

# On the difference between checking integrity constraints before or after updates

Davide Martinenghi

Politecnico di Milano  
Piazza Leonardo da Vinci 32  
20133 Milano, Italy  
`davide.martinenghi@polimi.it`

## Abstract

Integrity checking is a crucial issue, as databases change their instance all the time and therefore need to be checked continuously and rapidly. Decades of research have produced a plethora of methods for checking integrity constraints of a database in an incremental manner. However, not much has been said about *when* to check integrity. In this paper, we study the differences and similarities between checking integrity before an update (a.k.a. pre-test) or after (a.k.a. post-test) in order to assess the respective convenience and properties.

## 1 Introduction

Integrity checking has been a perennial topic in almost all database conferences, journals and research labs. Very large quantities of research activities and publications testify to the importance of the issue. The motivation which has stimulated these investigations is that integrity checking is practically unfeasible for significant amounts of stored data without a dedicated approach to optimize the process. Progress has been made with extensions of basic approaches to deductive, object-relational, XML-based, distributed and other kinds of advanced database technology, as surveyed in [66]. However, what has not changed much are the fundamental ideas that are already present in the seminal paper [76].

The basic idea is that, in most cases, a simplified form of the set of integrity constraints imposed on the database can be obtained from a given update (or just an update schema) and the current state of the database (or just the database schema). 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. Of course, efficiency is not unequivocally measurable. However, the number of stored facts

to be retrieved for constraint evaluation is a good rule of thumb. Another such rule is the number of literals in the simplified form. Also, the minimization (reduction or avoidance) of repair costs after having detected integrity violation is a factor to be considered when assessing the efficiency of integrity checking. To establish the new state is usually considered less costly than to undo it, but for concurrent transactions, more so in distributed and even more so in replicated databases, establishing the new state is a sizable and non-negligible cost factor (think of the concurrency control, management and communication rounds needed for distribution and different replication strategies), while roll-backs are less of a problem since they can be taken care of and optimized by the DBMS, making use of its transaction logs.

Integrity checking methods may differ in some of the assumptions. For instance, in [76] and in most of the publications on the same subject that came after it, a categorical premise for the correctness of the simplification approach 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. This assumption has been relaxed in [69], thereby introducing the notion of inconsistency-tolerant integrity checking. Other different assumptions may regard the class of integrity constraints and updates that are supported by the methods, as surveyed in [66].

Unlike previous surveys, in this paper we study the main differences between approaches that require to check integrity after all updates have been applied and those that check integrity before the updates. We shall see that these checks are generally non-interchangeable, unless the updates are of a specific kind.

## 2 Preliminaries

In this section, we recapitulate, classify and discuss the main characteristics of the simplification approach to integrity checking. We fix basic definitions and terminology, and thus the framework into which our formalizations in the remainder are cast. In Section 3 we introduce abstract notions of various classes of simplifications and discuss them in Section 4.

Throughout, we refer to the relational framework of deductive databases, i.e., relational databases with possibly recursive view definitions described in clause form [1, 79]. Thus, a *database* consists of a set of *facts* and a set of *rules*, i.e., *tuples* and *views*, in the terminology of the relational model.

An *integrity constraint* expresses a semantic invariant, i.e., a condition supposed to hold in each state of the database. In general, it can be expressed by any closed first-order logic formula in the language of the database on which it is imposed. Two kinds of normal form representations of integrity constraints which both incur no loss of generality are prominent in the literature: prenex normal form and denial form. The former, as defined and used, e.g., in [76, 56], has all quantifiers moved leftmost and all negation symbols moved innermost, by equivalent rewritings of the original formula. The latter, as defined and used, e.g., in [81, 53], has the form of datalog clauses with empty head, expressing

that, if their condition is satisfied, then integrity is violated, and may need dedicated view definitions to define these conditions by recurring on database facts. An *integrity theory* is a finite set of integrity constraints, to be thought of as being imposed on some database.

For simplicity, we limit ourselves to databases that have a unique standard model and no “unknown” facts (i.e., each fact is either true or false in the model), e.g., stratified databases, and assume that database semantics is defined by this model.

For a closed formula  $W$ , we write  $D \models W$  (resp.,  $D \not\models W$ ) to indicate that  $W$  evaluates to *true* (resp., *false*) in  $D$ ’s standard model. For a set of formulas  $\Gamma$ , we write  $D \models \Gamma$  (resp.,  $D \not\models \Gamma$ ) to indicate that for every (resp., some) formula  $W \in \Gamma$  we have  $D \models W$  (resp.,  $D \not\models W$ ). If  $W$  is an integrity constraint and  $\Gamma$  an integrity theory, it is usual to also say that  $D$  *satisfies* (resp., *violates*)  $W$  and  $\Gamma$ , respectively. An equivalent terminology is defined as follows.

**Definition 2.1 (Consistency)** *A database  $D$  is consistent with an integrity theory  $\Gamma$  iff  $D \models \Gamma$ .*

Informally, we have already spoken of database states. More formally, database states are determined by atomically executed updates. An *update*  $U$  is a mapping  $U : \mathcal{D} \mapsto \mathcal{D}$ , where  $\mathcal{D}$  is the space of databases as determined by a fixed, sufficiently rich language which need not be extended by any update. For simplicity, we only consider updates that may involve facts and rules in this paper, but no integrity constraints, which are thought of as immutable. For convenience, for any database  $D$ , let  $D^U$  denote the new database obtained by applying update  $U$  on  $D$ .

