

Can Integrity Tolerate Inconsistency?

Hendrik Decker¹ and Davide Martinenghi²

¹Instituto Tecnológico de Informática ²Free University of Bozen/Bolzano
Ciudad Politécnica de la Innovación Faculty of Computer Science
Campus de Vera, edificio 8G Piazza Domenicani, 3
E-46071 Valencia, Spain I-39100 Bolzano, Italy
`hendrik@iti.es` `martinenghi@inf.unibz.it`

Abstract. An unconditional and hitherto unquestioned basic requirement for integrity checking is that the data need to be consistent before the update, such that the success of a simplified test can guarantee the invariance of integrity after the update. We answer the question whether this consistency requirement can be relaxed with "Yes, at least sometimes, and to some extent."

1 Introduction

Virtually all known methods for simplifying the task of integrity checking require that integrity be satisfied in the old state, before the update. We have asked ourselves why that requirement would need to be so strict, and if it couldn't be relaxed to some extent. We found out that, for at least for two well-known methods in the literature, that requirement can indeed be relaxed, and quite significantly so. However, an even more general relaxation turned out to be impossible.

We have looked into the statements and proofs of the correctness results of the integrity checking method by Nicolas [14] as well as subsequent methods in order to see where exactly this requirement of consistency of the old state with its integrity constraints was needed or could not be given up. In this paper, we reconstruct that investigation and outline our findings.

2 Abstract Characterization of Simplification Methods

In this section, we provide an apparatus of definitions that capture the notion of simplification of integrity constraints in abstract terms. We adopt the terminology of deductive databases [1, 16], where a *database* consists of a set of *facts* and a set of *rules* (*tuples* and *views* in the relational model). An *integrity constraint* represents a semantic condition, expressed, e.g., as a closed first-order formula, which is supposed to always hold in the database. An *integrity theory* is a finite set of integrity constraints. Formulas are evaluated relative to a database; for simplicity, we refer to databases that have a unique standard model, such as stratified databases.

We write $D \models W$ (resp., $D \not\models W$), where W is a closed formula, to indicate that W evaluates to *true* (resp., *false*) in D 's standard model; a similar notation is used for an integrity theory, evaluated as the conjunction of its elements. We also say that a database is *consistent* if its associated integrity theory evaluates to *true* in it.

A database *update* U can be regarded as a mapping $U : \mathcal{D} \mapsto \mathcal{D}$, where \mathcal{D} is the space of databases; we indicate the image of D via update U as D^U . Here, we restrict our attention to updates that only involve facts.

Now, the integrity checking problem asks, given an integrity theory Γ , a database D consistent with Γ , and an update U , whether $D^U \models \Gamma$ holds, i.e., whether the new database still is consistent. Since evaluating Γ in D^U may be too time consuming a task, the problem is reformulated so as to take advantage of incrementality of updates and the fact that D is consistent and, therefore, possibly only a subset of D needs to be checked. It is customary to look for an alternative integrity theory \mathcal{T} (called a *simplification*), which is supposed to be simpler to evaluate than Γ , while yielding equivalent results. If \mathcal{T} is an integrity theory to be evaluated in D^U , it is called a post-test; if in D , then it is called a pre-test.

Definition 1. *Let Γ be an integrity theory and U an update. An integrity theory \mathcal{T} is a*

- post-test of Γ for U whenever $D^U \models \Gamma$ iff $D^U \models \mathcal{T}$
- pre-test of Γ for U whenever $D^U \models \Gamma$ iff $D \models \mathcal{T}$

for every database D consistent with Γ .

Clearly, Γ itself is a post-test of Γ for any update, but, as mentioned, one is interested in post-tests that are actually “simpler” than the original Γ . This is possible by exploiting the fact that the old state D is consistent with Γ to avoid redundant checks of cases that are already known to satisfy integrity.

Integrity checking strategies based on post-tests, such as [14, 7, 11], consist in: executing the update, checking the post-test and, if it fails to hold, correcting the database by performing a repair action (e.g., a rollback).

