Abstract—This paper describes the ODBA problem solution based on query rewriting techniques, introduces the DL-Lite logics for knowledge representation and the query rewriting algorithms for high-level data access. The RQR algorithm’s optimization capabilities are considered.

Keywords—ODBA; description logic; DL-Lite; query answering, query rewriting, OWL 2 QL.

I. INTRODUCTION

A conceptual interface for accessing the data stored in existing relational databases can be implemented via query rewriting techniques. Built on these techniques, the interface may be independent from DBMS and as well as from particular DB schemes[6]. The development of such interface is an actual problem of raising the abstraction level for working with data and high-level integration of information systems. The ontology representation format OWL 2 QL has been specially designed to use actual database technology for query answering via query rewriting. An efficient query rewriting algorithm QR[1] was introduced at the international OWLED-2009 workshop: it translates queries to ontologies into the queries to ordinal databases. The data complexity of RQR is no higher than P in the worst case. The additional advantage of the algorithm is that it can be used for more expressive descriptive logics – DL-Lite and higher. This paper describes the ODBA problem, introduces DL-Lite logics and query rewriting techniques, then analyses the RQR optimization capabilities.

II. ODBA PROBLEM

With conceptual modeling progress (OOP, UML) and system sophistication the need of providing an information system with a high-level interface for working with large amounts of data is appeared. Such interface may be provided if the knowledge domain is represented in an ontology description form (knowledge base). The data access problem though a high-level conceptual interface is an ontology-based data access (ODBA) [1]. The solution must satisfy the following requirements: 1) efficient query processing, which must be ideally executed with the same speed as the SQL queries over existing RDB, and 2) the query processing must use all advantages of relational technologies already used to store data.

III. ONTOLOGY-BASED KNOWLEDGE REPRESENTATION

Knowledge base, KB - is the knowledge domain description saving the relationships’ semantics between concepts. KB allows extracting data stored in a database (ABox), taking into account the constraints expressed at a higher conceptual level (TBox)[4]:

\[ KB = TBox + ABox, or K = (T, A) \]  

Where TBox (T) – terminological box – the conceptual data model, for instance, Entity-Relationship;

ABox (A) – assertional box – data set stored in a database.

An ontology can be called a particular instance of KB, represented on a formal KB description language. Description logic (DL) of a special expressivity power can be used as a knowledge representation language. The expressivity power is defined by the set of axioms allowed in TBox and ABox. On the one side, the language should be as more expressive as possible to completely describe the knowledge domain. On the other side, the reasoning problems over KB must have an acceptable computational complexity.

IV. SYNTAX AND AXIOMS OF THE DL-LITE FAMILY

The DL-Lite[1] language family is proposed for conceptual modeling in addition to UML and ER. The DL-Lite syntax:

\[ R ::= P_{k} \mid \neg P_{k}, \]  

\[ B ::= \bot \mid A_{k} \mid \exists q R, \]  

\[ C ::= B \mid \neg C \mid C_{1} \cap C_{2}, \]  

\[ \text{inv}(R) = \begin{cases} P_{k}^{+}, & \text{if } R = P_{k}, \\ P_{k}^{-}, & \text{if } R = \neg P_{k}. \end{cases} \]

TBox is a finite set of \( C_{1} \subseteq C_{2}, R_{1} \subseteq R_{2} \) - concept and role inclusion axioms.

\[ ABox \] is a finite set of \( A_{k}(a_{1}), \neg A_{k}(a_{1}), P_{k}(a_{1}, a_{2}) \) and \( \neg P_{k}(a_{1}, a_{2}) \) - assertions.

Where \( a_{1}, a_{2} \) - object name, \( A \) – concept name, \( P \) – role name, \( q \) – integer number.

