Scalable Uncertainty-tolerant Business Rules

Alfredo Cuzzocrea[1], Hendrik Decker[2⋆], and Francesc D. Muñoz-Escoí[2⋆]

[1] ICAR-CNR and University of Calabria, I-87036 Cosenza, Italy
[2] Instituto Tecnológico de Informática, UPV, E-46022 Valencia, Spain
cuzzocrea@si.deis.unical.it;hendrik@iti.upv.es;fmunyoz@iti.upv.es

Abstract. Business rules are of key importance for maintaining the correctness of business processes and the reliability of business data. When they take the form of integrity constraints, business rules also can help to contain the amount of uncertainty associated to business data and decisions based on those data. However, business rule enforcement may not scale up easily to systems with concurrent transactions. To a large extent, the problem is due to two common exigencies: the postulates of total and of isolated business rule satisfaction. In order to limit the accumulation of business rule violations, and thus of uncertainty, we are going to outline how a measure-based uncertainty-tolerant approach to business rules maintenance scales up to concurrent transactions. The scale-up is achieved by refraining from the postulates of total and isolated business rule satisfaction.

1 Introduction

For fully automating the reasoning and processing of business workflows, arguably the best choice is to invest in business rules and represent them as integrity constraints. That point of view has been convincingly argued for in [10].

Apart from business rules, another important application domain of integrity constraints is the field of data quality [38]. In [14, 13, 16], authors have argued that, to a large extent, conditions deemed necessary by the database designer for the data to have quality can be represented by integrity constraints. Consequently, data quality can be equated to the degree by which the data satisfy the integrity constraints that capture quality. Hence, such constraints can be conveniently called *quality constraints* [17].

In turn, uncertainty in databases can be understood as a lack of data quality. And, as already indicated, quality conditions can conveniently be represented in the syntax of integrity constraints. For the special case of primary key constraints, it has been pointed out in [11] that the violation of such constraints corresponds to the uncertainty of the data items that are identified by those keys. More generally, authors have argued in [9] that the violation of *any* integrity constraint accounts for a measurable degree of uncertainty. On the other hand, it is well-known that many database applications may involve considerable amounts of uncertain data.

⋆ supported by ERDF/FEDER and the MEC grant TIN2012-37719-C03-01 .

Hence, the point of departure for the remainder of this paper is the use and enforcement of business rules in databases as quality guards against uncertainty. Although business rules are meant to avoid uncertainty, they nevertheless need to be able to tolerate it, i.e., to perform reasonably well in the presence of uncertain data. The specific problem addressed in this paper is to make uncertainty-tolerant business rules work not only with regard to isolated singular updates of the database, but also on a much higher scale where arbitrarily many database transactions are executed concurrently. Concurrency of transactions is typical for most OLTP applications, for distributed systems and for transactions in the cloud. In [9], the scale-up of uncertainty tolerance to concurrent transactions has been addressed as a side issue. In this paper, it receives unabridged attention.

A large amount of methods for evaluating declarative integrity constraints have been proposed in the literature [33]. Also the use of business rules for data quality management has been documented, e.g., in [34] [40] [39]. Moreover, the management of concurrent transactions has been broadly covered in the literature [21] [25] [5] [26] [48].

This paper addresses two characteristic difficulties that impede a combination of approaches to control data uncertainty by checking quality constraints, on one hand, and uncertainty-containing processing of concurrent transactions, on the other. One of the two difficulties corresponds to a particular requirement that is traditionally imposed on all methods of integrity checking. The other corresponds to a particular requirement traditionally imposed on the design, implementation and use of concurrent transactions. It will turn out that both requirements are unrealistic and indeed not necessary to their full extent in practice.

The first requirement is that an update can be efficiently checked for integrity only if the state before the update totally satisfies all constraints, without exception. We call this requirement *the total integrity postulate*. The second is that, for guaranteeing integrity preservation by serializable concurrent transactions, each transaction is supposed to preserve integrity when executed in isolation. We call this requirement *the isolated integrity postulate*.

We point out that the isolated integrity postulate must not be confused with the well-known requirement of an isolated execution of transactions [26], i.e., that concurrent transactions should not step into each other's sphere of control [30], so that anomalies such as phantom updates, dirty or non-repeatable reads are avoided. That requirement usually is complied with by ensuring the serializability of schedules, or some relaxation thereof [48, 45]. However, serializability is independent of the isolated integrity postulate, requiring that integrity be preserved in isolation: while serializability can be guaranteed automatically by the scheduler of the DBMS, the isolated integrity postulate is usually expected to be complied with by the designers, programmers and users of transactions.