### 3 Kinds of simplifications

Traditionally, the integrity checking problem asks, given a set of integrity constraints  $\Gamma$ , a database  $D$  consistent with  $\Gamma$ , and an update  $U$ , whether  $D^U \models \Gamma$  holds, i.e., whether the new database is consistent with (i.e., satisfies) the integrity constraints. However, evaluating  $\Gamma$  in  $D^U$  may be prohibitively expensive. So, a reformulation of the problem is called for, trying to take advantage of the incrementality of updates. Traditionally, all such reformulations have been made under the assumption that the old state is consistent.

We will discuss two kinds of such reformulations that have commonly been dealt with in the literature. Both determine an alternative integrity theory  $\Upsilon$  (which by itself is later called a simplification), the evaluation of which is supposed to be simpler than to evaluate  $\Gamma$ , while yielding equivalent results. The first kind of such  $\Upsilon$  is determined to be evaluated in the new state.

**Definition 3.1 (Post-test)** *Let  $\Gamma$  be an integrity theory and  $U$  an update. An integrity theory  $\Upsilon$  is a post-test of  $\Gamma$  for  $U$  whenever  $D^U \models \Gamma$  iff  $D^U \models \Upsilon$  for every database  $D$  consistent with  $\Gamma$ .*

Clearly,  $\Gamma$  itself is a post-test of  $\Gamma$  for any update. As indicated, one is interested in producing a post-test that is actually “simpler” to evaluate than the original integrity constraints. This is traditionally achieved by exploiting the fact that the old state  $D$  is consistent with  $\Gamma$ , thus avoiding redundant checks of cases that are already known to satisfy integrity. Note that the process of integrity checking involving a post-test consists in: executing the update, checking the post-test and, if it fails to hold, correcting the database by performing a repair action, i.e., a rollback and optionally a modification of the update which won’t violate integrity. Well-known post-test-based approaches are described in [76, 57, 46, 77, 55].

The second kind of approach to deal with integrity checking incrementally is to determine an integrity theory  $\Sigma$  to be evaluated in the old state, i.e., to predict without actually executing the update whether the new, updated state will be consistent with the integrity constraints.

**Definition 3.2 (Pre-test)** *Let  $\Gamma$  be an integrity theory and  $U$  an update. An integrity theory  $\Sigma$  is a pre-test of  $\Gamma$  for  $U$  whenever  $D^U \models \Gamma$  iff  $D \models \Sigma$  for every database  $D$  consistent with  $\Gamma$ .*

Here, not only the consistency of the old state  $D$  with  $\Gamma$  is exploited, but also that inconsistent states can be prevented without executing the update, and, thus, without ever having to undo a violated new state. The integrity checking process involving a pre-test is therefore: check whether the pre-test holds and, if it does, execute the update. Examples of pre-test-based approaches are [78, 25]. Note that, depending on the requirements of availability and consistency of a given application, integrity checking with pre-tests is possibly better suited for concurrent transaction processing, particularly in distributed databases.

In the remainder, we refer to both post- and pre-tests as *simplifications* of the original integrity theory. It is tacitly assumed that given simplifications are, at least in significant classes of cases, indeed simpler to evaluate than the original constraints. A characterization of simplicity beyond what is mentioned in the introduction, e.g., by formal cost models, is out of the scope of this paper; cf. [61] for a discussion.

Now, whatever the definition of simplicity, we cannot directly compare the evaluation cost of a post-test with that of a pre-test since they refer to two different (viz. old and new) states. Therefore, it is desirable to have kindred reference pre- and post-tests for benchmarking given simplifications. Plain tests, as defined below, i.e., simplifications that do not exploit the fact that the old state satisfies integrity, may serve as such reference tests.

**Definition 3.3 (Plain test)** *Let  $\Gamma$  be an integrity theory and  $U$  an update.*

a) *An integrity theory  $\Sigma_0$  is a plain pre-test of  $\Gamma$  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  $\Upsilon_0$  is a plain post-test of  $\Gamma$  for  $U$ , denoted by  $\text{post}_0^U(\Gamma)$ , if the following holds:  $D^U \models \Upsilon_0$  iff  $D^U \models \Gamma$  for every database  $D$ .*

Clearly,  $\Gamma$  is a plain post-test of itself for any update. Note that each plain test is also a simplification. For any pre-test (resp., post-test), it is therefore

desirable that it be at least as simple to evaluate as the corresponding plain test.

It is straightforward to see that, for fixed  $\Gamma$  and  $U$ , all plain pre-tests of  $\Gamma$  for  $U$  are logically equivalent.

