On using Simplification and Correction Tables for Integrity Maintenance in Integrated Databases

Christiansen, Henning; Martinenghi, Davide

Published in:
Proceedings of LAAIC'06, Second International Workshop on Logical Aspects and Applications of Integrity Constraints, Included in Proc. DEXA 2006, Seventeenth Int'l Workshop on Database and Expert Systems Applications

Publication date:
2006

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Christiansen, H., & Martinenghi, D. (2006). On using Simplification and Correction Tables for Integrity Maintenance in Integrated Databases. In *Proceedings of LAAIC'06, Second International Workshop on Logical Aspects and Applications of Integrity Constraints, Included in Proc. DEXA 2006, Seventeenth Int'l Workshop on Database and Expert Systems Applications* (pp. 569-573). IEEE Computer Society Press.

On using simplification and correction tables for integrity maintenance in integrated databases

Henning Christiansen
Roskilde University
Computer Science Dept.
P.O.Box 260, DK-4000 Roskilde, Denmark
henning@ruc.dk

Davide Martinenghi
Free University of Bozen/Bolzano
Faculty of Computer Science
P.zza Domenicani, 3, I-39100 Bolzano, Italy
martinenghi@inf.unibz.it

Abstract

When a database is defined as views over autonomous sources, inconsistencies with respect to global integrity constraints are to be expected. This paper investigates the possibility of using simplification techniques for integrity constraints in order to maintain, in an incremental way, a correction table of virtual updates which, if executed, would restore consistency; access can be made through auxiliary views that take the table into account. The approach employs assumptions about local source consistency as well as cross-source constraints whenever possible.

1 Introduction

We study the problem of maintaining consistency of global integrity constraints in a data integration (DI) system, i.e., a system providing a reconciled and unified view of a set of possibly distributed and heterogeneous data sources. Our basic assumption is that sources are autonomous in the sense that the integrating systems, henceforth referred to as the *global* level, cannot update sources; the global level can ask queries to and receive answers from each source, and we assume, furthermore, that each source maintains its own local integrity constraints and provides information about the updates that it has experienced.

Under the assumption that all integrity constraints (ICs) at global as well as local levels are correct properties of the world, a violation of a global IC means that at least one of the sources has wrong (or lacking) information. To maintain global consistency, it seems natural to introduce a correction table (CT) of virtual updates which, if executed, would restore consistency; access can be made through auxiliary views that take the table into account (transformation of global queries into a set of source query followed by a composition step is a well-studied topic so we leave this aside).

Simplification of ICs is a principle introduced in the early 1980es for single databases [10] and which means to optimize the checking of ICs based on the assumption that the database was consistent before the update. This implies an incremental checking and maintenance of ICs so that (ideally) only a minimal number of tuples potentially affected by each update are checked. In the version of [10] and successor works, the check is performed after the update so that problematic updates need to be undone. This has been improved in later work by checking before each proposed update and thus problematic ones can be rejected and inconsistency effectively avoided. There are several problems involved in simplification which have hindered for a widespread application; see [9] for a survey. Our own recent work [6, 8] uncovers theoretical limitations and proposes an effective method that produce simplified checks for a very general class of updates; such updates can be parameterized so that the potentially complex symbolic transformations involved can be performed at database design time with actual parameters plugged into test queries at runtime. Typically, this method removes an order of magnitude or more. The principle has been applied to detect inconsistency in DI by [5] and in the present work we extend this with a correction table which makes it possible to maintain a virtual view of consistency. Similar tables have been used by [7], but the optimization by simplification seems new. Other approaches to consistency maintenance in DI, e.g., [3, 4, 12], modify source relations and do not incrementally trace changes. Instead of maintaining (virtual) consistency, approaches to consistent query-answering [1, 2] return only tuples that the different repairs of the database agree on.

In the following we introduce correction tables and consider their application for consistency maintenance using simplification. The results are still preliminary in the sense that no proofs are given and that no practical implementations have been tested.