The dispensability of the total integrity postulate has been unveiled in [18]. The isolated integrity requirement of concurrent transactions has been discussed and relaxed in [19]. In the cited references, the total integrity postulate has been shown to be superfluous, and the isolated integrity postulate has been significantly relaxed, both by a concept of inconsistency-tolerant integrity checking.

That concept was based on the notion of 'cases', i.e., instances of integrity constraints. Their violation can be tolerated as long as integrity checking can guarantee that the amount of violated cases is not increased by given updates. In [13], a significant generalization of inconsistency-tolerant integrity checking has been presented. It is based on arbitrary inconsistency measures. The latter permit the tolerance of integrity violation as long as integrity checking can guarantee that the amount of measured inconsistency is not increased by given updates. In this paper, we show that measure-based uncertainty checking also enables a significant weakening of the isolated integrity postulate.

In section 2, we characterize the postulates of total and isolated integrity. In section 3, we recapitulate measure-based uncertainty checking. We are going to see that it serves to get rid of the total integrity postulate as well as to relax the isolated integrity postulate. In section 4, we address related work, with an emphasis on integrity checking for concurrent transactions. If not specified otherwise, we use conventional terminology and notations for logic databases [1], as well as some basic notions of transaction concurrency [5].

2 Bad Postulates

The total integrity postulate is going to be explained in 2.1, the isolated integrity postulate in 2.2. Both postulates are unnecessary and actually bad, since they invalidate predictions of the traditional theories of transaction processing [26] and integrity maintenance [33] if databases contain uncertain data.

2.1 Integrity Checking with Totality

Integrity checking can be exceedingly costly, unless some simplification method is used [36]. That can be illustrated as follows. (As usual, lower-case letters x, y, z denote variables, in the example below.)

`Example 2.1.` *Let* emp *be a relation about employees, whose first column is a unique name and the second a project assigned to the employee. The formula* $I = \leftarrow \text{proj}(x, y), \text{proj}(x, z), y \neq z$ *is a primary key constraint on the first column of* proj*, a relation about projects, with unique identifiers in the first column. The foreign key constraint* $I' = \forall x, y\, \exists z\, (\text{emp}(x, y) \rightarrow \text{proj}(y, z))$ *on the second column of* emp *references the primary key of* proj*. Now, assume a transaction* T *that inserts* emp(Jack, p)*. Most integrity checking methods* $\mathcal{M}$ *ignore* I *for checking* T*, since* I *does not constrain* emp*. Rather, they only evaluate the case* $\exists z\, (\text{emp}(Jack, p) \rightarrow \text{proj}(p, z))$ *of* I'*, or its simplification* $\exists z\ \text{proj}(p, z)$*, since* emp(Jack, p) *becomes* true *by the transaction. If, e.g.,* (p, e) *is a row in* proj*,* $\mathcal{M}$ *accepts the insertion. If there is no tuple matching* (p, z) *in* proj*, then* $\mathcal{M}$ *signals a violation of integrity.* □

Proofs of the correctness of methods for simplified constraints checking in the literature all rely on the total integrity postulate, i.e., that integrity always

be totally satisfied, before updates are checked for preserving consistency. In practice, however, it is rather the exception than the rule that this postulate is complied with. In particular for applications such as business intelligence, distributed and replicated databases, legacy data maintenance, data warehousing, data federation, etc, a certain amount of uncertain data that violate constraints in committed states has to be lived with, at least temporarily.

Suppose that, for instance, the constraint I' in Example 2.1 is violated due to $emp(Jack, OO) \in D$ and a previous deletion of the OO project. Thus, by definition, no method that requires total integrity is equipped to check T, since not all constraints are satisfied. In practice however, if the project that *Jack* is assigned to is stored in the *proj* relation, T is rightfully sanctioned by all common implementations of integrity checking, as already indicated in Example 2.1. Example 3.1 in Subsection 3.1 will illustrate essentially the same point.

Hence, the total integrity postulate, which conventionally has always been imposed, does not approve the correctness of integrity checking in practice, since the latter often is performed in the presence of consistency violations. Fortunately, however, that postulate can be abolished without incurring any cost and without losing its essential guarantees, as shown in 3.1.

2.2 Integrity Checking with Isolation

Integrity preservation has been a fundamental concern already in the early literature on transaction processing. We cite from [21]: "*it is assumed that each transaction, when executed alone, transforms a consistent state into a consistent state; that is, transactions preserve consistency*". This is what we have called the isolated integrity postulate. (Recall that the execution of a transaction T is *isolated* when it is not concurrent with other transactions, or when the state transition effected by T is as if having been executed alone.) Thus, the isolated integrity postulate effectively presupposes the total integrity postulate. From the isolated integrity postulate, [21] [25] [5] [26] and many other authors have inferred the well-known result that then, also all serializable schedules of concurrent transactions preserve 'consistency', i.e., integrity.