Interpretation \( I \) (essentially, the particular instance of KB) is a pair if non-empty domain and an interpretation function

\[ (\Delta^{A}, \Delta^{R}) : a_{i}^{A} \in \Delta^{A}, A_{k}^{R} \subseteq \Delta^{A} \text{ and } P_{k}^{L} \subseteq \Delta^{A} \times \Delta^{A} \]

For each interpretation the unique name assumption (UNA) status is also specified. UNA affects on the computational complexity characteristics of \( I \):

\[ a_{i}^{A} \neq a_{j}^{A}, \text{ for all } i \neq j \]  

(UNA)

Languages of different expressive power are produced by restricting the set of allowed axioms. The main axioms:
\[(P_k^c)^2 = \{(y, x) \in \Delta^2 \times \Delta^2 | (x, y) \in P_k^c\}, \quad (8)\]

\[L^2 = \emptyset, \quad (9)\]

\[\geq q(R) = \{x \in \Delta^2 | y \in \Delta^2 | (x, y) \in R^2 \} \geq q, \quad (10)\]

\[-C)^2 | (C_1 \cap C_2)^2 = C_1^2 \cap C_2^2, \quad (11)\]

Where \(K\) means the cardinality of the following set.

Additional axioms reflect various relationships used in conceptual modeling:

\[\geq q(R, C)^2 = \{x \in \Delta^2 | y \in \Delta^2 | (x, y) \in R^2 \} \geq q\} \quad (13)\]

\[J \models \text{Dis}(R_1, R_2) \iff R_1^2 \cap R_2^2 = \emptyset, \quad (14)\]

\[J \models \text{Asym}(P_k) \iff P_k^2 \cap (P_k^c)^2 = \emptyset, \quad (15)\]

\[J \models \text{Sym}(P_k) \iff P_k^2 = (P_k^c)^2, \quad (16)\]

\[J \models \text{Irr}(P_k) \iff (x, x) \not\in P_k^2 \quad \text{for all} \ x \in \Delta^2, \quad (17)\]

\[J \models \text{Ref}(P_k) \iff (x, x) \in P_k^2 \quad \text{for all} \ x \in \Delta^2, \quad (18)\]

Where \(\models\) is a satisfaction relation in KB.

The common denominators of DL-Lite logics are the following – 1) it is not possible to assign particular roles only to certain concepts, that means all roles can be applied to every concept (3R,C), C = 1 ; 2) TBox axioms are only concept inclusions and cannot represent any kind of disjunctive information, for instance, that several concepts cover the whole domain.

V. MAIN PROBLEMS OF WORKING WITH KNOWLEDGE BASES

Given a KB K = (T, A) one may consider the following fundamental reasoning problems [5]:

A. Satisfiability

Check whether a model of K exists.

B. Instance checking

Given an object a and a concept C, check whether K \(\models\) C(a), or, in other words, whether a^2 \(\in\) C^2 for each J of K.

C. Query answering

Given a query q(X) and a tuple \(\bar{a}\) of objects from A, check whether K \(\models\) q(\(\bar{a}\)), or, in other words, whether \(\bar{a}\) is an answer to the q(X) query w.r.t. K.

The computational complexity of these problems depends on a number of variable and fixed input parameters. The input parameters are: the TBox size, |T|, the ABox size, |A|, the K = (T, A) size, the query q(X) size – the number of query parameters, |\(\bar{X}\)| = N.

The combined and data (by the amount of data to be processed) complexity are separately considered w.r.t. reasoning problems. The data complexity is the most important in ODBA problem context, so the TBox size is considered fixed, and the query size is negligible w.r.t. the size of ABox.

VI. EFFICIENT QUERY ANSWERING IN DL-LITE KBs

The maximal expressive language for conceptual modeling, for which the query answering complexity (data) will not exceed P, is DL – LiteHorn[1]. If UNA is accepted, then query answering in DL – LiteHorn[1](UNA) will have the least computational complexity by the amount of data - AC^0. This feature causes a very important fact:

Given a knowledge base K = (T, A) satisfying DL – LiteHorn with UNA and a conjunctive (with no disjunctions) query q(X). Then q(X) and TBox can be rewritten into a union of conjunctive SQL(q(X)) queries over ABox only, and the answer for this new query will be sound and complete[3].

