Information Systems 37 (2012) 677–711

Contents lists available at SciVerse ScienceDirect

Information Systems

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

Core schema mappings: Scalable core computations in data exchange ☆

Giansalvatore Mecca [a,*], Paolo Papotti [b], Salvatore Raunich [c]

[a] Dipartimento di Matematica e Informatica, Università della Basilicata, viale dell'Ateneo Lucano 10, I-85100 Potenza, Italy
[b] Università Roma Tre, Roma, Italy
[c] University of Leipzig, Leipzig, Germany

ARTICLE INFO

Article history:
Received 7 March 2011
Received in revised form
22 March 2012
Accepted 26 March 2012
Recommended by: L. Wong
Available online 3 April 2012

Keywords:
Schema mappings
Data exchange
Core computation

ABSTRACT

Research has investigated mappings among data sources under two perspectives. On the one side, there are studies of practical tools for schema mapping generation; these focus on algorithms to generate mappings based on visual specifications provided by users. On the other side, we have theoretical researches about data exchange. These study how to generate a solution – i.e., a target instance – given a set of mappings usually specified as tuple generating dependencies. Since the notion of a core solution has been formally identified as an optimal solution, it is very important to efficiently support core computations in mapping systems. In this paper, we introduce several new algorithms that contribute to bridge the gap between the practice of mapping generation and the theory of data exchange. We show how, given a mapping scenario, it is possible to generate an executable script that computes core solutions for the corresponding data exchange problem. The algorithms have been implemented and tested using common runtime engines to show that they guarantee very good performances, orders of magnitudes better than those of known algorithms that compute the core as a post-processing step.

1. Introduction

Integrating data coming from disparate sources is a crucial task in many applications. An essential requirement of any data integration task is that of manipulating *mappings* between sources. Mappings are executable transformations – say, SQL or XQuery scripts – that specify how an instance of the source repository should be translated into an instance of the target repository. We may identify two broad research lines in the literature.

On the one side, we have studies on practical tools and algorithms for *schema mapping generation*. In this case, the focus is on the development of systems that take as input an abstract specification of the mapping, usually made of a bunch of correspondences between the two schemas, and generate the mappings and the executable scripts needed to perform the translation. This research topic was largely inspired by the seminal papers about the Clio system [26,27]. The original algorithm has been subsequently extended in several ways [17,5,2,29,7] and various tools have been proposed to support users in the mapping generation process. More recently, a benchmark has been developed [1] to compare research mapping systems and commercial ones.

On the other side, we have theoretical studies about *data exchange*. Several years after the development of the initial Clio algorithm, researchers have realized that

☆ Portions of this paper have appeared under the title *Core Schema Mappings* in the Proceedings of the ACM SIGMOD 2009 Conference.

* Corresponding author. Tel. +39 0971 205855; fax. +39 0971 205897.
E-mail addresses: giansalvatore.mecca@gmail.com, giansalvatore.mecca@unibas.it (G. Mecca).

0306-4379/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2012.03.004

a more solid theoretical foundation was needed in order to consolidate the practical results obtained on schema mapping systems. This consideration has motivated a rich body of research in which the notion of a *data exchange problem* [12] was formalized, and a number of theoretical results were established. In this context, a *data exchange setting* is a collection of mappings – usually specified as *tuple generating dependencies* (*tgds*) [4] – that are given as part of the input; therefore, the focus is not on the generation of the mappings, but rather on the characterization of their properties. This has brought to an elegant formalization of the notion of a solution for a data exchange problem, and of operators that manipulate mappings in order, for example, to compose [14] or invert [11,3] them.

However, for a long time, these two research lines have progressed in a rather independent way. To give a clear example of this, consider the fact that there are many possible solutions for a data exchange problem. A natural question is the following: "which solution should be materialized by a mapping system?" A key contribution of data exchange research was the formalization of the notion of *core* [13] universal solution, which was identified as the "optimal" solution for a data exchange scenario. Informally speaking, the core universal solution has a number of nice properties: it is "irredundant", since it is the smallest among the solutions that preserve the semantics of the exchange, and it represents a "good" instance for answering conjunctive queries over the target database. It can therefore be considered a natural requirement for a schema mapping system to generate executable scripts that materialize core solutions.

Unfortunately, there is yet no schema mapping generation algorithm that natively produces executable scripts that compute the core. On the contrary, the solution produced by known schema mapping systems – called a *canonical solution* – typically contains quite a lot of redundancy. This is partly due to the fact that computing cores is a challenging task.

A possible approach to the generation of core solutions for a relational data exchange problem is the following: (i) first, to generate a canonical solution by chasing the source-to-target tgds; to do this, a mapping system typically generates an SQL or XQuery script that performs this step very efficiently, even on large source instances; (ii) then, to apply a post-processing algorithm for core identification.

Several polynomial algorithms have been identified to this end [13,18]. These algorithms provide a very general answer to the problem of computing core solutions for a data exchange setting. Also, an implementation of the core-computation algorithm in [18] has been developed [30] by using a combination of SQL for database access and a controlled form of recursive control-logic implemented in Java.

Although polynomial, experience with these algorithms shows that they hardly scale to large mapping scenarios. In fact, they exhaustively look for endomorphisms inside the canonical universal solution in order to identify which null values and which tuples can be removed. This kind of computation can take very high computing times, even on databases of a few thousand tuples, as shown in our experiments.

This paper makes several important contributions towards the goal of making the computation of core solutions a scalable functionality of mapping systems. More specifically:

- given a mapping scenario consisting of source-to-target tgds, we introduce a rewriting algorithm that generates a new set of dependencies that can be used to generate core solutions for the original tgds; these dependencies can be translated into an SQL script and ran inside any conventional database engine, thus achieving a very high degree of flexibility and performance;
- the algorithm has been implemented into the +Spicy mapping system; in the paper, we conduct an experimental evaluation on large mapping scenario that confirms the scalability of our solution;
- the rewriting algorithm is based on a new characterization of the core, in terms of *witness blocks* and *expansions*; we introduce these notions and show how they represent a natural tool for the rewriting of the given tgds.

The algorithms developed in this paper concentrate on mapping scenarios made of source-to-target tgds only. However, as it will be discussed in Section 2.1, they represent an essential building block for more general algorithms that handle larger classes of constraints.

2. Overview

Consider the mapping scenario informally described in Fig. 1, where also a source instance is shown. The source database contains tables about books coming from three different data sources, namely the *Internet Book Database* (*IBD*), the *Library of Congress* database (*LOC*), and the *Internet Book List* (*IBL*).

The desired mapping can be expressed using the following set of *tuple-generating dependencies (tgds)*:

$$m_1.\forall t,p : LOC(t,p) \rightarrow \exists I : Book(t,I) \wedge Publisher(I,p)$$

$$m_2.\forall t,id : IBLBook(t,id) \rightarrow Book(t,id)$$

$$m_3.\forall id,p : IBLPublisher(id,p) \rightarrow Publisher(id,p)$$

$$m_4.\forall t : IBDBook(t) \rightarrow \exists N : Book(t,N)$$

It can be seen how each source has a slightly different organization wrt the others. In particular, the *IBD* source contains data about book titles only; mapping m_4 copies titles to the *Book* table in the target. The *LOC* source contains book titles and publisher names in a single table; these are copied to the target tables by mapping m_1, which also "invents" a value to correlate the key and the foreign key. Finally, the *IBL* source contains data about books and their publishers in separate tables; these data are copied to the target by mappings m_2,m_3; note that in this case we do not need to invent any values.