In general, the requirements of total and isolated integrity seem to be illusionary, particularly for distributed multi-user databases, let alone for transactions in the cloud, where the 'eventual consistency' [46] of replicated data may compromise both integrity and isolation. Actually, it is hard to imagine than any agent who issues a transaction T would blindly believe in a consistency-preserving outcome of T by naïvely assuming that all concurrent transactions had been programmed with sufficient care to preserve integrity in isolation. Hitherto, there has been no theory to justify such optimism, in the presence of uncertainty. Yet now, such a justification is given in Section 3. In particular, we show in 3.4 that the consistency guarantees of uncertainty-tolerant integrity checking can be extended to concurrent transactions.

3 Uncertainty Tolerance

The purpose of business rules (i.e., integrity constraints) is to state and enforce quality properties of business data. However, uncertainty (i.e., violations of quality constraints that take the form of logical inconsistencies) are unavoidable in practice. Rather than insisting that all business rules must be totally satisfied at all times, it is necessary to tolerate unavoidable constraint violations.

Whenever time permits, attempts of reducing or repairing such manifestations of uncertainty can be made, while such attempts often are not affordable at update time. Thus, updates should be checkable for quality preservation, even if there are extant constraint violations, which can be dealt with later. That is the philosophy behind uncertainty-tolerant constraint checking, as revisited in 3.1. Technically speaking, constraint preservation means that the amount of measured uncertainty that manifests itself in constraint violations is not increased by a checked and approved transaction. In 3.4, we outline a generalization of the results in 3.1 to concurrent transactions.

Throughout the rest of the paper, let the symbols D, I, $\mathcal{IC}$, T, $\mathcal{M}$ stand for a database, an integrity constraint, a set of integrity constraints, a transaction and, resp., an integrity checking method. By $D(\mathcal{IC}) = TRUE$ and $D(\mathcal{IC}) = FALSE$, we denote that $\mathcal{IC}$ is satisfied or, resp., violated in D. Moreover, we suppose that all constraints are represented in prenex form, i.e., all quantifiers of variables appear leftmost. That includes the two most common forms of representing integrity constraints: as denials or in prenex normal form.

In general, each method $\mathcal{M}$ can be conceived as a mapping which takes triples $(D, \mathcal{IC}, T)$ as input, and returns either OK, which means that $\mathcal{M}$ sanctions T as integrity-preserving, or KO, which indicates that executing T would violate some constraint. Further, let D^T denote the database state obtained by applying the write set of T to D.

3.1 Removing the Total Integrity Constraint

In [18], authors have shown that, contrary to common belief, it is possible to get rid of the total integrity postulate for most approaches to integrity checking without any trade-off. Methods which continue to function well when this postulate is renounced are called inconsistency-tolerant. The basic idea is illustrated below.

`Example 3.1.` *Let I and I' be as in Example 2.1. Most integrity checking methods $\mathcal{M}$ accept the update* insert *(*Jack*, p) if, e.g., (p, e) is a row in* proj*. Now, the positive outcome of this integrity check is not disturbed if, e.g., also the tuple (p, f) is a row in* proj*. At first sight, that may be somewhat irritating, since I then is violated by two tuples about project p in the relation* proj*. In fact, the case $\leftarrow$ proj(p, e), proj(p, f), $e \neq f$ indicates an integrity violation. However, this violation has not been caused by the insertion just checked. It has been there before, and the assignment of* Jack *to p should not be rejected just because the data about p are not consistent. After all, it may be part of* Jack*'s new job to*

cleanse potentially inconsistent project data. In general, a transaction T that preserves the integrity of all consistent data without increasing the amount of extant inconsistency should not be rejected. And that is exactly what $\mathcal{M}$'s output indicates: no instance of any constraint that is satisfied in the state before T is committed is violated after T has been committed. □

3.2 Uncertainty Measures

Example 3.1 conveys that each update which does not increase the amount of inconsistency (i.e., integrity violation) can and should be accepted. For making precise what it means to have an increase of inconsistency or not, inconsistency needs to be measured. That can be formalized as follows.