2 Databases and data integration systems

We adopt DATALOG with default negation [11] referring to *clauses*, including *facts* (ground clauses whose body is empty, understood as *true*) for the tuples of a relation, denials (clauses whose head is empty, understood as false) for *integrity constraints* (ICs), and *rules* (all other clauses) to define views. An *integrity theory* is a satisfiable set of ICs. Built-in predicates including $=$ and $\neq$ are assumed with their standard meaning. Clauses are assumed to be *range restricted* meaning that any variable in a clause appears (also) in some positive database literal in the body of that clause. For example, the IC $\leftarrow p(X) \wedge \neg q(X)$ is range restricted as variable X appears in the positive literal $p(X)$. Only nonrecursive databases are considered, so a standard, least Herbrand model semantics can be used; $D \models \phi$ means that a formula ϕ holds in the standard model of database D.

Definition 2.1 *A* database $D = \langle F, \Delta, \Gamma \rangle$ *consists of a set of facts* F *(the* state*) and two constraint theories,* Δ *called the set of* trusted constraints *with* $F \models \Delta$*, and* Γ *referred to as the* ICs *of* D*;* $\Delta \cup \Gamma$ *is satisfiable. Database* D *is* consistent *whenever* $F \models \Gamma$ *and* inconsistent *otherwise. We may use* $D \models \phi$ *to indicate* $F \models \phi$*.*

This definition captures a broad collection of information systems, e.g., single databases in which Δ is enforced as hard constraints, and Γ are soft ones which are desired properties not always obeyed. The main focus in this paper is data integration systems:

- Each source relation is represented as a relation in D.
- Δ collects *local ICs* for the different sources, assumed to be enforced by separate mechanisms at the local level, plus *cross-source constraints*.
- The mapping from states of the local databases to a global database state is made using a mapping (e.g., view definitions) in such a way that *global integrity constraints* can have their predicates unfolded using that mapping so that we get integrity constraints Γ which are denials over the source relations.

As an example of a cross-source constraint, we may have that the domain of two attributes in different sources are disjoint. Global integrity constraints may easily be violated as sources are autonomous, only taking care of their own, local integrity constraints.

Example 2.1 *Consider two sources with predicate* m_1 *and* m_2*, and a global predicate* $m_0(husband, wife)$ *describing marriages and defined by a mapping specified by the view* $m_0(x, y) \leftarrow m_1(X, Y) \vee m_2(X, Y)$*. On all databases an integrity constraint is imposed enforcing non-bigamism of husbands:* $\Gamma_i = \{\leftarrow m_i(X, Y) \wedge m_i(X, Z) \wedge Y \neq Z\}$*, for* $i = 0, 1, 2$*. The trusted constraints are* $\Delta = \{\Gamma_1, \Gamma_2\}$*, and the unfolding of* Γ_0 *with respect to the mapping is*

$$\begin{aligned}\Gamma = \{ \quad & \leftarrow m_1(X, Y) \wedge m_1(X, Z) \wedge Y \neq Z, \\ & \leftarrow m_1(X, Y) \wedge m_2(X, Z) \wedge Y \neq Z, \\ & \leftarrow m_2(X, Y) \wedge m_1(X, Z) \wedge Y \neq Z, \\ & \leftarrow m_2(X, Y) \wedge m_2(X, Z) \wedge Y \neq Z \ \}\end{aligned}$$

The informally specified translation of DI systems into databases (in the sense of def. 2.1) implies some restrictions on the relationship between global and local databases. In a global-as-view setting, views defining the global database must be nonrecursive and certain chains of negations must have even length; see [6, 8] for details including generalization with quantifiers so that referential integrity can be expressed by ICs such as $\leftarrow p(X) \wedge \neg \exists Y q(X, Y)$.

In order to monitor and eventually maintain consistency, it is assumed that each source informs the global database about each local update in terms of which tuples that have been added and deleted. Thus if an update U indicates the addition of an atom A, this implies that A was not in the database before the update; similarly for deletions. An updates is indicated as a set of ground literals: the positive ones indicate addition, the negative ones deletion. For any literal A, we let $\neg\neg A$ indicate A, and for a set of literals U, $\neg U$ denotes $\{\neg L \mid L \in U\}$. Application and composition of updates are defined using the following notation.

