Ontology-Based Data Access Using Views

Juan F. Sequeda[1], Marcelo Arenas[2], and Daniel P. Miranker[1]

[1] Department of Computer Science, The University of Texas at Austin
{jsequeda,miranker}@cs.utexas.edu
[2] Department of Computer Science, PUC Chile
marenas@ing.puc.cl

1 Our Position

The OWL 2 QL profile, which is based on DL-Lite$_R$, has been designed so that query answering is possible using relational database technology via query rewriting. Unfortunately, given a query Q posed in terms of an OWL 2 QL ontology[1] O, the size of the rewritten query, Q_o, which can be evaluated directly on the relational database, is worst case exponential w.r.t the size of Q and O [1]. This means that the computation and evaluation of Q_o can be costly. Recent research focuses on creating rewriting algorithms that generates Q_o with a smaller size [3].

In this paper, we propose a new approach to answering SPARQL queries on OWL 2 QL ontologies over existing relationally stored data. Our proposal is to replace answering queries via query rewriting with *answering queries using views*. Our position is that SQL infrastructure can be leveraged in order to support effective SPARQL query answering on OWL 2 QL ontologies over existing relationally stored data. We present preliminary results that support our position.

Our position is inspired by our previous work on Ultrawrap [5], a system that can execute SPARQL queries at almost equivalent execution speed as its semantically equivalent SQL queries. Previous experimental studies demonstrated that existing approaches were several magnitudes slower. Our main insight was to represent the relational data as RDF triples using views. SPARQL queries are syntactically translated to SQL queries which operate on the views. We observed that two relational optimizations are needed in order for relational database to effectively execute SPARQL queries: detection of unsatisfiable conditions and self-join elimination. Given this past history, we ask ourselves if we can apply this same approach to answering SPARQL queries on OWL 2 QL ontologies.

We first present the status quo with a running example adapted from [3]. We then present our proposed approach and show initial results that support our position. Finally, we present the open issues and our next steps.

[1] In this paper, we use the term OWL 2 QL ontology to refer only to a TBox. Therefore, we assume that it only contains axioms.

M. Krötzsch and U. Straccia (Eds.): RR 2012, LNCS 7497, pp. 262–265, 2012.

2 The Status Quo

The status quo of systems that answer queries over an OWL 2 QL ontology mapped to a relational database, also known as Ontology-based Data Access (OBDA), consists of the following input: an OWL 2 QL ontology O, a relational database D and an initial mapping M from the database D to the ontology O. Queries posed over the ontology O are answered in three steps:

1. *Query Rewriting:* given a conjunctive query Q, and the ontology O, compute a union of conjunctive queries Q_o, which is a rewriting of Q w.r.t O.
2. *Query Unfolding:* given the mapping M, and the rewritten query Q_o, compute a SQL query Q_{SQL}.
3. *Query Evaluation:* the SQL query Q_{SQL} is evaluated on the relational database.

OBDA systems such as Quest [4] or Mastro [2] implement these three steps. Additionally, Calvanese et al. [1] and Perez-Urbina et al. [3] present query rewriting algorithms and part of systems such as QuOnto[2], Owlgres[3] and REQUIEM[4].

Throughout this paper, we will use the following running example. Consider the following OWL 2 QL ontology O with concepts Student, Professor and Person, and the following axioms:

$$\text{Student} \sqsubseteq \text{Person}$$
$$\text{Professor} \sqsubseteq \text{Person}$$

Consider the following relational database D consisting of tables `PROF(PID,NAME)` and `STUD(SID,NAME)`, and consider a mapping M from the database D to the ontology O defined as follows using Datalog notation:

$$\text{Student}(x) \leftarrow \texttt{STUD}(x, y)$$
$$\text{Professor}(x) \leftarrow \texttt{PROF}(x, y)$$

Now assume that $Q(x)$ is the query $\text{Person}(x)$ posed over O. The first step of the methodology just described is query rewriting: given the query Q and ontology O, Q is rewritten to Q_o:

$$Q_o(x) = \text{Student}(x) \vee \text{Professor}(x)$$

The next step is query unfolding: given the mapping M and the rewritten query Q_o, generate a SQL query Q_{SQL}:

```
SELECT SID FROM STUD   UNION ALL   SELECT PID FROM PROF
```

Finally, query Q_{SQL} is sent to the RDBMS where it is evaluated. Note that `UNION ALL` is used because it does not eliminate duplicate rows.