**Proposition 3.4** *Let  $\Gamma$  be an integrity theory and  $U$  an update. Then, for any two plain pre-tests  $\Sigma'$  and  $\Sigma''$  of  $\Gamma$  for  $U$ , we have  $\Sigma' \equiv \Sigma''$ .*

**Proof** *By applying definition 3.3 to  $\Sigma'$  and  $\Sigma''$ , one gets by transitivity that  $D \models \Sigma' \text{ iff } D \models \Sigma''$  for every  $D$ , i.e.,  $\Sigma' \equiv \Sigma''$ .  $\square$*

Conversely, not all plain post-tests are logically equivalent. As a counterexample, take, e.g.,  $\Gamma = \{\leftarrow p(a)\}$  and let  $U$  be the insertion of  $p(a)$ . Then  $\Upsilon = \text{false}$  is a post-test of  $\Gamma$  for  $U$  but  $\Upsilon \not\equiv \Gamma$ .

We conclude this section with an example of simplification of integrity constraints.

**Example 3.1** *Consider a database with the relations  $\text{rev}(S, R)$  (submission  $S$  assigned to reviewer  $R$ ),  $\text{sub}(S, A)$  (submission  $S$  authored by  $A$ ) and  $\text{pub}(P, A)$  (publication  $P$  authored by  $A$ ). Assume a policy establishing that no one can review a paper of his/her (former) coauthors. This is expressed by:*

$$\Gamma = \{ \leftarrow \text{rev}(S, R) \wedge \text{sub}(S, R), \\ \leftarrow \text{rev}(S, R) \wedge \text{sub}(S, A) \wedge \text{pub}(P, R) \wedge \text{pub}(P, A) \}$$

*Let  $U$  be the an update that inserts the facts  $\text{sub}(c, a)$  and  $\text{rev}(c, b)$  into the database, where  $a, b, c$  are some constants. A simplification of  $\Gamma$  for  $U$  (equivalent to what Nicolas' method would output) is as follows:*

$$\Sigma = \{ \leftarrow \text{sub}(c, b), \\ \leftarrow \text{rev}(c, a), \\ \leftarrow \text{pub}(P, b) \wedge \text{pub}(P, a), \\ \leftarrow \text{sub}(c, A) \wedge \text{pub}(P, b) \wedge \text{pub}(P, A), \\ \leftarrow \text{rev}(c, R) \wedge \text{pub}(P, R) \wedge \text{pub}(P, a) \}$$

*The simplified conditions given by  $\Sigma$  can be read as follows:*

- *b did not submit c*
- *a does not review c*
- *b is not coauthor of a*
- *b is not coauthor of an author of c*
- *c is not reviewed by a coauthor of a*

*These checks are much easier to execute than  $\Gamma$ , as they greatly reduce the space of tuples to be considered by virtue of the instantiation of variables with constants.*

## 4 Relationship between pre- and post-tests

In this section we compare pre-tests and post-tests and discuss their interchangeability. Note that we do this without referring to any specific simplification method or update language.

First, we show that, in general, a pre-test cannot be used as a post-test, nor vice versa.

**Example 4.1** Consider the integrity theory  $\Gamma = \{\leftarrow p(a) \wedge q(b)\}$  and an update  $U$  that exchanges the contents of  $p$  and  $q$ . Then  $\Sigma = \{\leftarrow q(a) \wedge p(b)\}$  is a pre-test but clearly not a correct post-test. Consider, e.g., a database  $D = \{p(a), p(b), q(a)\}$ ; we have  $D \models \Gamma, D^U \not\models \Gamma, D^U \models \Sigma$ , i.e., it does not hold that  $D^U \models \Sigma$  iff  $D^U \models \Gamma$ , although  $D$  is consistent with  $\Gamma$ . Similarly,  $\Upsilon = \{\leftarrow p(a) \wedge q(b)\}$  is a post-test, but not a pre-test, of  $\Gamma$  for  $U$ .

This result is not surprising, since we allow for updates representing any kind of transformation of the database, such as swapping the contents of two relations. We now introduce a class of updates that excludes an update such as  $U$  of example 4.1.

**Definition 4.1** An update  $U$  is idempotent if  $D^U = (D^U)^U$  for any database  $D$ .

Idempotent updates capture additions, deletions and changes of specific tuples, which are certainly among the most frequent kinds of updates. For idempotent updates, we can prove that a plain pre-test is also always a valid plain post-test.

**Proposition 4.2** Let  $\Gamma$  be an integrity theory and  $U$  an idempotent update. Then  $D^U \models \text{pre}_0^U(\Gamma)$  iff  $D^U \models \Gamma$  for any database  $D$ , i.e.,  $\text{pre}_0^U(\Gamma)$  is a plain post-test of  $\Gamma$  for  $U$ .

**Proof** Since  $U$  is idempotent, i.e.,  $D^U = (D^U)^U$  for any  $D$ , we have

$$(1) \quad D^U \models \Gamma \text{ iff } (D^U)^U \models \Gamma \text{ for any } D.$$

Since  $\text{pre}_0^U(\Gamma)$  is a plain pre-test of  $\Gamma$  wrt.  $U$ , we have

$$(2) \quad D^U \models \text{pre}_0^U(\Gamma) \text{ iff } (D^U)^U \models \Gamma \text{ for any } D^U \text{ (and thus for any } D).$$

By transitivity between (1) and (2) we obtain the thesis.  $\square$

More surprisingly, however, the converse does not hold, i.e., there are plain post-tests that are not plain pre-tests. In general, even for idempotent updates, there are pre-tests that are not post-tests and post-tests that are not pre-tests.