Definition 2.2 *Let* U *and* V *be sets of literals; the* composition $U \circ V$ *is defined as* $(U \cup V) \setminus \{L | \{L, \neg L\} \subseteq U \cup V\}$.

Definition 2.3 *Let* $D = \langle F, \Delta, \Gamma \rangle$ *be a database. An* update U *(for* D*) is a set of ground source literals such that* $L \in U$ *implies* $F \not\models L$*. The updated database is defined as* $D^U = \langle F \circ U, \Delta, \Gamma \rangle$*; it is assumed that* $D^U \models \Delta$.

As mentioned in the introduction, our proposal for providing a consistent view of an inconsistent database or DI system is to introduced a globally maintained table of virtual correction, which we conceive as an update which, if it were executed, would restore consistency.

Definition 2.4 *A* correction table (CT) *for a database* $D = \langle F, \Delta, \Gamma \rangle$ *is an update* R *to* D *so that* $D^{\mathsf{R}} \models \Gamma \cup \Delta$*;* R *is* minimal *when there is no other* $\mathsf{R}' \subset \mathsf{R}$ *which is a CT for* D.

Trivially, for a consistent database D and an update U for which D^U is inconsistent, $\neg U$ is a CT (although not necessarily a minimal one).

In the remainder of this paper we study how such tables can be constructed and maintained (i.e., given an existing table and an update, how to provide a new table) supported by simplification techniques. As will be noticed below, it is not always a minimal table which is the desired one.

3 Simplification

Here we review the framework of [6, 8]. Central to this approach is a syntactic replacement operator, called $\mathsf{After}^U(\cdot)$, which transforms a constraint theory Φ into another Ψ that evaluates in any state D in the same way as Φ would evaluate in an updated state D^U. Basically, the operator unfolds each atom according to a view that incorporates the update and subsequently performs trivial truth preserving transformations as to keep the specific syntax of denials. We skip the definition in favour of an example.

Example 3.1 *Consider constraint theory Γ_1 from example 2.1 and update $U = \{m_1(a, b)\}$. We have*

$$\begin{aligned}\mathsf{After}^U(\Gamma_1) = \{&\leftarrow m_1(X,Y) \wedge m_1(X,Z) \wedge Y \neq Z,\\ &\leftarrow m_1(X,Y) \wedge X = a \wedge Z = b \wedge Y \neq Z,\\ &\leftarrow X = a \wedge Y = b \wedge m_1(X,Z) \wedge Y \neq Z,\\ &\leftarrow X = a \wedge Y = b \wedge X = a \wedge Z = b \wedge Y \neq Z\}.\end{aligned}$$

Proposition 3.1 *Let D be a database, U an update, and Φ a constraint theory. Then $D \models \mathsf{After}^U(\Phi)$ iff $D^U \models \Phi$.*

The result of After corresponds to what in Hoare's logic is called a weakest precondition. Notice that a shift of notation leads to a useful alternative formulation,

$$D^U \models \mathsf{After}^{\neg U}(\Phi) \quad \text{iff} \quad D \models \Phi. \tag{1}$$

Simplification involves an optimization phase which can employ (e.g.) the hypothesis that the state before the update is consistent. We indicate with $\mathsf{Optimize}_\Theta$ an optimization operator which employs that a constraint theory Θ holds in a given state (also referred to as "background knowledge"). Any such optimization operator must satisfy the following soundness and idempotency conditions.

$$\text{For any } D \text{ with } D \models \Theta,\ D \models \mathsf{Optimize}_\Theta(\Phi) \text{ iff } D \models \Phi$$

$$\mathsf{Optimize}_\Theta(\mathsf{Optimize}_\Theta(\Phi)) = \mathsf{Optimize}_\Theta(\Phi)$$

An Optimize operator is described in [6, 8] in terms of a nontrivial combination of different proof procedures that remove, from the input constraint theory, all denials and literals that can be proved redundant. In the end one obtains an output constraint theory with a minimized number of literals, which in most cases is a good approximation of the "most efficient" theory to evaluate. These details are not central to the present paper, and we will simply assume that there exists an optimization procedure, referred to as Optimize, having the above properties.