Based on this fact, query rewriting allows one to obtain a knowledge base over a traditional database, as well as to work with data at the conceptual level independently from a certain database scheme, and effectively use all advantages provided by modern relational DBMS.

VII. QUERY REWRITING ALGORITHMS FOR OWL 2 QL AND HIGHER

For information systems working with large amounts of data, mostly performing the query answering problems, the W3C consortium’s proposed the OWL 2 QL standard. This standard based on less expressive, than DL – LiteHorn, the DL – Litecore subset of axioms (another designation - DL – LiteR). The complexity of all reasoning problems over DL – Litecore ontologies does not exceed polynomial. This significant restriction’s been added because the equality or inequality of objects in OWL is to be specified explicitly with no UNA (or not UNA) implicit assumption. To keep the reasoning problems’ complexity constant and UNA-independent for ontologies built in compliance with the OWL 2 QL standard, it’s been decided not to include axioms, which allow one to define function dependencies and numeral restrictions over concepts. These axioms strongly affect the reasoning complexity, which depends on the fact whether UNA or not UNA is assumed in the ontology.

Query rewriting techniques and algorithms are intensively developed for OWL 2 QL to provide mechanisms for high-level conceptual query answering over existing databases.

Currently two algorithms have been designed and implemented[2]: CGLLR and RQR.

The CGLL algorithm for DL-Lite has been implemented in several systems, such as QuOnto, Owlgres, ROWLKit. The RQR algorithm for DL-Lite+ was introduced in 2009 and implemented in REQUEST. Both algorithms, CGLL and RQR, retrieve the same results of query rewriting. During the rewriting process each algorithm produces a large number – about several thousand - UCQ (unique conjunctive query). This results in complicated SQL queries with too many unions, which can be impracticable to DBMS.

The algorithms have been tested on computers with equal configuration. The testing data included 9 ontologies of the DL – LiteR [2] expressivity level, corresponding to the OWL 2
queries and qualified

on check,

ffiffi

model.

to a particular database for each database and for each unique
rewriting
efficient on real databases
ans

algorithms, first of all, one should find out how much query
preferable

CGLLR in the worst case.

number of UCQ.

rewriting based on prerequisite mappings. One may suppose that the
query rewriting algorithm efficiency may also significantly
depend on a particular mapping representation. Currently, there are
no standards and examined formalisms to define such
mappings.

Further optimizations can be applied to RQR: forward and
backward subsumption check, query condensation and other.
Additional experiments with these optimizations are needed.
Besides, full features of OWL 2 QL (especially, data types)
must be supported in RQR, and a new series of experiments
will be required to get reliable results of checking the RQR
efficiency with the complete support for DL – LiteR .

VIII. FUTURE WORK

The experiment results demonstrate that RQR is more
preferable for query rewriting, than CGLLR[2].

For researching into practical usage aspects of these
algorithms, first of all, one should find out how much query
answering based on described query rewriting techniques is
efficient on real databases. The obvious obstacle for query
rewriting approach is the need of mapping a conceptual model
to a particular database for each database and for each unique
model. However, it is an additional abstraction layer
requirement, which is inevitable to raise the abstraction level of
data access interface.

In further experiments the testing data must include queries,
which are to be transformed into SQL queries to real databases
based on prerequisite mappings. One may suppose that the
query rewriting algorithm efficiency may also significantly
depend on a particular mapping representation. Currently, there are
no standards and examined formalisms to define such
mappings.

Further optimizations can be applied to RQR: forward and
backward subsumption check, query condensation and other.
Additional experiments with these optimizations are needed.
Besides, full features of OWL 2 QL (especially, data types)
must be supported in RQR, and a new series of experiments
will be required to get reliable results of checking the RQR
efficiency with the complete support for DL – LiteR .