**Proposition 4.3** Let  $\Gamma$  be an integrity theory and  $U$  an idempotent update. Then

- (1) there is a pre-test  $\Sigma$  of  $\Gamma$  for  $U$  such that it does not hold that  $D^U \models \Sigma$  iff  $D^U \models \Gamma$  for any database  $D$  consistent with  $\Gamma$ , i.e.,  $\Sigma$  is not a post-test of  $\Gamma$  for  $U$ .
- (2) there is a plain post-test  $\Upsilon$  of  $\Gamma$  for  $U$  such that it does not hold that  $D \models \Upsilon$  iff  $D^U \models \Gamma$  for any database  $D$  consistent with  $\Gamma$ , i.e.,  $\Upsilon$  is not a pre-test of  $\Gamma$  for  $U$ .

**Proof**

(1) Let  $D$  be a state consistent with  $\Gamma$ . We have:

$$\begin{aligned} D \models \Sigma & \text{ iff } D^U \models \Gamma \text{ } (\Sigma \text{ pre-test}) \\ & \text{ iff } (D^U)^U \models \Gamma \text{ } (U \text{ idempotent}) \\ & \text{ iff } D^U \models \Sigma, \text{ if } D^U \models \Gamma \text{ } (\Sigma \text{ pre-test}) \end{aligned}$$

So, the only possibility is a situation where  $D \not\models \Sigma$ ,  $D^U \models \Sigma$ , and  $D^U \not\models \Gamma$ , which happens, e.g., with  $\Gamma = \{\leftarrow p(a)\}$ ,  $\Sigma = \{\leftarrow \neg p(a)\}$ ,  $D = \emptyset$  and  $U$  an update such that  $D^U = \{p(a)\}$ . To conclude the proof, we show that the chosen  $\Sigma$  is a pre-test. Indeed, for any  $D$  such that  $D \models \Gamma$ , then  $D \models \Sigma$ , and  $D^U \models \Gamma$ . (2) The same  $D$ ,  $U$  and  $\Gamma$  as in the previous point can also be used for the second case by considering the plain post-test  $\Upsilon = \Gamma$ . We have  $D \models \Gamma$ ,  $D^U \not\models \Gamma$ ,  $D \models \Upsilon$  and  $D^U \not\models \Upsilon$ .  $\square$

Another interesting aspect regarding pre-tests and post-tests is whether their evaluation is at all affected by the update. Proposition 4.2 immediately implies that the evaluation of a plain pre-test is not affected by the update, as stated in the following corollary.

**Corollary 4.4** *Let  $\Gamma$  be an integrity theory and  $U$  an idempotent update. Then  $D \models \text{pre}_0^U(\Gamma)$  iff  $D^U \models \text{pre}_0^U(\Gamma)$  for any  $D$ .*

**Proof** By definition of  $\text{pre}_0^U(\Gamma)$ , we have

$$D \models \text{pre}_0^U(\Gamma) \text{ iff } D^U \models \Gamma \text{ for any } D,$$

and, by transitivity with the claim of proposition 4.2, we have the thesis.  $\square$

However, this does not hold in general for pre-tests or (plain) post-tests, as demonstrated in the example of the proof of proposition 4.3.

The following table summarizes the results presented in this section. We indicate with  $\text{Pre}^U(\Gamma)$  (resp.,  $\text{Post}^U(\Gamma)$ ) the set of all pre-tests (resp., post-tests) of integrity theory  $\Gamma$  for  $U$ , and use a 0 subscript to indicate the set of all plain pre-tests (resp., post-tests) of  $\Gamma$  for  $U$ .

for any $\Gamma$	any $U$	$U$ idempotent
$\text{Pre}^U(\Gamma) \subseteq \text{Post}^U(\Gamma)?$	<b>no</b>	<b>no</b>
$\text{Post}^U(\Gamma) \subseteq \text{Pre}^U(\Gamma)?$	<b>no</b>	<b>no</b>
$\text{Pre}_0^U(\Gamma) \subseteq \text{Pre}^U(\Gamma)?$	<b>yes</b>	<b>yes</b>
$\text{Post}_0^U(\Gamma) \subseteq \text{Post}^U(\Gamma)?$	<b>yes</b>	<b>yes</b>
$\text{Pre}_0^U(\Gamma) \subseteq \text{Post}^U(\Gamma)?$	<b>no</b>	<b>yes</b>

## 5 Related work

Simplification of integrity constraints has been recognized by a large body of research as a powerful technique for optimization of integrity checking. Several approaches to simplification require the update transaction to be performed *before* the resulting state is checked for consistency with a post-test [76, 56, 81, 46, 29, 55]. As opposed to that, to pre-test the feasibility of an update with respect to an integrity theory allows for avoiding both the execution of the update and, particularly, the restoration of the database state before the update, which may be very costly. Pre-test-based methods are, e.g., [48, 50, 78, 54, 55, 25, 37, 66, 24, 64, 65, 59, 60, 22, 61, 58, 23, 26], including a few industrial attempts, e.g., [16, 3]. Other methods provide simplifications that may require the availability of both the old and the new state, assuming that the database keeps track of the old state before committing an update, [82, 83].