One important application of simplification in a traditional, single database is to produce optimal tests that can check whether a given update will destroy ICs Γ on the background that Γ holds in the present state prior to the update. Such a test is produced as $\mathsf{Optimize}_\Gamma(\mathsf{After}^U(\Gamma))$ which can be tested *before* the update, and if it fails, the update is not even performed and expensive rollback operations to restore consistency are unnecessary.

Example 3.2 *For Γ_1 and U defined in examples 2.1 and 3.1, the optimized condition is calculated as $\{\leftarrow m_1(a, Y) \wedge Y \neq b\}$, which indicates that for a database D to keep consistency at source 1 after update U, husband a must not already be married to a wife different from b.*

The simplification process itself can be very complex, and [6, 8] describe an extension for parameterized update patterns so that simplification can be done at database design time, and actual data values are plugged into the tests at runtime. (You may think of a and b in the example as parameters, to be replaced by, say, "John" and "Mary" when update m_1("John", "Mary") is encountered.

4 Integrity maintenance in DI systems

The situation for data DI is different, as the sources act independently and the global database is informed about an update *after* it is has taken place at the sources: simplified constraints are needed which apply in the updated state. Suppose as a first simplified case that global consistency of a database $D = \langle F, \Delta, \Gamma\rangle$ was given before an update U. An optimal test for consistency in the updated state can be produced as follows, with the background knowledge of all formulas known to hold in D^U; notice that we apply principle (1) to get useful formulas.

$$\mathsf{Optimize}_{\mathsf{After}^{\neg U}(\Gamma)\cup\Delta\cup U\cup\mathsf{After}^{\neg U}(\Delta)}(\Gamma) \tag{2}$$

Example 4.1 *Consider the database of example 2.1 and update $U = \{m_1(a, b)\}$. The task is now to find an optimal test which, under the given conditions, is equivalent to Γ_0. Expression (2) above gives $\{\leftarrow m_2(a, Z) \wedge b \neq Z\}$. Clearly, this denial is much simpler to evaluate than the original Γ, and it is difficult imagine another denial satisfying the same semantics requirements which executes faster.*

Adding the cross-source constraint that the domains of husbands in either source are disjoint ($\leftarrow m_1(X, Y) \wedge m_2(X, Y)$) makes formula (2) amount to true, i.e., under this assumption, U cannot introduce global inconsistency.

In the general case, we cannot expect the integrated database to be consistent even before a particular update, but we maintain a CT (def. 2.4) in an incremental way in order to provide a consistent of the database. The current CT is in itself useful for providing background knowledge about the current database state. Assume a database $D^U = \langle F, \Delta, \Gamma\rangle$ following an update U and let R' be a CT for the state D prior to the update. The following set Θ collects properties about the current state D^U.

$$\Theta = \mathsf{After}^{\neg U\circ\mathsf{R}'}(\Gamma)\cup\Delta\cup U\cup\mathsf{After}^{\neg U}(\Delta)\cup\mathsf{After}^{\neg U\circ\mathsf{R}'}(\Delta)$$

Notice that we used After backwards to simulate reversal of the update followed by application of the table R' with the effect of simulating a consistent database. The trusted constraints Δ are utilized for all states available physically or by "simulation". The following property shows how simplification can be used for characterizing an updated CT.

Proposition 4.1 *Let $D = \langle F, \Delta, \Gamma\rangle$ be a database, U an update, and R' a CT for D. Then R is a CT for D^U iff*

$$D^U \models \mathsf{Optimize}_{\Theta}(\mathsf{After}^{\mathsf{R}}(\Gamma \cup \Delta))$$

where Θ is a set of formulas with $D^U \models \Theta$; this holds especially for the value of Θ indicated above.

Given a proposal for a CT R, the proposition provides an optimized check for whether R is in fact a CT. The task is now to provide effective ways of producing candidate CTs.