With pre-tests, one tries to predict consistency of the updated state without actually executing the update. Besides consistency of the old state D with Γ , one also exploits the fact that inconsistent states can be avoided without executing the update, and, thus, without ever needing to undo a violated new state. Integrity checking with pre-tests, as in [15, 5], consists in: checking whether the pre-test holds and, if so, executing the update.

Both post- and pre-tests are referred to as *simplifications* of the original integrity theory.

In order to characterize the degree of simplification of a pre- or post-test, we exhibit reference pre- and post-tests, called *plain* tests, corresponding to somewhat non-simplified conditions that do not exploit the fact that the old state satisfies integrity,

Definition 2 (Plain test). *Let Γ be an integrity theory and U an update.*

a) An integrity theory Σ_0 is a plain pre-test of Γ for U , denoted by $\text{pre}_0^U(\Gamma)$, if the following holds: $D \models \Sigma_0$ iff $D^U \models \Gamma$ for every database D .

b) An integrity theory \mathcal{T}_0 is a plain post-test of Γ for U , denoted by $\text{post}_0^U(\Gamma)$, if the following holds: $D^U \models \mathcal{T}_0$ iff $D^U \models \Gamma$ for every database D .

Clearly, Γ is a plain post-test of itself for any update. Any pre-test (resp., post-test) can be considered as a proper simplification if it is simpler (for whatever criterion of simplicity¹) to evaluate than the corresponding plain test.

Due to space limitation, we refrain from giving a formal account on how to achieve simplification of integrity constraints and refer to surveys on the subject, e.g., [13]. Rather, we conclude this section with an example of simplification, often mentioned in the literature, showing the gains of a specialized approach to integrity checking.

Example 1. Consider a database with the relation $m(H, W)$ (husband H married to wife W), and an integrity constraint preventing husbands' bigamy, expressed as $\Gamma = \{\leftarrow m(X, Y) \wedge m(X, Z) \wedge Y \neq Z\}$. Let U be the an update that inserts the marriage $m(a, b)$ into the database. A simplification of Γ for U (equivalent to what Nicolas' method would output) is $\Sigma = \{\leftarrow m(a, Y) \wedge Y \neq b\}$, i.e., to ensure consistency of the updated state one only needs to check that a is not already married to a wife different from b . This is much easier to execute than Γ , as the the space of tuples to be considered is greatly reduced by virtue of the instantiation of variables with constants.

3 Towards Inconsistency Tolerance

In some contexts, certain violations of integrity constraints may be considered acceptable or even unavoidable, e.g., in distributed or federated systems or when data come from unverified sources. In practice, inconsistencies are more of a rule than an exception in daily experience with DBMSs. Methods that cope with the presence of inconsistencies, commonly referred to as *paraconsistent*, are therefore highly in need.

We now discuss to which extent well-known techniques for integrity checking, that have been designed under the unrealistic premise that all data need to be consistent, can in fact be used also in the presence of inconsistency. Their application, in fact, will gradually improve compliance of data with the integrity constraints.

Any pre- or post-test-based simplification of integrity constraints is called inconsistency-tolerant if it guarantees preservation of satisfaction of constraints that were satisfied before the update. To this end, we introduce the notion of case, to formalize the thought that integrity typically is violated not as a whole, but only by cases; satisfied cases of constraints can be separated from violated cases.

¹ Typically, such criteria correspond to some notion of syntactic minimality, such as the number of literals in an integrity theory or the checking space of the queries associated with the integrity constraints; see [5] for discussion.

Definition 3 (Global variable, Case). Let $W = \forall x_1 \dots \forall x_q (W')$ ($q \geq 0$) be an integrity constraint in prenex normal form.

a) Each variable x in W that is \forall -quantified but not dominated by any \exists quantifier (i.e., \exists does not occur left of the quantifier of x in W) is called a global variable of W .

b) The formula $W'\zeta$ is called a case of W if ζ is a substitution such that $\text{Range}(\zeta) \subseteq \{x_1, \dots, x_q\}$ and $\text{Image}(\zeta) \cap \{x_1, \dots, x_q\} = \emptyset$.

Cases of an integrity constraint need not be ground, and each constraint W as well as each variant of W is a case of W .

Now we can define inconsistency tolerance of simplified integrity checking.