In [46], an adaptation of subsumption checking (called *partial subsumption*) is used to generate simplification as the “difference” (called *residual*) between an integrity constraint and a clause representing an update.

Qian’s method [78] generates pre-tests for integrity checking based on the observation that a simplified integrity constraint can be regarded as a weakest precondition for having a consistent updated state, in the same sense as in Hoare’s logic [49, 41] for imperative languages, and by assuming consistency of the database before the update.

Simplification of integrity constraints for update patterns resembles the notion of program *specialization* used in partial evaluation, which is the process of creating a specialized version of a given program (in this case, a general integrity checker) with respect to known input data (here, the update), as investigated in [55].

More generally, integrity checking can be seen 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 [47, 42].

Simplification can also be obtained by resorting to decision procedures for query containment [43, 14, 15, 4], as shown in [25].

We intentionally did not do so before, but at this point it seems worth mentioning that several simplification methods accept instantiable or parameterizable *patterns* of updates instead of specific updates, e.g. [48, 46, 25]. Thus, given such a pattern at schema specification time, it is possible to compile a simplification of the integrity theory for all updates matching that pattern. For instance, if constants  $a$ ,  $b$  and  $c$  in the update of Example 3.1 were specified as simple placeholders for constants (called *dummy constants* in [48] and *parameters* in [25]), and thus not assumed to be necessarily different, a pre-test-based simplification would also include an integrity constraint that checks that  $a \neq b$ .

Logic programming-based approaches such as [81, 53] do not take into account irrelevant clauses for refuting denial constraints, even if they would take part in an unnoticed case of inconsistency that has not been caused by the checked update but by some earlier event. Moreover, such approaches do not exhibit any explosive behavior as predicted classical logic in the presence of in-



consistency. In other words, query evaluation procedures based on SL-resolution can fairly well be called inconsistency-tolerant or “paraconsistent” in a procedural sense, as done, e.g., in [52, 27]. The declarative inconsistency tolerance of simplifications for improving integrity checking has been studied in several works [40, 36, 34, 32, 31, 39, 35, 33, 30, 38].

The related problem of restoring integrity of a database once inconsistencies are discovered is tackled by calculating a *repair*, i.e., a consistent database that is as close as possible to the original, inconsistent database. Since the seminal contribution [2], many authors have studied the problem of providing consistent answers to queries posed to an inconsistent database. These techniques certainly add to inconsistency tolerance in databases, but cannot be directly used to detect inconsistencies for integrity checking purposes (i.e., by posing integrity constraints as queries). Along the same lines, active rules have been considered as a means to restore a consistent database [20, 18, 28].

Last, we mention work on incomplete databases which also considers integrity constraints that are not satisfied in a given database state as something to be dealt with constructively, instead of banning it from consideration, as most integrity checking methods do [86, 85, 11, 63]. However, that work is not interested in integrity checking simplifications that could be used in such databases. Rather, it is dealing with the issue of satisfiability and its computational complexity, as related to an open world assumption by which the space of possible “closed worlds” (i.e. database states without missing information) that would satisfy integrity are studied. The theme of this paper is to simplify the checking of integrity satisfaction in the presence of inconsistency, not to ask for the satisfiability of integrity constraints in the absence of complete information. (Basic similarities and differences of satisfaction and satisfiability of integrity are addressed in [5].)

Relevant new directions of research regard all those areas where integrity constraints are used to characterize useful scenarios in which query answering plays an important role. Among these, we mention *i)* *access patterns*, which are constraints indicating which attributes of a relation schema are used as input and which ones are used as output [10, 12, 62, 8, 9, 6, 13, 7], *ii)* *top- $K$  queries*, where the constraints specify a limit on the number of results that the query should return, including constraints on proximity or diversity [70, 67, 17, 45, 44, 69, 19, 68, 84, 51], *iii)* *taxonomies* and context information, which may be used to pose constraint on the granularity of the data and to reason about it [75, 73, 74, 71, 72, 80, 21].

## 6 Conclusion

We have discussed and compared the two main abstract families of methods that can be used to incrementally check integrity constraints: pre-tests and post-tests. These are simplifications to be checked before or, respectively, after the update is executed (while update commitment is supposed to occur only after a successful check). In order to not only talk about some selected, specific

methods, albeit well-known ones, we have characterized declarative and procedural aspects of simplification-based integrity checking in a manner which is largely independent of concrete methods. Unsurprisingly, pre-tests and post-tests are not interchangeable, not only in terms of the convenience of executing the ones or the others in practical situations (pre-tests may actually be preferred in case of updates to be rejected), but also of their semantic properties. Somewhat surprisingly, however, their applicability is mostly asymmetric, even for the simple case of idempotent updates.