It should be noticed that we need just *some* CT R' to apply the proposition; it need not be minimal and we can ignore the complexity given by that fact that there exponentially many different (minimal) tables. [4]

5 Maintenance of correction tables

In the following we characterize CTs for different cases in ways which may be used for effective implementation. In general, we cannot rely on evaluation of simplified ICs only, which we can indicate by a simple example. Assume IC $\leftarrow p(X) \wedge q(X)$ and update $\{p(a)\}$ to a consistent state; the simplified check is thus $\leftarrow q(a)$. If the simplified test fails, an inspection of it indicates the CT $\{\neg q(a)\}$. However, the simplification removed (as part of its job) all references to the atom in the update (since it is obviously true in the updated state), so looking at the simplified ICs only is prone for missing corrections that neutralize part of the recent update, e.g., $\{\neg p(a)\}$ in the present example. On the other hand, inspection of the instance of the original IC, $\leftarrow p(a) \wedge q(a)$, gives both results immediately.

Definition 5.1 *Let D be a database and Σ a constraint theory. Let* $\mathsf{Ground}(\Sigma)$ *be the set of all ground instances of denials in Σ;* $\mathsf{Falsifiers}(D, \Sigma)$ *is defined as* $\{\sigma \in \mathsf{Ground}(\Sigma) | D \not\models \sigma\}$.

We define $\mathsf{PotentialCorrections}(D, \Sigma)$ *as* $\{\neg L \mid (\leftarrow \cdots \wedge L \wedge \cdots) \in \mathsf{Falsifiers}(D, \Sigma)\}$*; however, if* $(\leftarrow) \in \Sigma$, $\mathsf{PotentialCorrections}(D, \Sigma) = \bot$. *A* potential correction set *for Σ in D is a subset $C \subseteq \mathsf{PotentialCorrections}(D, \Sigma)$ such that each denial of* $\mathsf{Falsifiers}(D, \Sigma)$ *contains some literal M with $\neg M \in C$; C is* minimal *if no subset of it is a potential correction set for Σ.*

The sets defined above are finite since every component is finite and Σ is range restricted; they can be found by posing Σ as a query to D followed by straightforward processing of the result. In the following we consider an updated database $D^U = \langle F, \Delta, \Gamma\rangle$ and a given CT R' for D; Θ refers to the background theory described in the previous section.

Case 1: Consistently signed IC. We say that D is *consistently signed* whenever the same predicate cannot occur positively as well as negatively in $\Delta \cup \Gamma$. Intuitively, this means that correcting one failed IC instance $\leftarrow \cdots A \cdots$ by adding $\neg A$ to the CT cannot create other failing IC instance. The following characterization relies on After and Optimize preserving the consistent sign property.

Claim: Any minimal CT for a consistently signed database is a subset of $\neg U \cup \mathsf{PotentialCorrections}(D^U, \Sigma)$ where $\Sigma = \mathsf{Optimize}_{\Theta}(\Gamma)$.

When $\neg U \cup \mathsf{PotentialCorrections}(D^U, \Sigma)$ is of manageable size it is feasible to generate a minimal CT by trying out different subsets in a systematic manner. If preference is given to the most recent information given by U, a CT can be found by first selecting a minimal potential correction set for Σ, and if this is not sufficient, we begin testing which elements of U needs to be virtually deleted. The involved calculation can be performed using only simplified ICs, the original ICs are not evaluated.

Case 2: Case 1 extended with data verification. Assume that there is a way to check database facts against the real world, e.g., by an infallible human expert. Our idea is to maintain a CT that contains only verified information and that the expert should be asked as little as possible. This has two implications. Firstly, minimality is not relevant and secondly, the table should keep verified information still after a source update has made it redundant. This means that the table does not conform with our definition of an update, but we avoid this problem by keeping the redundant data in a separate table. What we need in this case is to produce a new CT by extending the existing R'.

If $D^U \models \mathsf{Optimize}_{\Theta}(\mathsf{After}^{\mathsf{R}'}(\Gamma))$ nothing is necessary, otherwise we may construct firstly a minimal potential correction set for $\mathsf{Optimize}_{\Theta}(\mathsf{After}^{\mathsf{R}'}(\Gamma))$ by asking the expert which facts should be virtually added or deleted. If that is not sufficient, we can consult the expert concerning the elements of U one by one until a consistent view is gained. In any non-construed example we expect this procedure to involve only very few steps; only simplified ICs are checked.

