Chapter XXXVIII
Inconsistency-Tolerant Integrity Checking

Hendrik Decker
Instituto Technológico de Informática, Spain
Ciudad Politécnica de la Innovación, Spain

Davide Martinenghi
Politecnico di Milano, Italy

INTRODUCTION

Integrity checking has been a perennial topic in almost all database conferences, journals, and research labs. The importance of the issue is testified by very large amounts of research activities and publications. They are motivated by the fact that integrity checking is practically unfeasible for significant amounts of stored data without a dedicated approach to optimize the process. Basic early approaches have been extended to deductive, object-relational, XML- (extensible markup language) based, distributed, and other kinds of advanced database technology. However, the fundamental ideas that are already present in the seminal paper (Nicolas, 1982) have not changed much.

The basic principle is that, in most cases, a so-called simplification, that is, 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 is authorized only if the check of the simplification yields that integrity is not violated. Here, *simplified* essentially means *more efficiently evaluated* at update time. A general overview of the field of simplified integrity checking is provided in Martinenghi, Christiansen, and Decker (2006).

A common point of view by which the need for integrity checking is justified can be characterized as follows. Whenever a database contains erroneous, unwanted, or faulty information, that is, data that violate integrity, answers to queries cannot be trusted. Hence, simplification methods for integrity checking usually address this issue in a very drastic way: In order to avoid possibly wrong answers that are due to integrity violation, incorrect stored data that cause inconsistency need to be completely prevented. However, this drastic

attitude is most often unrealistic: The total absence of unwanted, incorrect, or unexpected data is definitely an exception in virtually all real-world scenarios. Still, it is desirable to preserve the good data in the database while preventing more bad ones from sneaking in and, thus, further diminish the trustworthiness of answers to queries.

The intolerant attitude of the simplification approach of integrity checking toward data that violate integrity is reflected in Nicolas (1982) and virtually all publications on the same subject that came after it. They all postulate the categorical premise of total integrity satisfaction, that is, that each constraint must be satisfied in the old database state, given when an update is requested but not yet executed. Otherwise, correctness of simplification is not guaranteed.

As opposed to the attention granted to integrity checking in academia, support for the declarative specification and efficient evaluation of semantic integrity in practical systems has always been relatively scant, apart from standard constructs such as constraints on column values, or primary and foreign keys in relational database tables. Various reasons have been identified for this lack of practical attention. Among them, the logically abstract presentation of many of the known simplification methods is often mentioned. Here, we focus on another issue of integrity checking that we think is even more responsible for a severe mismatch between theory and practice: Hardly any database ever is in a perfectly consistent state with regard to its intended semantics. Clearly, this contradicts the fundamental premise that the database must always satisfy integrity. Thus, due to the intolerance of classical logic with respect to inconsistency, integrity checking is very often not considered an issue of practical feasibility or even relevance.

Based on recent research results, we are going to argue that inconsistency is far less harmful for database integrity than as suggested by commonly established results. We substantiate our claim by showing that, informally speaking, the consistent part of a possibly inconsistent database can be preserved across updates. More precisely, we show that, if the simplified form of an integrity theory is satisfied, then each instance of each constraint that has been satisfied in the old state continues to be satisfied in the new updated state, even if the old database is not fully consistent. Therefore, such an approach can rightfully be called inconsistency tolerant. Yet, we are also going to see that the use of inconsistency-tolerant integrity checking methods prevents an increase of inconsistency and may even help to decrease it.

BACKGROUND

Throughout, we refer to the relational framework of deductive databases, that is, relational databases, with possibly recursive view definitions described in clause form (Abiteboul, Hull, & Vianu, 1995). Thus, a database consists of a set of facts and a set of rules, that is, tuples and view definitions, respectively, in the terminology of the relational model.

An integrity constraint (or, shortly, constraint) expresses a semantic invariant, in other words, a condition that is 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. Usually, without loss of generality, constraints are either represented in prenex normal form (i.e., with all quantifiers moved leftmost and all negation symbols moved innermost) or as denials (i.e., datalog clauses with empty heads). Such a denial expresses that, if its condition is satisfied, then integrity is violated. An integrity theory is a finite set of constraints.

We limit ourselves to databases with a unique standard model. For a closed formula W and a database D, we write $D \models W$ (resp., $D \not\models W$) to indicate that W evaluates to true (resp., false) in the standard model of D. For a set of formulas Γ, we write $D \models \Gamma$ (resp., $D \not\models \Gamma$) to indicate that, for each (resp., some) formula W in Γ, we have $D \models W$ (resp., $D \not\models W$). For a constraint W and an integrity theory Γ, it is also usual to say that D satisfies (resp., violates) W and Γ, respectively.