[2] `http://www.dis.uniroma1.it/quonto/`
[3] `http://pellet.owldl.com/owlgres/`
[4] `http://www.cs.ox.ac.uk/projects/requiem/`

3 Our Proposal: Answering Queries Using Views

Our proposal is to replace answering queries via query rewriting with answering queries using views. We do this in two steps. First, we represent the mappings using SQL views, which is a union of SQL queries. We call this view the *Tripleview*. Second, we compile the axioms of the OWL 2 QL ontology into SQL queries and add them to the Tripleview. Note that this is only done once and not for each time a query is executed. With our approach, we avoid rewriting queries at run time and let the relational database do the query unfolding. In order to further explain our approach, consider the same running example: the OWL 2 QL ontology O, the relational database D and the initial mapping M shown in Section 2. First, we represent the mapping M in the Tripleview:

```
CREATE VIEW Tripleview(s, p, o) AS
SELECT SID AS s, "rdf:type" AS p, "Student" AS o FROM STUD
UNION ALL
SELECT PID AS s, "rdf:type" AS p, "Professor" AS o FROM PROF
```

The next step is to compile the OWL 2 QL ontology axioms into SQL queries. Following our example, we need to add two additional queries to the Tripleview, generating the following:

```
CREATE VIEW Tripleview(s, p, o) AS
SELECT SID AS s, "rdf:type" AS p, "Student" AS o FROM STUD
UNION ALL
SELECT PID AS s, "rdf:type" AS p, "Professor" AS o FROM PROF
UNION ALL
SELECT SID AS s, "rdf:type" AS p, "Person" AS o FROM STUD
UNION ALL
SELECT PID AS s, "rdf:type" AS p, "Person" AS o FROM PROF
```

Everything up to now is done before any query is executed. After the Tripleview is created, we can execute queries. For example, consider the query Q written in SPARQL:

```
SELECT ?x WHERE { ?x rdf:type Person }
```

This query is then syntactically translated to a SQL query on the Tripleview and evaluated directly on the RDBMS without any further processing:

```
SELECT s FROM Tripleview WHERE p = "rdf:type" AND o = "Person"
```

4 Does This Work in Practice?

To test if our approach works in practice, we created the example database and implemented the Tripleview on Microsoft SQL Server and executed the query. The resulting query plan is shown in Fig. 1.

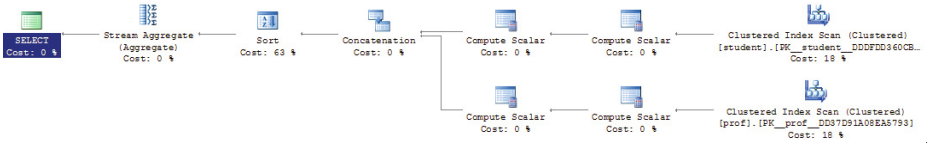

Fig. 1. The physical query plan for our running example on Microsoft SQL Server

The logical query plan consists of the Tripleview with the union of four queries. We observe that the physical query plan generated by SQL Server consists of a union of two queries. Therefore, the SQL optimizer determined which queries in the Tripleview were not going to satisfy the original query and transformed the original logical query plan into the optimal physical plan shown in Fig. 1. This transform is the *detection of unsatisfiable conditions* optimization. Additionally, we observe that query Q_{SQL} generates the same query plan.

This preliminary result supports our position that by answering queries using views, the SQL infrastructure can be leveraged to support effective SPARQL query answering on OWL 2 QL ontologies over existing relationally stored data. Notice that we are not claiming that query rewriting is not needed. On the contrary, we are trying to get the best of both worlds. For example, query rewriting algorithms, such as the ones presented in [2,3], would need to be modified to generate the views. Additionally, we need to investigate how much of OWL 2 QL can be represented in views. Our first findings show that any axioms that is not of the form $A \sqsubseteq \exists R$ (i.e. sub-class, sub-property, equivalent class, equivalent property, etc), can be represented in views. Nevertheless, the possibility of representing existential axioms of the form $A \sqsubseteq \exists R$ depends on the ability of a relational database to generate the equivalent of a blank node.

We are currently implementing this approach in Ultrawap and planning to evaluate it against other OBDA systems.

References

1. Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The dl-lite family. J. Autom. Reason. 39(3), 385–429 (2007)
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. Pérez-Urbina, H., Horrocks, I., Motik, B.: Efficient Query Answering for OWL 2. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 489–504. Springer, Heidelberg (2009)
4. Rodríguez-Muro, M., Calvanese, D.: Quest, a system for ontology based data access. In: OWLED (2012)
5. Sequeda, J.F., Miranker, D.P.: Ultrawrap: Sparql execution on relational data. Technical Report TR-12-10, The University of Texas at Austin, Department of Computer Sciences (2012)