Case 3: General ICs. Dropping the consistent sign assumption means that an attempt to correct one failed IC instance may generate other failing instances which in turn may need to be corrected; notice that we also need to monitor possible effects on the trusted constraints. This indicates that the integrity check needs to be repeated, including calculation of potential correction sets. The following nondeterministic algorithm calculates all possible minimal CTs (when followed by a straightforward thinning process).

$\mathsf{R} := \emptyset$
repeat
  $\Sigma := \mathsf{Optimize}^{\Theta}(\mathsf{After}^{\mathsf{R}}(\Gamma \cup \Delta))$
  if $D^U \models \Sigma$, **return**(R), **otherwise**
  $C := \mathsf{PotentialCorrections}(D^U, \Sigma) \cup U \setminus \mathsf{R} \setminus \neg\mathsf{R}$
  if $C = \emptyset$ or $C = \bot$, **abort**, **otherwise**
  $\mathsf{R} := \mathsf{R} \cup \{L\}$ for some $L \in C$

In most cases, we expect the number of iterations to be small, and if adapted for data verification, the cases where one correction produces new violations will be rare.

Example 5.1 *In our running example, assume a correction table $CT = \emptyset$ and the update $m_1(a, b)$. In case $m_2(a, c)$ is indicated by second source, the algorithm produces two different new minimal CTs, $\{m_1(a, b)\}$ and $\{m_2(a, c)\}$.*

Assuming instead an initial $CT = \{m_2(a, b)\}$ we may expect this table be output unchanged as the sole minimal CT for the updated DI system.

6 Discussion

The runtime applications of simplification is a potentially high computational cost in the algorithms indicated above which may outweigh the reduction in orders of magnitude compared with checking of unsimplified ICs.

The exact complexity of simplification is currently not known, but it is independent of the database size, being a function of the size of the ICs and the updates. However, the use of parameters [6, 8] may make it possible to unfold (or partially evaluate) the algorithm into a decision tree where choices are made from queries posed to the source.

Comparing with approaches to consistent answering from in an inconsistent DI system (e.g., [4]), we notice that the use of a CT is less pessimistic in that it may provide more answers to a query. The assumption of a human expert who is always online may be a bit problematic, but a combination of consistent answering and a CT maintained manually through a simplification-based interface, can make it possible to collect problematic updates over time and only involve the expert now and then.

Acknowledgement: The first author is supported by the CONTROL project, funded by Danish Natural Science Research Council.

References

[1] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

[2] Leopoldo E. Bertossi and Jan Chomicki. Query answering in inconsistent databases. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer, 2003.

[3] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[4] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In Leopoldo E. Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2005.

[5] Henning Christiansen and Davide Martinenghi. Simplification of integrity constraints for data integration. In Dietmar Seipel and Jose Maria Turull Torres, editors, *Foundations of Information and Knowledge Systems, Third International Symposium (FoIKS 2004)*, volume 2942 of *Lecture Notes in Computer Science*, pages 31–48. Springer, 2004.

[6] Henning Christiansen and Davide Martinenghi. On simplification of database integrity constraints. *Fundamenta Informaticae*, 71:1–47, 2006.

[7] Suzanne M. Embury, Sue M. Brandt, John S. Robinson, Iain Sutherland, Frank A. Bisby, W. A. Gray, Andrew C. Jones, and Richard J. White. Adapting integrity enforcement techniques for data reconciliation. *Inf. Syst.*, 26(8):657–689, 2001.

[8] Davide Martinenghi. *Advanced Techniques for Efficient Data Integrity Checking*. PhD thesis, Roskilde University, Denmark, in *Datalogiske Skrifter*, 105, `http://www.ruc.dk/dat/forskning/skrifter/DS105.pdf`, 2005.

[9] Davide Martinenghi, Henning Christiansen, and Hendrik Decker. Integrity checking and maintenance in relational and deductive databases – and beyond. In *Intelligent Databases: Technologies and Applications*. Idea Group Inc., 2006. To appear.

[10] Jean-Marie Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.

[11] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I & II*. Computer Science Press, 1988/89.

[12] Jef Wijsen. On condensing database repairs obtained by tuple deletions. In *DEXA Workshops*, pages 849–853. IEEE Computer Society, 2005.