## 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] M. Benedikt, G. Bruns, J. Gibson, R. Kuss, and A. Ng. Automated Update Management for XML Integrity Constraints. In *Informal Proceedings of PLAN-X Workshop*, 2002.
- [4] P. A. Bonatti. On the decidability of containment of recursive datalog queries - preliminary report. In A. Deutsch, editor, *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France*, pages 297–306. ACM, 2004.
- [5] F. Bry, H. Decker, and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *EDBT*, pages 488–505, 1998.
- [6] A. Calì, D. Calvanese, and D. Martinenghi. Optimization of query plans in the presence of access limitations. In *Proc. of the ICDT 2007 Workshop on Emerging Research Opportunities in Web Data Management (EROW 2007)*, CEUR Electronic Workshop Proceedings, <http://ceur-ws.org/>, pages 33–47, 2007.
- [7] A. Calì, D. Calvanese, and D. Martinenghi. Dynamic Query Optimization under Access Limitations and Dependencies. *Journal of Universal Computer Science*, 15(21):33–62, 2009.
- [8] A. Calì and D. Martinenghi. Conjunctive Query Containment under Access Limitations. In *Proceedings of Conceptual Modeling - ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, October 20-24, 2008*, pages 326–340, 2008.
- [9] A. Calì and D. Martinenghi. Querying Data under Access Limitations. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 50–59, 2008.
- [10] A. Calì and D. Martinenghi. Optimizing Query Processing for the Hidden Web (Tutorial). In *Advances in Web Technologies and Applications, Proceedings of the 12th Asia-Pacific Web Conference, APWeb 2010, Busan, Korea, 6-8 April 2010*, page 397, 2010.
- [11] A. Calì and D. Martinenghi. Querying incomplete data over extended er schemata. *Theory and Practice of Logic Programming*, 10(3):291–329, 2010.

- [12] A. Cali and D. Martinenghi. Querying the deep web (tutorial). In *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings*, pages 724–727, 2010.
- [13] A. Cali, D. Martinenghi, and D. Carbotta. Query optimisation for web data sources: minimisation of the number of accesses (Extended Abstract). In *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD 2007, 17-20 June 2007, Torre Canne, Fasano, BR, Italy*, pages 316–323, 2007.
- [14] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 149–158. ACM Press, 1998.
- [15] D. Calvanese, G. De Giacomo, and M. Vardi. Decidable containment of recursive queries. In D. Calvanese, M. Lenzerini, and R. Motwani, editors, *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings*, volume 2572 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2003.
- [16] S. Carrico, B. Ewbank, T. G. Griffin, J. Meale, and H. Trickey. A tool for developing safe and efficient database transactions. In *XV International Switching Symposium of the World Telecommunications Congress. April, 1995*, pages 173–177, 1995.
- [17] I. Catallo, E. Ciceri, P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k diversity queries over bounded regions. *ACM Transactions on Database Systems*, 38(2):10, 2013. Extended version of [45].
- [18] 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.
- [19] S. Ceri, D. Martinenghi, and M. Tagliasacchi. Cost-aware rank-join algorithms. Technical report, Politecnico di Milano, 2009.
- [20] 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.
- [21] H. Christiansen and D. Martinenghi. Symbolic Constraints for Meta-Logic Programming. *Applied Artificial Intelligence*, 14(4):345–367, 2000.
- [22] H. Christiansen and D. Martinenghi. Simplification of database integrity constraints revisited: A transformational approach. In *Logic Based Program Synthesis and Transformation, 13th International Symposium LOPSTR 2003, Uppsala, Sweden, August 25-27, 2003, Revised Selected Papers*, volume 3018 of *Lecture Notes in Computer Science*, pages 178–197. Springer, 2004.
- [23] H. Christiansen and D. Martinenghi. Simplification of integrity constraints for data integration. In *Foundations of Information and Knowledge Systems, Third International Symposium, FoIKS 2004, Wilhelminenburg Castle, Austria, February 17-20, 2004, Proceedings*, volume 2942 of *Lecture Notes in Computer Science*, pages 31–48. Springer, 2004.
- [24] H. Christiansen and D. Martinenghi. Incremental integrity checking: Limitations and possibilities. In *Logic for Programming, Artificial Intelligence, and Reasoning*,