Definition 4 (Inconsistency tolerance). Let Γ be an integrity theory, U an update and ϕ a case of a constraint in Γ . A pre-test Σ (resp., post-test Υ) of Γ for U is inconsistency-tolerant whenever $D^U \models \phi$ if $D \models \Sigma$ (resp., $D^U \models \phi$ if $D^U \models \Upsilon$) for all databases D consistent with ϕ .

A simplification method is inconsistency-tolerant if, for any input integrity theory Γ and update U , its output is an inconsistency-tolerant pre-test (or post-test) of Γ for U .

We first note that inconsistency tolerance is not enjoyed in general, which immediately limits our ambition of possibly characterizing all conceivable simplification methods as inconsistency-tolerant. Take, e.g., $D = \{p(a)\}$, $\Gamma = \{\leftarrow p(X)\}$, $\phi = \leftarrow p(b)$ and let U be the update such that $D^U = \{p(a), p(b)\}$. We have $D \models \phi$, $D^U \not\models \phi$ as required. Integrity theory $\Upsilon = \exists X(p(X) \wedge X \neq b)$ is a post-test of Γ for U , since, whenever Γ holds, Υ evaluates to *false*, which it should, since U always invalidates Γ . With a similar argument we can conclude that $\Sigma = \exists X p(X)$ is a pre-test of Γ for U . We have $D \models \Sigma$ and $D^U \models \Upsilon$, although $D \models \phi$ and $D^U \not\models \phi$.

Yet, inconsistency tolerance is enjoyed by plain pre-tests and post-tests.

Proposition 1. Let Γ be an integrity theory, U an update. Any $\text{pre}_0^U(\Gamma)$ (resp., $\text{post}_0^U(\Gamma)$) is an inconsistency-tolerant pre-test (resp., post-test) of Γ for U .

3.1 Relaxing Nicolas' simplification theorem

In [14] and most publications on the same subject that came after it, an unconditional premise for the correctness of simplified integrity checking has been that the constraints to be checked for a given update U are supposed to be satisfied in the "old" state, i.e., the database state given when U is requested. Otherwise, correctness of simplification is not guaranteed.

The basic idea of Nicolas' method is that a simplified form of the set of integrity constraints imposed on the database is obtained from a given update and the current state of the database. Thus, integrity, which is supposed to be an invariant of all possible database states, is checked upon each update request, which in turn becomes effective if the check yields that integrity is not violated. Here, "simplified" essentially means more efficiently evaluated at update time.

Somewhat more formally, the essence of the approach in [14] can be summarized as follows.

For a database D , an integrity constraint W in prenex conjunctive normal form and a tuple r to be inserted into some relational table R , Nicolas' simplification method automatically generates a simplification $\Gamma_R^+ = W\gamma_1 \wedge \dots \wedge W\gamma_m$, $m \geq 0$, where the γ_i are substitutions obtained from unifiers of r and m different occurrences of negated atoms in W that unify with r . More precisely, the γ_i are restrictions of mgu's of R with negated atoms in W , restricted to the variables that are \forall -quantified and not dominated by an \exists quantifier in W . For convenience, let us call such variables in W *global variables*. (Arguments are symmetric for deletions.)

Essentially, the main theorem in [14], as recapitulated below, states that if W is known to hold in the old state D , then W also holds in the new, updated state D^U iff Γ_R^+ holds in D^U .

Theorem 1. [14] *Let D be a database, W an integrity constraint, U the insertion of a tuple r , into a relation R , and Γ_R^+ the simplification of W . If W is true in D , then:*

W is true in D^U iff Γ_R^+ is true in D^U .

First, we note that the only-if half of this theorem is in fact uninteresting because simplified integrity checking does not want to evaluate all of W in D^U , but only its simplified form Γ_R^+ , for ensuring that W remains satisfied in the new state. Hence, we are going to focus on the if-half, from now on.

Obviously, the requirement that the integrity constraint W be satisfied in the old state D (i.e., W is true in D) is very central in the theorem above. Now, we want to relax this premise. To simply drop it would not leave anything to look at. The intuition of what we are after is that there are only some violated instances W (which can be found when evaluating all of W , not just a simplified form, against D), while all of the "rest" of W is satisfied. And it is this big rest that we are interested in, hoping that the given update will not introduce more violated instances.

