manner described in Section 2.1; *(ii)* query execution that is based on standard relational technologies; *(iii)* rewriting and optimization. We will consider the last task in more details.

Rewriting and optimization. Also, QUEST performs query rewriting and optimization for one-time queries, which is done in QUEST by means of query rewriting into SQL. Given an input query Q, two steps are performed at query transformation time: *(i)* query rewriting, where Q is transformed into a new query Q_0 that takes into account the semantics of the OBDA ontology; *(ii)* query unfolding, where Q_0 is transformed into a single SQL query using the mappings of the OBDA system. We provide now an intuitive description of these two steps.

– **Query rewriting**. [3] Query rewriting uses the ontology axioms to compute a set of queries that encode the semantics of the ontology, such that if we evaluate the union of these queries over the data, we will obtain sound and complete answers. The query rewriting process works using the ontology axioms and unification to

generate more specific queries from the original, more general query. In QUEST, this process is iterative, and stops when no more queries can be generated.
We note that the query rewriting algorithm is a variation of the `TreeWitness` rewriter [8] optimized in order to obtain a fast query rewriting procedure that generates a minimal amount of queries.
- **Query unfolding**. Query unfolding uses the mapping program (see [14]) to transform the rewritten query into SQL. First, QUEST transforms the rewritten query into a Datalog program. Then, the program is resolved [9] against the mapping program, until no resolution step can be applied. At each resolution step, QUEST replaces an atom formulated in terms of ontology predicates, with the body of a mapping rule. In case there is no matching rule for an atom, that sub-query is logically empty and is eliminated. Finally, when no more resolution steps are possible, we have a new Datalog program, formulated in terms of database tables, that can be directly translated into a union of select-project-join SQL queries. Also at this step, QUEST makes use of query containment w.r.t. dependencies to detect redundant queries and to eliminate redundant join operations in individual queries (i.e., using primary key metadata). Last, QUEST always attempts to generate simple queries, with no sub-queries or structurally complex SQL. This is necessary to ensure that most relational database engines are able to generate efficient execution plans.

3.2 Pegasus

Another option for the query transformation phase is the PEGASUS implementation [11]. PEGASUS adapts the well-known Chase & Backchase (C&B) algorithm [5] and outperforms existing C&B implementations by usually two orders of magnitude. Originally, C&B was introduced in the context of semantic query optimization minimizing conjunctive queries under integrity constraints. Just recently, it was proved that C&B can be reconfigured for OBDA. More precisely, it can be used to compute perfect reformulations exploiting its logical properties. Thus, PEGASUS can be used in several stages of the query transformation process:

- semantic query optimization at SPARQL level, thus removing redundant join operations w.r.t the ontology,
- query rewriting, i.e. computing the perfect reformulation, and
- semantic query optimization at SQL level, i.e. optimizing SQL queries according to the constraints encoded in a database schema.

Generally speaking, C&B takes a basic graph pattern query and an ontology as input and applies two phases to optimize the input query: (i) the classical chase procedure [10,1] is used to deploy a preprocessing step in which a universal plan is computed and (ii) then in the proceeding backchase phase all (exponentially many) subqueries of the universal plan are enumerated and checked for equivalence to the original query, i.e. the backchase phase is the actual optimization process.

C&B makes use of the classical chase algorithm, which does not necessarily terminate. While semantic query optimization typically assumes a finite chase, OBDA does not necessarily do so. Thus, C&B have been extended in such a way that it can handle infinite chase sequences e.g. for DL-Lite ontologies. Furthermore, C&B have been extended to handle SPARQL 1.1 queries beyond basic graph patterns.

PEGASUS works in a bottom-up manner, i.e. during the Backchase phase it enumerates subqueries by increasing number of triple patterns. However, this is not done in a naive way. PEGASUS heavily makes use of pruning and optimization strategies to avoid inspecting a large portion of unnecessary queries at all. The main optimization steps in PEGASUS can be summarized as follows:

- The guided backchase avoids inspecting subqueries that will be subsumed by other queries in the perfect rewriting.
- Avoiding containment checks between queries with many triple patterns by reducing containment checks to sets of queries with few triple patterns only.
- A necessary condition for query containment that can be easily computed. PEGASUS performs full containment checks only when this condition is satisfied.

Applying these optimization steps input SPARQL (or SQL) queries can be efficiently rewritten to a form in which they can then be further processed.

4 Conclusions

In this paper we considered the query transformation and optimization problems in context of query answering in the Optique OBDA system. We introduced the Query Transformation component of the system and discussed the options of how this component can be implemented: QUEST and PEGASUS. We briefly introduced these two implementations and discussed their peculiarities.

References

1. Beeri, C., Vardi, M.Y.: A Proof Procedure for Data Dependencies. J. ACM 31(4), 718–741
2. Calvanese, D., Giacomo, G.D., 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 2(1), 43–53 (2011)
3. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family. JAR 39(3) (2007)
4. Crompton, J.: Keynote talk at the W3C Workshop on Semantic Web in Oil & Gas Industry: Houston, TX, USA, 9–10 December (2008), available from `http://www.w3.org/2008/12/ogws-slides/Crompton.pdf`
5. Deutsch, A., Popa, L., Tannen, V.: Physical data independence, constraints, and optimization with universal plans. pp. 459–470. VLDB '99 (1999)
6. Giese, M., Calvanese, D., Haase, P., Horrocks, I., Ioannidis, Y., Kllapi, H., Koubarakis, M., Lenzerini, M., Möller, R., Özep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Schmidt, M., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In: Rajendra Akerkar: Big Data Computing. Florida : Chapman and Hall/CRC. To appear. (2013)
7. Haase, P., Schmidt, M., Schwarte, A.: The information workbench as a self-service platform for linked data applications. In: COLD (2011)
8. Kikot, S., Kontchakov, R., Zakharyaschev, M.: Conjunctive Query Answering with OWL 2 QL. In: KR (2012)
9. Leitsch, A.: The resolution calculus. Texts in theoretical computer science, Springer (1997)
10. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing Implications of Data Dependencies. ACM Trans. Database Syst. 4(4), 455–469 (1979)
11. Meier, M.: The backchase revisited. Submitted for Publication (2013)
12. Rodriguez-Muro, M., Calvanese, D.: Dependencies: Making Ontology Based Data Access Work. In: AMW (2011)
13. Rodriguez-Muro, M., Calvanese, D.: High Performance Query Answering over DL-Lite Ontologies. In: KR (2012)
14. Rodriguez-Muro, M., Calvanese, D.: Quest, an OWL 2 QL Reasoner for Ontology-based Data Access. In: OWLED (2012)