- 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*, volume 3835 of *Lecture Notes in Computer Science*, pages 712–727. Springer, 2005.
- [25] H. Christiansen and D. Martinenghi. On Simplification of Database Integrity Constraints. *Fundamenta Informaticae*, 71(4):371–417, 2006.
  - [26] H. Christiansen and D. Martinenghi. On using simplification and correction tables for integrity maintenance in integrated databases. In *Proceedings of the Second International Workshop on Logical Aspects and Applications of Integrity Constraints (LAAIC06) 8 September 2006, Krakow, Poland*, pages 569–576. IEEE Computer Society, 2006.
  - [27] H. Decker. Historical and computational aspects of paraconsistency in view of the logic foundation of databases. In L. Bertossi, G. Katona, K.-D. Schewe, and B. Thalheim, editors, *Semantics in Databases*, volume 2582 of *LNCS*, pages 63–81. Springer, 2001.
  - [28] H. Decker. Translating advanced integrity checking technology to sql. In J. H. Doorn and L. C. Rivero, editors, *Database integrity: challenges and solutions*, pages 203–249. Idea Group Publishing, 2002.
  - [29] H. Decker and M. Celma. A slick procedure for integrity checking in deductive databases. In P. Van Hentenryck, editor, *Logic Programming: Proc. of the 11th International Conference on Logic Programming*, pages 456–469. MIT Press, Cambridge, MA, 1994.
  - [30] H. Decker and D. Martinenghi. Avenues to flexible data integrity checking. In *Proceedings of the International Workshop on Flexible Database and Information System Technology (FlexDBIST-06) 6 September 2006, Krakow, Poland*, pages 425–429. IEEE Computer Society, 2006.
  - [31] H. Decker and D. Martinenghi. Can Integrity Tolerate Inconsistency? (Extended Abstract). In *Proceedings of the Fourteenth Italian Symposium on Advanced Database Systems, SEBD 2006, Portonovo, Italy, June 18-21, 2006*, pages 32–39, 2006.
  - [32] H. Decker and D. Martinenghi. Checking violation tolerance of approaches to database integrity. In *Advances in Information Systems, 4th International Conference, ADVIS 2006, Izmir, Turkey, October 18-20, 2006, Proceedings*, volume 4243 of *Lecture Notes in Computer Science*, pages 139–148. Springer, 2006.
  - [33] H. Decker and D. Martinenghi. Integrity checking for uncertain data. In *Proceedings of the Second Twente Data Management Workshop on Uncertainty in Databases 6 June 2006, Enschede, The Netherlands*, CTIT Workshop Proceedings Series WP06-01, pages 41–48. University of Twente, 2006.
  - [34] H. Decker and D. Martinenghi. A relaxed approach to integrity and inconsistency in databases. In *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13-17, 2006, Proceedings*, volume 4246 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2006.
  - [35] H. Decker and D. Martinenghi. Getting rid of straitjackets for flexible integrity checking. In *Proceedings of the 2nd International Workshop on Flexible Database and Information System Technology (FlexDBIST-07)*, pages 360–364, 2007.

- [36] H. Decker and D. Martinenghi. Classifying integrity checking methods with regard to inconsistency tolerance. In *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 15-17, 2008, Valencia, Spain*, pages 195–204, 2008.
- [37] H. Decker and D. Martinenghi. Database integrity checking. In M. Khosrow-Pour, editor, *Encyclopedia of Information Science and Technology (Second Edition)*, volume II, pages 961–966. Information Science Reference, 2008.
- [38] H. Decker and D. Martinenghi. Inconsistency-tolerant integrity checking. In V. Ferragine, J. Doorn, and L. Rivero, editors, *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends*, volume II, chapter XXXVIII, pages 348–357. Information Science Reference, 2009.
- [39] H. Decker and D. Martinenghi. Modeling, measuring and monitoring the quality of information. In *Proceedings of the 4th International Workshop on Quality of Information Systems (QoIS 2009)*, pages 212–221, 2009.
- [40] H. Decker and D. Martinenghi. Inconsistency-tolerant Integrity Checking. *IEEE Transactions on Knowledge & Data Engineering*, 23(2):218–234, 2011.
- [41] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- [42] G. Dong and J. Su. Incremental Maintenance of Recursive Views Using Relational Calculus/SQL. *SIGMOD Record*, 29(1):44–51, 2000.
- [43] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS '98. Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 139–148, New York, NY 10036, USA, 1998. ACM Press.
- [44] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Efficient diversification of top-k queries over bounded regions. In *SEBD*, pages 139–146, 2012.
- [45] P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Top-k bounded diversification. In *Proceedings of the 2012 ACM SIGMOD/PODS Conference – SIGMOD 2012, Scottsdale, Arizona, USA, May 20-24, 2012*, pages 421–432, 2012.
- [46] 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.
- [47] A. Gupta and I. S. Mumick, editors. *Materialized views: techniques, implementations, and applications*. MIT Press, 1999.
- [48] L. Henschen, W. McCune, and S. Naqvi. Compiling constraint-checking programs from first-order formulas. In H. Gallaire, J. Minker, and J.-M. Nicolas, editors, *Advances In Database Theory, February 1-5, 1988, Los Angeles, California, USA*, volume 2, pages 145–169. Plenum Press, New York, 1984.
- [49] C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
- [50] 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.

- [51] I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Rank-join algorithms for search computing. In S. Ceri and M. Brambilla, editors, *Search Computing: Challenges and Directions*, pages 211–224. 2009.
- [52] R. A. Kowalski. *Logic for Problem Solving*. Elsevier, 1979.
- [53] R. A. Kowalski, F. Sadri, and P. Soper. Integrity checking in deductive databases. In P. M. Stocker, W. Kent, and P. Hammersley, editors, *VLDB 1987, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 61–69. Morgan Kaufmann, 1987.
- [54] S. Y. Lee and T. W. Ling. Further improvements on integrity constraint checking for stratifiable deductive databases. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB’96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 495–505. Morgan Kaufmann, 1996.
- [55] 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.
- [56] J. W. Lloyd, L. Sonenberg, and R. W. Topor. Integrity constraint checking in stratified databases. *Journal of Logic Programming*, 4(4):331–343, 1987.
- [57] J. W. Lloyd and R. W. Topor. A basis for deductive database systems II. *Journal of Logic Programming*, 30(1):55–67, 1986.
- [58] D. Martinenghi. A simplification procedure for integrity constraints. <http://www.dat.ruc.dk/~dm/spic/index.html>, 2003.
- [59] D. Martinenghi. Optimal database locks for efficient integrity checking. In *Eighth East-European Conference on Advances in Databases and Information Systems (ADBIS 2004), Budapest, Hungary, 22-25 September 2004, local proceedings*, pages 64–77, 2004.
- [60] D. Martinenghi. Simplification of integrity constraints with aggregates and arithmetic built-ins. In *Flexible Query Answering Systems, 6th International Conference, FQAS 2004, Lyon, France, June 24-26, 2004, Proceedings*, volume 3055 of *Lecture Notes in Computer Science*, pages 348–361. Springer, 2004.
- [61] D. 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.
- [62] D. Martinenghi. Access pattern. In H. C. van Tilborg and S. Jajodiathe, editors, *Encyclopedia of Cryptography and Security (Second Edition)*, pages A17–A20. Springer, 2011.
- [63] D. Martinenghi. On the dependency on the size of the data when chasing under conceptual dependencies. *CoRR*, submit/0863729, 2013.
- [64] D. Martinenghi and H. Christiansen. Efficient integrity checking for databases with recursive views. In *Advances in Databases and Information Systems, 9th East European Conference, ADBIS 2005, Tallinn, Estonia, September 12-15, 2005, Proceedings*, volume 3631 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2005.
- [65] D. Martinenghi and H. Christiansen. Transaction management with integrity checking. In *16th International Conference on Database and Expert Systems Applications*, volume 3588 of *Lecture Notes in Computer Science*, pages 606–615. Springer, 2005.

