Project-Join-Repair: An Approach to Consistent Query Answering Under Functional Dependencies

Jef Wijsen

Université de Mons-Hainaut, Mons, Belgium
jef.wijsen@umh.ac.be
http://staff.umh.ac.be/Wijsen.Jef/

Abstract. *Consistent query answering* is the term commonly used for the problem of answering queries on databases that violate certain integrity constraints. We address this problem for universal relations that are inconsistent with respect to a set of functional dependencies. In order to obtain more meaningful repairs, we apply a project-join dependency prior to repairing by tuple deletion. A positive result is that the additional project-join can yield tractability of consistent query answering.

1 Motivation

In the Internet age, data are ubiquitous and cannot be expected to be globally consistent. The database systems in which such data are stored and queried, should be capable of handling this *inconsistency* phenomenon. A way to deal with the problem is to rectify the database before proceeding to queries. Since there is usually no single best way to solve an inconsistency, we will generally end up with a set of possible *database repairs*. Such set of possible databases defines an *incomplete database*, a concept that has been studied for a long time [1]. When a query is asked on an incomplete database, the *certain query answer* is defined as the intersection of the answers to the query on every possible database. The motivation for intersecting query answers should be clear: although we do not know which database is the "right" one, we do know that it will (at least) return this certain answer. If the incomplete database is made up of repairs, the certain answer has also been called the *consistent answer*.

Repairing with respect to functional and key dependencies is commonly done by tuple deletion (inserting new tuples will not take away the inconsistency anyway). In this article, we propose a novel approach, termed "project-join-repair," for repairing a universal relation subject to a set of functional dependencies (fd's). We next motivate the approach by three examples.

Example 1. The first row in the following relation states that on January 7, twenty units of product P1 have been shipped to customer C1 called A. Jones. The constraints are $\mathsf{Cid} \rightarrow \mathsf{CName}$ (the same identifier cannot be used for different customers) and, since quantities are daily totals per product and client, $\{\mathsf{Date}, \mathsf{Pid}, \mathsf{Cid}\} \rightarrow \mathsf{Qty}$. The first fd is violated, because C1 appears with two

H. Legind Larsen et al. (Eds.): FQAS 2006, LNAI 4027, pp. 1–12, 2006.

different names. We could repair this relation by deleting either the first or the second tuple. However, it may be more meaningful to assume that names are mistaken, and that "A. Johnson" should read "A. Jones," or *vice versa*.

I	Date	Pid	Qty	Cid	CName
	7 Jan	P1	20	C1	A. Jones
	8 Feb	P2	15	C1	A. Johnson

We propose a novel way to make the intended rectification. First, we apply the join dependency $\bowtie[\{\mathsf{Date}, \mathsf{Pid}, \mathsf{Qty}, \mathsf{Cid}\}, \{\mathsf{Cid}, \mathsf{CName}\}]$, that is, we take the join of the projections on $\{\mathsf{Date}, \mathsf{Pid}, \mathsf{Qty}, \mathsf{Cid}\}$ and $\{\mathsf{Cid}, \mathsf{CName}\}$. This join dependency (jd) corresponds to a lossless-join decomposition in third normal form (3NF). The lossless-join property means that applying the jd will not insert new tuples into consistent relations. However, since one fd is violated in our example, the join contains two new tuples (followed by $*$).

$\pi_{\mathsf{Date},\mathsf{Pid},\mathsf{Qty},\mathsf{Cid}}(I)$

Date	Pid	Qty	Cid
7 Jan	P1	20	C1
8 Feb	P2	15	C1

$\pi_{\mathsf{Cid},\mathsf{CName}}(I)$

Cid	CName
C1	A. Jones
C1	A. Johnson

$\pi_{\mathsf{Date},\mathsf{Pid},\mathsf{Qty},\mathsf{Cid}}(I) \bowtie \pi_{\mathsf{Cid},\mathsf{CName}}(I)$

Date	Pid	Qty	Cid	CName	
7 Jan	P1	20	C1	A. Jones	
8 Feb	P2	15	C1	A. Johnson	
7 Jan	P1	20	C1	A. Johnson	$(*)$
8 Feb	P2	15	C1	A. Jones	$(*)$

Next, we take as repairs the maximal (under set inclusion) consistent subsets of the join relation. This gives us the two intended repairs:

J_1	Date	Pid	Qty	Cid	CName
	7 Jan	P1	20	C1	A. Jones
	8 Feb	P2	15	C1	A. Jones

J_2	Date	Pid	Qty	Cid	CName
	8 Feb	P2	15	C1	A. Johnson
	7 Jan	P1	20	C1	A. Johnson

Example 2. The following relation I is subject to the functional dependencies $\mathsf{Name} \to \{\mathsf{Birth}, \mathsf{Sex}, \mathsf{ZIP}\}$ and $\mathsf{ZIP} \to \mathsf{City}$. The latter fd is violated.

I	Name	Birth	Sex	ZIP	City
	An	1964	F	7000	Mons
	Ed	1962	M	7000	Bergen

Again, instead of deleting either tuple, it is reasonable to assume that Mons should be Bergen, or *vice versa*.[1] We can obtain this effect by first applying the jd $\bowtie[\{\mathsf{Name}, \mathsf{Birth}, \mathsf{Sex}, \mathsf{ZIP}\}, \{\mathsf{ZIP}, \mathsf{City}\}]$ and then repair by tuple deletion. Here are the two repairs that result from this project-join-repair scenario:

J_1	Name	Birth	Sex	ZIP	City
	An	1964	F	7000	Mons
	Ed	1962	M	7000	Mons

J_2	Name	Birth	Sex	ZIP	City
	An	1964	F	7000	Bergen
	Ed	1962	M	7000	Bergen

Again, the intended rectifications have been made.

[1] Bergen is actually the Dutch name for the city of Mons situated in the French speaking part of Belgium.