Now, we can look formally at cases of W that are satisfied before the update. In fact, it is possible to show by tracing Nicolas' proof line by line that such cases of W will remain satisfied after the update if the simplified form of W evaluates to true.

Theorem 2. *Let D be a database, W be an integrity constraint, r a tuple to be inserted, and Γ_R^+ the simplification of W wrt r generated by Nicolas' method. Let $W\zeta$ be a case of W such that $W\zeta$ holds in D . Then, $W\zeta$ also holds in D^U if Γ_R^+ holds in D^U .*

3.2 Inconsistency Tolerance of Other Methods

Reconstructing the proof of Nicolas' simplification theorem in an inconsistency-tolerance perspective was possible, since the notion of case, albeit hidden, is

already present in the original proof, which exhibits sets of substitutions to be applied to the integrity constraints.

The main simplification theorem in [12] and also its proof are formal generalizations of [14]. And in fact it is possible to show that also the method in [12] is inconsistency-tolerant, in the same sense as [14].

However, similar relaxations are much harder for other methods, and certainly leave space for future work, although we conjecture that virtually all main approaches to simplification enjoy inconsistency tolerance. For example, in [5], simplification is based on notions such as provability and query containment that can be regarded as replaceable off-the-shelf tools. The method of [5] aims to produce a pre-test expressed as a set of denials, starting from a plain pre-test, and minimizing it in the number of literals and denials. It is very easy to show that removing a literal from a denial makes the denial logically stronger (i.e., it entails the denial with the literal). As a consequence, if the denial was inconsistency tolerant, so is the denial with a literal less. However, removing a denial from an integrity theory makes the theory logically weaker and, thus, it cannot be concluded that inconsistency tolerance is preserved. In fact, one can even build counter-examples showing that, in principle, inconsistency tolerance is not preserved, based on the declarative definitions of plain pre-test and minimization thereof. However, the method of [5] procedurally selects one *specific* plain pre-test (different from the one used to build a counter-example), and we conjecture that this is the key to guarantee inconsistency tolerance of the method.

4 Related work

Simplification has been recognized by a large body of research as a necessary technique for optimization of integrity checking. For each specific update or update pattern, a specialized integrity theory can be derived that guarantees consistency of the updated state, provided that the old state is also consistent. We have discussed approaches based on post-tests [14, 11] and pre-tests [9, 15, 5]. More generally, integrity checking can be regarded as a special case of materialized view maintenance: integrity constraints are defined as views that must always remain empty for the database to be consistent [8, 6].

Approaches based on logic programming such as [17] do not take into account irrelevant clauses for refuting denial constraints. Unlike classical logic, such approaches do not exhibit any explosive behavior in the presence of inconsistency. In this respect, they may well be characterized as inconsistency-tolerant or “paraconsistent” in a procedural sense, as done in [10]. However, to the best of our knowledge, the declarative inconsistency tolerance of simplifications for improving integrity checking, as investigated in this paper, has never been studied nor even defined beforehand. We actually reckon that all the mentioned approaches can be reconsidered in terms of this declarative understanding of inconsistency tolerance.

The related problems of restoring integrity once inconsistencies are detected (tackled since [2] with the notion of *repair*) and of using active rules for much

the same purpose [4, 3], certainly give way to inconsistency tolerance, but cannot be directly used to detect inconsistencies for integrity checking purposes.

5 Conclusion

Intuitively, it seems unrealistic to assume that integrity in databases is always completely satisfied. This, however, is exactly the premise for virtually all known approaches to integrity checking. More precisely, all correctness results in the literature about more efficient ways to check integrity by evaluating simplifications of integrity constraints upon each update fundamentally rely on the requirement that integrity must be satisfied in the old state, i.e., before the update is executed.

The unease about this intuitive conflict has motivated us to have a closer look into some well-known methods for integrity checking. We wanted to see if the consistency requirement could be relaxed somehow. On the basis of the notion of a “case” of an integrity constraint, it turned out to be fairly straightforward to generalize the main results of [14] and [12], including the original proofs.