- [66] D. Martinenghi, H. Christiansen, and H. Decker. Integrity checking and maintenance in relational and deductive databases - and beyond. In Z. Ma, editor, *Intelligent Databases: Technologies and Applications*, chapter X, pages 238–285. Idea Group Publishing, 2006.
- [67] D. Martinenghi and M. Tagliasacchi. Proximity Rank Join. *Proceedings of the VLDB Endowment*, 3(1):352–363, 2010.
- [68] D. Martinenghi and M. Tagliasacchi. Top-k pipe join. In *Proceedings of the 4th International Workshop on Ranking in Databases (DBRank 2010)*, pages 16–19. IEEE Computer Society Press, 2010.
- [69] D. Martinenghi and M. Tagliasacchi. Cost-Aware Rank Join with Random and Sorted Access. *IEEE Transactions on Knowledge & Data Engineering*, 24(12):2143–2155, 2012.
- [70] D. Martinenghi and M. Tagliasacchi. Proximity measures for rank join. *ACM Transactions on Database Systems*, 37(1), 2012. Extended version of [67].
- [71] D. Martinenghi and R. Torlone. A logical approach to context-aware databases. In *Proceedings of the 6th Conference of the Italian Chapter of AIS (itAIS 2009)*, 2009. Post-proceedings published in [74].
- [72] D. Martinenghi and R. Torlone. A model and a language for context-aware databases. Technical Report RT-DIA-152-2009, Dipartimento di Informatica e Automazione, Università degli studi Roma Tre, July 2009.
- [73] D. Martinenghi and R. Torlone. Querying context-aware databases. In *Flexible Query Answering Systems, 8th International Conference, FQAS 2009, Roskilde, Denmark, October 26-28, 2009. Proceedings*, pages 76–87, 2009.
- [74] D. Martinenghi and R. Torlone. A logical approach to context-aware databases. In A. D’Atri, M. D. Marco, A. Braccini, and F. Cabiddu, editors, *Management of the Interconnected World*, pages 211–220. 2010. Extended version of [71].
- [75] D. Martinenghi and R. Torlone. Querying Databases with Taxonomies. In *Proceedings of Conceptual Modeling - ER 2010, 29th International Conference on Conceptual Modeling, Vancouver, BC, Canada, November 1-4, 2010*, pages 377–390, 2010.
- [76] J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, 18:227–253, 1982.
- [77] A. Olivé. Integrity constraints checking in deductive databases. In G. M. Lohman, A. Sernadas, and R. Camps, editors, *17th International Conference on Very Large Data Bases, September 3-6, 1991, Barcelona, Catalonia, Spain, Proceedings*, pages 513–523. Morgan Kaufmann, 1991.
- [78] 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.
- [79] K. Ramamohanarao and J. Harland. An introduction to deductive database languages and systems. *VLDB Journal*, 3(2):107–122, 1994.
- [80] A. Rauseo, D. Martinenghi, and L. Tanca. Context through answer set programming (abstract). In *Proceedings of the EDBT 2011 workshops, 3rd International Workshop on Logic in Databases (LID 2011)*, page 58, 2011.

- [81] 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.
- [82] R. Seljée. A new method for integrity constraint checking in deductive database. *Data & Knowledge Engineering*, 15(1):63–102, 1995.
- [83] R. Seljée and H. C. M. de Swart. Three types of redundancy in integrity checking: An optimal solution. *Data & Knowledge Engineering*, 30(2):135–151, 1999.
- [84] M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Ranking with Uncertain Scoring Functions: Semantics and Sensitivity Measures. In *Proceedings of the 2011 ACM SIGMOD/PODS Conference – SIGMOD 2011, Athens, Greece, June 12–16, 2011*, pages 805–816, 2011.
- [85] R. van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356, 1998.
- [86] M. Y. Vardi. On the integrity of databases with incomplete information. In *PODS*, pages 252–266, 1986.