`Definition 3.1.` *We say that $(\mu, \preceq)$ is an* uncertainty measure *(in short, a* measure*) if μ maps tuples $(D, \mathcal{IC})$ to a metric space that is partially ordered by $\preceq$. If $\preceq$ is understood, we simply identify a measure $(\mu, \preceq)$ with μ.* □

`Example 3.2.` *A binary border-case measure β is given by $\beta(D, \mathcal{IC}) = D(\mathcal{IC})$, with the natural ordering $TRUE \prec FALSE$ of the range of β, i.e., quality constraint satisfaction ($D(\mathcal{IC}) = TRUE$) means lower uncertainty than quality constraint violation ($D(\mathcal{IC}) = FALSE$). In fact, β is used by all conventional integrity checking methods, for deciding whether a given transaction T on a database D that satisfies its constraints $\mathcal{IC}$ should be accepted (if $D^T(\mathcal{IC}) = TRUE$) or rejected (if $D^T(\mathcal{IC}) = FALSE$).* □

More, less trivial uncertainty measures are defined and discussed in [9]. For instance, the function that maps pairs $(D, \mathcal{IC})$ to the cardinality of the set of cases (instances) of violated constraints is a convenient uncertainty measure. Inconsistency can also be measured by taking such sets themselves, as elements of the powerset of all cases of $\mathcal{IC}$, together with the subset ordering.

3.3 Integrity Checking with Uncertainty: Generalizing the Process

In accordance with [9], uncertainty-tolerant integrity checking can now be defined as follows, for databases D, integrity theories $\mathcal{IC}$ and transactions T.

`Definition 3.2.` *Let $\mathcal{M}$ be a mapping from triples $(D, \mathcal{IC}, T)$ to $\{OK, KO\}$, so that T is either accepted or, resp. rejected, and $(\mu, \preceq)$ an uncertainty measure. $\mathcal{M}$ is called a* sound*, resp.,* complete *method for integrity checking if, for each triple $(D, \mathcal{IC}, T)$, (1) or, resp., (2) holds.*

$$\mathcal{M}(D, \mathcal{IC}, T) = OK \;\Rightarrow\; \mu(D^T, \mathcal{IC}) \preceq \mu(D, \mathcal{IC}). \tag{1}$$

$$\mu(D^T, \mathcal{IC}) \preceq \mu(D, \mathcal{IC}) \;\Rightarrow\; \mathcal{M}(D, \mathcal{IC}, T) = OK. \tag{2}$$

If (1) holds, then $\mathcal{M}$ is also called measure-based*, and, in particular, μ-*based. □

Definition 3.2 generalizes the traditional definition of sound and complete integrity checking significantly, in two ways. The first essential upgrade is that, traditionally, the measure used for sizing constraint violations in a database with regard to its associated integrity theory is binary, and thus very coarse: $\mathcal{IC}$ is either *violated* or *satisfied* in D, i.e., there is no distinction with regard to different amounts of uncertainty. As opposed to that, the range of an uncertainty measure μ may be arbitrarily fine-grained. The second upgrade is equally significant: traditionally, the total integrity postulate is imposed, i.e., $D(\mathcal{IC}) = TRUE$ is required. As opposed to that, this postulate is absent in Definition 3.2, i.e., $\mathcal{M}$ does not need to worry about extant constraint violations.

Definition 3.2 formalizes that a method $\mathcal{M}$ is uncertainty-tolerant if its output OK for a given transaction T guarantees that the amount of uncertainty in $(D, \mathcal{IC})$ as measured by μ is not increased by executing T on D. Moreover, each transaction that, on purpose or by happenstance, repairs some inconsistent instance(s) of any constraint without introducing any new violation will be OK-ed too by $\mathcal{M}$. This means that, over time, the amount of integrity violations will decrease, as long as an uncertainty-tolerant method is used for checking each transaction for integrity preservation.

Note that it follows by the definition above that each uncertainty-tolerant $\mathcal{M}$ returns KO for any transaction the commitment of which would violate a hitherto satisfied instance of some constraint. It is then up to the agent who has called $\mathcal{M}$ for checking integrity to react appropriately to the output KO

A defensive reaction is to simply cancel and reject the transaction. A more offensive reaction could be to modify ('repair') the database, the constraints or the transaction, so that an increase of the amount of integrity violations is undone. Such measure-based database repairs are dealt with in [15].

3.4 Relaxed Integrity Checking with Isolation

To say, as the isolated integrity postulate does, that a transaction T "preserves integrity in isolation", means the following: For a given set $\mathcal{IC}$ of integrity constraints and each state D of a given database schema, each $I \in \mathcal{IC}$ is satisfied in D^T if I is satisfied in D.

