Data & Knowledge Engineering 69 (2010) 545–572

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

The consistency extractor system: Answer set programs for consistent query answering in databases

Monica Caniupán [a,*], Leopoldo Bertossi [b]

[a] Universidad del Bío-Bío, Depto. Sistemas de Información, Concepción, Chile
[b] Carleton University, School of Computer Science, Ottawa, Canada

A R T I C L E I N F O

Article history:
Received 9 March 2009
Received in revised form 24 January 2010
Accepted 27 January 2010
Available online 4 February 2010

Keywords:
Databases
Integrity constraints
Inconsistency
Answer set programs

A B S T R A C T

We describe the *Consistency Extractor System* (*Cons Ex*) that computes consistent answers to Datalog queries with negation posed to relational databases that may be inconsistent with respect to certain integrity constraints. In order to solve this task, *Cons Ex* uses answers set programming. More precisely, *Cons Ex* uses disjunctive logic programs with stable models semantics to specify and reason with the *repairs*, i.e. with the consistent virtual instances that minimally depart from the original database. The consistent information is invariant under all repairs. *Cons Ex* achieves efficient query evaluation by implementing *magic sets* techniques. We describe the general methodology, its optimizations for query answering, and the architecture of the system. We also present encouraging experimental results.

1. Introduction

Integrity constraints (ICs) capture the semantics of data with respect to the external reality that the database is expected to model. Databases should satisfy their ICs, but in practice, databases may become inconsistent with respect to them [10]. Nevertheless, in most cases only a small portion of the data is inconsistent (i.e. participates in inconsistency with respect to the ICs). In consequence, an inconsistent database can still give us useful and semantically correct information. The process of characterizing and obtaining consistent answers to queries is called *Consistent Query Answering* (CQA) [12].

Consistent query answering makes sense if, to make thing worse, it becomes impossible, undesirable or too difficult to restore the consistency of the database by applying some form of materialized data cleaning. Conventional data cleaning might be a non-deterministic and expensive process, that also leads to a loss of potentially useful information. In some cases, we might actually have no permission to modify data or a clear way about how to proceed in this direction. This is the case, for example, when autonomous and independent data sources are virtually integrated.

Enforcing consistency at query time is an alternative to enforcing consistency at the instance level. This idea is applicable in, among other situations, (a) virtual data integration, (b) in the case of a single database on which, for better performance purposes, some ICs are not enforced, (c) in the materialization of a database whose content is obtained from other sources. In the latter case, the ETL process (Extract, Transform, and Load) can be seen, modeled and implemented as the materialization of a process of CQA.

Before computing consistent answers to queries, they have to be formally characterized, in precise logical terms. This was first done in [4]: Consistent answers to a query are invariant under all the forms of restoring consistency by minimal changes on the database. The alternative consistent versions of the original instance are called *repairs*. So, *consistent answers* are those that can be obtained as usual answers from every repair. The notion of consistent answer is related in spirit to the notion of

* Corresponding author.
E-mail addresses: mcaniupa@ubiobio.cl (M. Caniupán), bertossi@scs.carleton.ca (L. Bertossi).

0169-023X/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2010.01.005

certain answer as found, for example, in virtual data integration [1]: Certain answers are those that are true of all the possible legal instances for the integration system.

More precisely, given a relational database instance D and a set IC of ICs, a (minimal) repair [4] is an instance D' of the same schema, that satisfies IC, and differs from D by a minimal set of whole database tuples under set inclusion. Repairs do not have to be materialized; actually there may be too many of them [10]. In principle, they are virtual instances that are used to give a model-theoretic definition of consistent answer. Mechanism for CQA can and have to be assessed against this semantic definition.

Example 1. Consider the database schema $Student(id, name)$. The functional dependency (FD) $id \rightarrow name$ establishes that each student identifier is associated with a unique name value. The first two tuples of the following database instance D, that can be the result of the integration of two data sources, violate the FD:

Student	
id	name
1	*smith*
1	*peter*
2	*jones*

Consistency can be minimally restored by deleting either tuple $Student(1, smith)$ or tuple $Student(1, peter)$. If we delete both tuples, the resulting database is not a repair since it does not satisfy the minimality requirement. Therefore, there are two database repairs:

	Student		
id	name	id	name
1	*smith*	1	*peter*
2	*jones*	2	*jones*

We can see that certain information persists in the repairs, e.g. tuple $Student(2, jones)$ is in both of them, reflecting the fact that it does not participate in the violation of the FD. On the other hand, the "inconsistent tuples" $Student(1, smith)$ and $Student(1, peter)$ do not persist in all the repairs. If we want to know the id of student *jones*, we can pose the query $Student(x, jones)$. The answer to this query is $\langle 2 \rangle$ in both repairs, therefore the consistent answer is $\langle 2 \rangle$.

Moreover, for the boolean disjunctive query $Student(1, smith) \vee Student(1, peter)$, the consistent answer is *yes*, since each repair satisfies one of the disjuncts in the query. Notice that if we had simultaneously deleted all the tuples participating in an inconsistency, we would have lost this kind of information.

Already in [4] some computational mechanisms were presented that do not use or compute the repairs, but pose a new, rewritten query to the given inconsistent database. The answers to the new query are the consistent answers to the original query. Cf. [12] for a survey containing more recent results of this kind.

The algorithm for CQA proposed in [4] implemented and slightly extended in [25] is applicable to limited classes of queries and ICs, e.g. projection-free conjunctive queries, functional dependencies, full inclusion dependencies. In these cases, the first-order (FO) query can be rewritten into a new FO query that posed and answered as usual to given instance, obtains the consistent answers to the original query. Other FO query rewriting methods for CQA were presented in [27,39,46]. However, they are still limited in their applicability, which is due to the intrinsic higher data complexity of CQA. Cf. [10,12] for surveys in this direction, and [60] for more recent results about non-FO rewritability of CQA. The on-the-fly, at query time, resolution of inconsistencies is what makes the FO rewriting for CQA difficult or impossible. This is in contrast to, for example, querying databases through DL-Lite ontologies by FO query rewriting [20], where the ontology basically extends the underlying database without being in logical conflict with it.

As a consequence, languages for query rewriting than are more expressive than FO Logic became necessary. Actually, they first and naturally emerged when logic programs were used to specify the repairs, with the idea of query this compact specification of repairs in order to obtain the consistent answers. In several papers [5,8,9,15,16,32,43], database repairs were specified as the stable models of disjunctive logic programs with stable model semantics [40] (aka. *answer set programs*). The logic programs that specify the repairs are called *repair programs*.

The logic programs introduced in [16] are most general and take into consideration the possible occurrences of null values in the databases. Furthermore, they capture the use of null values for restoring consistency with respect to referential ICs (RICs). Actually, in [9,16] it was shown that there is a one-to-one correspondence between the stable models of the repair program and the repairs with respect to *RIC*-acyclic sets of ICs, i.e. sets of constraints that do not present cycles involving RICs (cf. Section 2).