We also observe that all of the performance gains obtained by such simplification methods are inherited by their inconsistency-tolerant counterparts. The practical significance of these results is further demonstrated by the fact that, as experiments have shown, simplified integrity constraints execute faster than the original constraints and their execution time does not depend on the number of inconsistencies in the database. Besides, by applying an inconsistency-tolerant integrity checking method, the percentage of the data in a relational database that participate in inconsistencies will necessarily decrease in the new state if the update consists only of insertions, since the number of inconsistent cases cannot grow, while the total number of tuples increases. Although the same conclusion cannot be drawn for deletions, if deletions are distributed proportionally over consistent and inconsistent cases, in the long run, integrity will improve also in percentage. This is especially advantageous for the federation of databases, where, initially, there will be a fair amount of inconsistency (think, e.g., of a business taking over a former competitor). Inconsistency-tolerant integrity checking will automatically help making the database consistent over time, by reducing the percentage of inconsistency.

Future work will investigate in which sense also other methods for simplified integrity checking are inconsistency-tolerant, such as [5] and [17]. For the latter, it will probably be necessary to introduce a more procedural notion of inconsistency tolerance, because the simplification of Sadri and Kowalski is obtained “on the fly”, by running a resolution proof procedure. We also intend to investigate the feasibility of implementing inconsistency-tolerant integrity checking in replicated databases.

Acknowledgements Hendrik Decker acknowledges support from the Spanish government research grant TIC2003-09420-CO2. Davide Martinenghi is partly supported by the TONES IST project financed within the European Union 6th Framework Programme under contract number FP6-7603.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. M. Arenas, L. E. Bertossi, and J. Chomicki. Querying database repairs using logic programs with exceptions. In *FQAS 2000*, pages 27–41. Physica-Verlag Heidelberg New York, A Springer-Verlag Company, 2000.
3. S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Automatic generation of production rules for integrity maintenance. *ACM Transactions on Database Systems (TODS)*, 19(3):367–422, 1994.
4. S. Ceri and J. Widom. Deriving production rules for constraint maintainance. In D. McLeod, R. Sacks-Davis, and H.-J. Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 566–577. Morgan Kaufmann, 1990.
5. H. Christiansen and D. Martinenghi. On simplification of database integrity constraints. *Fundamenta Informaticae*, 71(4):371–417, 2006.
6. G. Dong and J. Su. Incremental Maintenance of Recursive Views Using Relational Calculus/SQL. *SIGMOD Record*, 29(1):44–51, 2000.
7. J. Grant and J. Minker. Integrity constraints in knowledge based systems. In H. Adeli, editor, *Knowledge Engineering Vol II, Applications*, pages 1–25. McGraw-Hill, 1990.
8. A. Gupta and I. S. Mumick, editors. *Materialized views: techniques, implementations, and applications*. MIT Press, 1999.
9. A. Hsu and T. Imielinski. Integrity checking for multiple updates. In S. B. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985*, pages 152–168. ACM Press, 1985.
10. R. A. Kowalski. *Logic for Problem Solving*. Elsevier, 1979.
11. M. Leuschel and D. de Schreye. Creating specialised integrity checks through partial evaluation of meta-interpreters. *Journal of Logic Programming*, 36(2):149–193, 1998.
12. J. W. Lloyd, L. Sonenberg, and R. W. Topor. Integrity constraint checking in stratified databases. *Journal of Logic Programming*, 4(4):331–343, 1987.
13. D. Martinenghi, H. Chtistiansen, and H. Decker. Integrity checking and maintenance in relational and deductive databases, and beyond. In Z. Ma, editor, *Intelligent Databases: Technologies and Applications*, page to appear. Idea Group Publishing, 2006.
14. J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.
15. X. Qian. An effective method for integrity constraint simplification. In *Proceedings of the Fourth International Conference on Data Engineering, February 1-5, 1988, Los Angeles, California, USA*, pages 338–345. IEEE Computer Society, 1988.
16. K. Ramamohanarao and J. Harland. An introduction to deductive database languages and systems. *VLDB Journal*, 3(2):107–122, 1994.
17. F. Sadri and R. Kowalski. A theorem-proving approach to database integrity. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 313–362. Morgan Kaufmann, Los Altos, CA, 1988.