Now, we are going to apply the concept of uncertainty-tolerant constraint checking in 3.1 not only to transactions executed in isolation, but also to concurrent transactions. Thus, we are going to abandon the above postulate "if I is satisfied in D" and weaken the consequence "each $I \in \mathcal{IC}$ is satisfied in D^T" according to Definition 3.2.

In [19], authors could show that this is possible for integrity checking methods that preserve all satisfied cases of integrity constraints, while tolerating cases that are violated in the state before a given transaction is executed. By an analogous (though more abstract) argument, the isolated integrity postulate can be weakened as follows.

For each state $(D, \mathcal{IC})$ of a given database schema, each uncertainty measure $(\mu, \preceq)$ and each transaction T,

$$\mu(D^T, \mathcal{IC}) \preceq \mu(D, \mathcal{IC}) \tag{3}$$

must hold whenever T is executed in isolation.

Under this postulate, it is then possible to infer the general result that (3) will continue to hold if T and all transaction that are concurrent with T are serializable, and (3) holds for each transaction T' that is concurrent with T whenever T' is executed in isolation, where (3) is obtained from (3) by replacing T with T'.

We point out that this result does not endorse that each case should be checked individually. On the contrary: integrity checking can proceed as for systems without concurrency, i.e., no built-in nor any external routine that takes part in the integrity checking process needs to be modified. The result just says that, if the method returns OK, then everything that was satisfied in the state before the transaction will remain satisfied after the transaction has committed, also for concurrent transactions.

The essential difference of this relaxation with regard to the traditional result, which imposes the general isolated integrity postulate, is the following. In the relaxed result, isolated integrity preservation only is asked to hold for individual cases. Since simplified integrity checking always focuses on cases that are relevant for the write set of a given transaction T, only these cases are guaranteed to remain satisfied by a successful integrity check. All non-relevant cases of the same or any other constraints may possibly be violated by concurrent or preceding transactions. Such violations are detected only if the respective transactions are checked too. If not, such violations are tolerated by each uncertainty-tolerant method that is used to check T.

To conclude this subsection, the following two points must be mentioned. Firstly, the relaxation of the isolated integrity postulate outlined above still asks for the serializability, i.e., a highly demanding isolation level, of all concurrent transactions. This means that we cannot expect that integrity guarantees of the form (3) would continue to hold in general if the isolation level is lowered. (For a general critique of lowering isolation levels, see [4].) Future work of ours is intended to investigate possible isolation level relaxations such that sufficient integrity guarantees can still be given.

The second point refers to the evolution of database schemata, and in particular to changes of the set of integrity constraints in the schema. The generalized form of uncertainty-tolerant integrity checking, as presented in 3.3, can be shown to be further generalizable for transactions that also involve schema updates including changes of the quality constraints [12]. Unfortunately, that is not the case for the consequences obtained from the relaxed isolated integrity postulate in 3.4. In fact, already the guarantees provided by the traditional isolated integrity postulate cannot be maintained for changes in the set of constraints. Thus, more work is necessary in order to establish sufficiently general conditions under which any integrity guarantees can be made for concurrent transactions across evolving database schemata.

4 Related Work

Most papers about the maintenance of constraints do not deal with transaction concurrency. On the other hand, most papers that do address concurrent transactions take it for granted that, if transaction were checked for constraint preservation in isolation, then it would pass that test successfully, i.e., they do not care how integrity is ensured.

As an exception, the work documented in [31], addresses both problem areas. However, the proposed solutions are application-specific (flight reservation) and seem to be quite ad-hoc. Also the author of [43] is aware of the problem, and argues convincingly to not be careless about consistency issues. However, with regard to semantic integrity violations in concurrent scenarios, he only exhibits a negative result (the CAP theorem [23]), but does not investigate uncertainty-tolerant solutions. There do exists solutions for reconciling consistency, availability and partition tolerance in distributed systems, e.g., [46] [44]. However, the consistency they are concerned with is either transaction consistency (i.e., the avoidance of dirty reads, unrepeatable reads and phantom updates) or replication consistency (i.e., that all replicas consist of identical copies, so that there are no stale data), not the semantic consistency that is the contrary of uncertainty.

In the seminal papers [24, 20, 28, 21, 3, 22], a distinction is made between integrity violations caused either by anomalies of concurrency or by semantic errors. In [20, 28], concurrency is not dealt with in depth. In [24, 21, 3, 22], integrity is not looked at in detail. Also in later related work, either concurrency or integrity is largely passed by, except in [6] [32], and in papers (e.g., [41]) that do not check each transaction for constraint preservation, but only the final state reached at the end of a history of concurrent transactions.

In this paper, we have not dealt with final-state integrity because we are interested in the integrity of each committed state, i.e, in guarantees that can be made for each individual transaction. Moreover, for each serializable history H, integrity is preserved in the final state of H if each transaction in H preserves integrity, even if there are extant constraint violations.

The author of [6] observes that integrity checks are read-only actions without effect on other operations, possibly except abortions due to integrity violation. Some scheduling optimizations made possible by the unobtrusive nature of read actions for integrity checking are discussed in [6].

A different, more DBMS-oriented solution, which however is proprietary, has been proposed in [37]:

"Making constraints immediate at the end of a transaction is a way of checking whether COMMIT can succeed. You can avoid unexpected rollbacks by setting constraints to IMMEDIATE as the last statement in a transaction. If any constraint fails the check, you can then correct the error before committing the transaction."

However, as long as the semantics of IMMEDIATE are not well-defined, it seems to be difficult to identify against which state a constraint is evaluated. In particular, the meaning of IMMEDIATE seems to be entirely speculative if there are locks which prevent immediate access (whatever that may mean).

A non-proprietary, partially automatic proposal to re-program concurrent transactions such that conflicts at commit time are avoided is proposed in [32]. The authors outline how to augment transactions with read actions for simplified constraint checking and with locks, so that their serializable execution guarantees integrity preservation. However, ad-hoc transactions are not considered in [32].

As opposed to solutions that are proprietary or involve transaction design, we propose a different approach: For each transaction T, the DBMS should determine autonomously (either by using a built-in procedure or some external device) whether the state transition effected by T preserves integrity, and react accordingly. In this paper, we have uncovered ways to overcome some of the obstacles that hitherto may have turned away researchers and developers from striving for such solutions.

For replicated database systems, the interplay of built-in integrity checking, concurrency and replication consistency has been studied in [35]. In that paper, solutions are provided for enabling integrity checking even in systems where the isolation level of transactions is lowered to *snapshot isolation* [4]. However, uncertainty tolerance in the sense of coping with extant integrity violations has not been considered in [35]. Thus, for the snapshot-isolation-based replication of databases, more research is necessary in order to clarify which consistency guarantees can be given when inconsistency-tolerant integrity checking methods are used in the presence of inconsistent cases of constraints.

5 Discussion

Since the beginnings of the field of computational databases, the obligation of maintaining the integrity of business rules in multi-user systems, and thus the avoidance of uncertainty, has remained with the designers, implementers, administrators and end users of transaction processing. More precisely, integrity maintenance in concurrency-enabled systems is delegated to a multitude of individual human actors who, on one hand, have to trust on each other's unfailing compliance with the integrity requirements, but, on the other hand, usually do not know each other.

The author expects that, in the long run, this unfortunate distribution of responsibilities will give way to declarative specifications of integrity constraints that can be supported automatically, just the way some fairly simple kinds of constraints are supported already for serial schedules in centralized, non-distributed database systems. An early attempt in this direction is reported in [27], where, however, concurrency is hardly an issue. Likewise, the work in [29] largely passes by concurrency.

Anyway, we have aimed in our work on uncertainty-containing transactions to keep as close to the declarative paradigm as possible. The advantage of declarativity is to free users and application programmers from having to worry about quality preservation. That is, the database designer should formalize business rules as declarative integrity constraints in SQL and leave everything else to the integrity checking module of the DBMS. That module may be built into the

DBMS core or run on top of it. In any case, the enforcement of the business rules should be as transparent to the user as concurrency, distribution and replication.

However, as seen above, well-known authors of concurrency theory require what is virtually impossible, on a grand scale: that all transactions should be programmed such that they guarantee the preservation of all constraints in isolation [25] [5] [48]. So, database designers and users are asked to program transactions in a way such that all semantically uncertain situations are avoided. This obviously may amount to a formidable task in complex systems.

Hence, the motivating objective of this paper has been to enable an automated enforcement of business rules for concurrent transactions. We have identified two obstacles that, in the past, have prevented to attain that goal: the postulates of total and isolated integrity.

For overcoming the traditional misbelief that integrity can be checked efficiently for a transaction T only if the state before T totally complies with all constraints, we have revisited the work in [18]. There, it has been shown that the total integrity postulate can be waived without further ado, for most (though not all) integrity checking methods. Fortunately, the postulate also is unnecessary for deferred checking of key constraints and other common built-in integrity constructs in DBMSs on the market.

We have seen that the advantages of making the total integrity postulate dispensable even extend to relaxing the isolated integrity postulate. More precisely, the use of an uncertainty-tolerant quality checking method to enforce business rules for concurrent serializable transactions guarantees that no transaction can violate any instance of any constraint that has been satisfied in the state before committing if all transactions preserve the integrity of the same instance in isolation. Conversely stated, our result guarantees that, if any violation happens, then no transaction that has been correctly and successfully checked for integrity preservation by an inconsistency-tolerant method can be held responsible for that. The most interesting aspect of this result is that it even holds in the presence of extant uncertain data that violates any integrity constraint.

6 Outlook to Future Work

Important contemporary areas where uncertainty tolerance is paramount are systems for streaming data [7], linked data [8] and big data [47], as well as hybrid approaches for approximate reasoning [2]. In the data stores of such systems, data integrity and the isolation level of concurrent transactions usually is severely compromised. We have seen that, in general, more research is needed for systems involving such compromises. In particular, for non-serializable histories of concurrent transactions, it should be interesting to elaborate a precise theory of different kinds of database states. Such a theory should allow to differentiate between states that are committed, states that are "seen" by a transaction and states that are "seen" by (human or programmed) agents that have issued the transaction. and which consistency guarantees can be made by which methods for transitions between those states. This area of research is important because

most commercial database management systems compromise the isolation level of transactions in favor of a higher transaction throughput, while leaving the problem of integrity preservation to the application programmers. First steps in this direction have been proposed in [19].

Another important, possibly even more difficult area of upcoming research is that of providing uncertainty-containing transactions not only in distributed and replicated systems with remote clients and servers, but also for databases in the cloud, for big volumes of data and for No-SQL data stores. These are going to be the objectives of impending projects. So far, there are only some special-purpose solutions (e.g., [49]), which lack genericity (or, at least, the generalizability of which is less than obvious). After all, a move away from the universality-obsessed attitude toward solutions to technical problems in the field of databases will be the way of the future [42].

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. Ajith Abraham. Hybrid approaches for approximate reasoning. *Journal of Intelligent and Fuzzy Systems* 23(2-3):41–42, 2012.
3. Rudolf Bayer. Integrity, concurrency, and recovery in databases. In *ECI*, pages 79–106, 1976.
4. Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ansi sql isolation levels. In *SIGMOD Conference*, pages 1–10, 1995.
5. Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
6. Stefan Böttcher. Improving the concurrency of integrity checks and write operations. In Serge Abiteboul and Paris C. Kanellakis, editors, *ICDT'90, Third International Conference on Database Theory, Paris, France, December 12-14, 1990, Proceedings*, volume 470 of *Lecture Notes in Computer Science*, pages 259–273. Springer, 1990.
7. Alfredo Cuzzocrea. Optimization issues of querying and evolving sensor and stream databases. *Information Systems*, 39:196–198, 2014.
8. Alfredo Cuzzocrea, Carson Kai-Sang Leung and Syed Khairuzzaman Tanbeer. Mining of Diverse Social Entities from Linked Data. In K. Selçuk Candan, Sihem Amer-Yahia, Nicole Schweikardt, Vassilis Christophides, Vincent Leroy (Eds.): *Proc. Workshops of the EDBT/ICDT 2014 Joint Conference*, CEUR Workshop Proceedings, 269–274, 2014.
9. Alfredo Cuzzocrea, Rubén de Juan-Marín, Hendrik Decker, and Francesc D. Muñoz-Escoí. Managing uncertainty in databases and scaling it up to concurrent transactions. In *SUM*, pages 30–43, 2012.
10. C. J. Date. *What not how: the business rules approach to application development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
11. Alexandre Decan, Fabian Pijcke, and Jef Wijsen. Certain conjunctive query answering in sql. In *SUM*, pages 154–167, 2012.
12. Hendrik Decker. Causes for inconsistency-tolerant schema update management. In *ICDE Workshops*, pages 157–161, 2011.

13. Hendrik Decker. Causes of the violation of integrity constraints for supporting the quality of databases. In *ICCSA (5)*, pages 283–292, 2011.
14. Hendrik Decker. Data quality maintenance by integrity-preserving repairs that tolerate inconsistency. In *QSIC*, pages 192–197, 2011.
15. Hendrik Decker. Partial repairs that tolerate inconsistency. In *ADBIS*, pages 389–400, 2011.
16. Hendrik Decker. New measures for maintaining the quality of databases. In *ICCSA (4)*, pages 170–185, 2012.
17. Hendrik Decker. Answers that have quality. In *ICCSA (2)*, pages 543–558, 2013.
18. Hendrik Decker and Davide Martinenghi. Inconsistency-tolerant integrity checking. *IEEE Trans. Knowl. Data Eng.*, 23(2):218–234, 2011.
19. Hendrik Decker and Francesc D. Muñoz-Escoí. Revisiting and improving a result on integrity preservation by concurrent transactions. In *OTM Workshops*, pages 297–306, 2010.
20. Kapali P. Eswaran and Donald D. Chamberlin. Functional specifications of subsystem for database integrity. In Douglas S. Kerr, editor, *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*, pages 48–68. ACM, 1975.
21. Kapali P. Eswaran, Jim Gray, Raymond A. Lorie, and Irving L. Traiger. The notions of consistency and predicate locks in a database system. *Commun. ACM*, 19(11):624–633, 1976.
22. G. Gardarin. Integrity, consistency, concurrency, reliability in distributed database management systems. In *Distributed Databases*, pages 335–351. 1980.
23. Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.
24. J. Gray, R. Lorie, and G. Putzolu. Granularity of locks in a shared data base. In *Very Large Data Bases, 1st International Conference*, pages 428–451. ACM Press, 1975.
25. Jim Gray. The transaction concept: Virtues and limitations (invited paper). In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 144–154. IEEE Computer Society, 1981.
26. Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1992.
27. Paul W. P. J. Grefen. Combining theory and practice in integrity control: A declarative approach to the specification of a transaction modification subsystem. In Rakesh Agrawal, Seán Baker, and David A. Bell, editors, *19th International Conference on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland, Proceedings*, pages 581–591. Morgan Kaufmann, 1993.
28. M. Hammer and D. McLeod. Semantic integrity in a relational data base system. In *Very Large Data Bases, 1st International Conference*, pages 25–47. ACM Press, 1975.
29. H. Ibrahim. Checking integrity constraints - how it differs in centralized, distributed and parallel databases. In *DEXA Workshops*, pages 563–568, 2006.
30. Charles T. Davies Jr. Data processing spheres of control. *IBM Systems Journal*, 17(2):179–198, 1978.
31. Nancy A. Lynch, Barbara T. Blaustein, and Michael Siegel. Correctness conditions for highly available replicated databases. In *PODC*, pages 11–28, 1986.
32. Davide Martinenghi and Henning Christiansen. Transaction management with integrity checking. In *DEXA*, pages 606–615, 2005.

33. Davide Martinenghi, Henning Christiansen, and Hendrik Decker. Integrity checking and maintenance in relational and deductive databases and beyond. In *In Zongmin Ma, editor, Intelligent Databases: Technologies and Applications*, pages 238–285. Idea Group Publishing, 2006.
34. Tony Morgan. *Business Rules and Information Systems: Aligning IT with Business Goals (Unisys Series)*. Addison-Wesley Professional, 2002.
35. Francesc D. Muñoz-Escoí, María Idoia Ruiz-Fuertes, Hendrik Decker, José Enrique Armendáriz-Iñigo, and José Ramón González de Mendívil. Extending middleware protocols for database replication with integrity support. In *OTM Conferences (1)*, pages 607–624, 2008.
36. J.-M. Nicolas. Logic for improving integrity checking in relational data bases. *Acta Informatica*, (18):227–253, 1982.
37. http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_10003.htm, Oracle, 2013.
38. L. Pipino, Y. Lee, and R. Yang. Data quality assessment. *Commun. ACM*, 45(4):211–218, 2002.
39. Ronald G Ross. *Business Rule Concepts: Getting to the Point of Knowledge. 2nd ed.* 1998.
40. Ronald G. Ross. *Principles of the Business Rule Approach (Addison-Wesley Information Technology Series)*. Addison-Wesley Professional, 2003.
41. Abraham Silberschatz and Zvi M. Kedem. Consistency in hierarchical database systems. *J. ACM*, 27(1):72–80, 1980.
42. Michael Stonebraker. Technical perspective - one size fits all: an idea whose time has come and gone. *Commun. ACM*, 51(12):76, 2008.
43. Michael Stonebraker. Errors in Database Systems, Eventual Consistency, and the CAP Theorem, 2010.
44. Michael Stonebraker. In search of database consistency. *Commun. ACM*, 53(10):8–9, 2010.
45. K. Vidyasankar. Serializability. In *Encyclopedia of Database Systems*, pages 2626–2632. 2009.
46. Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.
47. Gerhard Weikum. Wheres the Data in the Big Data Wave? ACM SIGMOD Blog, 2013. http://wp.sigmod.org/?p=786
48. Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.
49. Roverli Pereira Ziwich, Elias Procópio Duarte Jr., and Luiz Carlos Pessoa Albini. Distributed integrity checking for systems with replicated data. In *ICPADS (1)*, pages 363–369, 2005.